ТЕХНОСФЕРА

Лекция №2

Многопоточное программирование

Дмитрий Калугин-Балашов

Литература

Стивенс У.

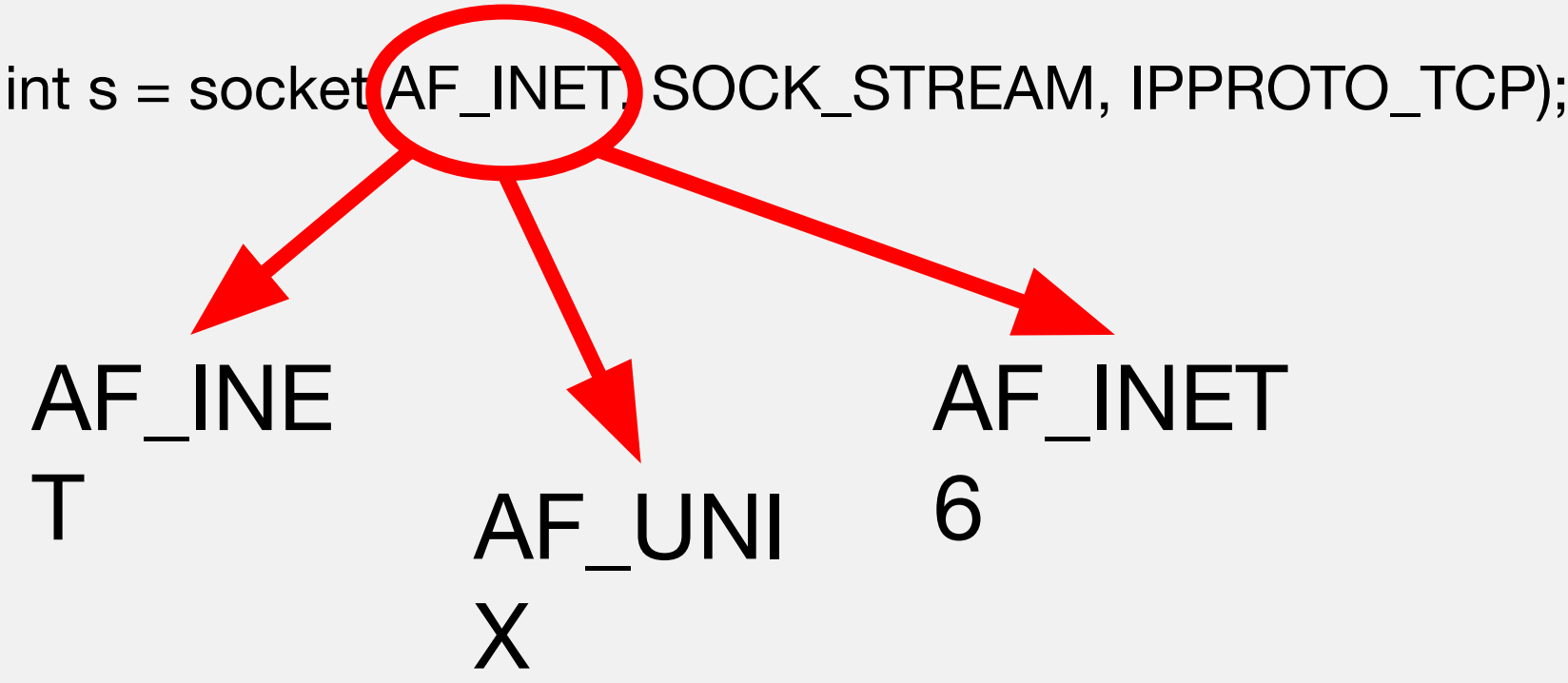
UNIX. Разработка сетевых приложений.

W. Richard Stevens.

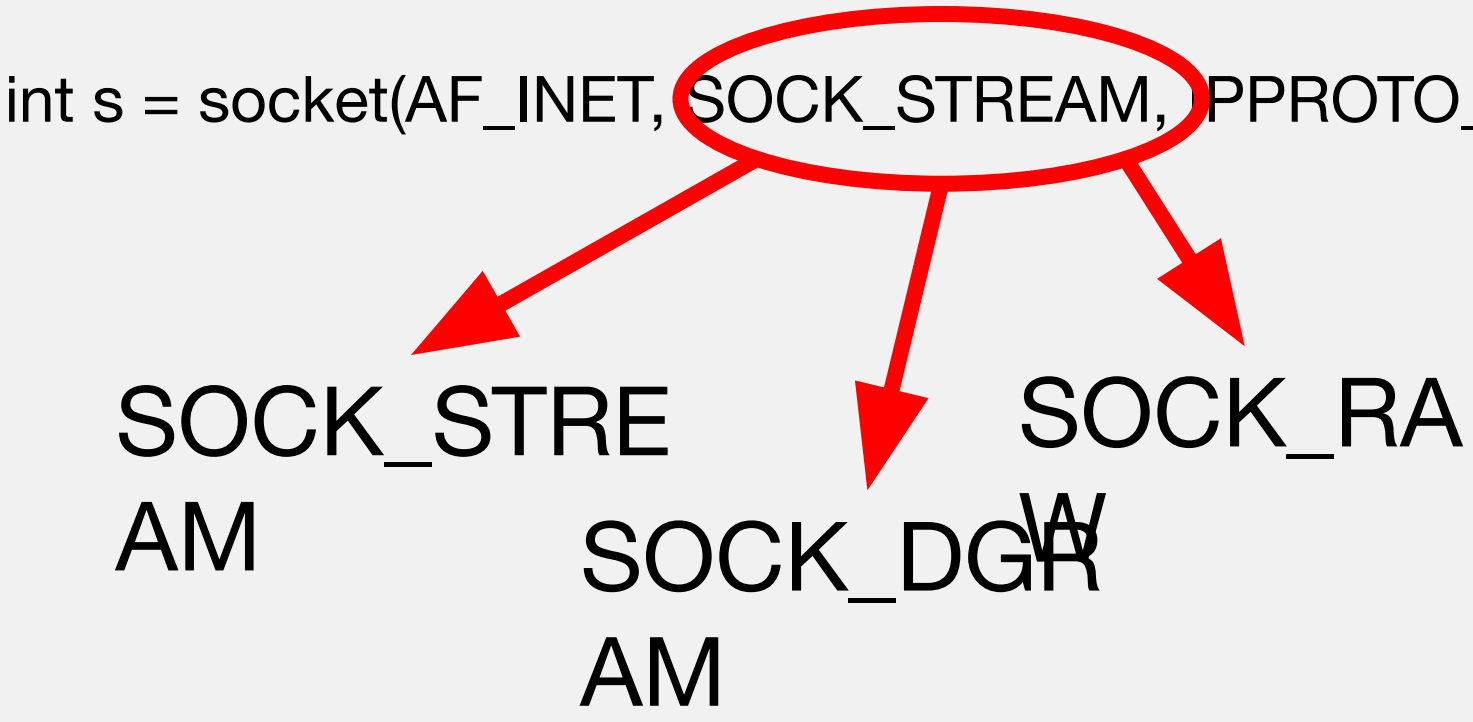
UNIX Network Programming

Сокеты Беркли

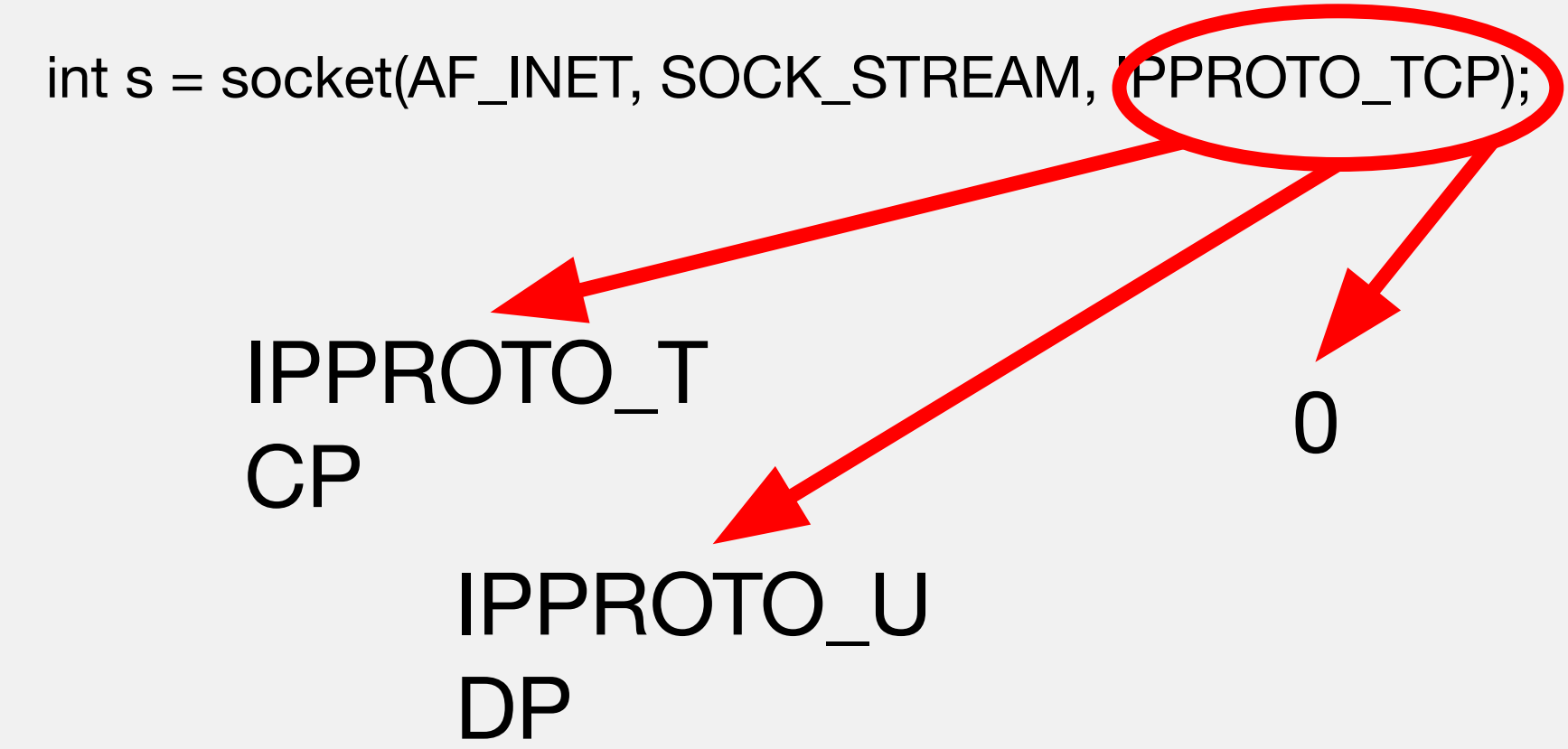
Сокеты Беркли



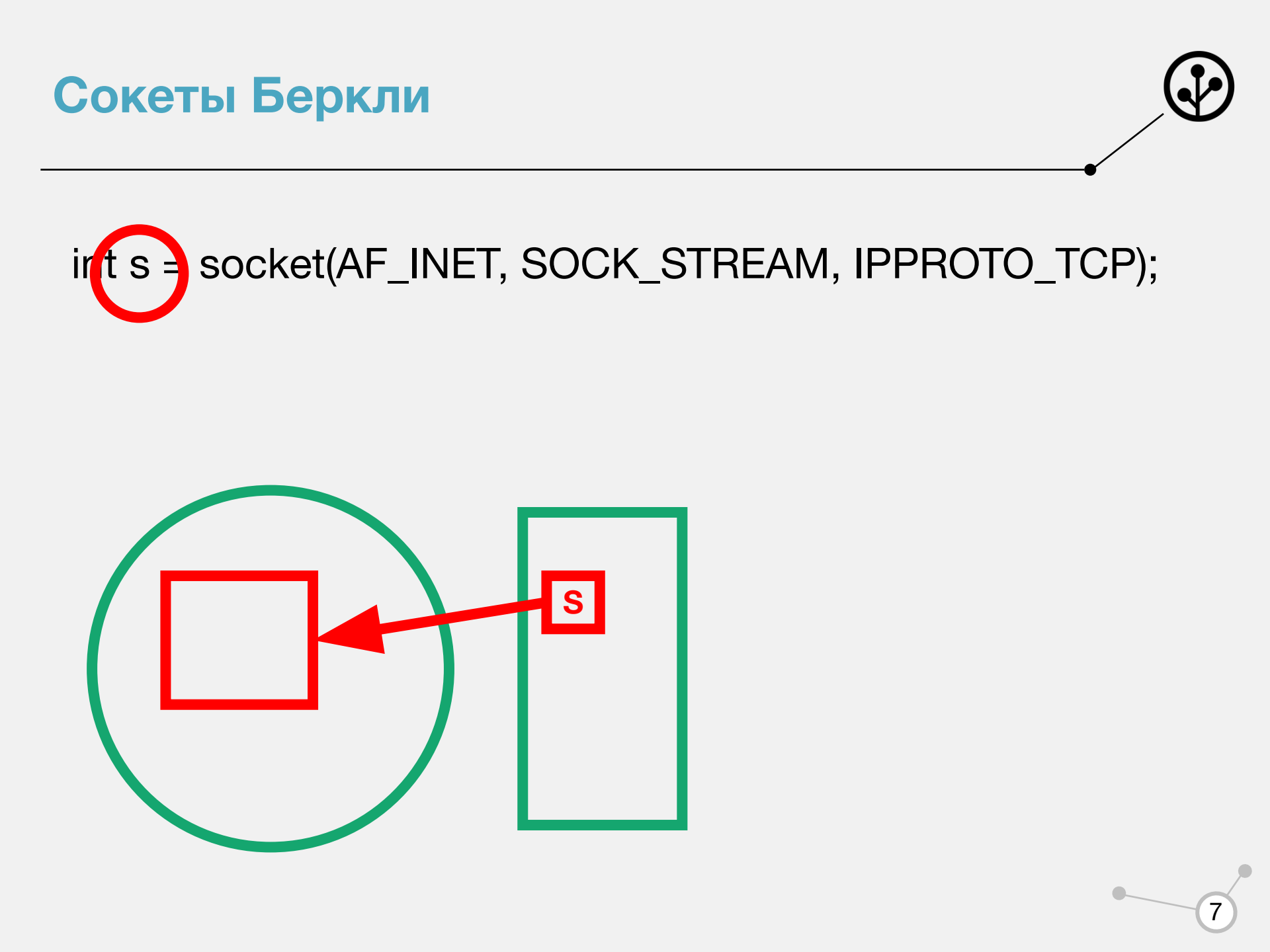
Сокеты Беркли



Сокеты Беркли

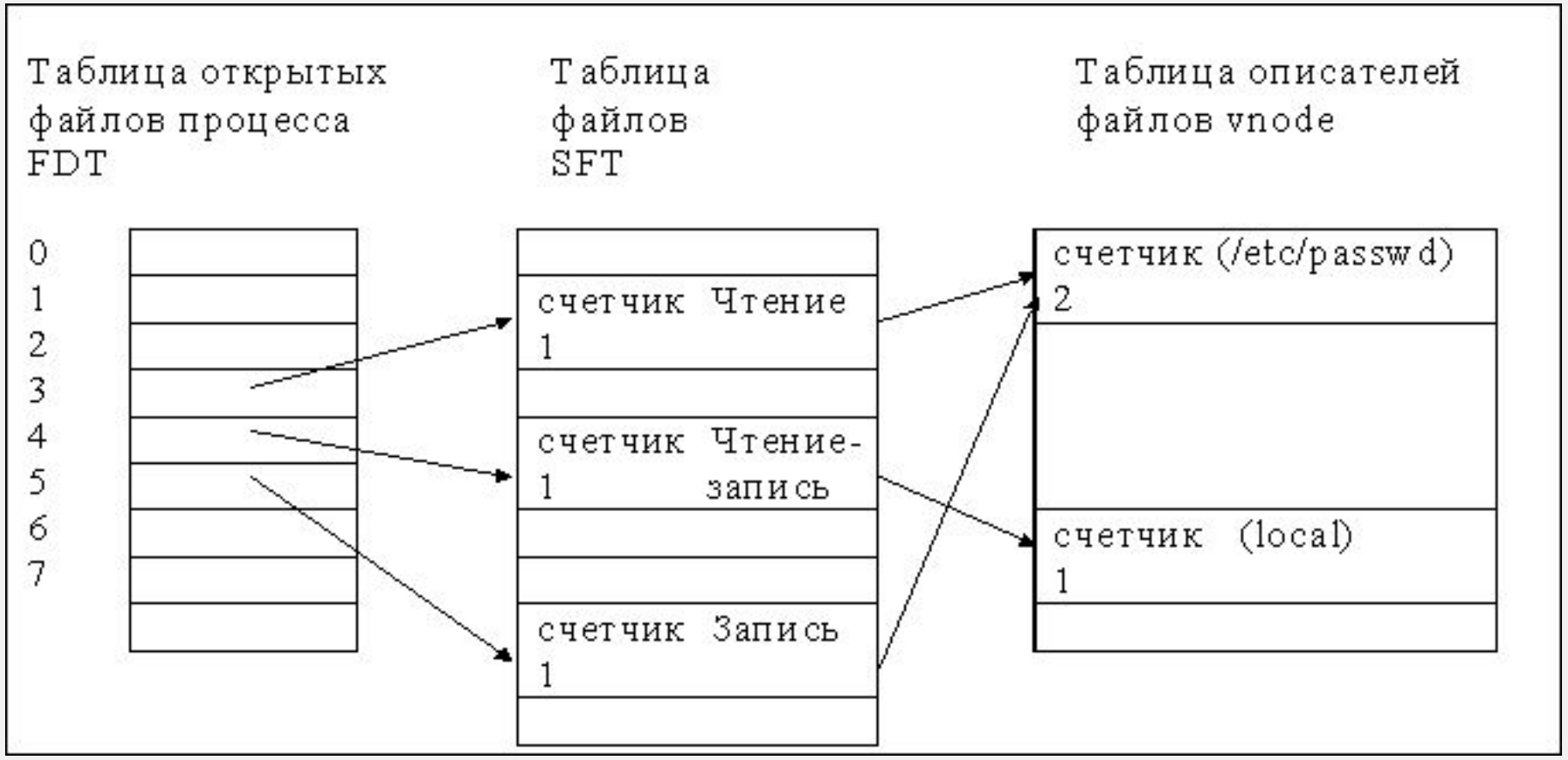


Сокеты Беркли

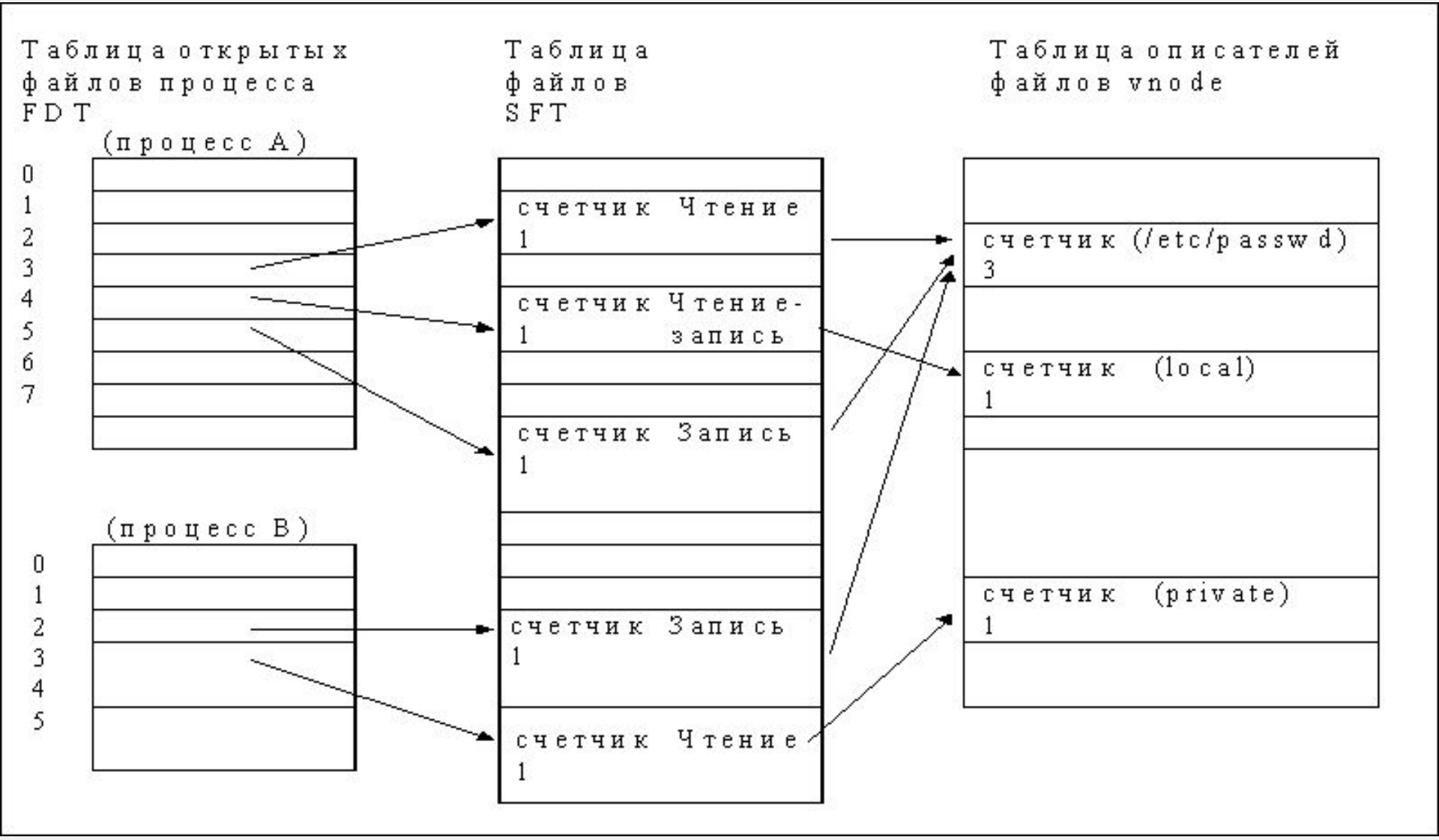


```
int s = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
```

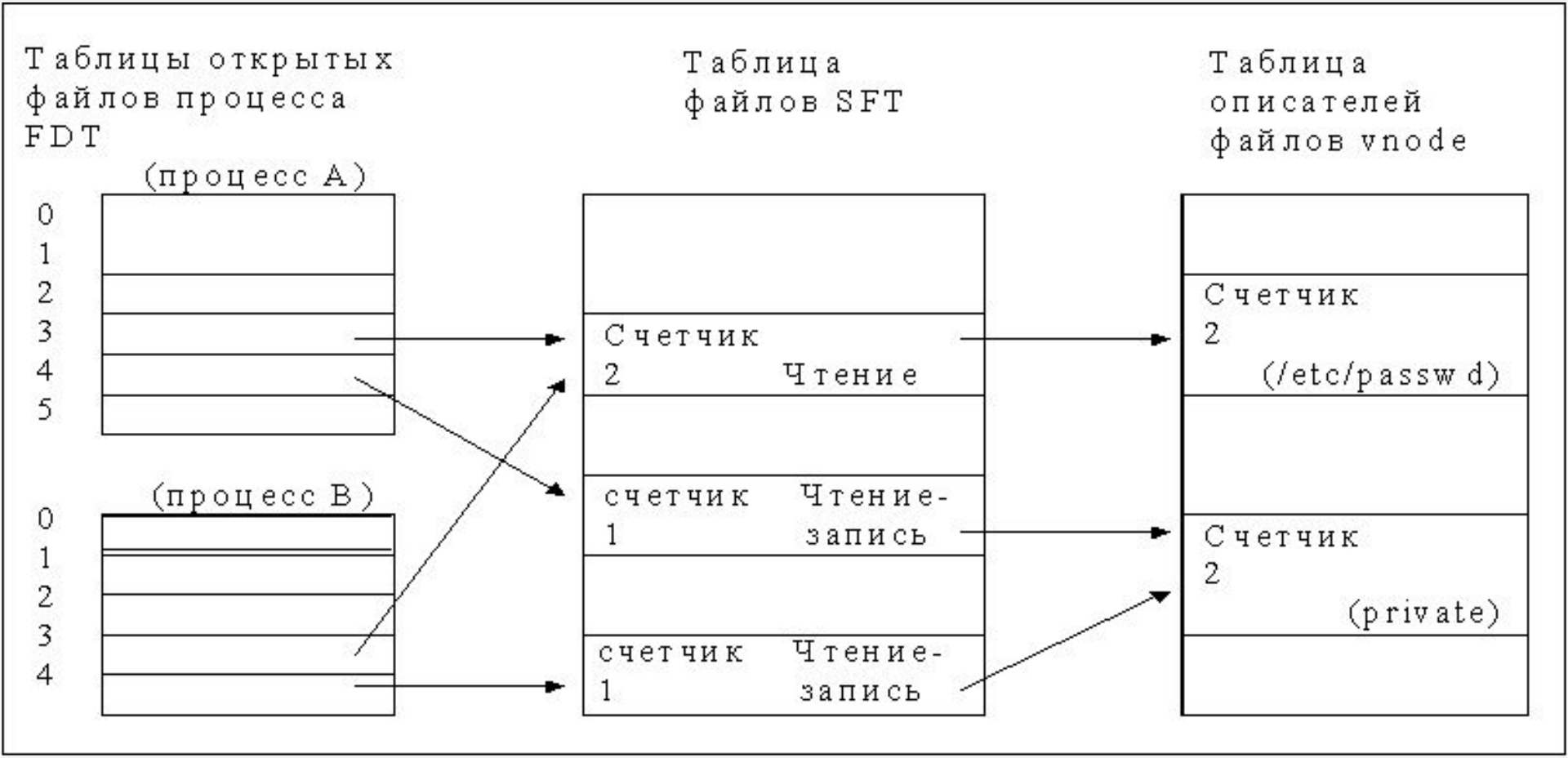
Сокеты Беркли



Сокеты Беркли



Сокеты Беркли



Сокеты Беркли

```
bind(s, (struct sockaddr *)sa, sizeof(sa));
```

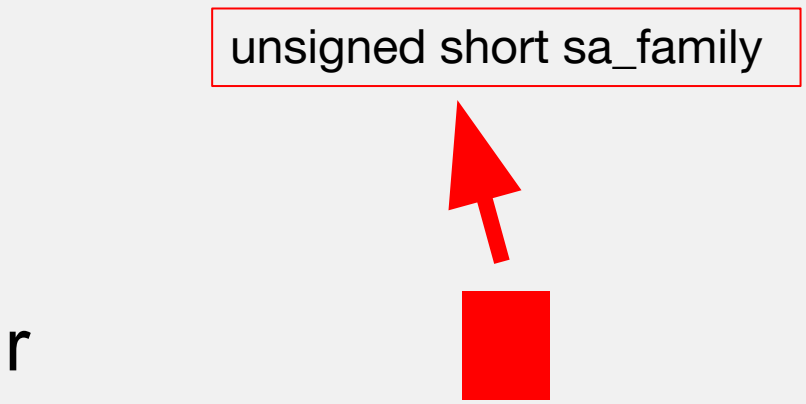
Сокеты Беркли

```
bind(s, (struct sockaddr *)sa, sizeof(sa));
```

Сокеты Беркли

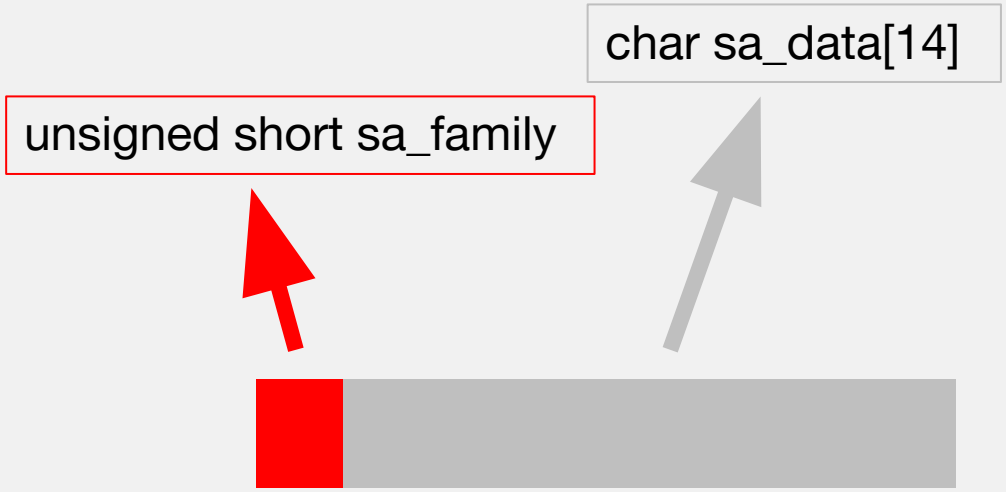
struct sockaddr

Сокеты Беркли



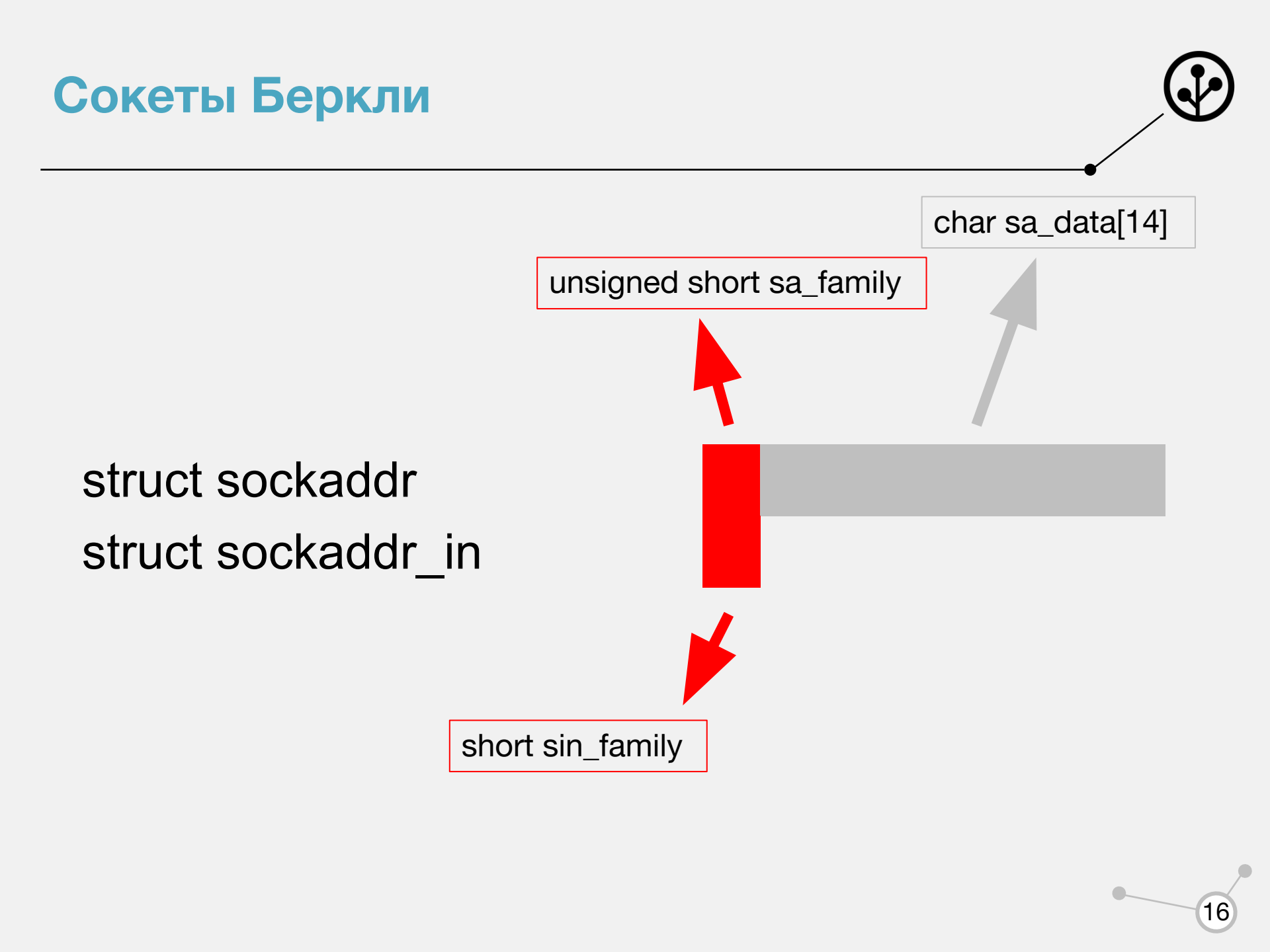
struct sockaddr

Сокеты Беркли



struct sockaddr

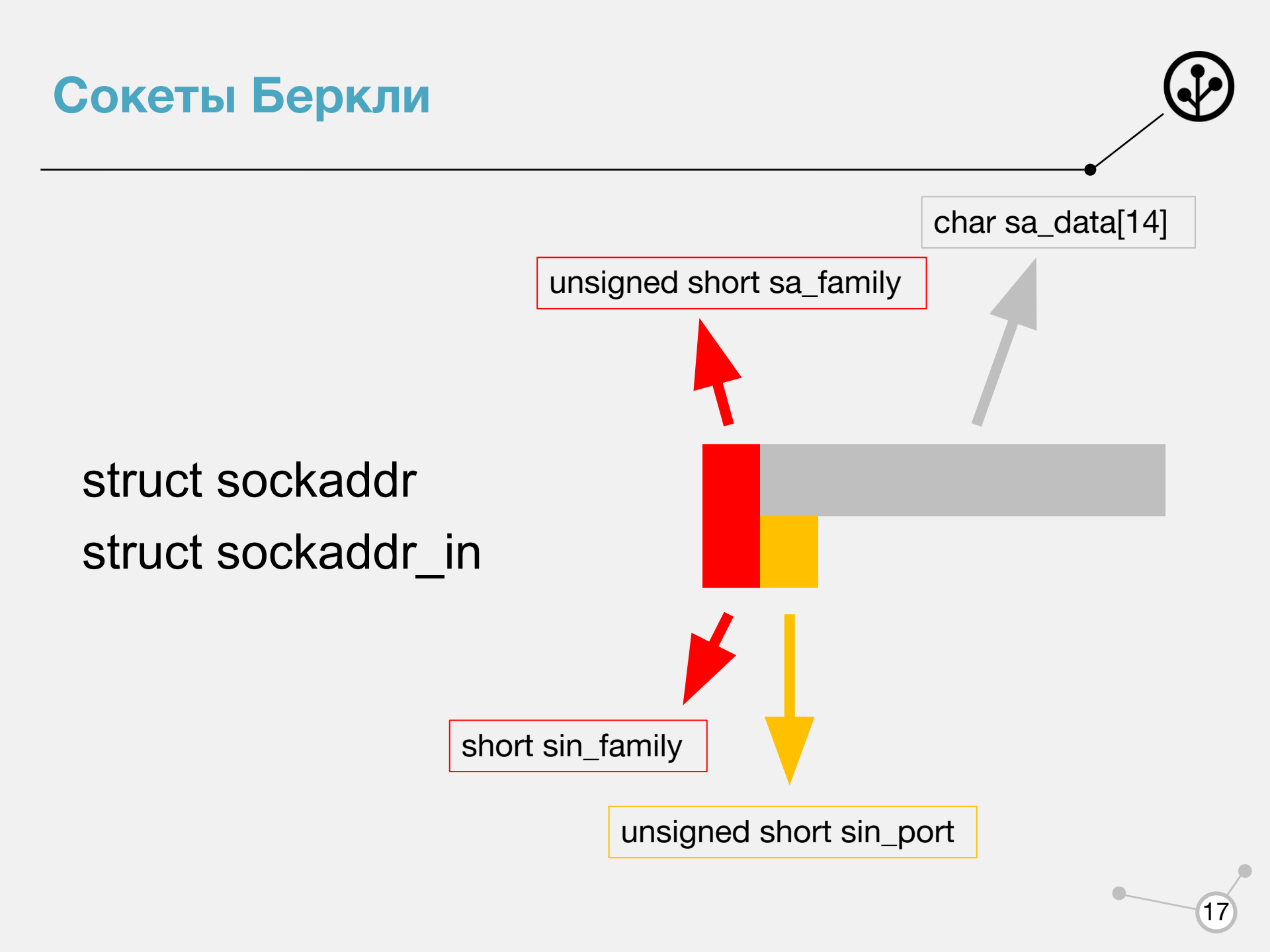
Сокеты Беркли



struct sockaddr

struct sockaddr_in

Сокеты Беркли



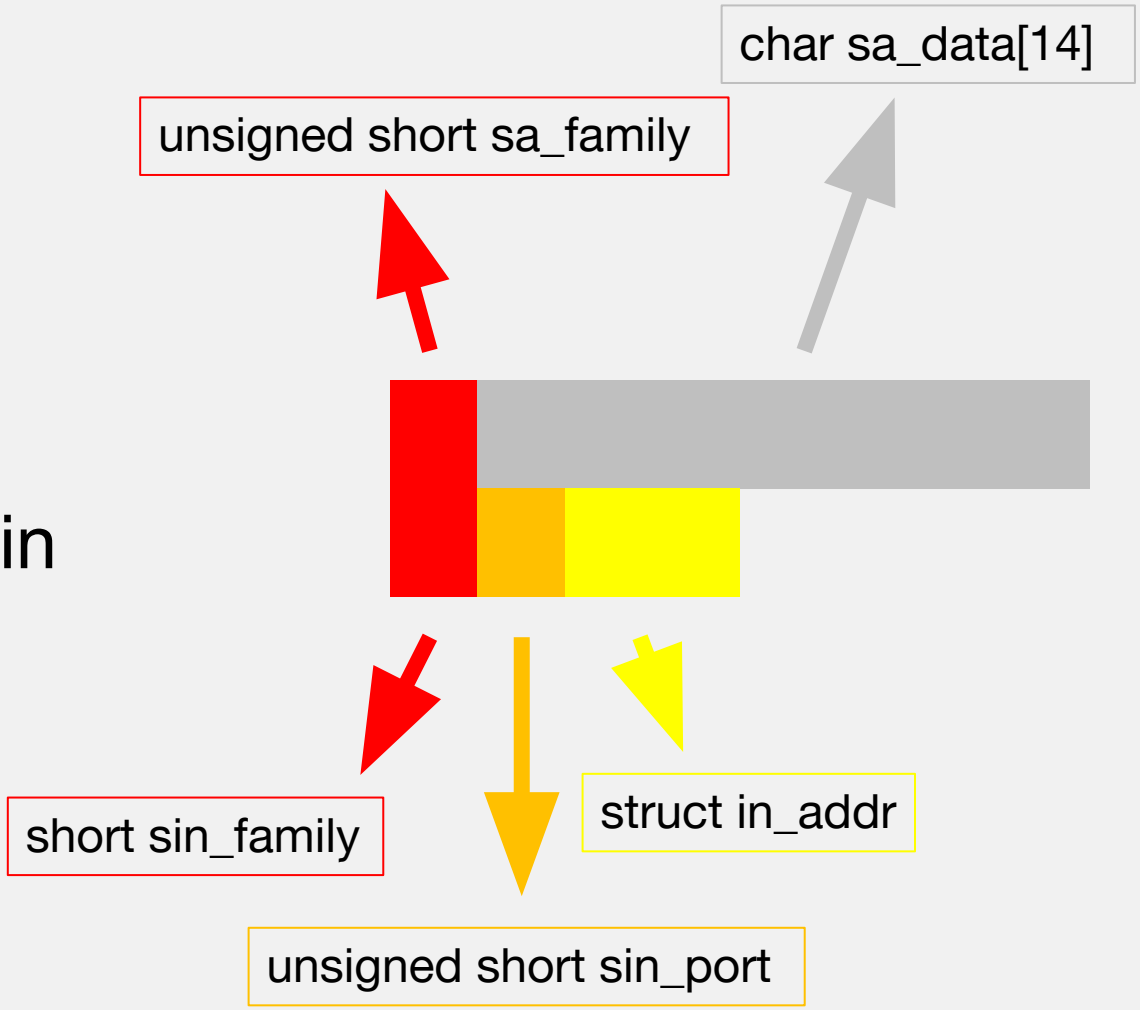
struct sockaddr

struct sockaddr_in

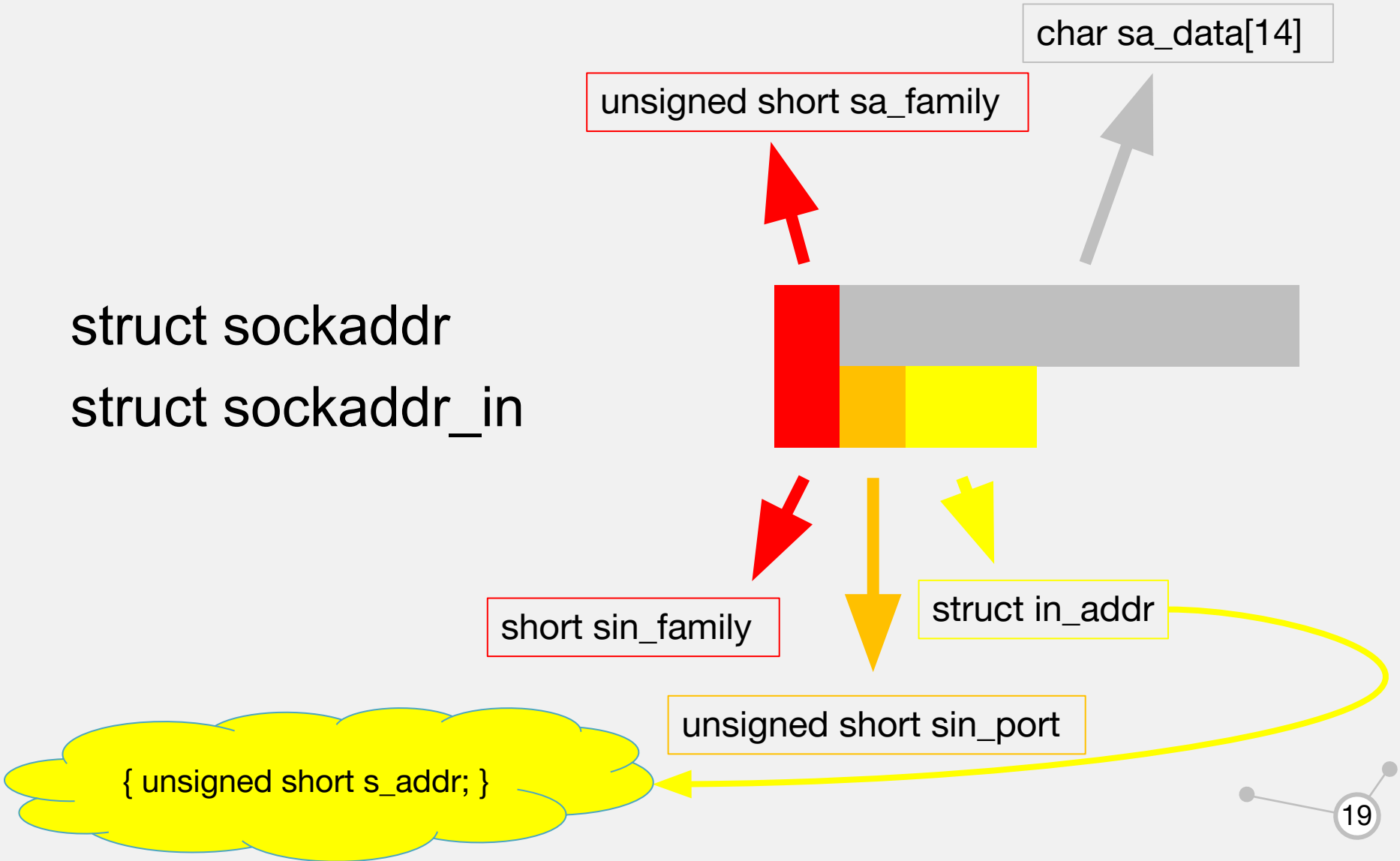
Сокеты Беркли

struct sockaddr

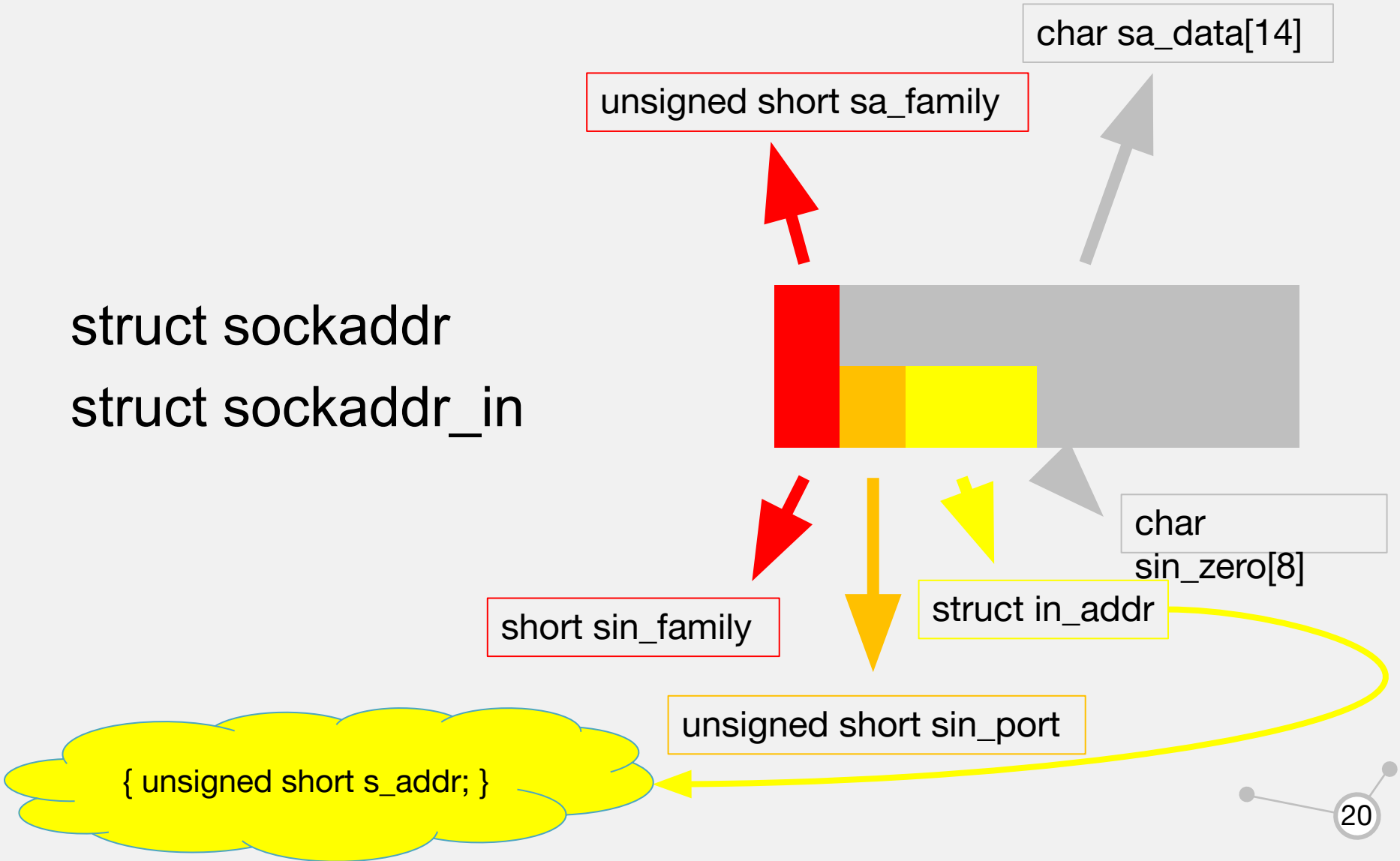
struct sockaddr_in



Сокеты Беркли



Сокеты Беркли



IPv4

Заполнение структуры sockaddr_in

```
1.    struct sockaddr_in SockAddr;
```

IPv4

Заполнение структуры sockaddr_in

```
1.    struct sockaddr_in SockAddr;
2.    memset(&SockAddr, 0, sizeof(SockAddr));
```

Заполнение структуры sockaddr_in

```
1.  struct sockaddr_in SockAddr;
2.  // memset(&SockAddr, 0, sizeof(SockAddr));
3.  bzero(&SockAddr, sizeof(SockAddr));
```

Заполнение структуры sockaddr_in

```
1.  struct sockaddr_in SockAddr;
2.  // memset(&SockAddr, 0, sizeof(SockAddr));
3.  bzero(&SockAddr, sizeof(SockAddr));
4.  SockAddr.sin_family = AF_INET;
```

IPv4

Заполнение структуры sockaddr_in

```
1.  struct sockaddr_in SockAddr;
2.  // memset(&SockAddr, 0, sizeof(SockAddr));
3.  bzero(&SockAddr, sizeof(SockAddr));
4.  SockAddr.sin_family = AF_INET;
5.  SockAddr.sin_port = htons(12345);
```

IPv4

Заполнение структуры sockaddr_in

```
1.  struct sockaddr_in SockAddr;
2.  // memset(&SockAddr, 0, sizeof(SockAddr));
3.  bzero(&SockAddr, sizeof(SockAddr));
4.  SockAddr.sin_family = AF_INET;
5.  SockAddr.sin_port = htons(12345);
6.  SockAddr.sin_addr.s_addr = htonl(INADDR_ANY);
```

Заполнение структуры sockaddr_in

```
1.  struct sockaddr_in SockAddr;
2.  // memset(&SockAddr, 0, sizeof(SockAddr));
3.  bzero(&SockAddr, sizeof(SockAddr));
4.  SockAddr.sin_family = AF_INET;
5.  SockAddr.sin_port = htons(12345);
6.  // SockAddr.sin_addr.s_addr = htonl(INADDR_ANY);
7.  SockAddr.sin_addr.s_addr = htonl(INADDR_LOOPBACK);
```

IPv4

Заполнение структуры sockaddr_in

```
1.  struct sockaddr_in SockAddr;
2.  // memset(&SockAddr, 0, sizeof(SockAddr));
3.  bzero(&SockAddr, sizeof(SockAddr));
4.  SockAddr.sin_family = AF_INET;
5.  SockAddr.sin_port = htons(12345);
6.  // SockAddr.sin_addr.s_addr = htonl(INADDR_ANY);
7.  // SockAddr.sin_addr.s_addr = htonl(INADDR_LOOPBACK);
8.  SockAddr.sin_addr.s_addr = inet_addr("178.63.66.215");
```

Заполнение структуры sockaddr_in

```
1.  struct sockaddr_in SockAddr;
2.  // memset(&SockAddr, 0, sizeof(SockAddr));
3.  bzero(&SockAddr, sizeof(SockAddr));
4.  SockAddr.sin_family = AF_INET;
5.  SockAddr.sin_port = htons(12345);
6.  // SockAddr.sin_addr.s_addr = htonl(INADDR_ANY);
7.  // SockAddr.sin_addr.s_addr = htonl(INADDR_LOOPBACK);
8.  // SockAddr.sin_addr.s_addr = inet_addr("178.63.66.215");
9.  struct hostent * hp = gethostbyname("rvncerr.org");
10. bcopy(hp->h_addr, &(SockAddr.sin_addr.s_addr), hp->h_length);
```

Заполнение структуры sockaddr_in

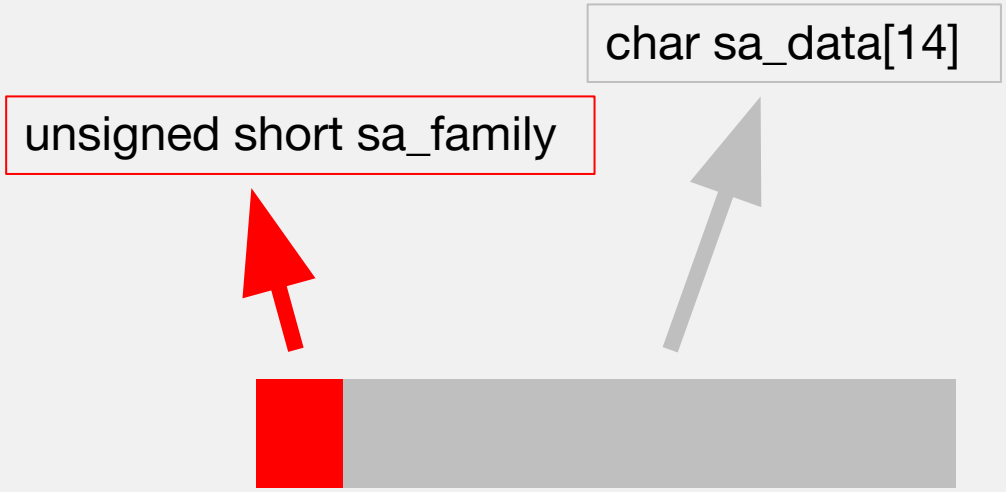
```
1.  struct hostent
2.  {
3.      char *h_name;
4.      char **h_aliases;
5.      int h_addrtype;
6.      int h_length;
7.      char **h_addr_list;
8.  }
```

IPv4

Заполнение структуры sockaddr_in

```
1.  struct hostent
2.  {
3.      char *h_name;
4.      char **h_aliases;
5.      int h_addrtype;
6.      int h_length;
7.      char **h_addr_list;
8.  }
9.  #define h_addr h_addr_list[0]
```

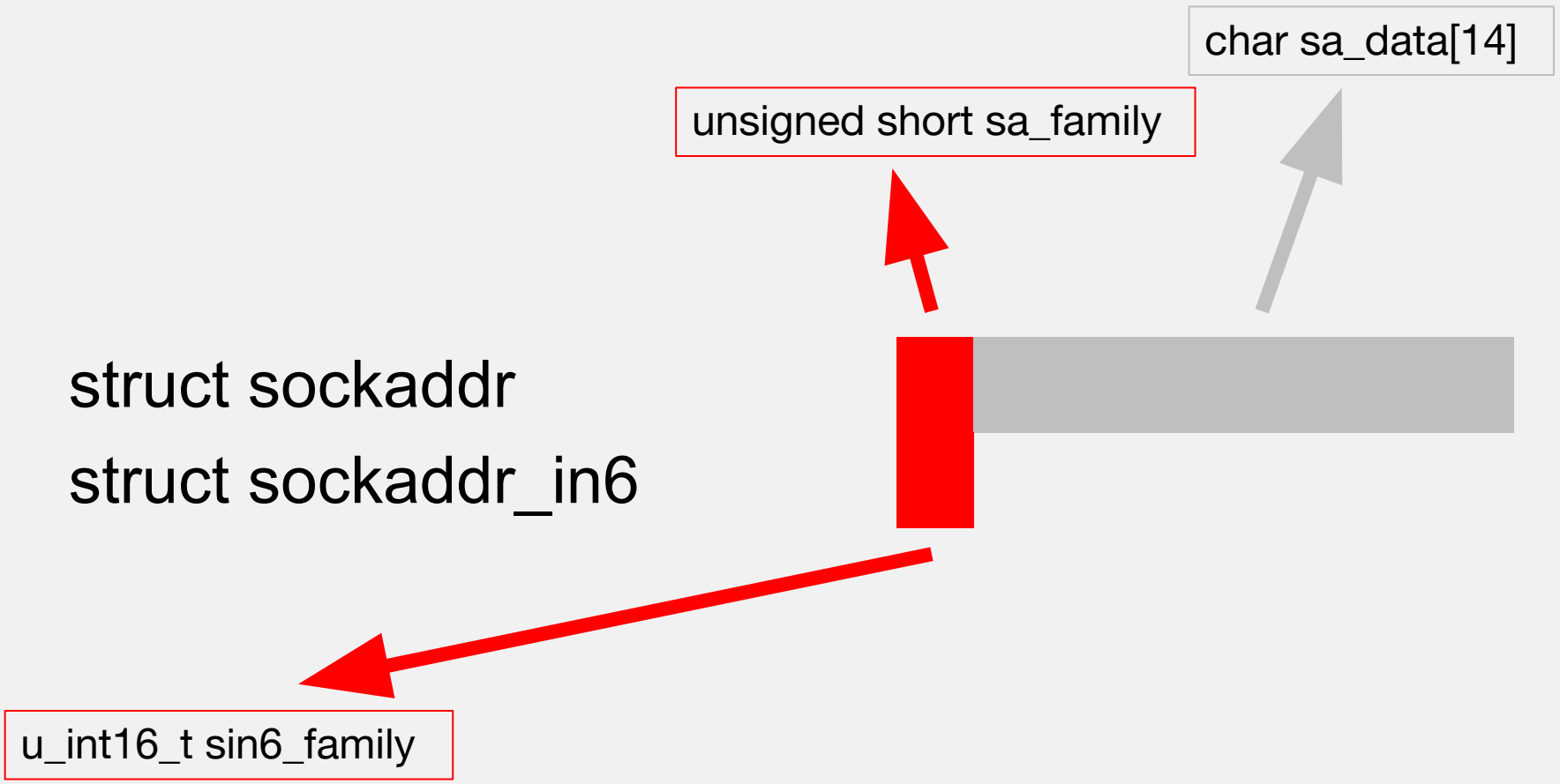
IPv6



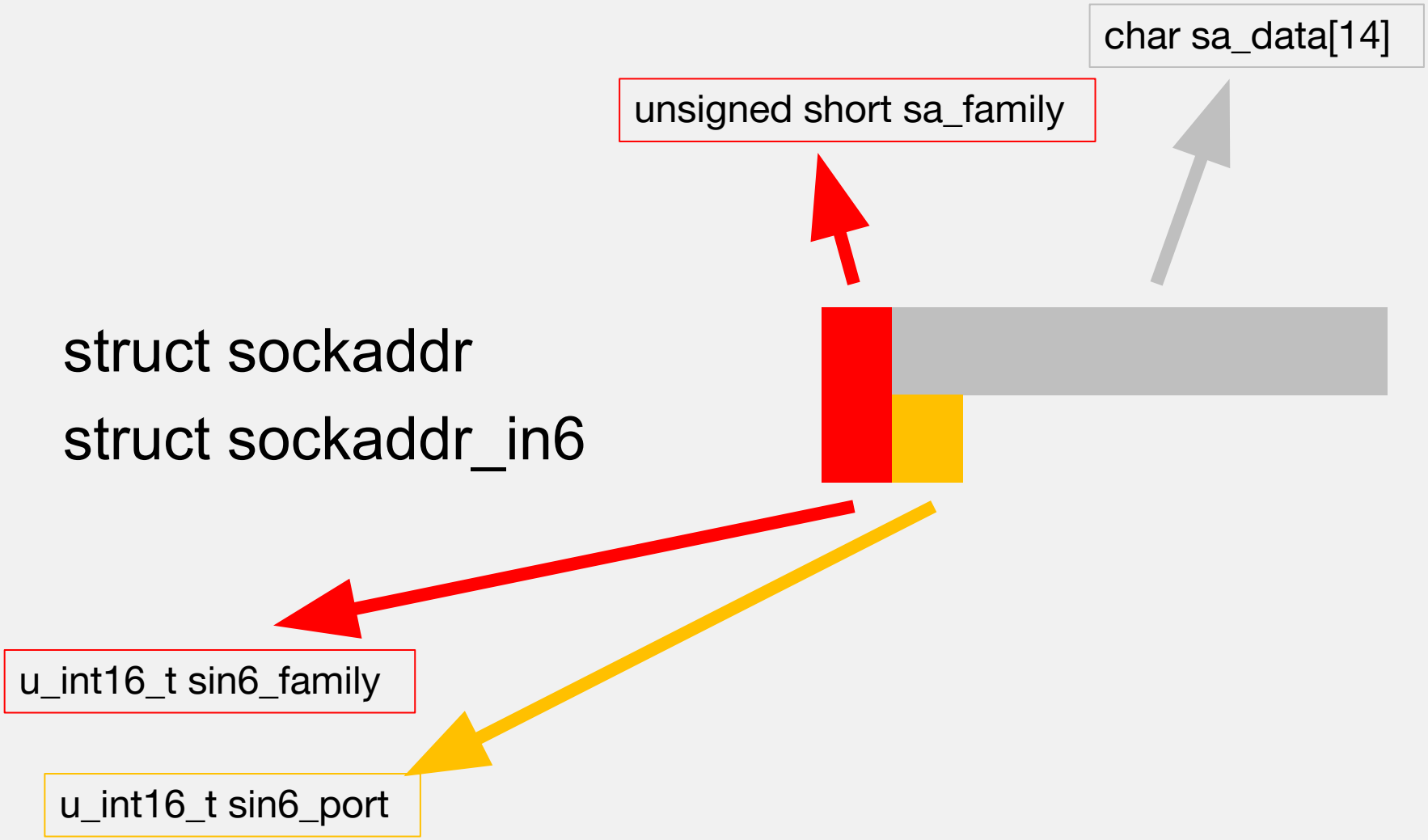
struct sockaddr

struct sockaddr_in6

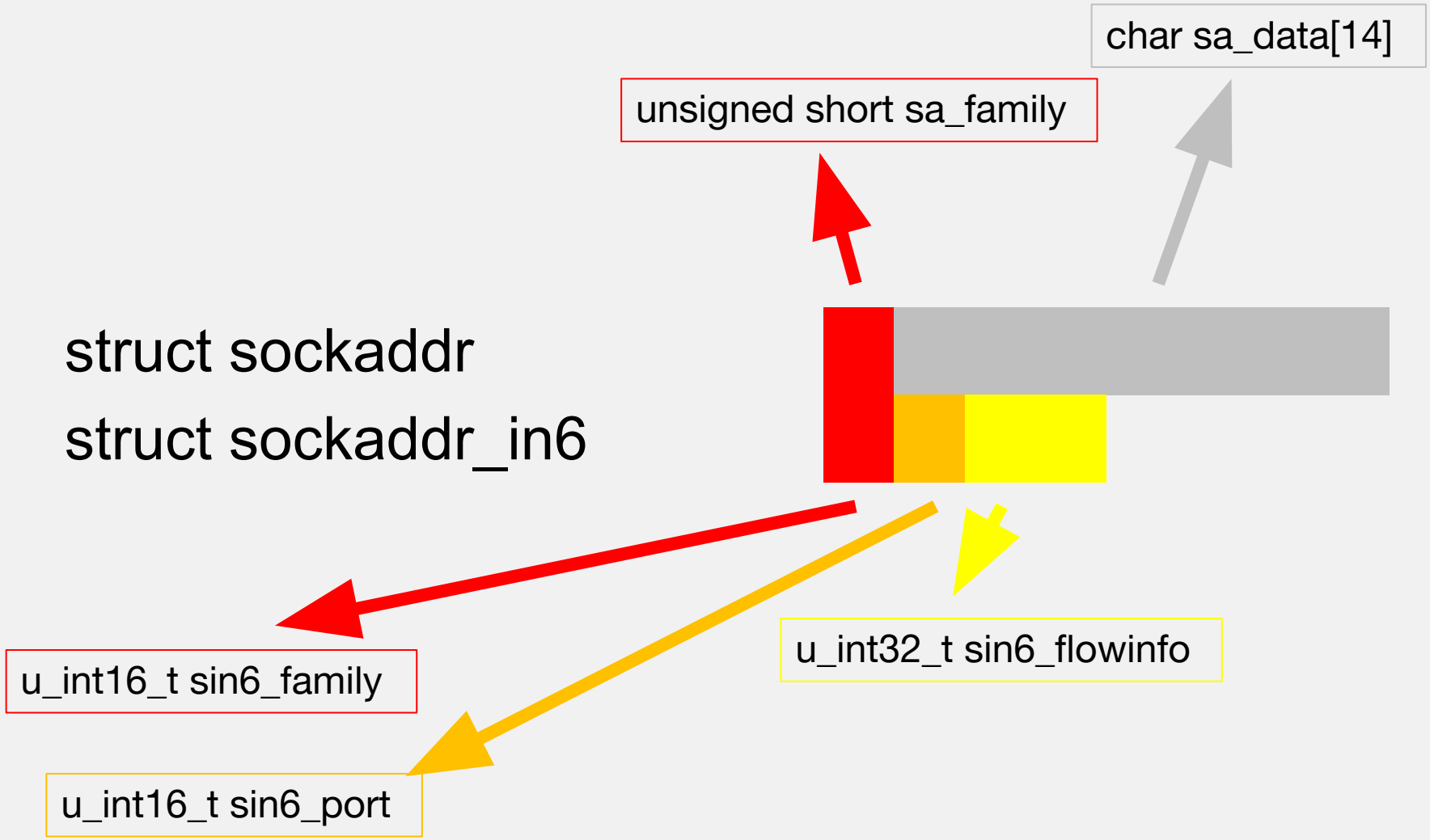
IPv6


char sa_data[14]
unsigned short sa_family
struct sockaddr
struct sockaddr_in6
u_int16_t sin6_family

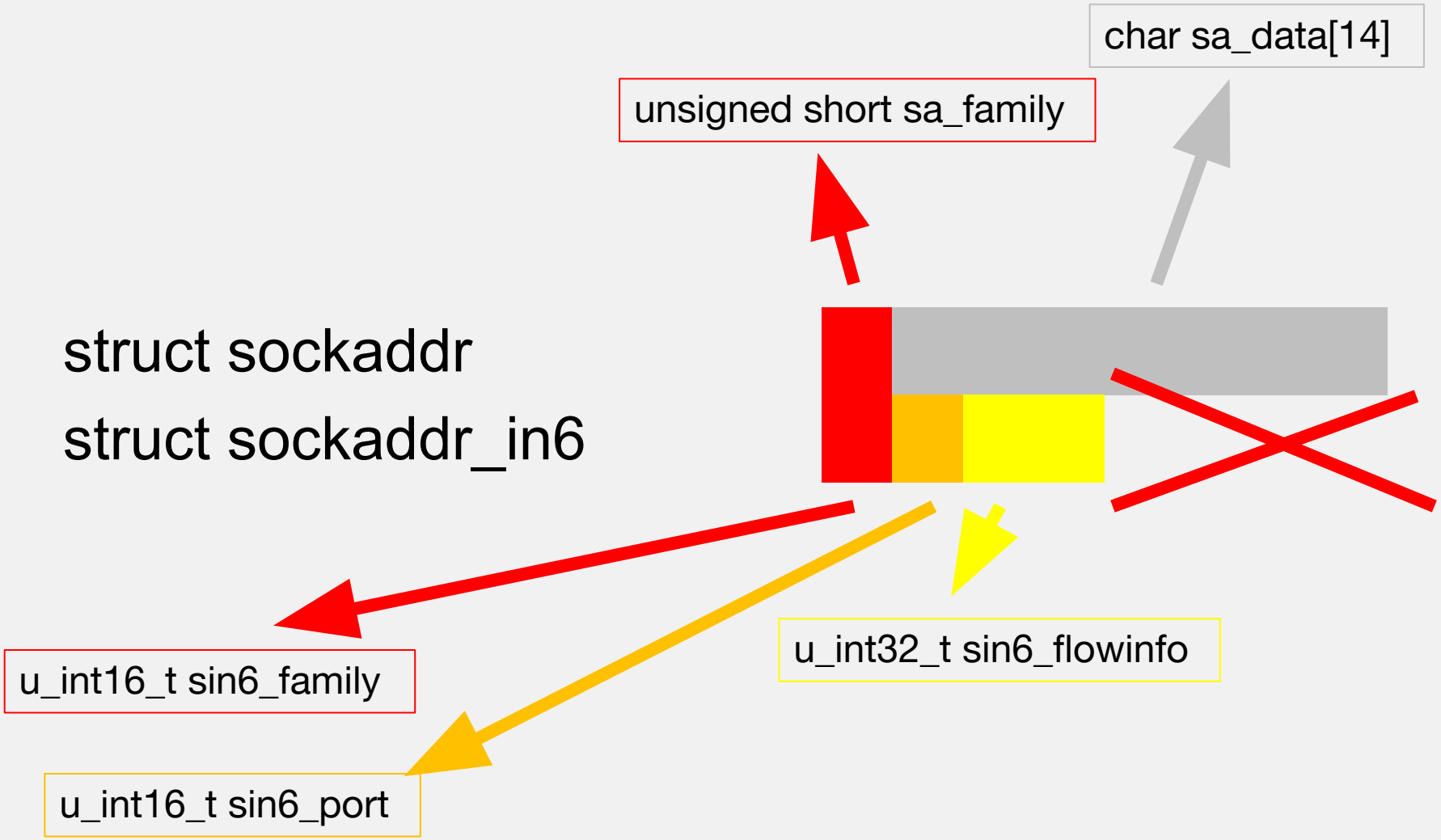
IPv6


char sa_data[14]
unsigned short sa_family
struct sockaddr
struct sockaddr_in6
u_int16_t sin6_family
u_int16_t sin6_port

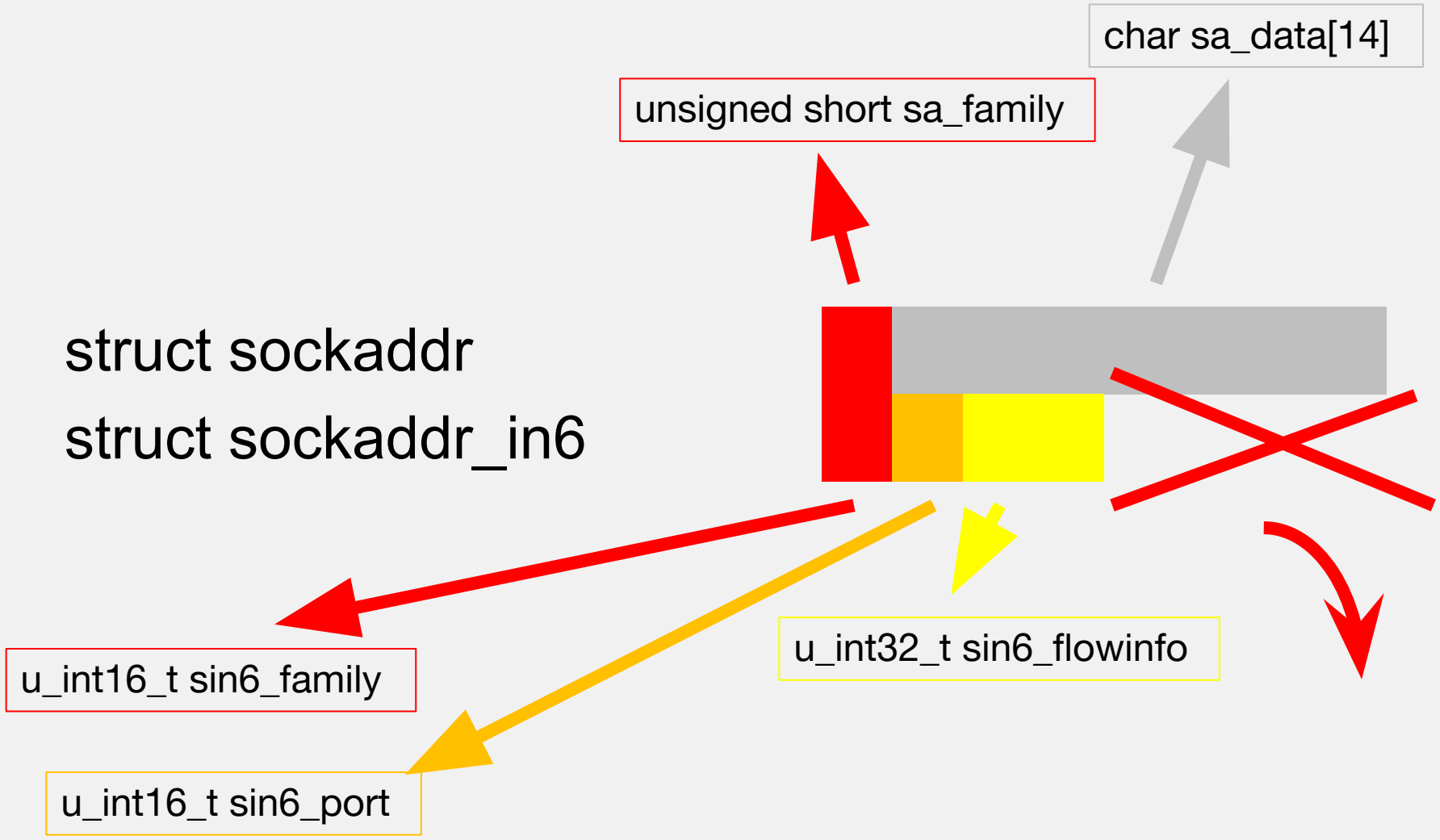
IPv6


char sa_data[14]
unsigned short sa_family
struct sockaddr
struct sockaddr_in6
u_int16_t sin6_family
u_int32_t sin6_flowinfo
u_int16_t sin6_port

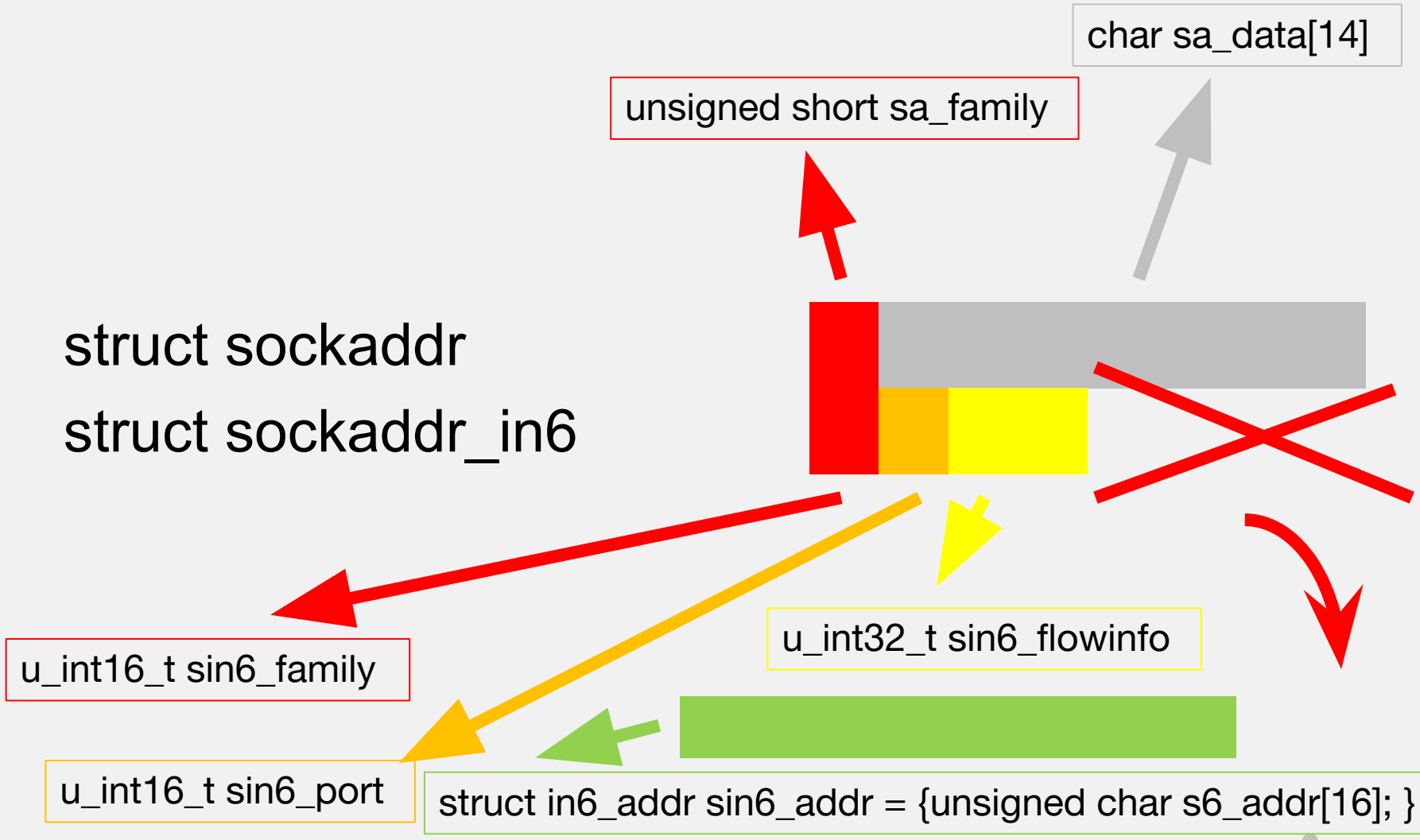
IPv6


char sa_data[14]
unsigned short sa_family
struct sockaddr
struct sockaddr_in6
u_int16_t sin6_family
u_int32_t sin6_flowinfo
u_int16_t sin6_port

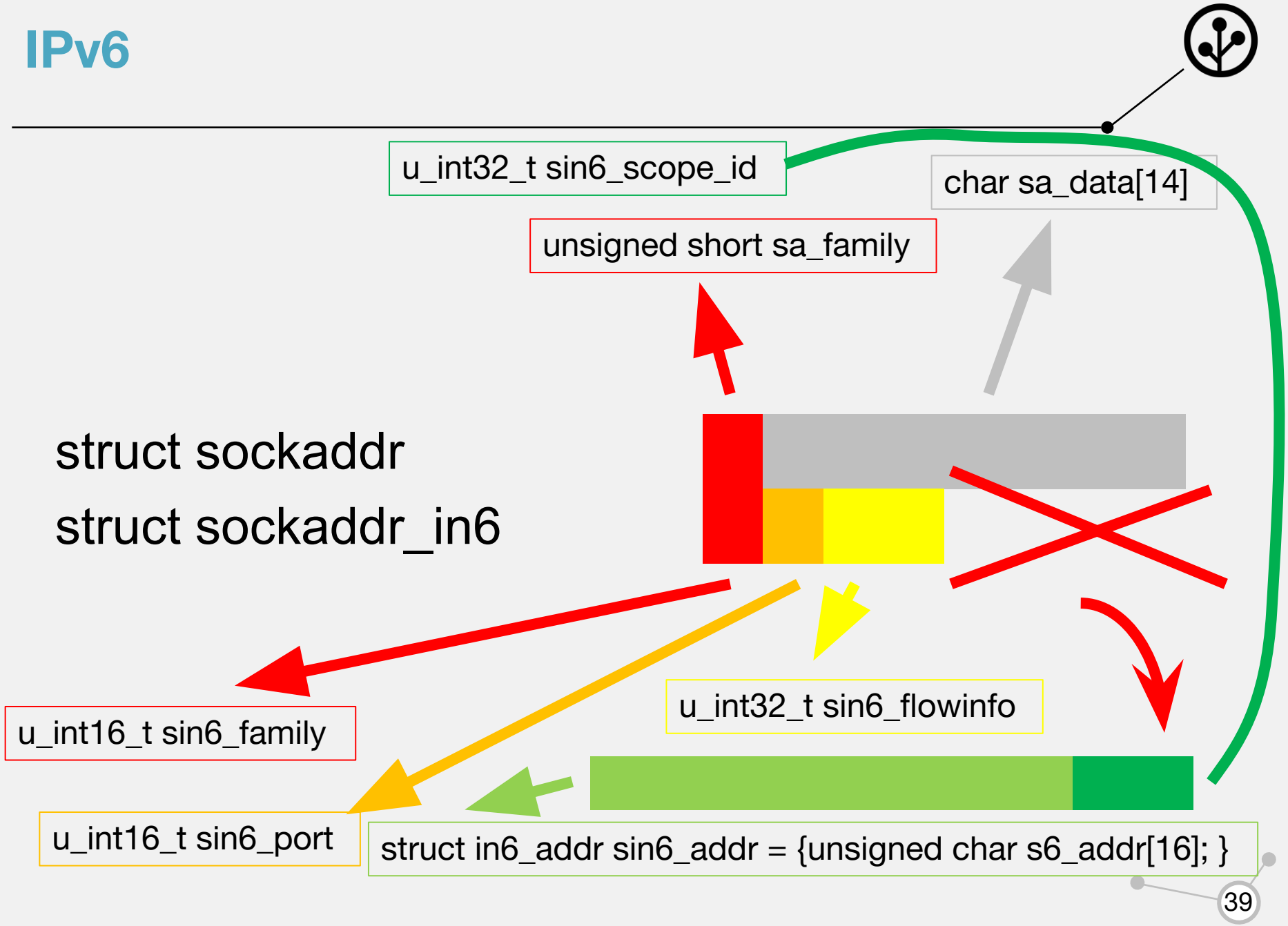
IPv6


char sa_data[14]
unsigned short sa_family
struct sockaddr
struct sockaddr_in6
u_int16_t sin6_family
u_int16_t sin6_port
u_int32_t sin6_flowinfo

IPv6


char sa_data[14]
unsigned short sa_family
struct sockaddr
struct sockaddr_in6
u_int16_t sin6_family
u_int32_t sin6_flowinfo
u_int16_t sin6_port
struct in6_addr sin6_addr = {unsigned char s6_addr[16]; }

IPv6


u_int32_t sin6_scope_id
char sa_data[14]
unsigned short sa_family
struct sockaddr
struct sockaddr_in6
u_int16_t sin6_family
u_int32_t sin6_flowinfo
u_int16_t sin6_port
struct in6_addr sin6_addr = {unsigned char s6_addr[16]; }

Заполнение структуры sockaddr_in6

```
1.    SockAddr.sin_addr.s_addr = inet_addr(“178.63.66.215”);
```

Заполнение структуры sockaddr_in6

```
1.    // SockAddr.sin_addr.s_addr = inet_addr("178.63.66.215");
2.    inet_pton(AF_INET, "178.63.66.215", &(SockAddr.sin_addr));
```

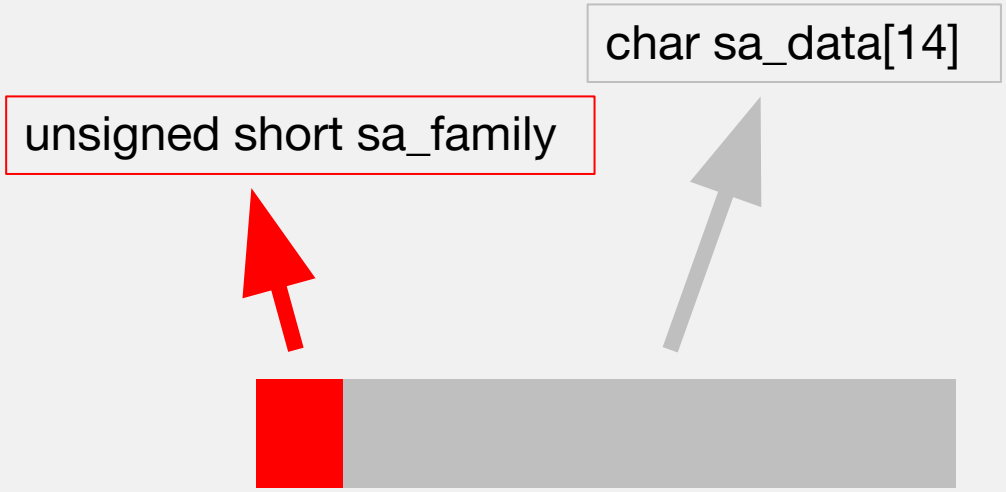
Заполнение структуры sockaddr_in6

```
1.  // SockAddr.sin_addr.s_addr = inet_addr("178.63.66.215");
2.  inet_pton(AF_INET, "178.63.66.215", &(SockAddr.sin_addr));

3.  inet_pton(AF_INET6, "2001:db8:8714:3a90::12",
    &(SockAddr6.sin6_addr));
```

Заполнение структуры sockaddr_un

```
1.  struct sockaddr_un addr;
2.  memset(&addr, 0, sizeof(addr));
3.  addr.sun_family = AF_UNIX;
4.  strncpy(addr.sun_path, “/tmp/server.sock”,
    sizeof(addr.sun_path)-1);
```

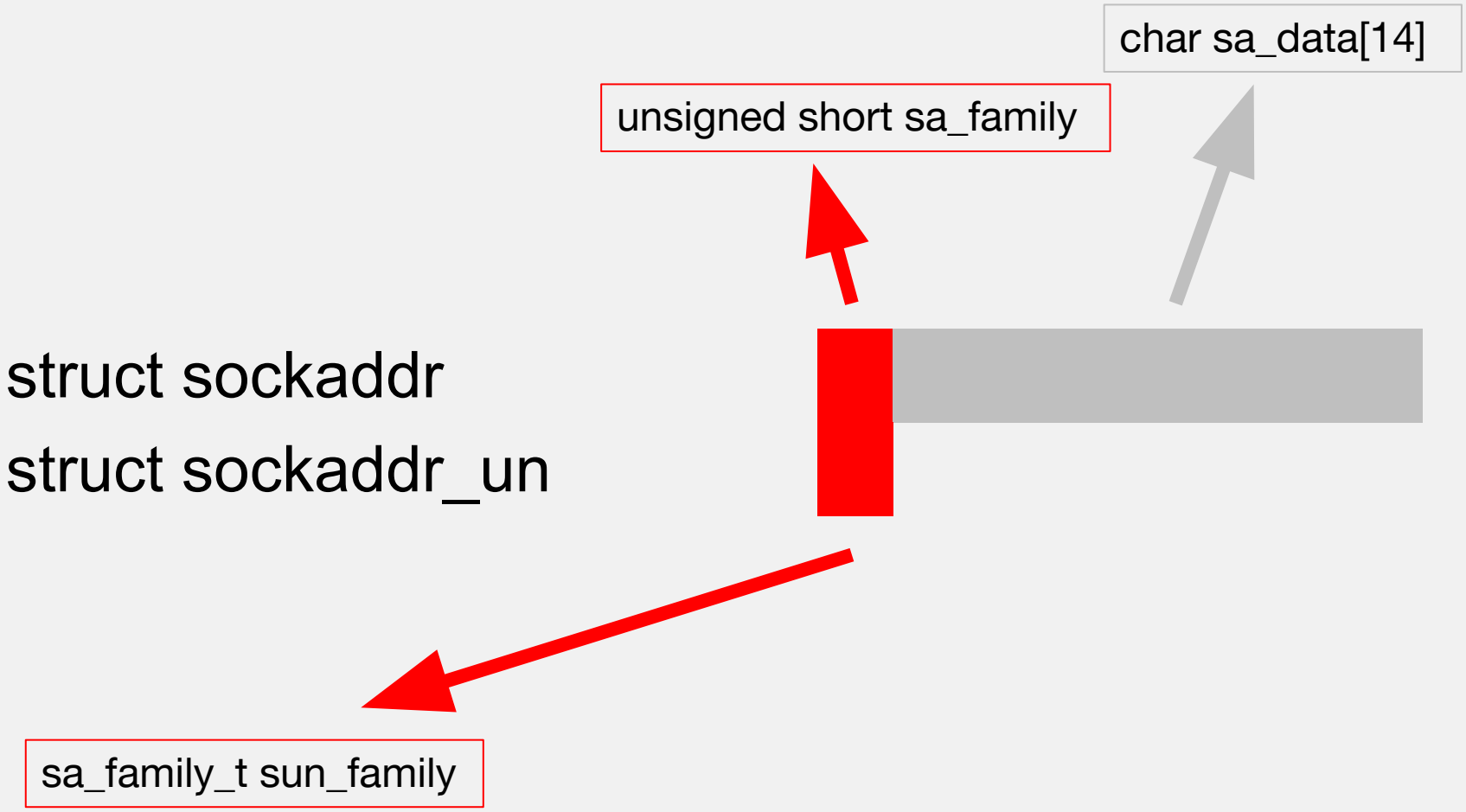
UDS



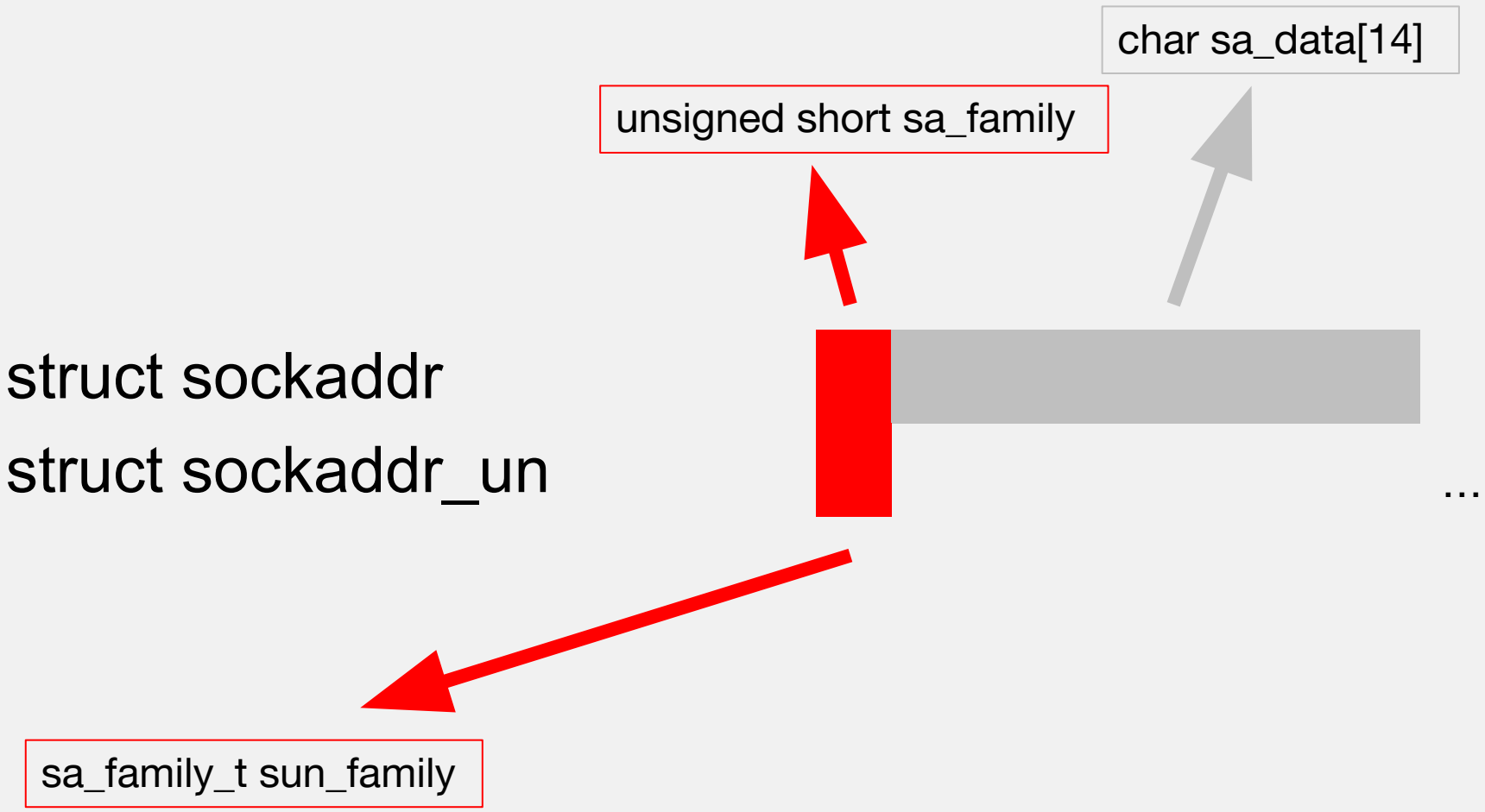
struct sockaddr

struct sockaddr_un

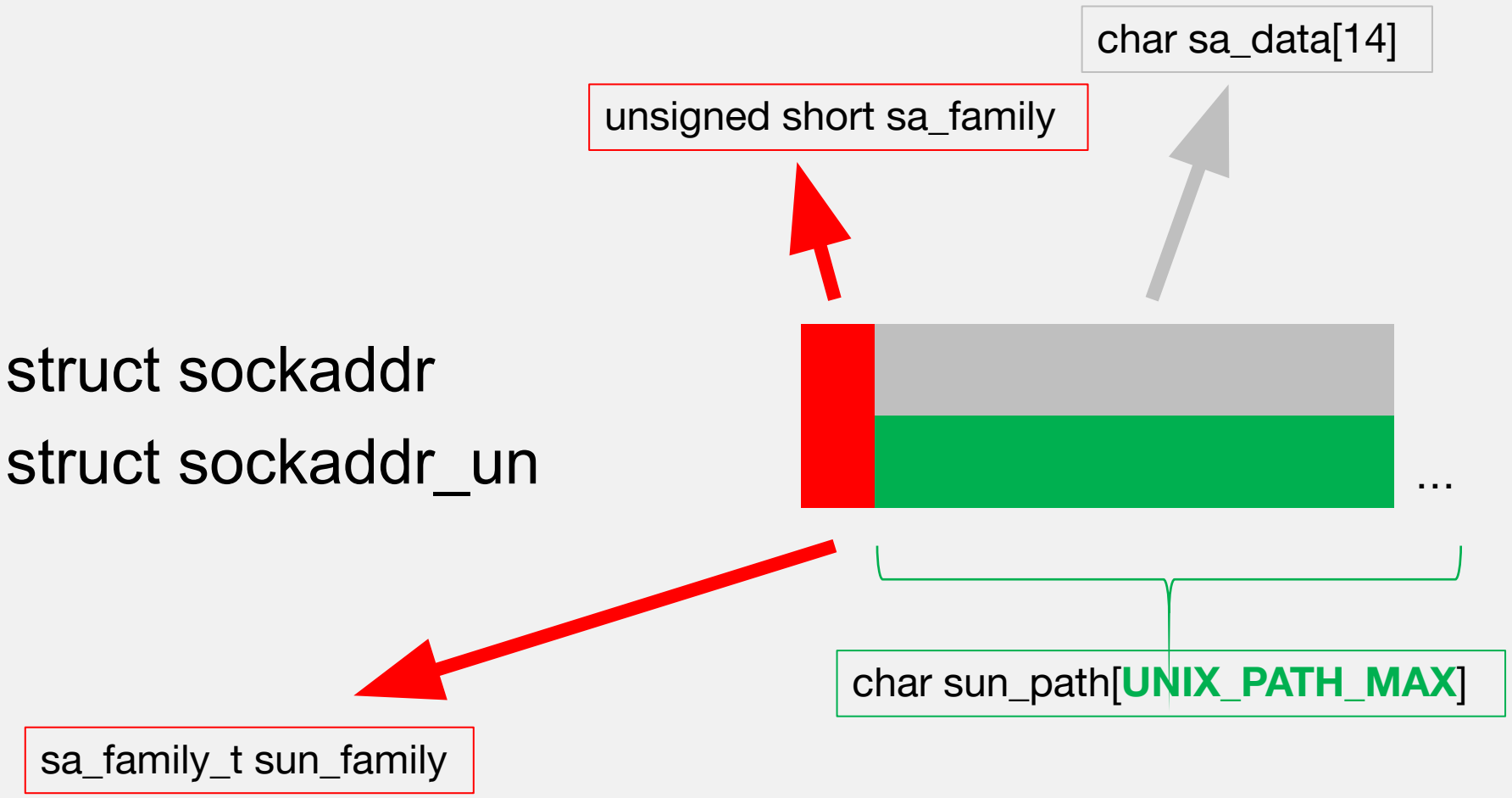
UDS


char sa_data[14]
unsigned short sa_family
struct sockaddr
struct sockaddr_un
sa_family_t sun_family

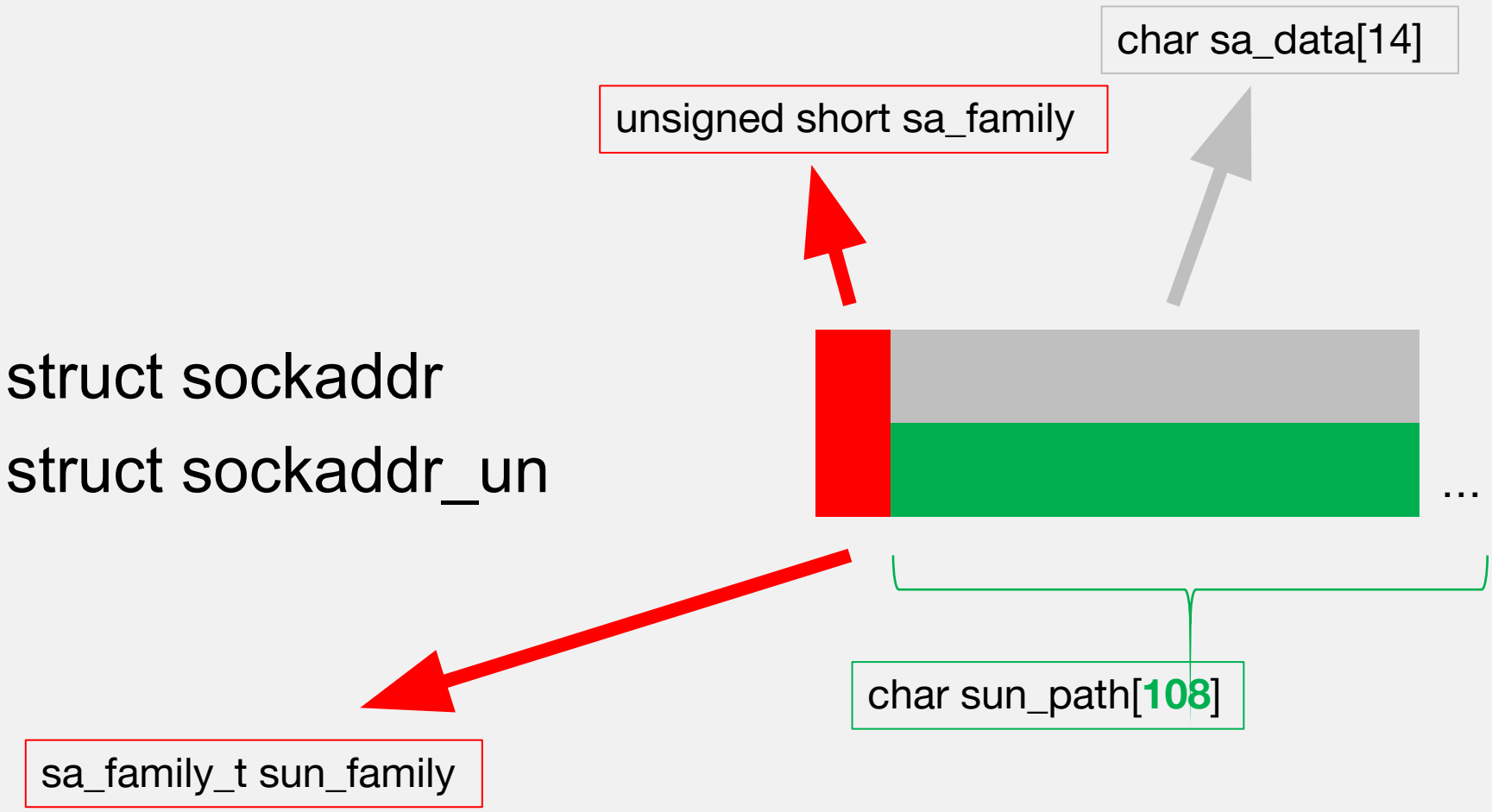
UDS


char sa_data[14]
unsigned short sa_family
struct sockaddr
struct sockaddr_un
...
sa_family_t sun_family

UDS


char sa_data[14]
unsigned short sa_family
struct sockaddr
struct sockaddr_un
...
char sun_path[UNIX_PATH_MAX]
sa_family_t sun_family

UDS


char sa_data[14]
unsigned short sa_family
struct sockaddr
struct sockaddr_un
...
char sun_path[108]
sa_family_t sun_family

Сокеты Беркли

```
listen(s, SOMAXCONN /* 128 */);
```

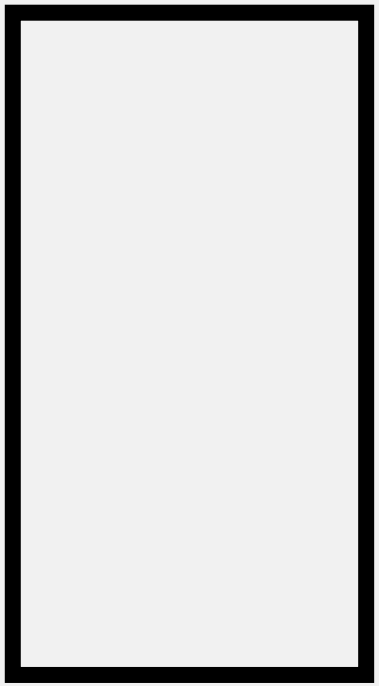
Сокеты Беркли

```
int SlaveSocket = accept(MasterSocket, 0, 0);
```

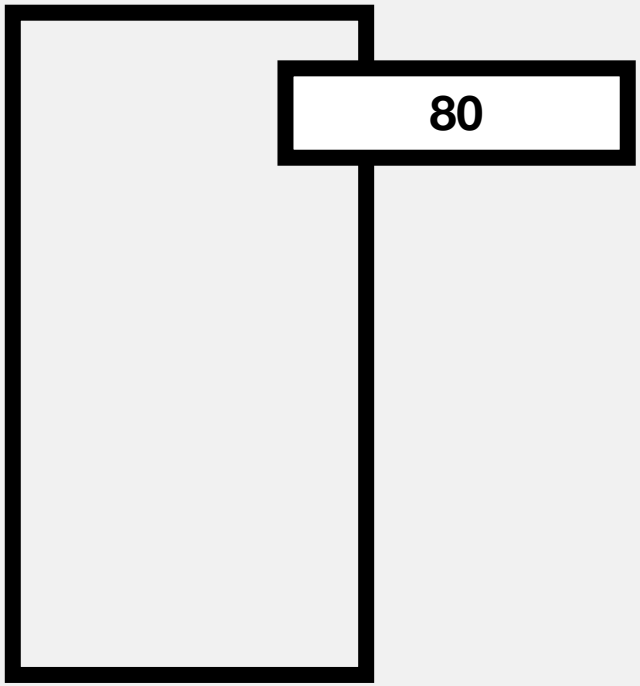
Сокеты Беркли

```
int SlaveSocket = accept(MasterSocket, 0, 0);

int SlaveSocket = accept(MasterSocket,
    struct sockaddr *addr, socklen_t *addrlen);
```

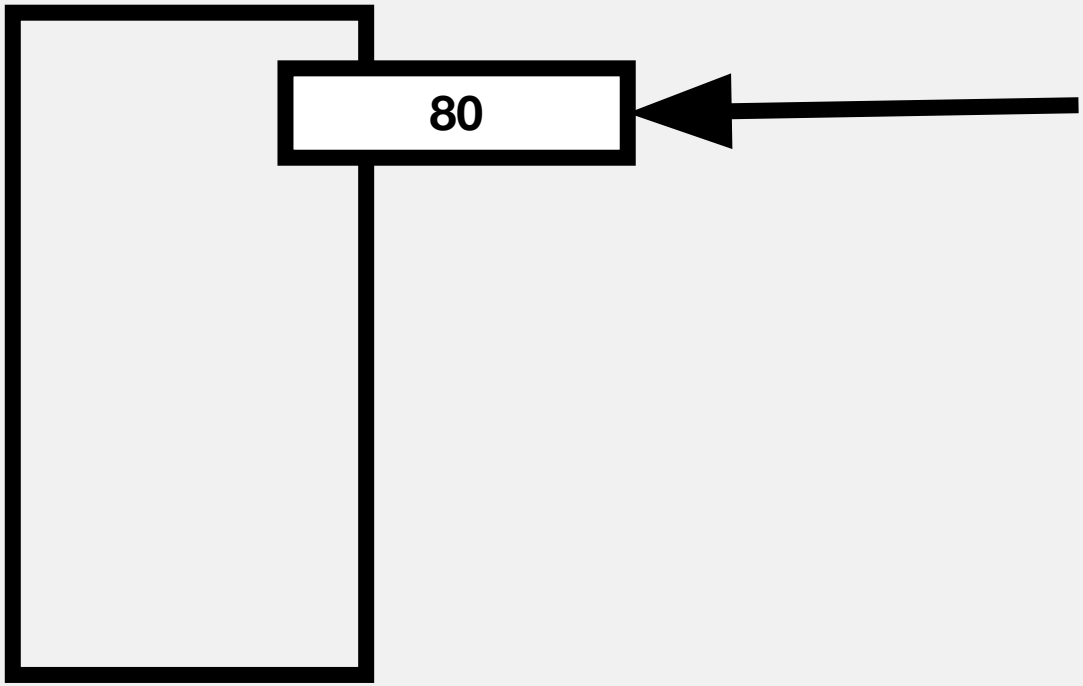
Сокеты Беркли



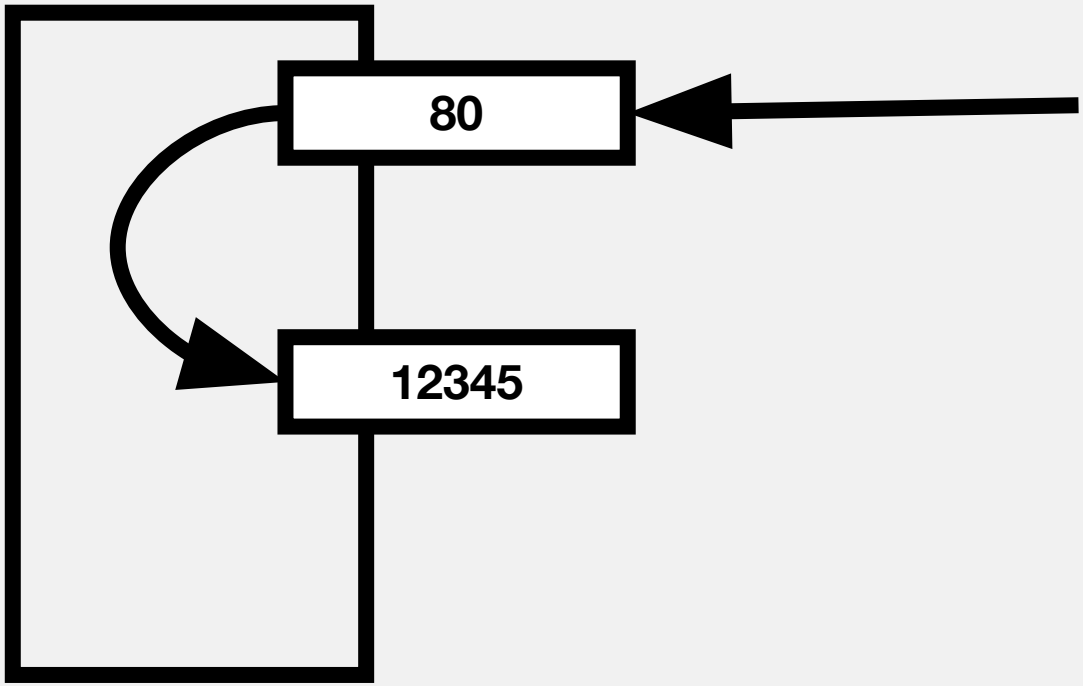
Сокеты Беркли



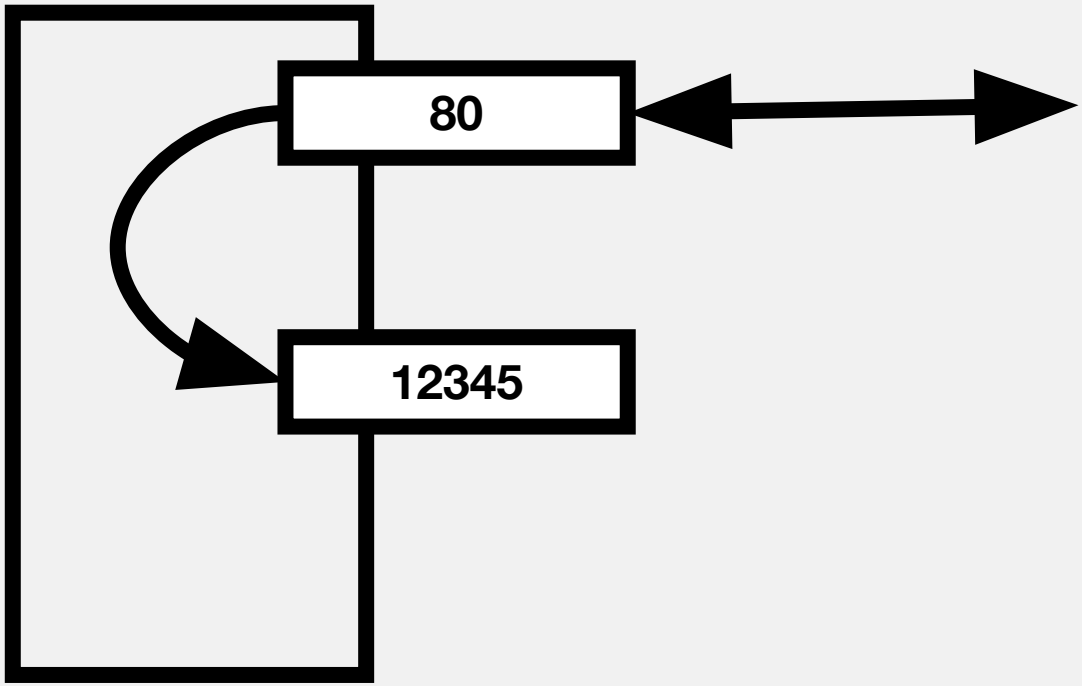
Сокеты Беркли



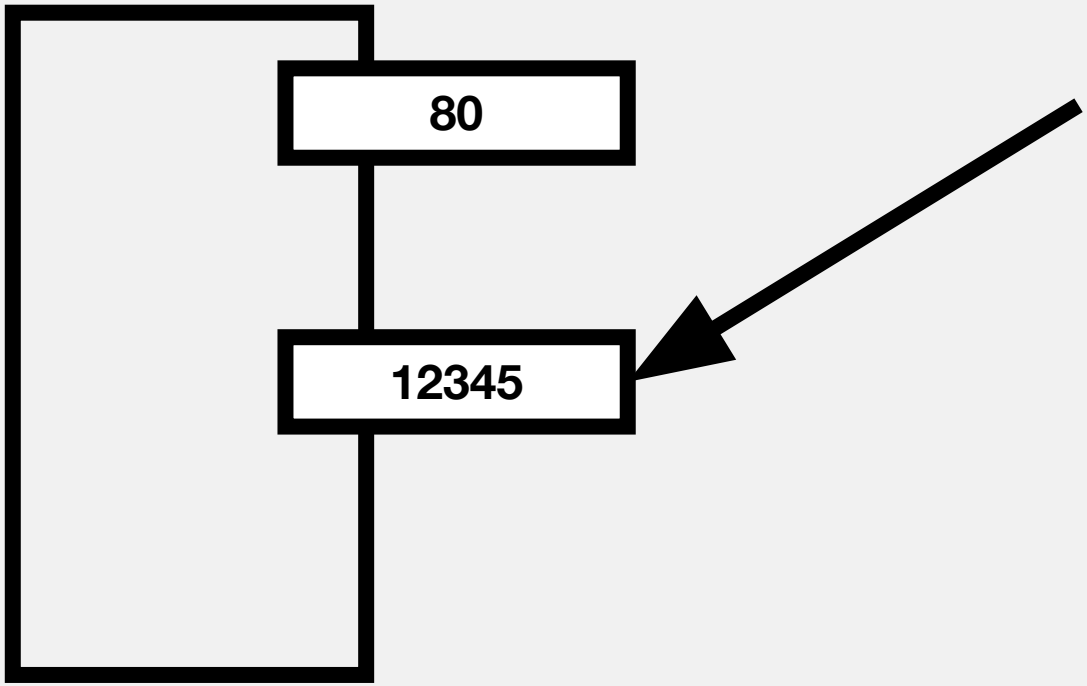
Сокеты Беркли



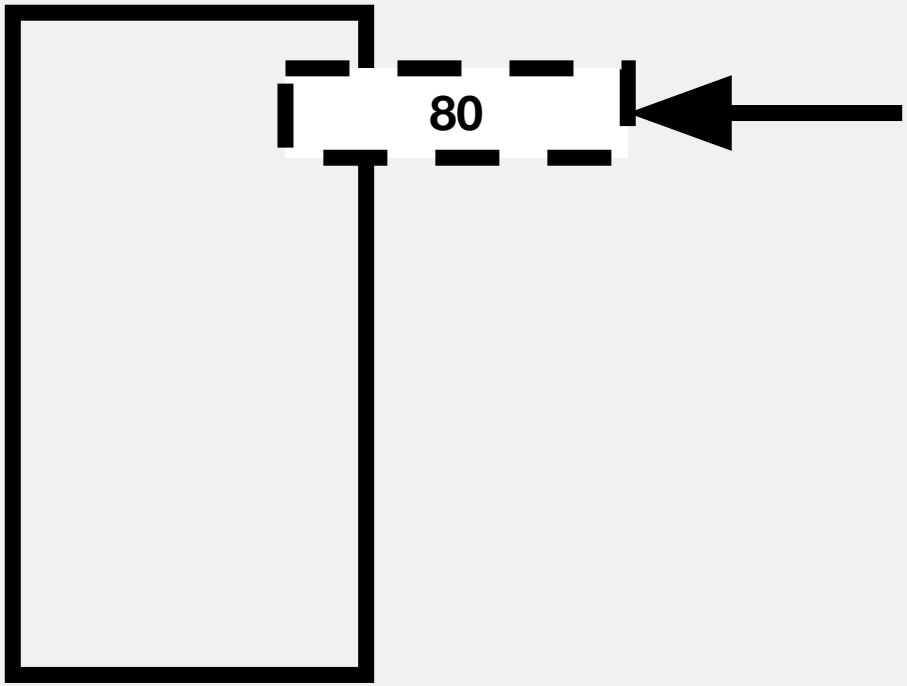
Сокеты Беркли



Сокеты Беркли



Сокеты Беркли



Сокеты Беркли

TCP-сервер

```
1.    int MasterSocket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
```

Сокеты Беркли

TCP-сервер

```
1.  int MasterSocket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);

2.  struct sockaddr_in SockAddr;
3.  SockAddr.sin_family = AF_INET;
4.  SockAddr.sin_port = htons(12345);
5.  SockAddr.sin_addr.s_addr = htonl(INADDR_ANY);
```

Сокеты Беркли

TCP-сервер

```
1.  int MasterSocket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);

2.  struct sockaddr_in SockAddr;
3.  SockAddr.sin_family = AF_INET;
4.  SockAddr.sin_port = htons(12345);
5.  SockAddr.sin_addr.s_addr = htonl(INADDR_ANY);

6.  bind(MasterSocket, (struct sockaddr *)&SockAddr,
    sizeof(SockAddr));
```

Сокеты Беркли

TCP-сервер

```
1.  int MasterSocket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);

2.  struct sockaddr_in SockAddr;
3.  SockAddr.sin_family = AF_INET;
4.  SockAddr.sin_port = htons(12345);
5.  SockAddr.sin_addr.s_addr = htonl(INADDR_ANY);

6.  bind(MasterSocket, (struct sockaddr *)&SockAddr,
    sizeof(SockAddr));

7.  listen(MasterSocket, SOMAXCONN);
```

Сокеты Беркли

TCP-сервер

```
1.  int MasterSocket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);

2.  struct sockaddr_in SockAddr;
3.  SockAddr.sin_family = AF_INET;
4.  SockAddr.sin_port = htons(12345);
5.  SockAddr.sin_addr.s_addr = htonl(INADDR_ANY);

6.  bind(MasterSocket, (struct sockaddr *)&SockAddr,
    sizeof(SockAddr));

7.  listen(MasterSocket, SOMAXCONN);

8.  while(true)
9.  {
10.     int SlaveSocket = accept(MasterSocket, 0, 0);
11.     // ...
12. }
```

Сокеты Беркли

TCP-клиент

```
1.  int ClientSocket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);

2.  struct sockaddr_in SockAddr;
3.  SockAddr.sin_family = AF_INET;
4.  SockAddr.sin_port = htons(12345);
5.  SockAddr.sin_addr.s_addr = htonl(INADDR_LOOPBACK);

6.  connect(ClientSocket, (const void*) &SockAddr, sizeof
    (SockAddr));
```

Сокеты Беркли

```
shutdown(ClientSocket, SHUT_RDWR);
shutdown(MasterSocket, SHUT_RDWR);

SHUT_RDWR
SHUT_RD
SHUT_WR

close(ClientSocket);
close(MasterSocket);
```

Сокеты Беркли

ssize_t read(**int** fd, **void** *buf, **size_t** count);

ssize_t write(**int** fd, **const void** *buf, **size_t** count);

Сокеты Беркли

~~**ssize_t** read(**int** fd, **void** *buf, **size_t** count);~~

~~**ssize_t** write(**int** fd, **const void** *buf, **size_t** count);~~

ssize_t recv(**int** s, **void** *buf, **size_t** len, **int** flags);

ssize_t send(**int** s, **const void** *buf, **size_t** len, **int** flags);

Сокеты Беркли

~~**ssize_t** read(**int** fd, **void** *buf, **size_t** count);~~

~~**ssize_t** write(**int** fd, **const void** *buf, **size_t** count);~~

ssize_t recv(**int** s, **void** *buf, **size_t** len, **int** flags);

ssize_t send(**int** s, **const void** *buf, **size_t** len, **int** flags);

MSG_NOSIGNAL

Сокеты Беркли

~~**ssize_t** read(**int** fd, **void** *buf, **size_t** count);~~

~~**ssize_t** write(**int** fd, **const void** *buf, **size_t** count);~~

ssize_t recv(**int** s, **void** *buf, **size_t** len, **int** flags);

ssize_t send(**int** s, **const void** *buf, **size_t** len, **int** flags);

signal(SIGPIPE, SIG_IGN);

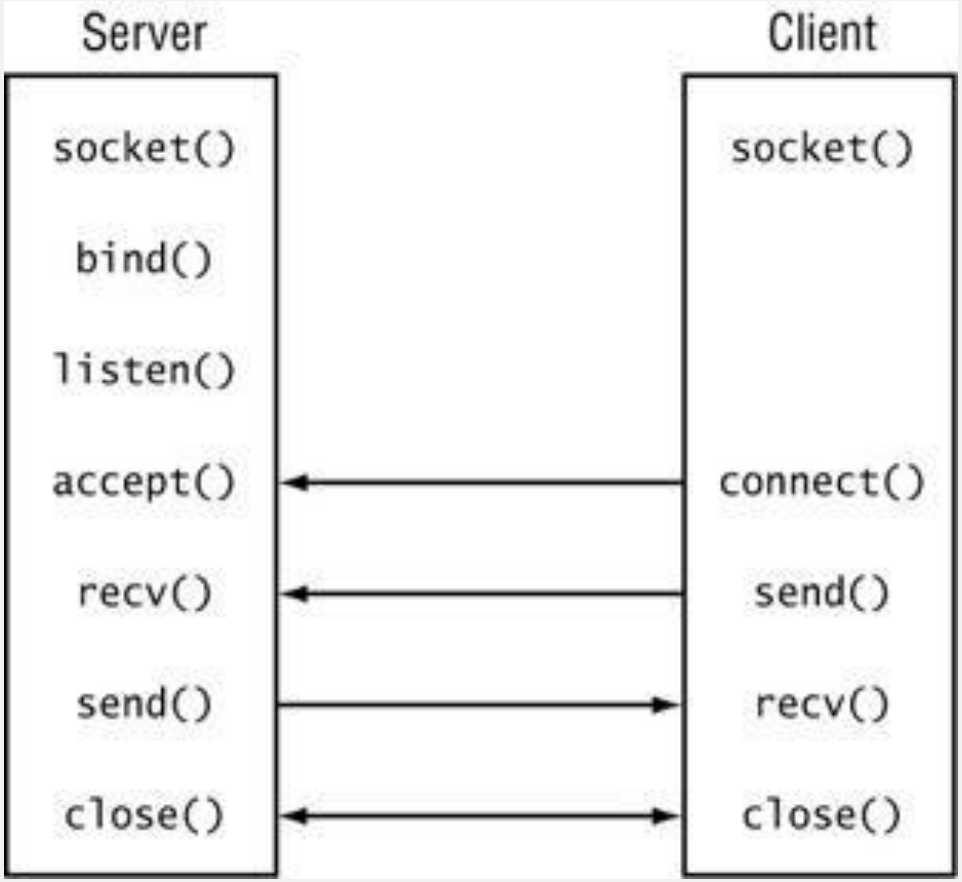
MSG_NOSIGNAL

Сокеты Беркли

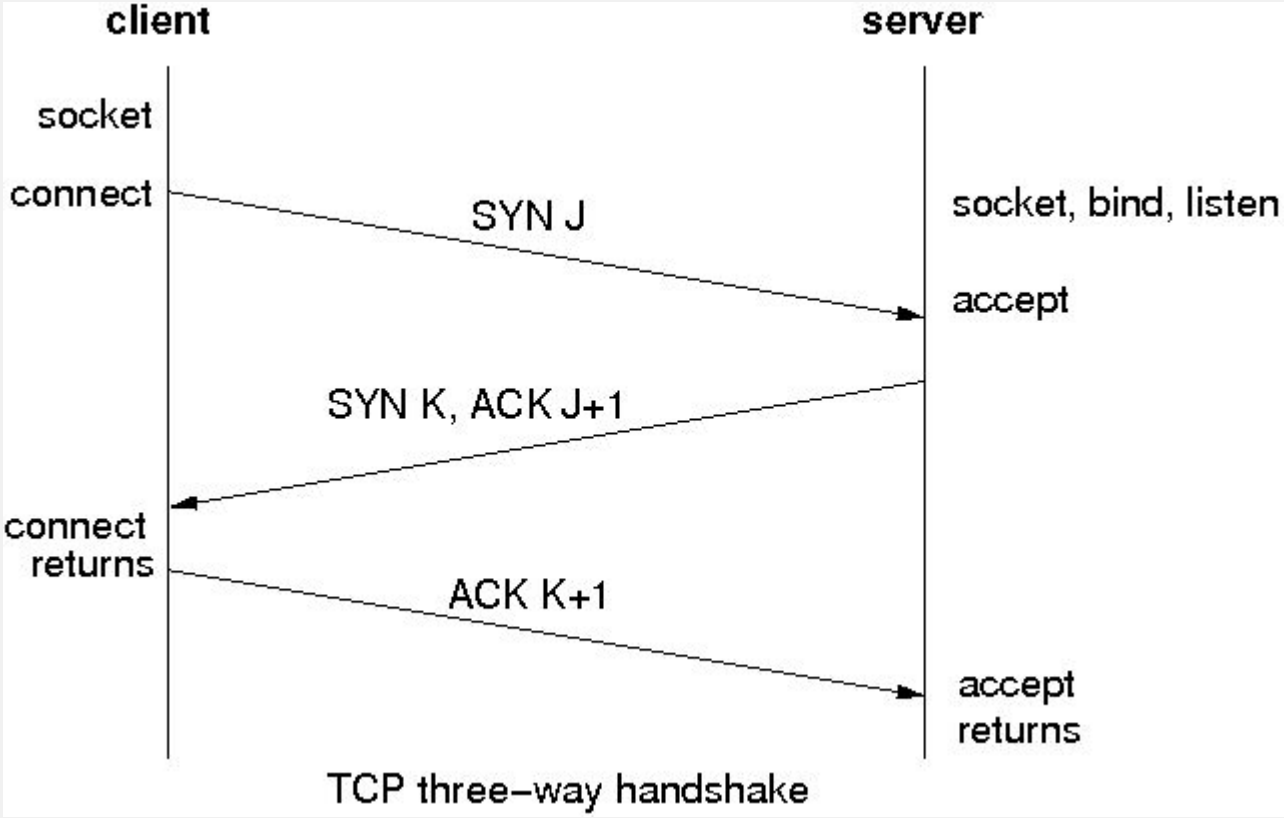
ssize_t sendto(**int** s, **const** **void** *buf, **size_t** len, **int** flags, **const struct** sockaddr *to, **socklen_t** tolen);

ssize_t recvfrom(**int** s, **void** *buf, **size_t** len, **int** flags, **struct** sockaddr *from, **socklen_t** *fromlen);

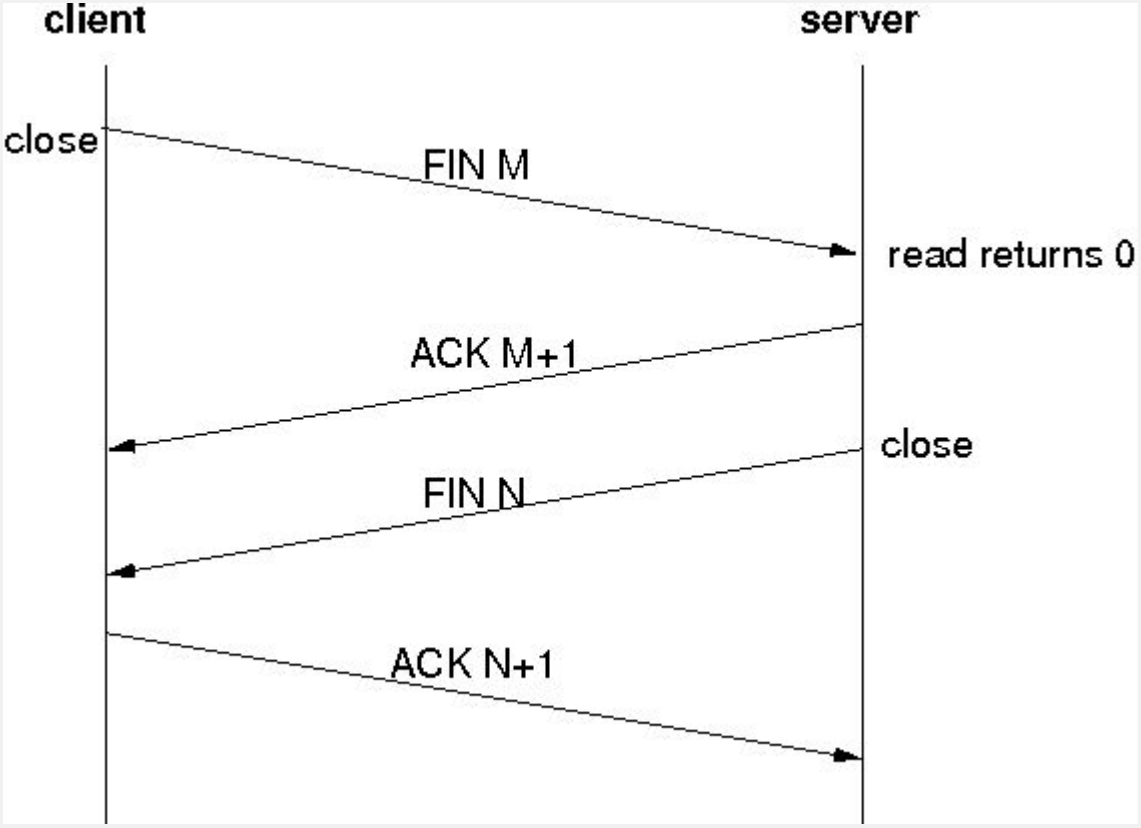
Сокеты Беркли



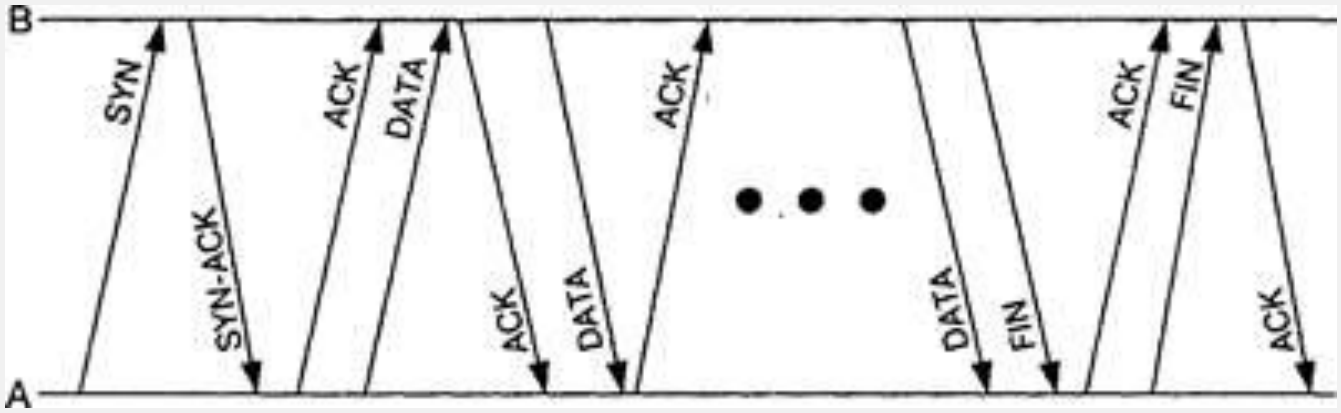
Сокеты Беркли

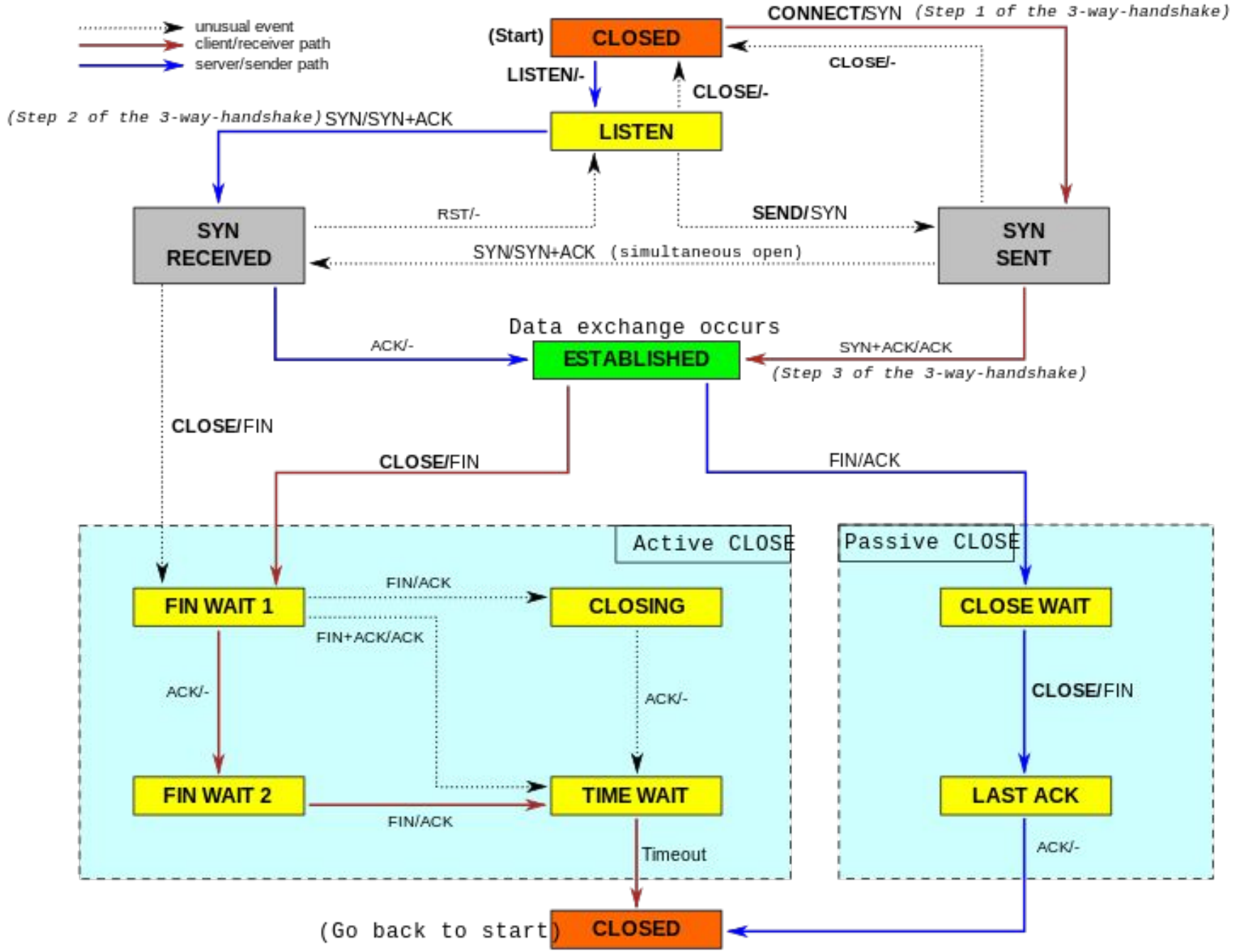


Сокеты Беркли



Сокеты Беркли




unusual event
client/receiver path
server/sender path
(Start)
CLOSED
CONNECT/SYN
(Step 1 of the 3-way-handshake)
CLOSE/-
LISTEN/-
CLOSE/-
LISTEN
(Step 2 of the 3-way-handshake)
SYN/SYN+ACK
SYN RECEIVED
RST/-
SEND/SYN
SYN SENT
SYN/SYN+ACK (simultaneous open)
Data exchange occurs
ACK/-
ESTABLISHED
SYN+ACK/ACK
(Step 3 of the 3-way-handshake)
CLOSE/FIN
CLOSE/FIN
FIN/ACK
Active CLOSE
Passive CLOSE
FIN WAIT 1
FIN/ACK
CLOSING
FIN+ACK/ACK
CLOSE WAIT
ACK/-
ACK/-
CLOSE/FIN
FIN WAIT 2
TIME WAIT
LAST ACK
FIN/ACK
Timeout
ACK/-
(Go back to start)
CLOSED

Сокеты Беркли

Неблокирующий сокет

```
1.  int set_nonblock(int fd)
2.  {
3.      int flags;
4.      #if defined(O_NONBLOCK)
5.      if (-1 == (flags = fcntl(fd, F_GETFL, 0))) flags = 0;
6.      return fcntl(fd, F_SETFL, flags | O_NONBLOCK);
7.      #else
8.      flags = 1;
9.      return ioctl(fd, FIOBIO, &flags);
10.     #endif
11. }
```

Сокеты Беркли

Использование setsockopt

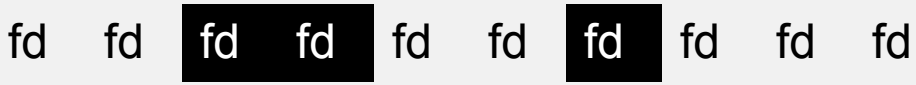
```
1.  int optval = 1;
2.  setsockopt(MasterSocket, SOL_SOCKET, SO_REUSEADDR, &optval,
    sizeof(optval));

3.  struct timeval tv;
4.  tv.tv_sec = 16;
5.  tv.tv_usec = 0;
6.  setsockopt(SlaveSocket, SOL_SOCKET, SO_RCVTIMEO, (char*) &tv,
    sizeof(tv));
7.  setsockopt(SlaveSocket, SOL_SOCKET, SO_SNDTIMEO, (char*) &tv,
    sizeof(tv));
```

Мультиплексирование

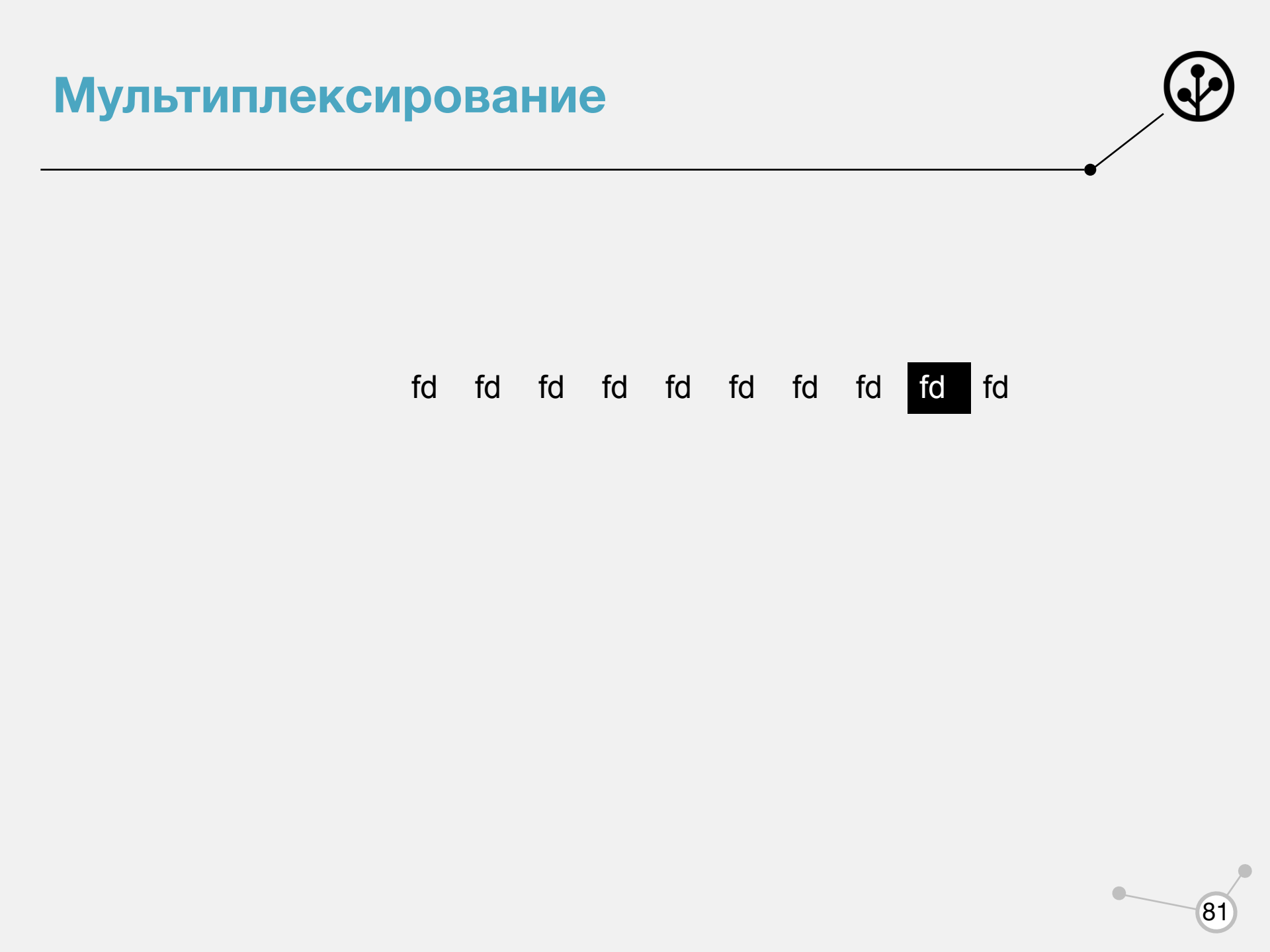
Мультиплексирование

fd fd fd fd fd fd fd fd fd fd

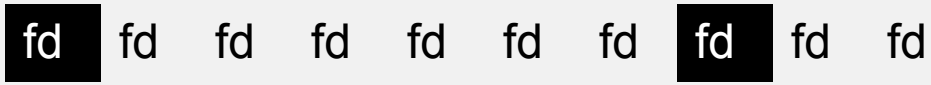
Мультиплексирование



Мультиплексирование



Мультиплексирование



Мультиплексирование

Why?

Мультиплексирование

Why?

CPU!

Мультиплексирование

Работа с select

```
1.    fd_set Set;
```

Мультиплексирование

Работа с select

```
1.    fd_set Set;
2.    FD_ZERO(&Set);
3.    FD_SET(MasterSocket, &Set);
```

Мультиплексирование

Работа с select

```
1.  fd_set Set;
2.  FD_ZERO(&Set);
3.  FD_SET(MasterSocket, &Set);
4.  for(auto Iter = SlaveSockets.begin(); Iter !=
    SlaveSockets.end(); Iter++)
5.  {
6.      FD_SET(*Iter, &Set);
7.  }
```

Мультиплексирование

Работа с select

```
1.  fd_set Set;
2.  FD_ZERO(&Set);
3.  FD_SET(MasterSocket, &Set);
4.  for(auto Iter = SlaveSockets.begin(); Iter !=
    SlaveSockets.end(); Iter++)
5.  {
6.      FD_SET(*Iter, &Set);
7.  }

8.  int Max = std::max(MasterSocket,
    *std::max_element(SlaveSockets.begin(), SlaveSockets.end()));
```

Мультиплексирование

Работа с select

```
1.  fd_set Set;
2.  FD_ZERO(&Set);
3.  FD_SET(MasterSocket, &Set);
4.  for(auto Iter = SlaveSockets.begin(); Iter !=
    SlaveSockets.end(); Iter++)
5.  {
6.      FD_SET(*Iter, &Set);
7.  }

8.  int Max = std::max(MasterSocket,
    *std::max_element(SlaveSockets.begin(), SlaveSockets.end()));
9.  select(Max+1, &Set, NULL, NULL, NULL);
```

Мультиплексирование

Работа с select

```
1.  fd_set Set;
2.  FD_ZERO(&Set);
3.  FD_SET(MasterSocket, &Set);
4.  for(auto Iter = SlaveSockets.begin(); Iter !=
    SlaveSockets.end(); Iter++)
5.  {
6.      FD_SET(*Iter, &Set);
7.  }

8.  int Max = std::max(MasterSocket,
    *std::max_element(SlaveSockets.begin(), SlaveSockets.end()));
9.  select(Max+1, &Set, NULL, NULL, NULL);

10. for(auto Iter = SlaveSockets.begin();
11.   Iter != SlaveSockets.end();
12.   Iter++)
13.     if(FD_ISSET(*Iter, &Set)) { /* ... */ }
```

Мультиплексирование

Работа с select

```
1.  fd_set Set;
2.  FD_ZERO(&Set);
3.  FD_SET(MasterSocket, &Set);
4.  for(auto Iter = SlaveSockets.begin(); Iter !=
    SlaveSockets.end(); Iter++)
5.  {
6.      FD_SET(*Iter, &Set);
7.  }

8.  /* ... */

9.  if(FD_ISSET(MasterSocket, &Set))
10. {
11.   int SlaveSocket = accept(MasterSocket, 0, 0);
12.   SlaveSockets.insert(SlaveSocket);
13. }
```

Мультиплексирование

Работа с poll

```
1.    struct pollfd Set[POLL_SIZE];
2.    Set[0].fd = MasterSocket;
3.    Set[0].events = POLLIN;
```

Мультиплексирование

Работа с poll

```
1.  struct pollfd Set[POLL_SIZE];
2.  Set[0].fd = MasterSocket;
3.  Set[0].events = POLLIN;

4.  unsigned int Index = 1;
5.  for(auto Iter = SlaveSockets.begin(); Iter !=
    SlaveSockets.end(); Iter++)
6.  {
7.    Set[Index].fd = *Iter;
8.    Set[Index].events = POLLIN;
9.    Index++;
10. }
11. unsigned int SetSize = 1 + SlaveSockets.size();
```

Мультиплексирование

Работа с poll

```
1.  struct pollfd Set[POLL_SIZE];
2.  Set[0].fd = MasterSocket;
3.  Set[0].events = POLLIN;

4.  unsigned int Index = 1;
5.  for(auto Iter = SlaveSockets.begin(); Iter !=
    SlaveSockets.end(); Iter++)
6.  {
7.    Set[Index].fd = *Iter;
8.    Set[Index].events = POLLIN;
9.    Index++;
10. }
11. unsigned int SetSize = 1 + SlaveSockets.size();

12. poll(Set, SetSize, -1);
```

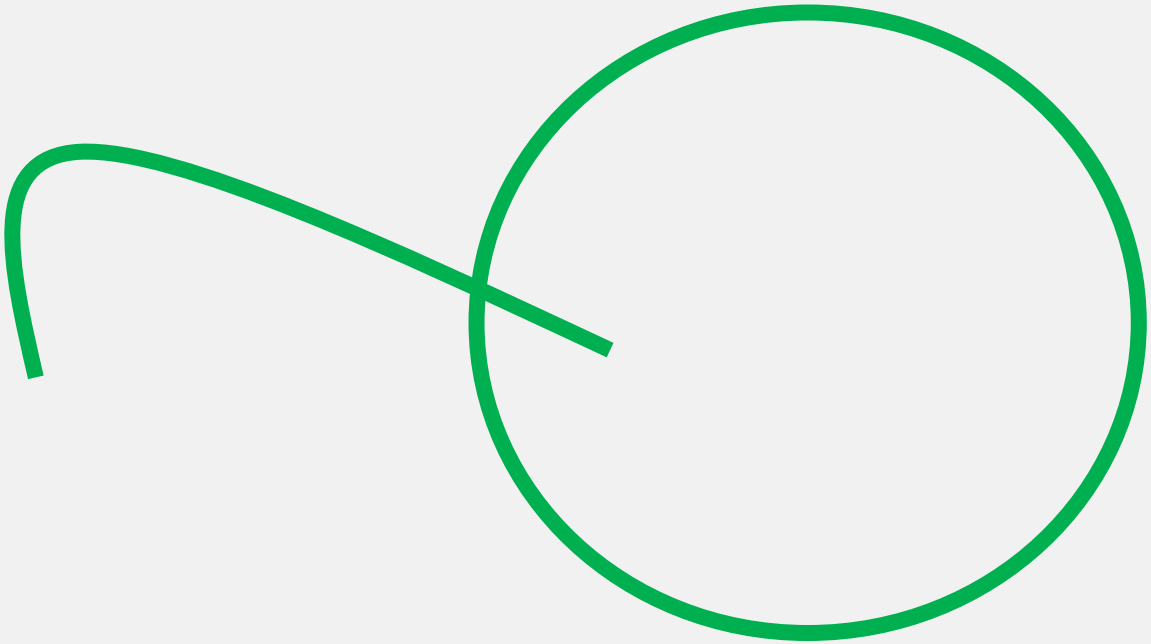
Мультиплексирование

Работа с poll

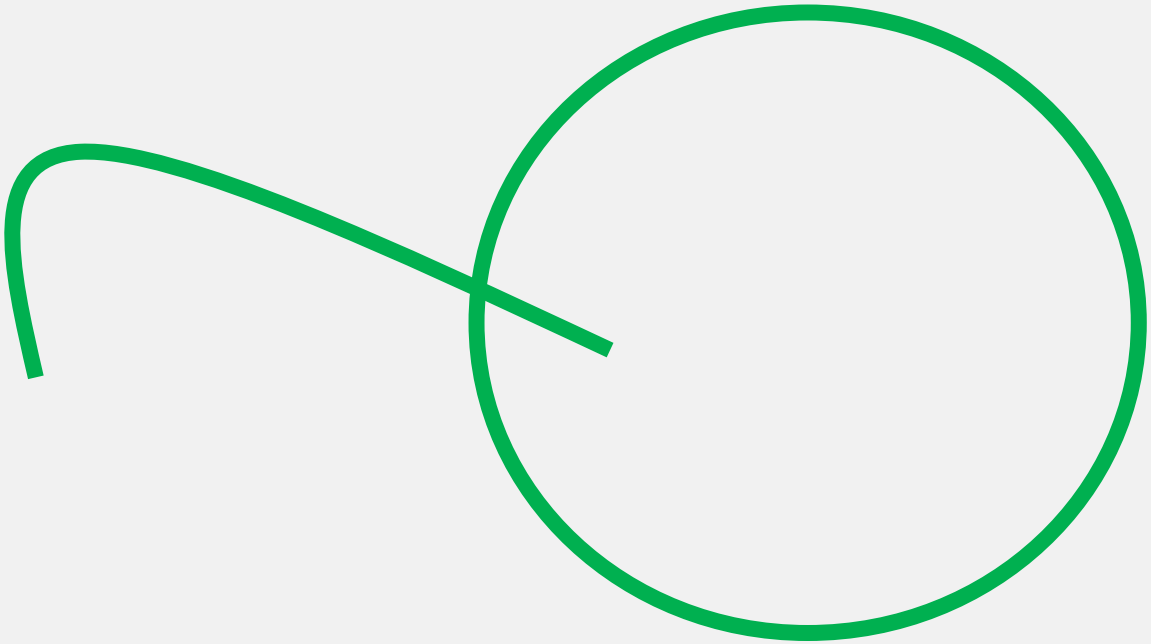
```
1.  struct pollfd Set[POLL_SIZE];
2.  Set[0].fd = MasterSocket;
3.  Set[0].events = POLLIN;

4.  /* ... */

5.  poll(Set, SetSize, -1);

6.  for(unsigned int i = 0; i < SetSize; i++)
7.  {
8.    if(Set[i].revents & POLLIN)
9.    {
10.     /* ... */
11.   }
12. }
```

C10K Problem

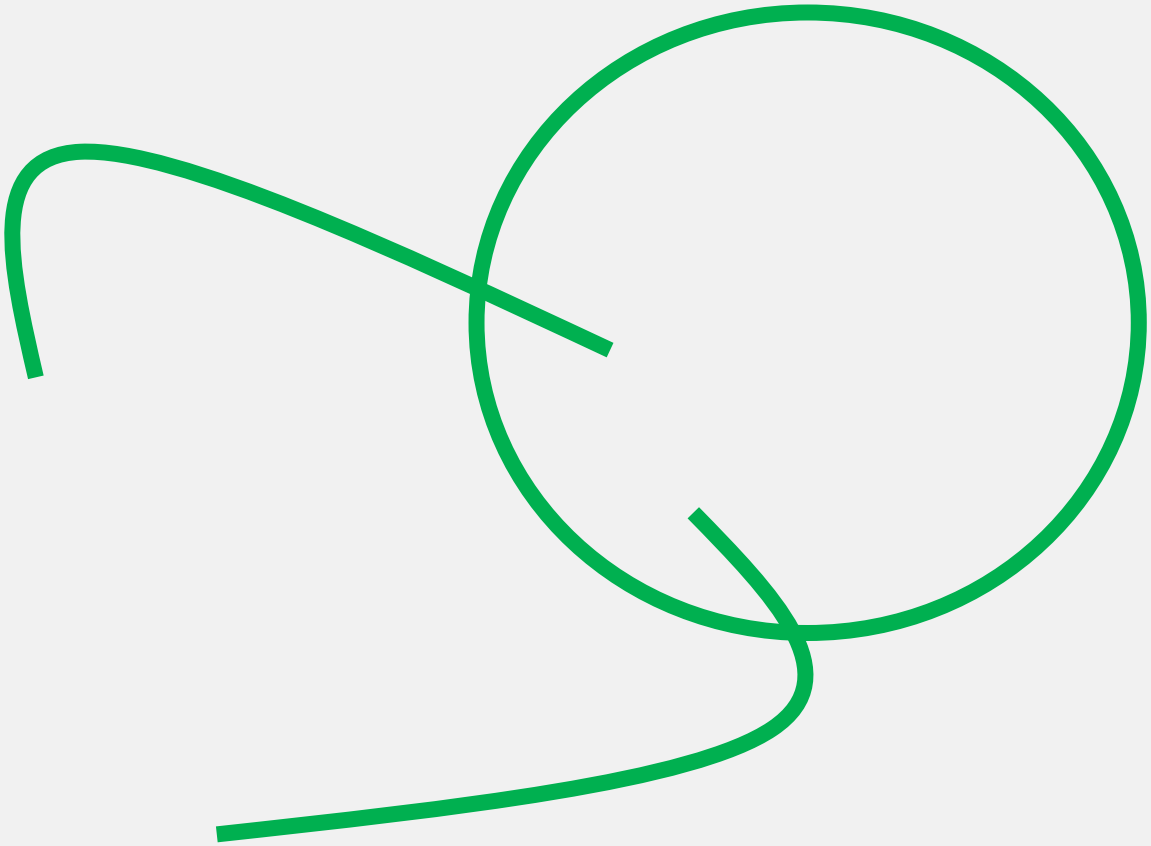
C10K Problem



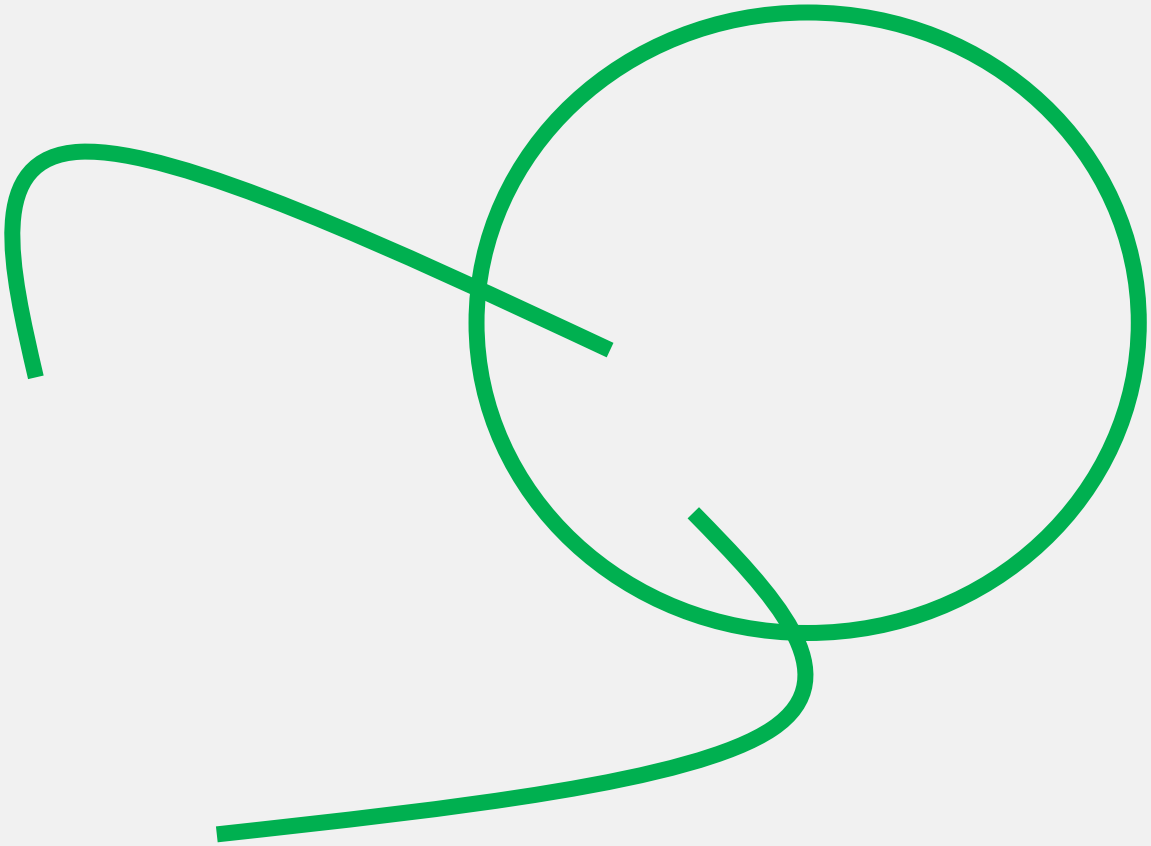
C10K Problem



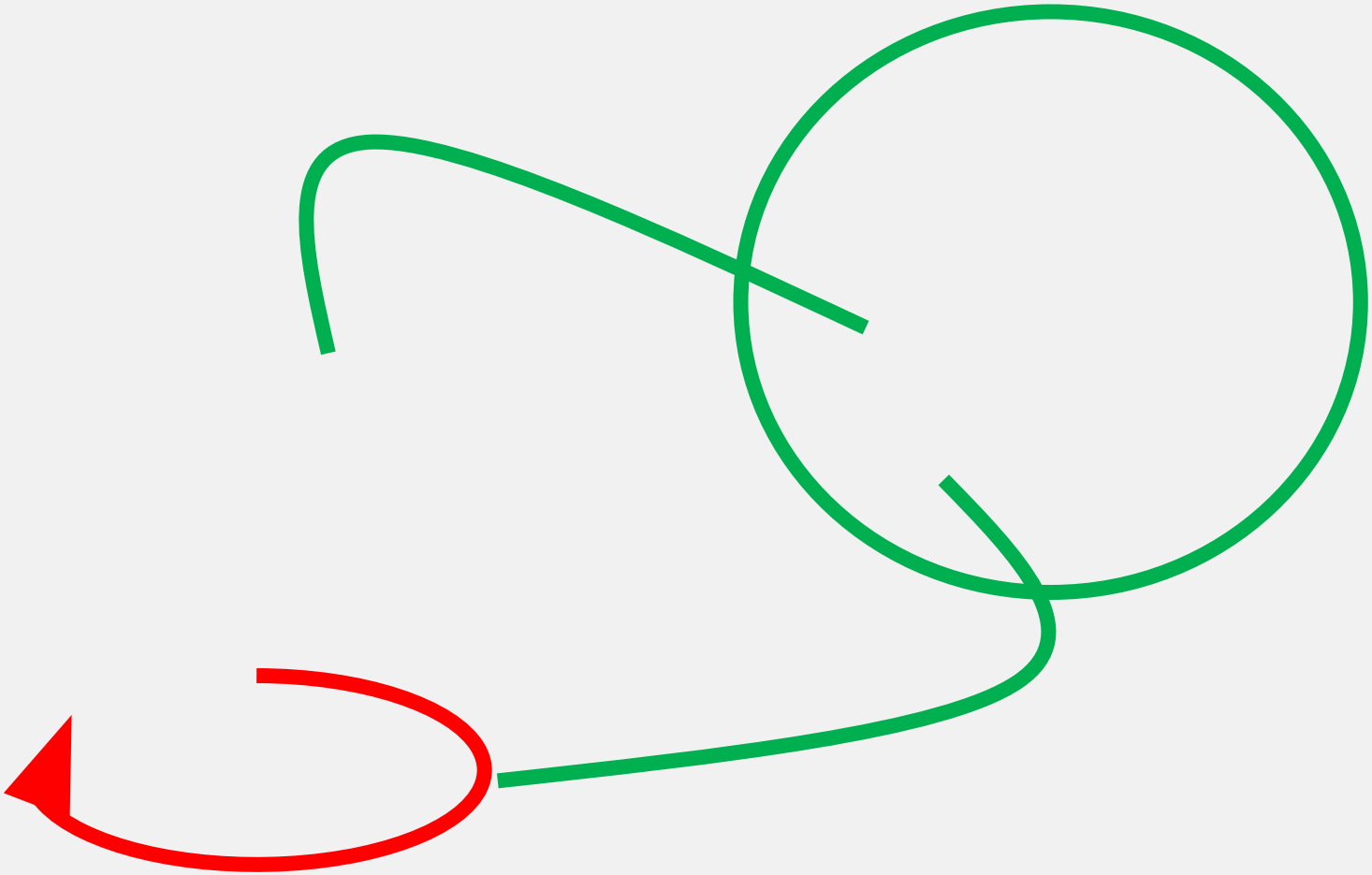
C10K Problem



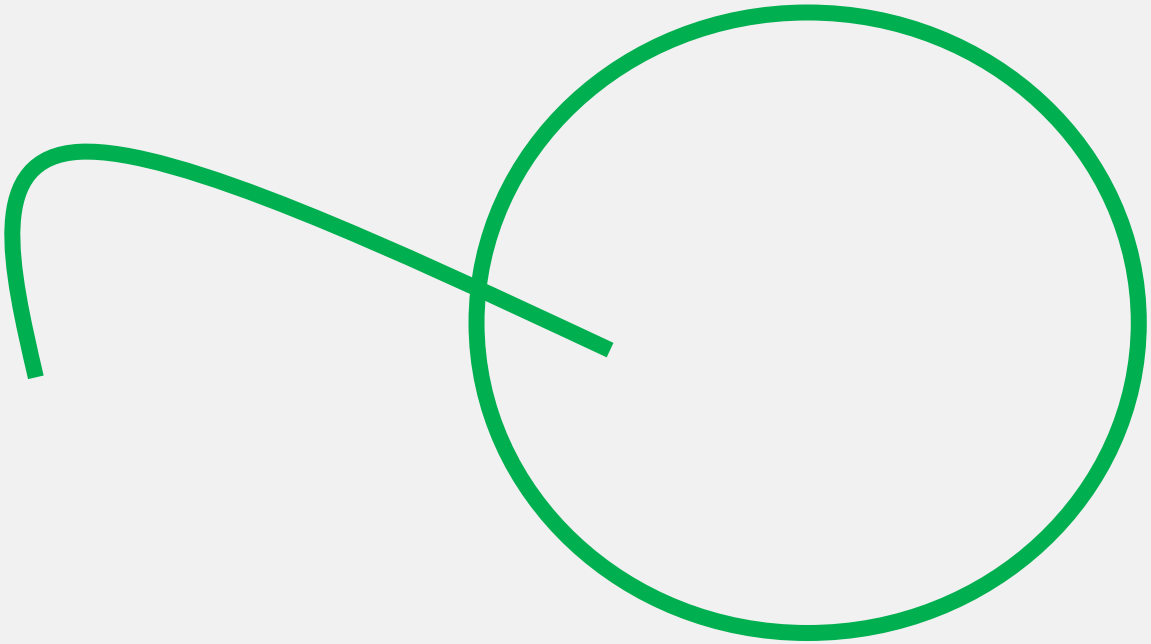
C10K Problem



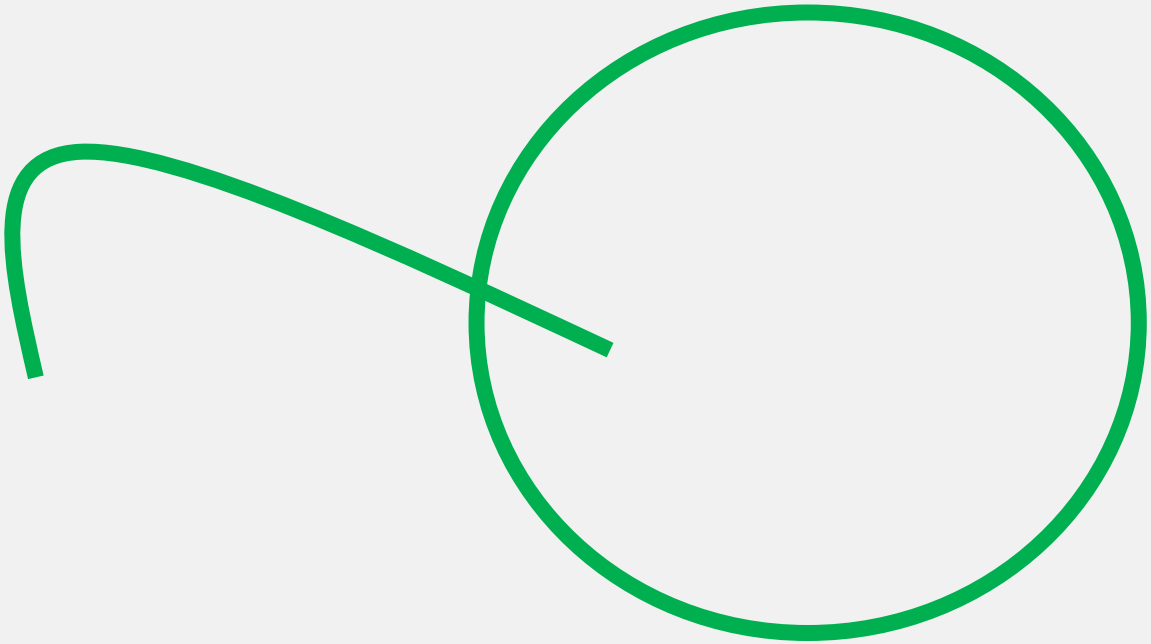
C10K Problem



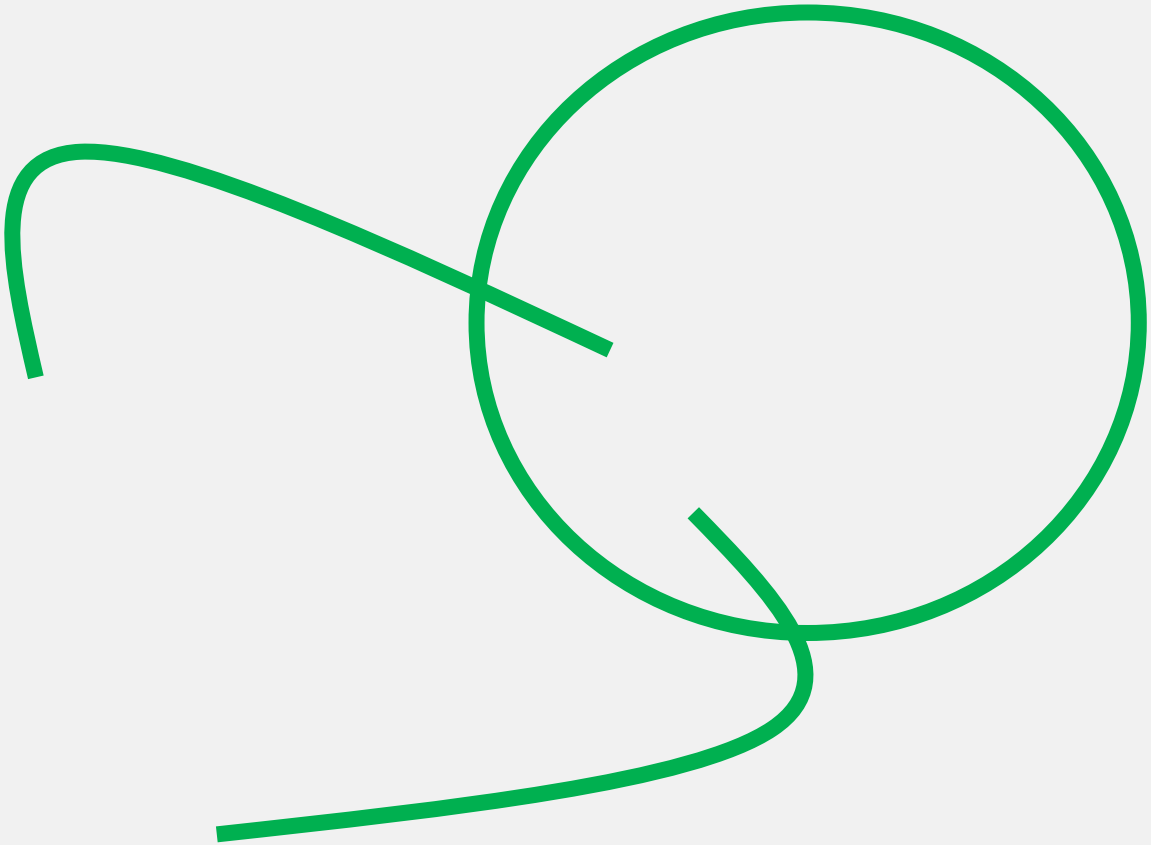
C10K Problem



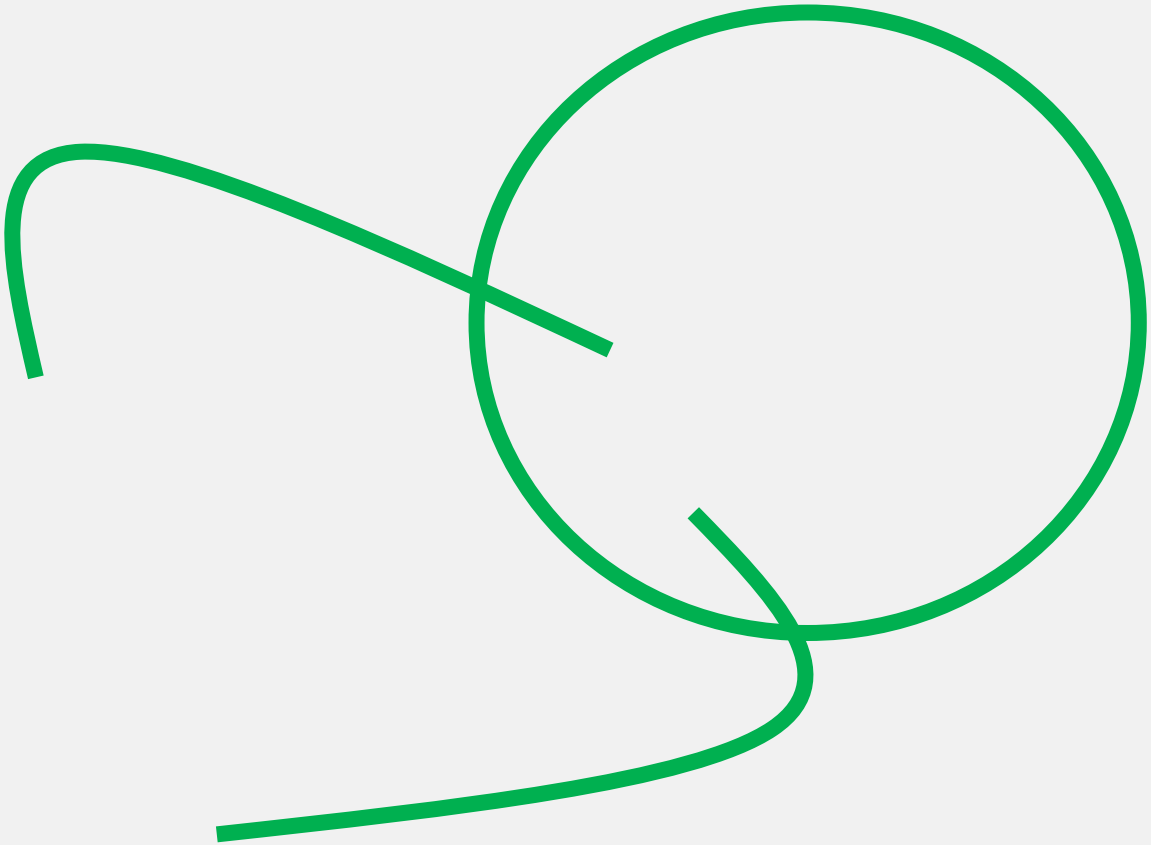
C10K Problem



C10K Problem



C10K Problem

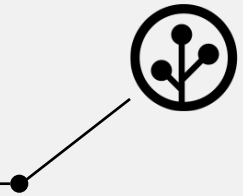


Мультиплексирование

Работа с epoll

```
1.    int EPoll = epoll_create1(0);
```

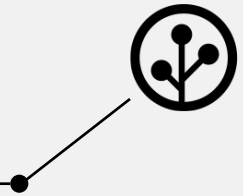
Мультиплексирование



Работа с epoll

```
1.  int EPoll = epoll_create1(0);

2.  struct epoll_event Event;
3.  Event.data.fd = MasterSocket;
4.  Event.events = EPOLLIN | EPOLLET /* edge triggered */;
```

Мультиплексирование



Работа с epoll

```
1.  int EPoll = epoll_create1(0);

2.  struct epoll_event Event;
3.  Event.data.fd = MasterSocket;
4.  Event.events = EPOLLIN | EPOLLET; /* edge triggered */

5.  epoll_ctl(EPoll, EPOLL_CTL_ADD, MasterSocket, &Event);
```

Мультиплексирование

Работа с epoll

```
1.   int EPoll = epoll_create1(0);

2.   struct epoll_event Event;
3.   Event.data.fd = MasterSocket;
4.   Event.events = EPOLLIN | EPOLLET; /* edge triggered */

5.   epoll_ctl(EPoll, EPOLL_CTL_ADD, MasterSocket, &Event);

6.   while(true)
7.   {
8.     int N = epoll_wait(EPoll, Events, MAX_EVENTS, -1);
9.     for(unsigned int i = 0; i < N; i++)
10.    {
11.      /* ... */
12.    }
13.  }
```

Мультиплексирование

Работа с epoll

```
1.  struct epoll_event * Events;
2.  Events = (struct epoll_event *) calloc(MAX_EVENTS,
    sizeof(struct epoll_event));

3.  /* ... */

4.  while(true)
5.  {
6.    int N = epoll_wait(EPoll, Events, MAX_EVENTS, -1);
7.    for(unsigned int i = 0; i < N; i++)
8.    {
9.      /* ... */
10.   }
11. }
```

Мультиплексирование

Работа с epoll

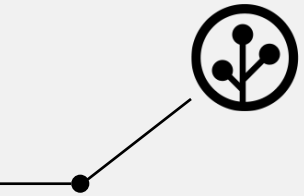
```
1.  struct epoll_event * Events;
2.  Events = (struct epoll_event *) calloc(MAX_EVENTS,
    sizeof(struct epoll_event));

3.  /* ... */

4.  while(true)
5.  {
6.    int N = epoll_wait(EPoll, Events, MAX_EVENTS, -1);
7.    for(unsigned int i = 0; i < N; i++)
8.    {
9.      if((Events[i].events & EPOLLERR)||(Events[i].events &
    EPOLLHUP))
10.     { /* ... */ }
11.   }
12. }
```

Мультиплексирование

Работа с kqueue

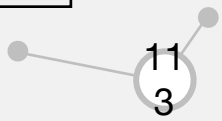
```
1.    int KQueue = kqueue();
```

Мультиплексирование

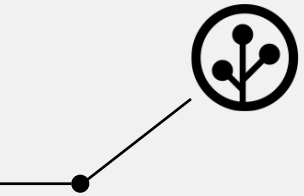


Работа с kqueue

```
1.  int KQueue = kqueue();

2.  struct kevent KEvent;
3.  bzero(&KEvent, sizeof(KEvent));
4.  EV_SET(&KEvent, MasterSocket, EVFILT_READ, EV_ADD, 0, 0, 0);
5.  kevent(KQueue, &KEvent, 1, NULL, 0, NULL);
```

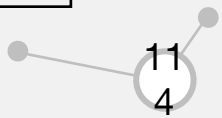


Мультиплексирование



Работа с kqueue

```
1.   int KQueue = kqueue();

2.   struct kevent KEvent;
3.   bzero(&KEvent, sizeof(KEvent));
4.   EV_SET(&KEvent, MasterSocket, EVFILT_READ, EV_ADD, 0, 0, 0);
5.   kevent(KQueue, &KEvent, 1, NULL, 0, NULL);

6.   while(true)
7.   {
8.     bzero(&KEvent, sizeof(KEvent));
9.     kevent(KQueue, NULL, 0, &KEvent, 1, NULL);
10.    if(KEvent.filter == EVFILT_READ) { /* ... */ }
11.  }
```



Мультиплексирование

Работа с kqueue

```
1.  if(KEvent.ident == MasterSocket)
2.  {
3.    int SlaveSocket = accept(MasterSocket, 0, 0);
4.    bzero(&KEvent, sizeof(KEvent));
5.    EV_SET(&KEvent, SlaveSocket, EVFILT_READ, EV_ADD, 0, 0, 0);
6.    kevent(KQueue, &KEvent, 1, NULL, 0, NULL);
7.  }
8.  else
9.  {
10.   /* ... */
11. }
```

Raw-сокеты

Raw-сокеты

```
int RAWSocket = socket(AF_INET, SOCK_RAW,
                       IPPROTO_RAW);
```

```
int RAWSocket = socket(AF_INET, SOCK_RAW,
                       IPPROTO_TCP);
```

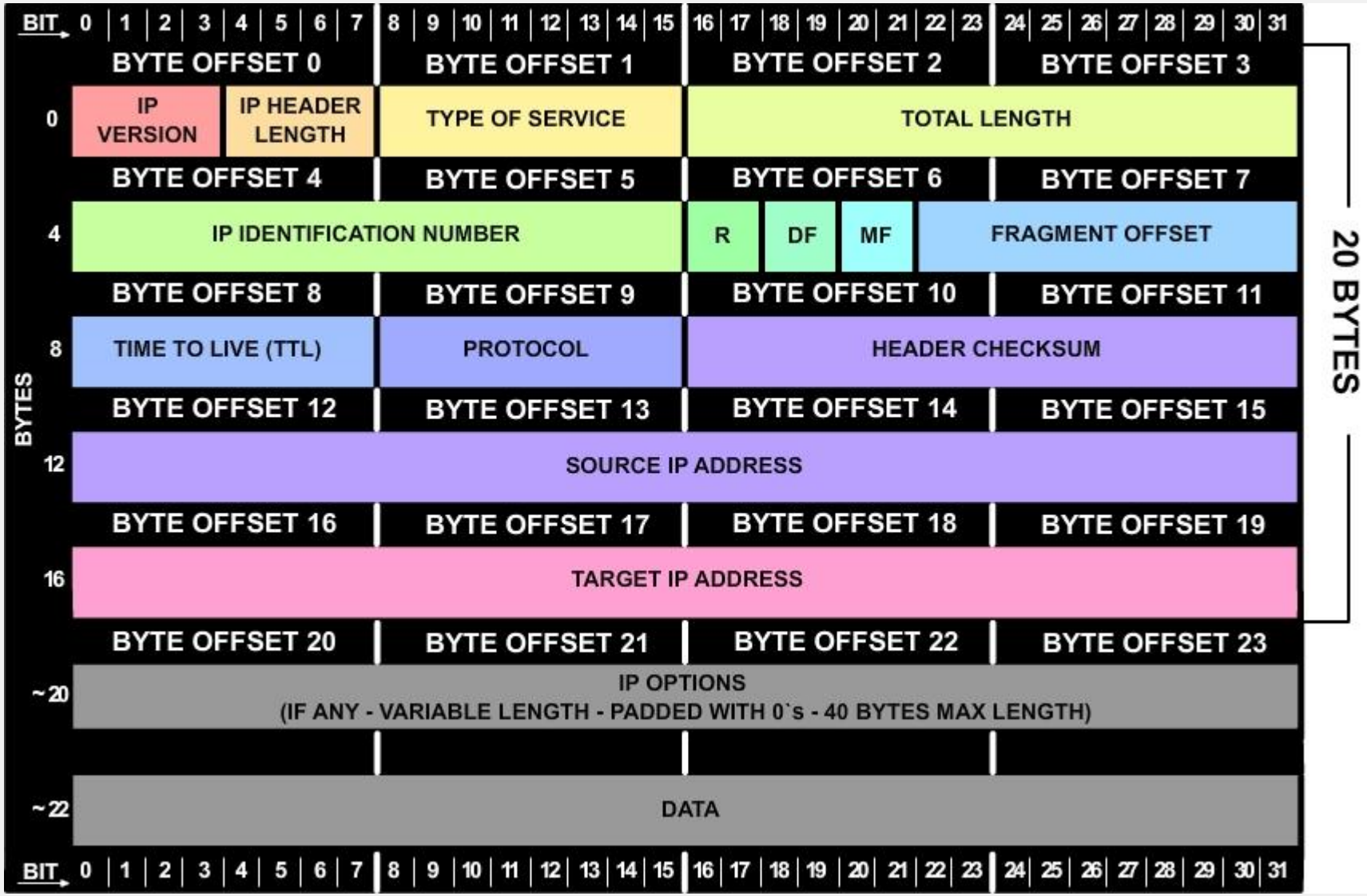
```
int tmp = 1;
setsockopt(sock, 0, IP_HDRINCL, & tmp, sizeof(tmp));
```

```
int RAWSocket = socket(PF_PACKET, SOCK_RAW,
                       <protocol>);
```

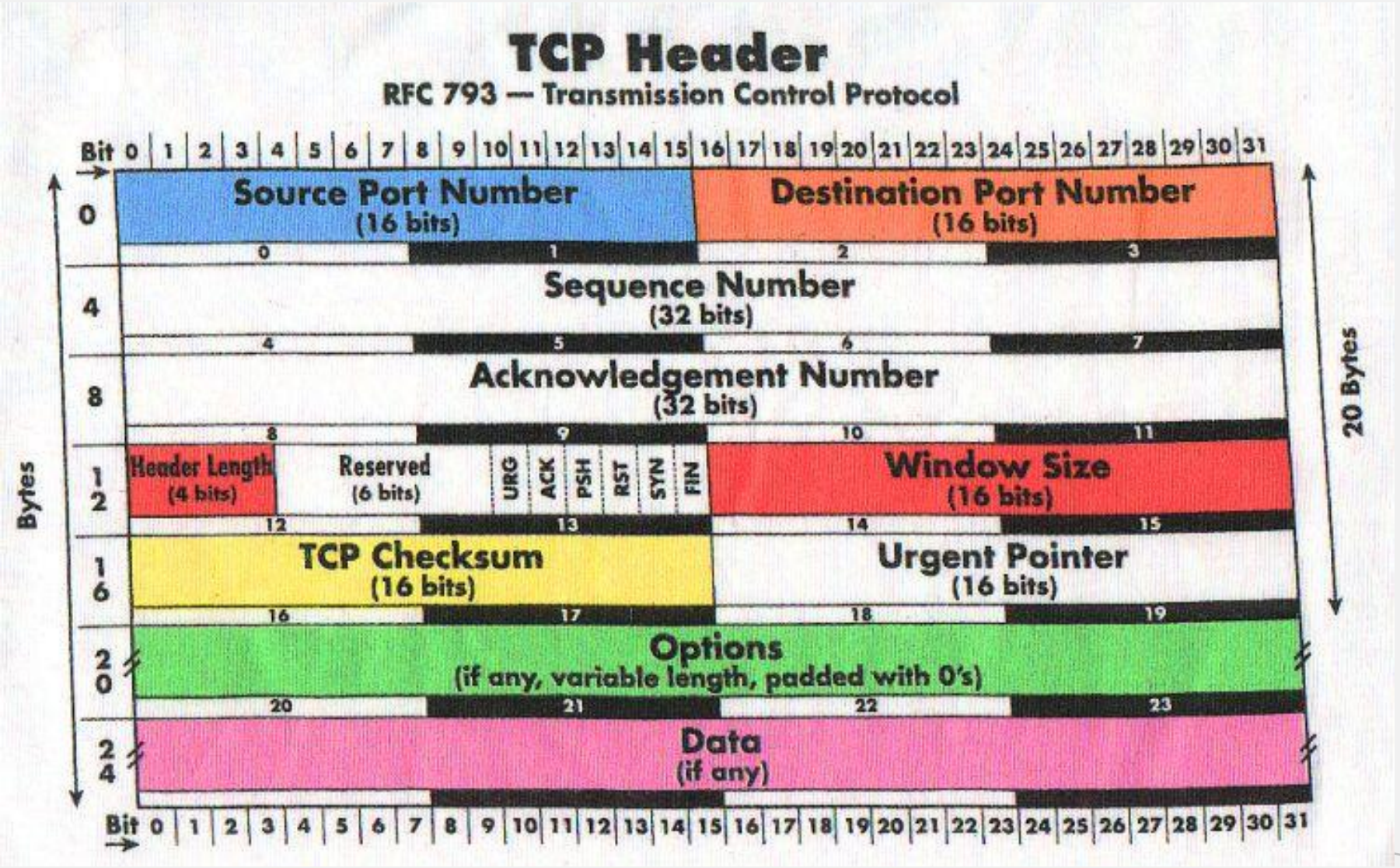
Raw-сокеты

http://www.pdbuchan.com/rawsock/rawsock.html

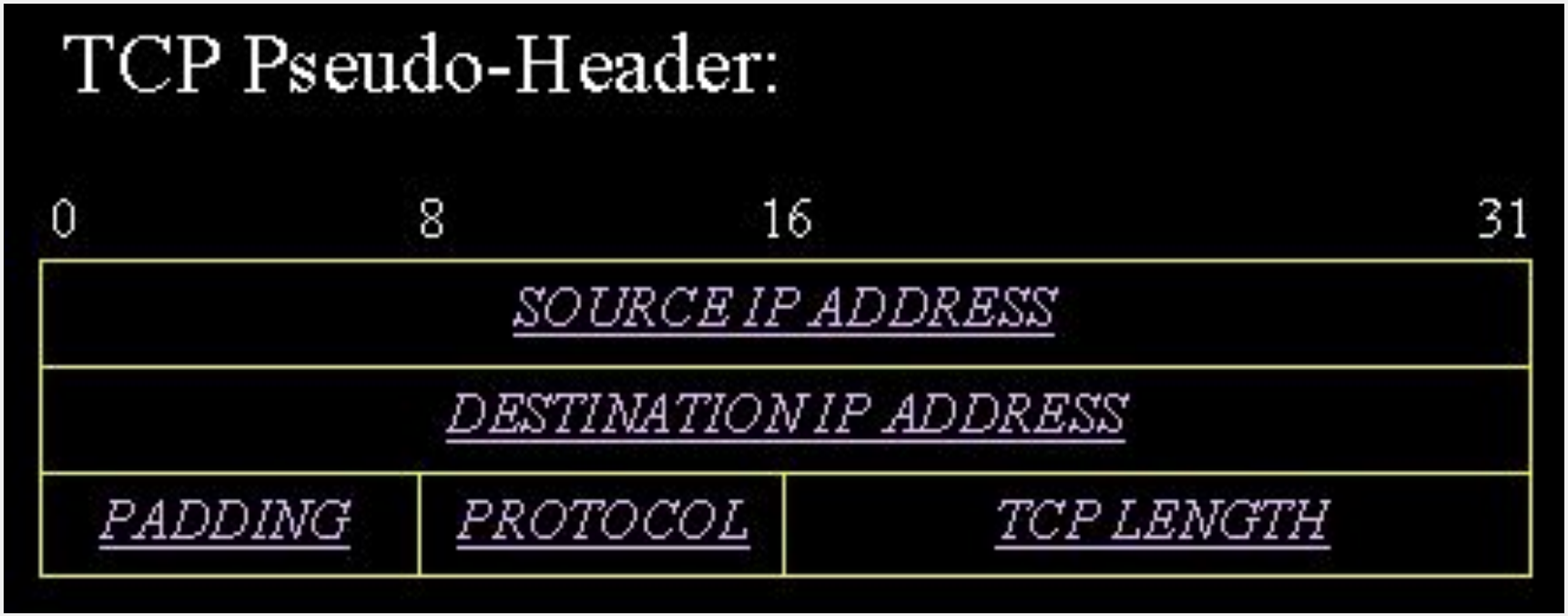
Raw-сокеты


BIT 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
BYTE OFFSET 0
BYTE OFFSET 1
BYTE OFFSET 2
BYTE OFFSET 3
0
IP VERSION
IP HEADER LENGTH
TYPE OF SERVICE
TOTAL LENGTH
BYTE OFFSET 4
BYTE OFFSET 5
BYTE OFFSET 6
BYTE OFFSET 7
4
IP IDENTIFICATION NUMBER
R
DF
MF
FRAGMENT OFFSET
BYTE OFFSET 8
BYTE OFFSET 9
BYTE OFFSET 10
BYTE OFFSET 11
8
TIME TO LIVE (TTL)
PROTOCOL
HEADER CHECKSUM
BYTE OFFSET 12
BYTE OFFSET 13
BYTE OFFSET 14
BYTE OFFSET 15
12
SOURCE IP ADDRESS
BYTE OFFSET 16
BYTE OFFSET 17
BYTE OFFSET 18
BYTE OFFSET 19
16
TARGET IP ADDRESS
BYTE OFFSET 20
BYTE OFFSET 21
BYTE OFFSET 22
BYTE OFFSET 23
~20
IP OPTIONS
(IF ANY - VARIABLE LENGTH - PADDED WITH 0`s - 40 BYTES MAX LENGTH)
~22
DATA
BIT 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
BYTES
20 BYTES

Raw-сокеты


TCP Header
RFC 793 — Transmission Control Protocol
Bit 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
0
Source Port Number
(16 bits)
Destination Port Number
(16 bits)
0 1 2 3
4
Sequence Number
(32 bits)
4 5 6 7
8
Acknowledgement Number
(32 bits)
8 9 10 11
12
Header Length
(4 bits)
Reserved
(6 bits)
URG
ACK
PSH
RST
SYN
FIN
Window Size
(16 bits)
12 13 14 15
16
TCP Checksum
(16 bits)
Urgent Pointer
(16 bits)
16 17 18 19
20
Options
(if any, variable length, padded with 0's)
20 21 22 23
24
Data
(if any)
Bit 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
Bytes
20 Bytes

Raw-сокеты


TCP Pseudo-Header:
0
8
16
31
SOURCE IP ADDRESS
DESTINATION IP ADDRESS
PADDING
PROTOCOL
TCP LENGTH

Домашнее задание №1

- Создать HTTP-сервер.
- Сборка через make.
- Запуск:

  ./wwwd -d <dir> -h <ip> -p <port>

- Реализация HEAD/GET/POST.
- Статусы 200 и 404.
- В каталоге <dir> - html и jpeg файлы.

Срок сдачи

3 апреля (Коллоквиум)

ТЕХНОСФЕРА

Спасибо за внимание!

Дмитрий Калугин-Балашов

rvncerr@rvncerr.org