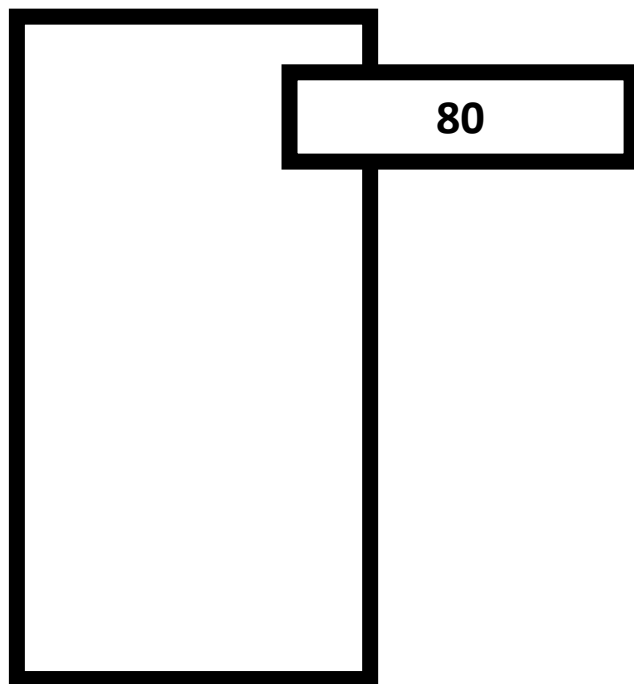


Курс «Проектирование больших систем на языке C++»

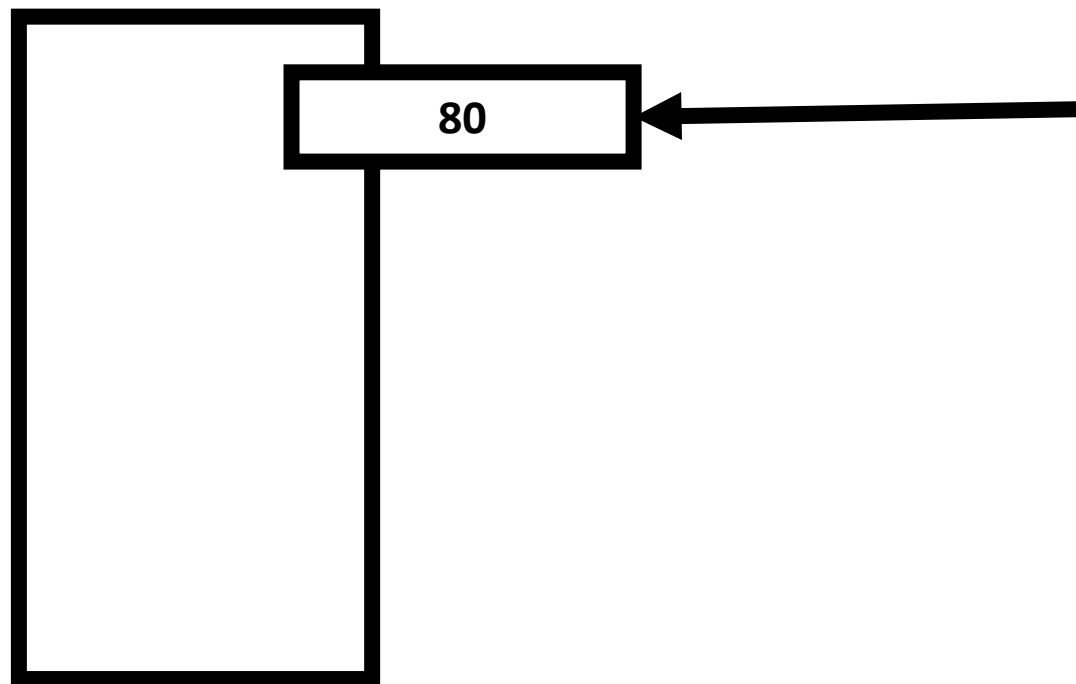
Лекция

Сокеты

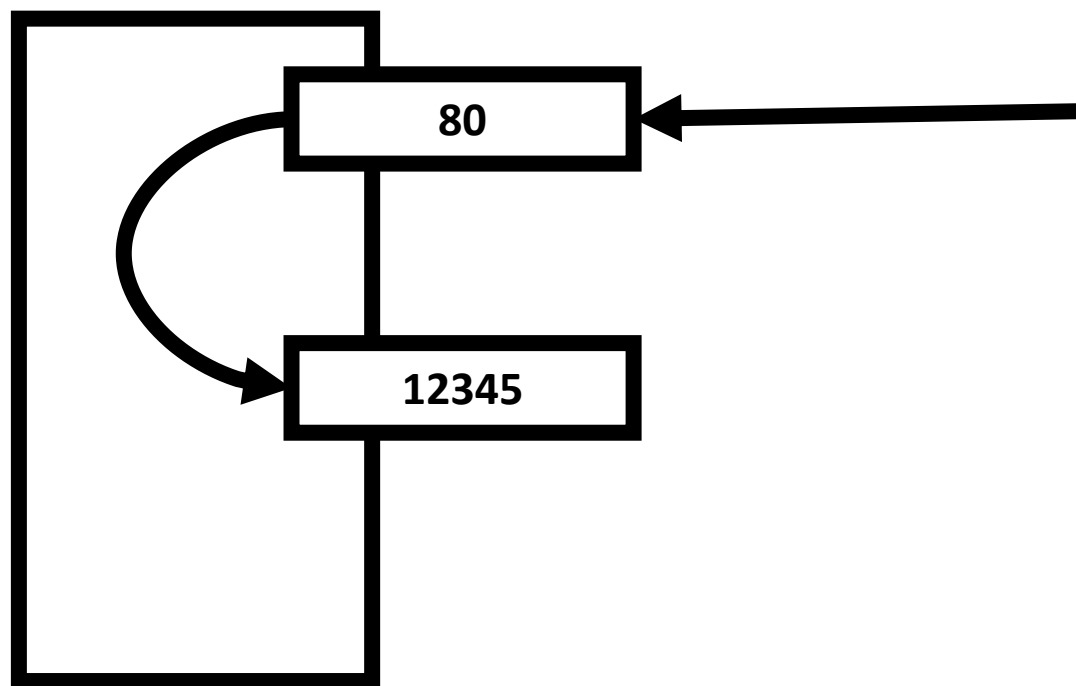
Сокеты Беркли



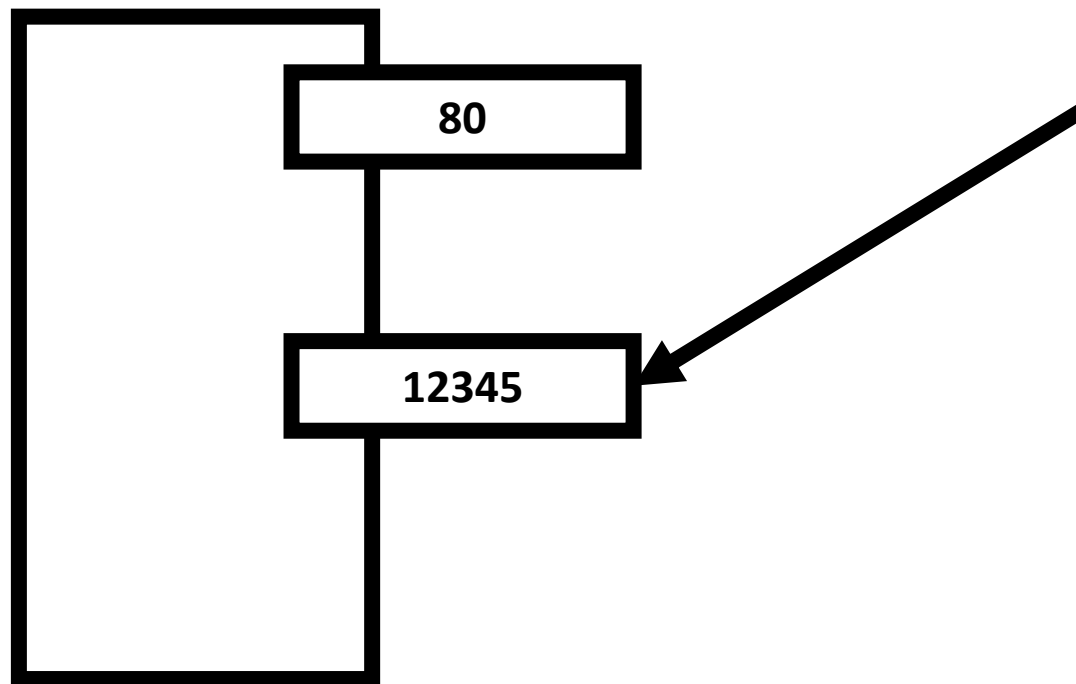
Сокеты Беркли



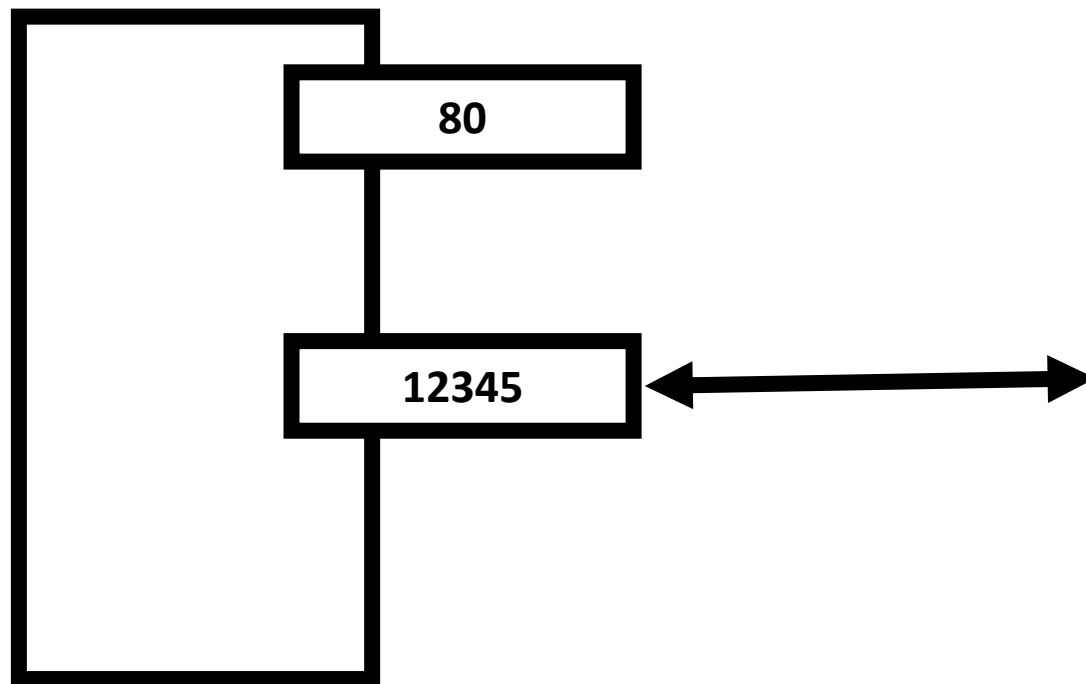
Сокеты Беркли



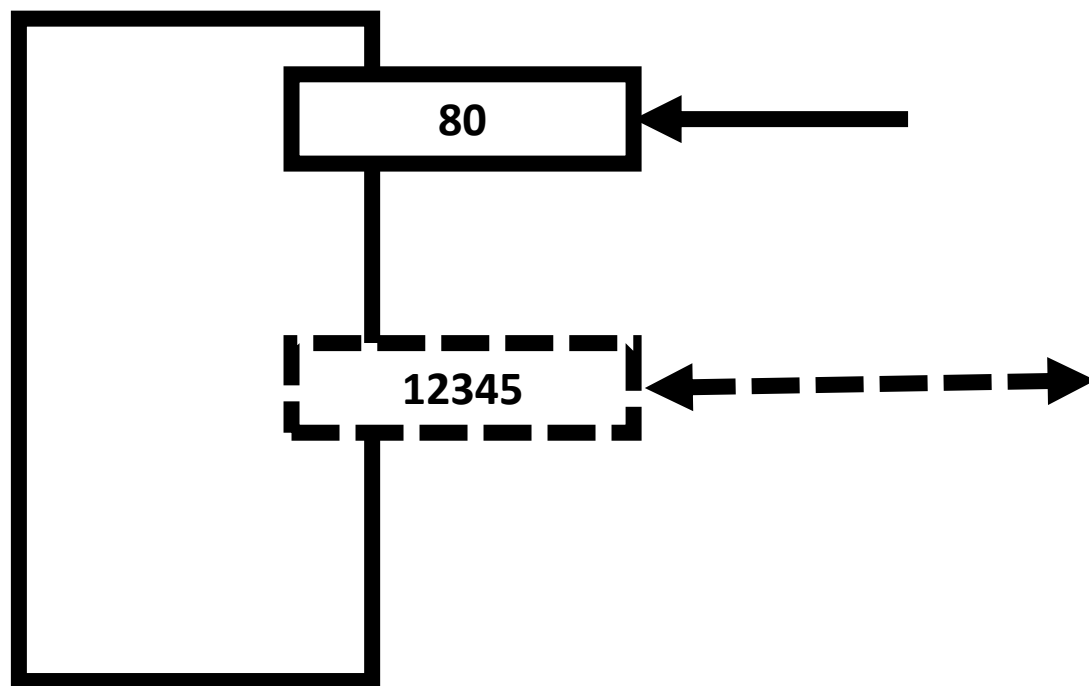
Сокеты Беркли



Сокеты Беркли

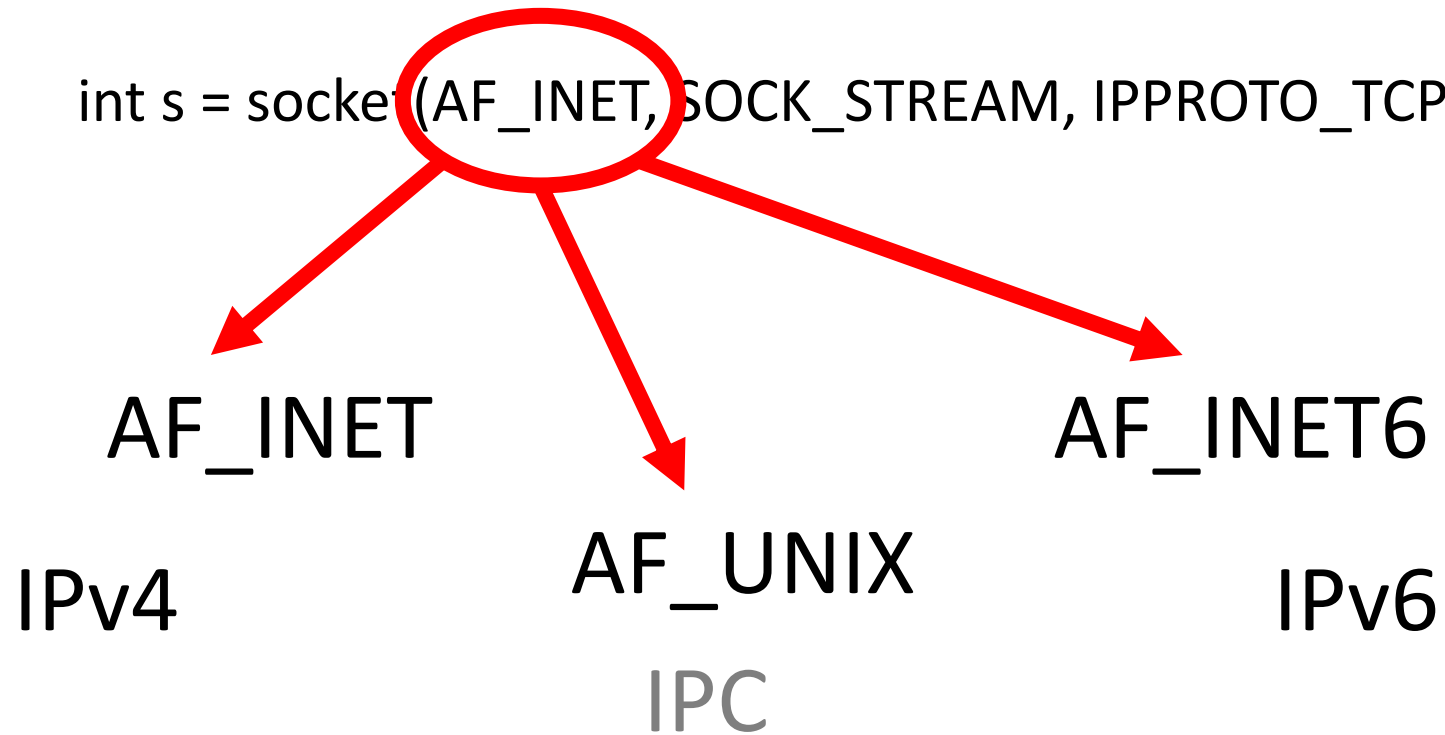


Сокеты Беркли



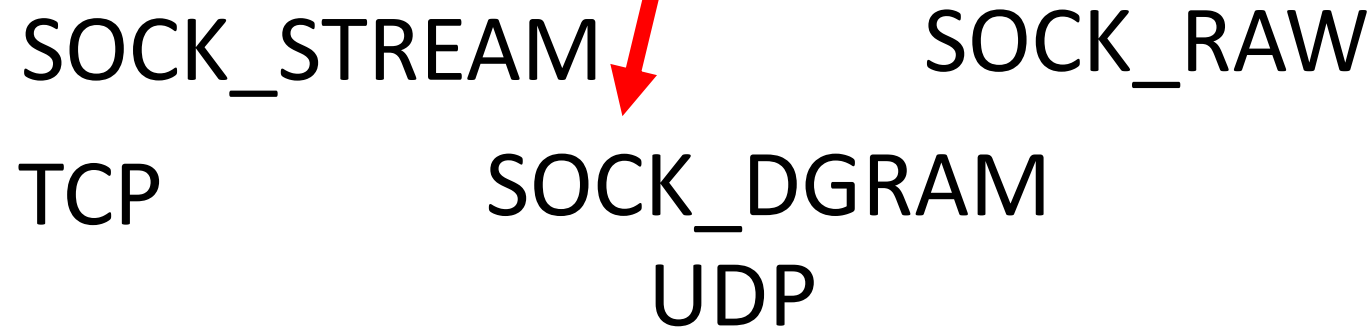
Сокеты Беркли

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



Сокеты Беркли

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



Сокеты Беркли

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

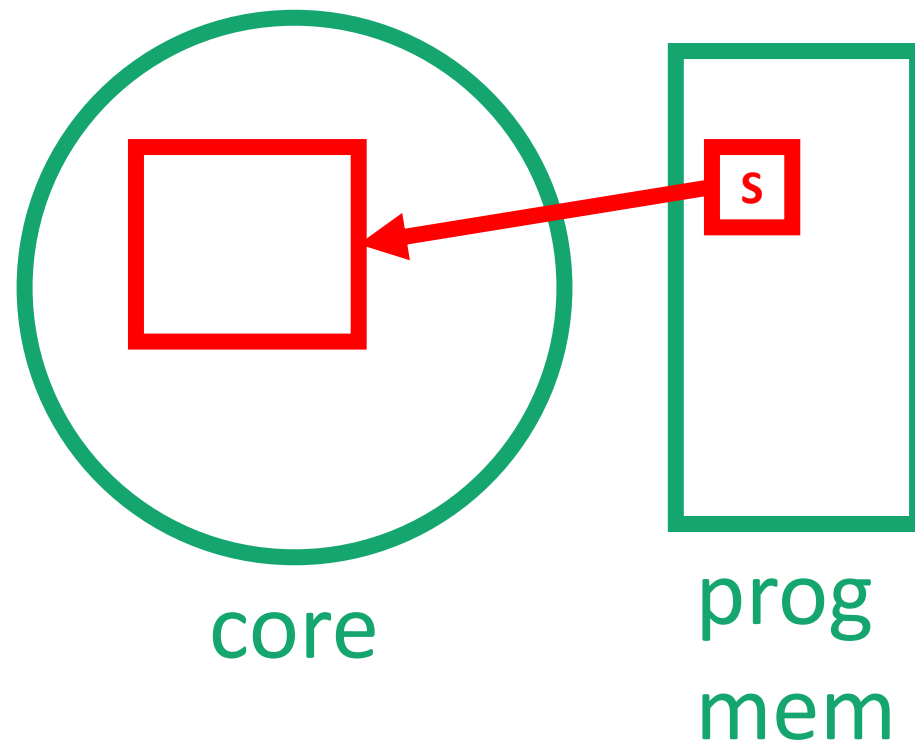
IPPROTO_TCP

0

IPPROTO_UDP

Сокеты Беркли

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



Сокеты Беркли

Таблицы открытых
файлов процесса
FDT

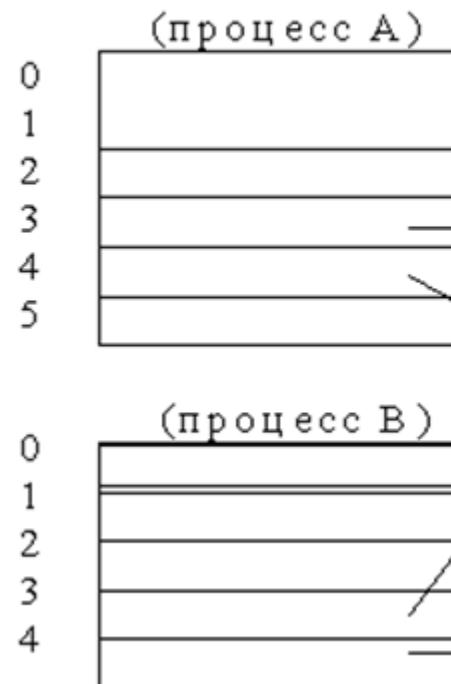


Таблица
файлов SFT

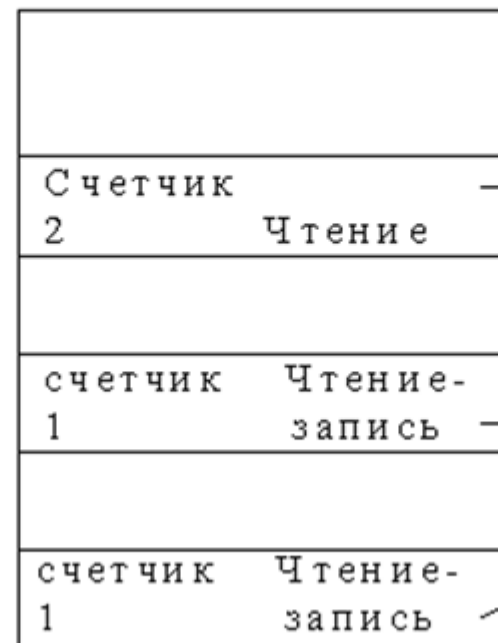


Таблица
описателей
файлов vnode



Сокеты Беркли

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

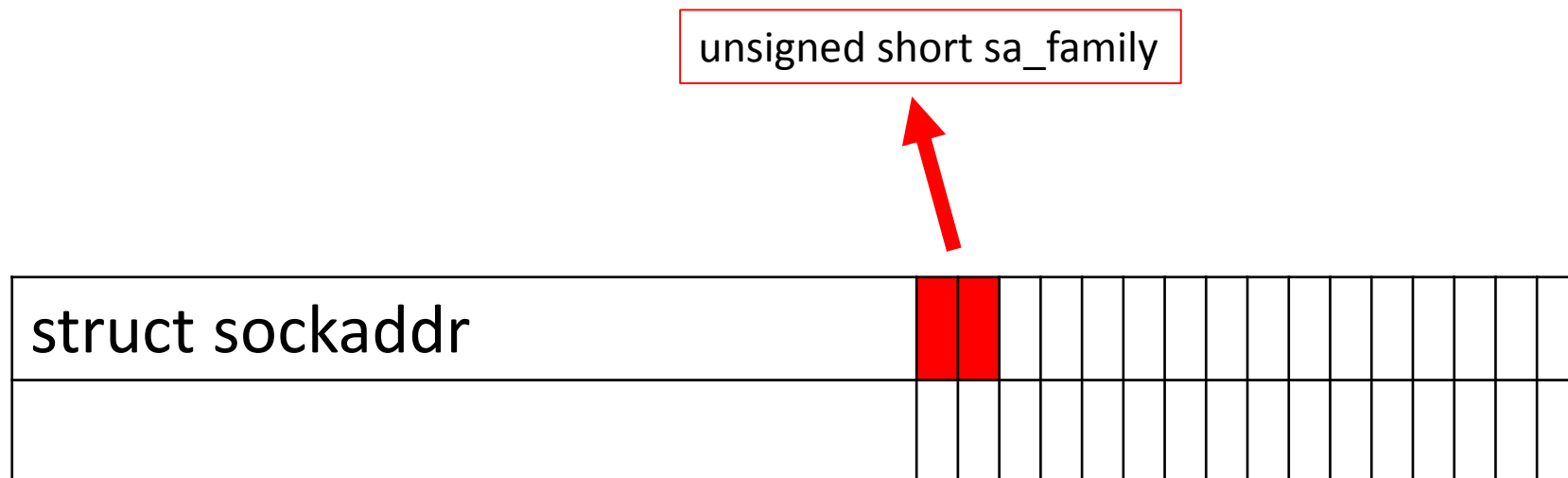
Сокеты Беркли

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

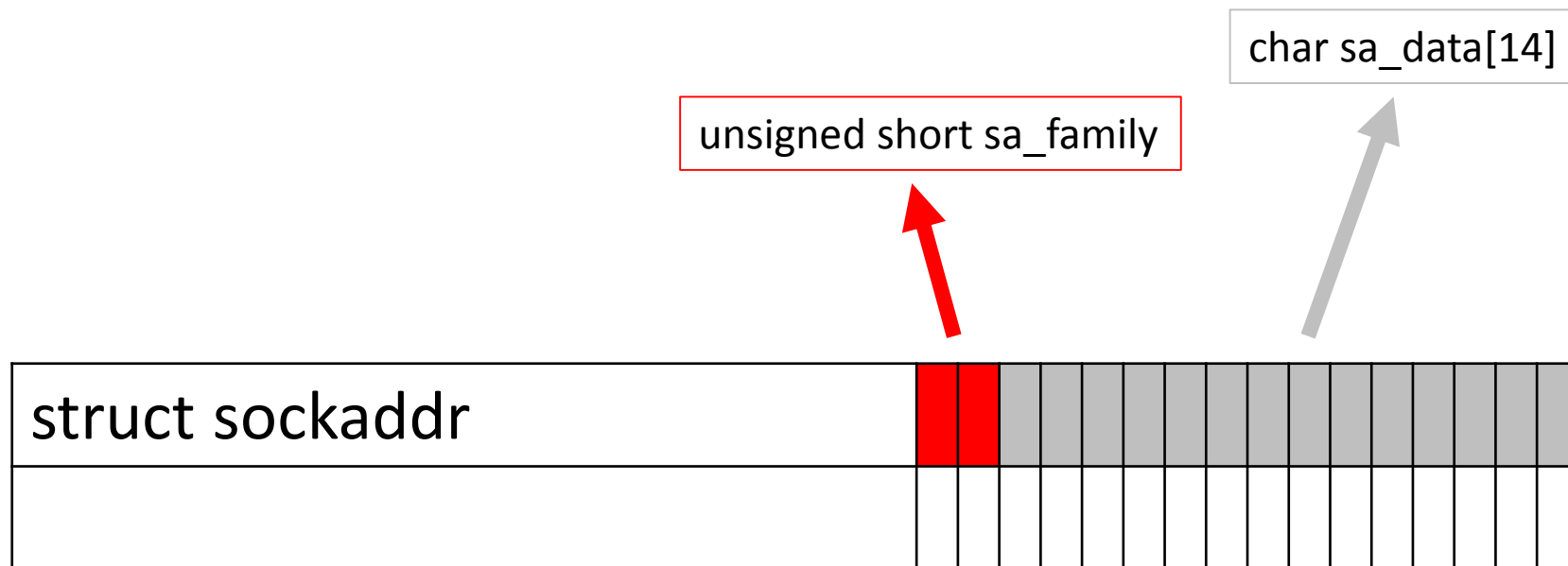
Сокеты Беркли

struct sockaddr																	

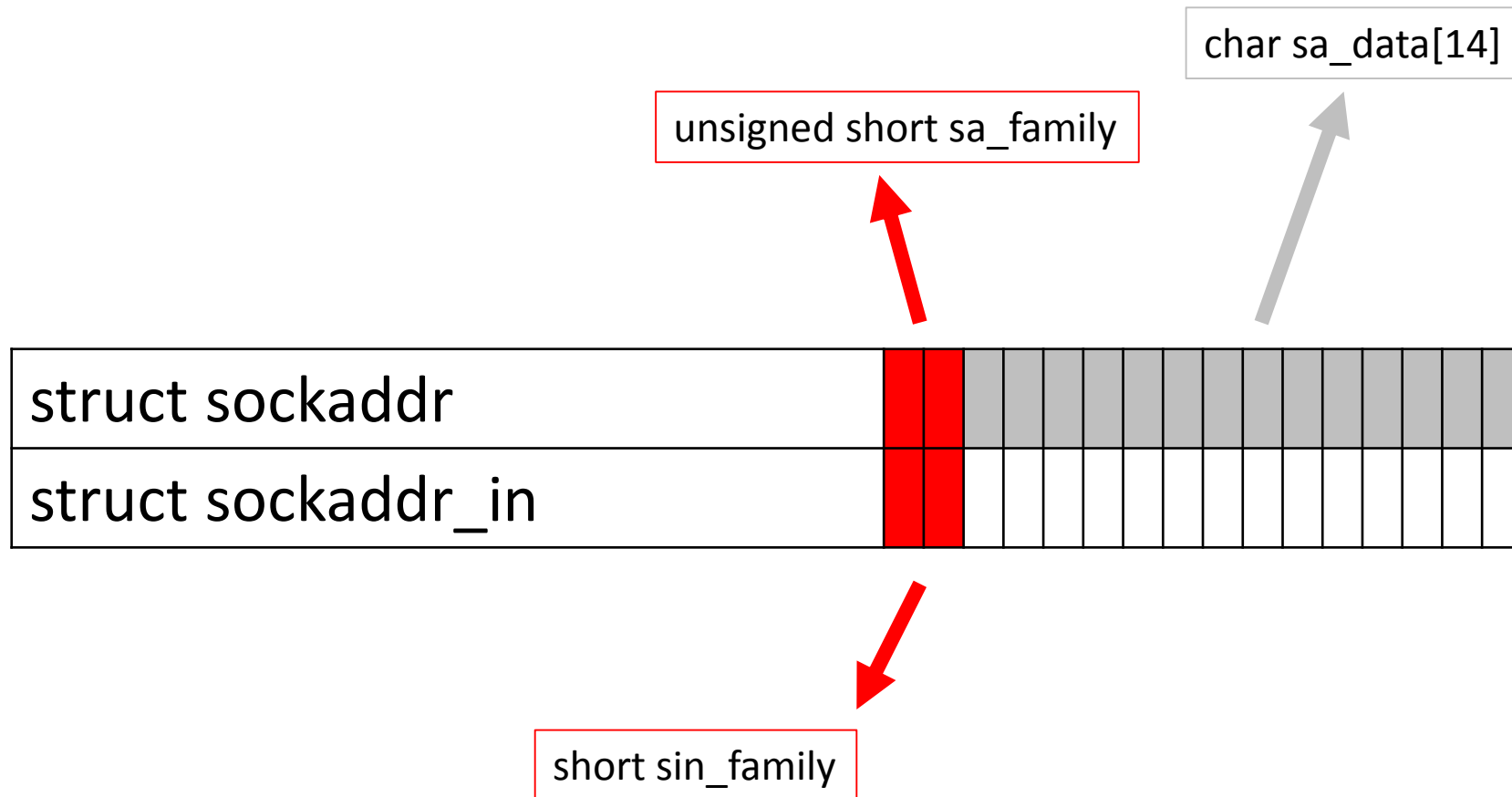
Сокеты Беркли



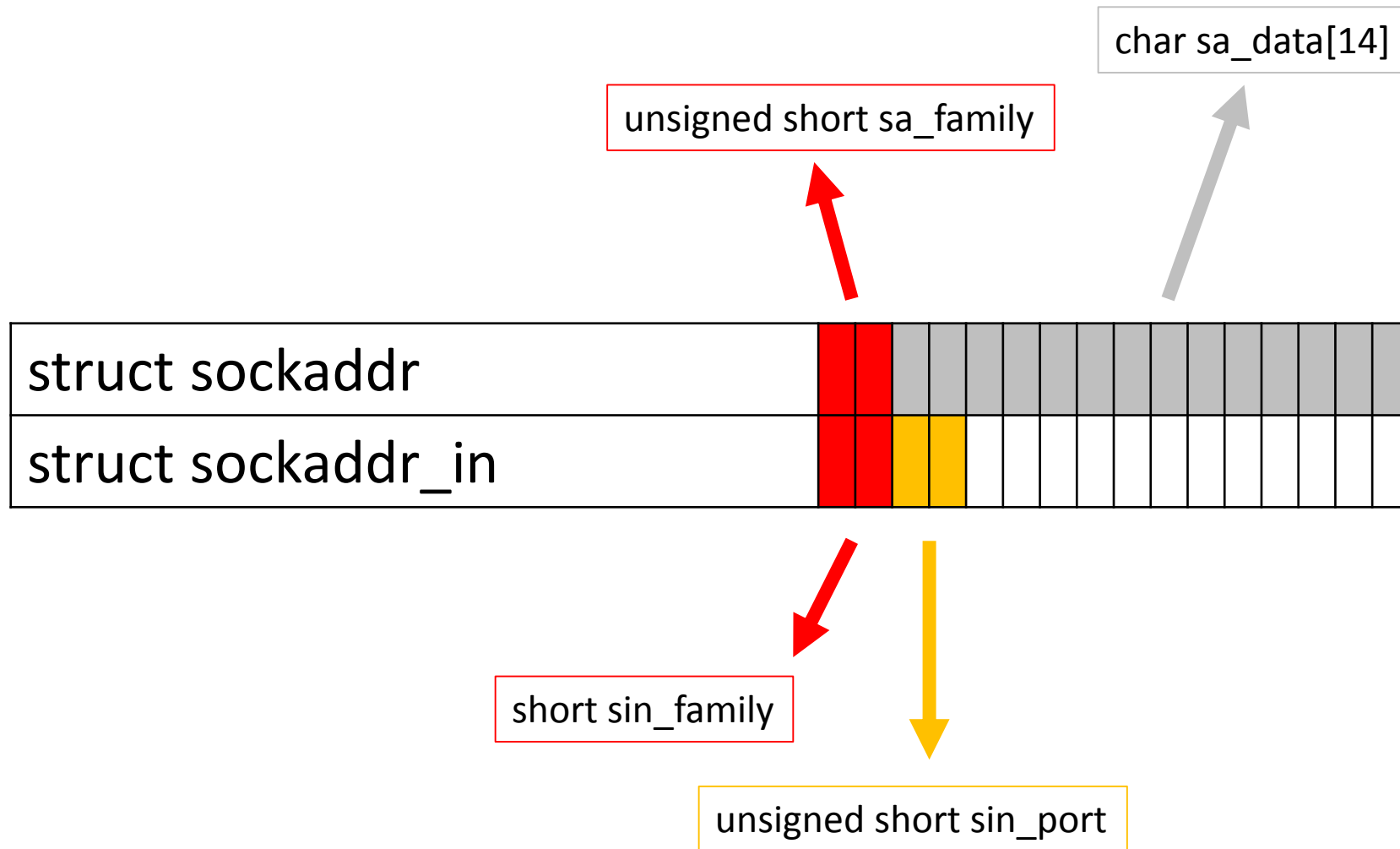
Сокеты Беркли



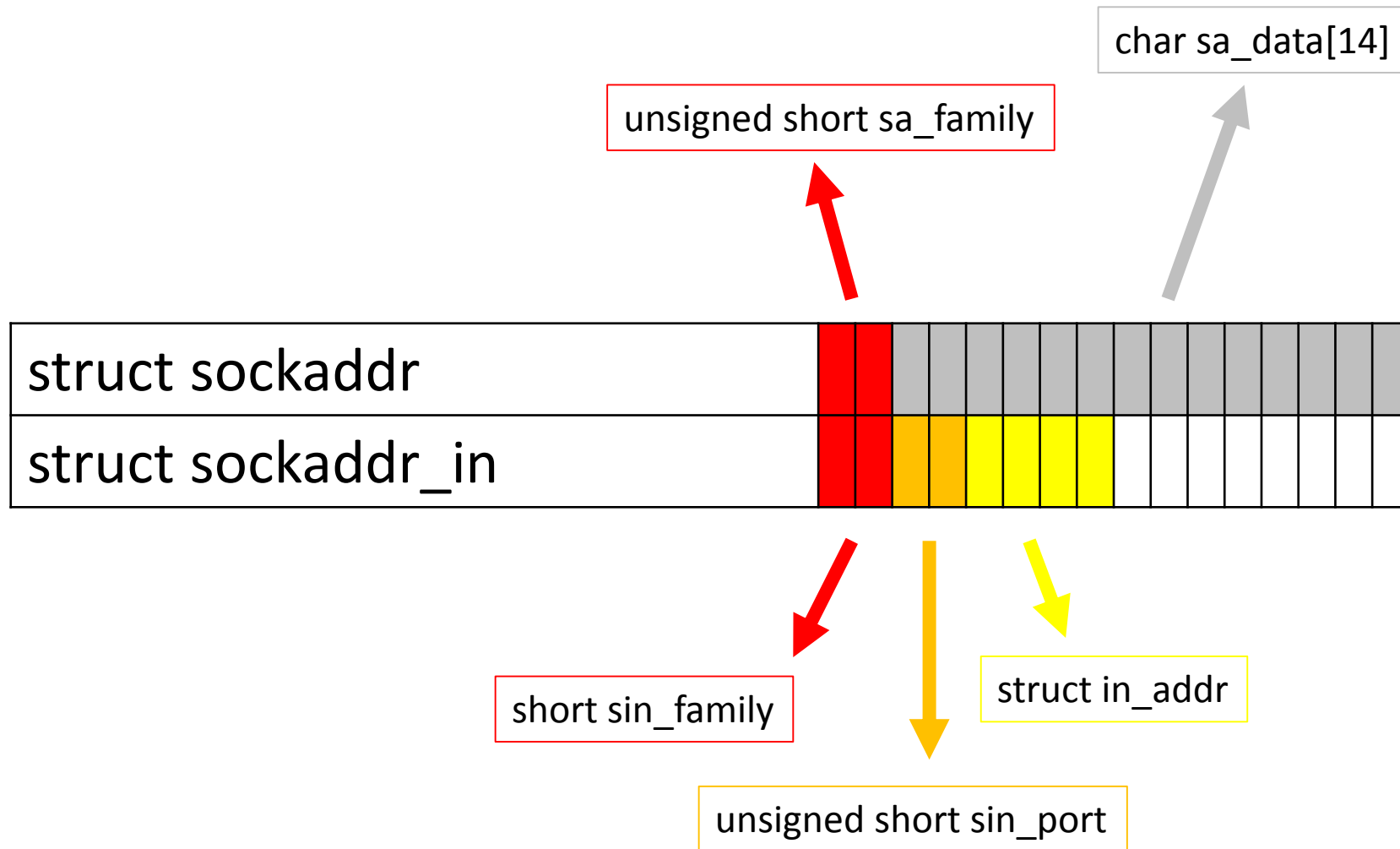
Сокеты Беркли



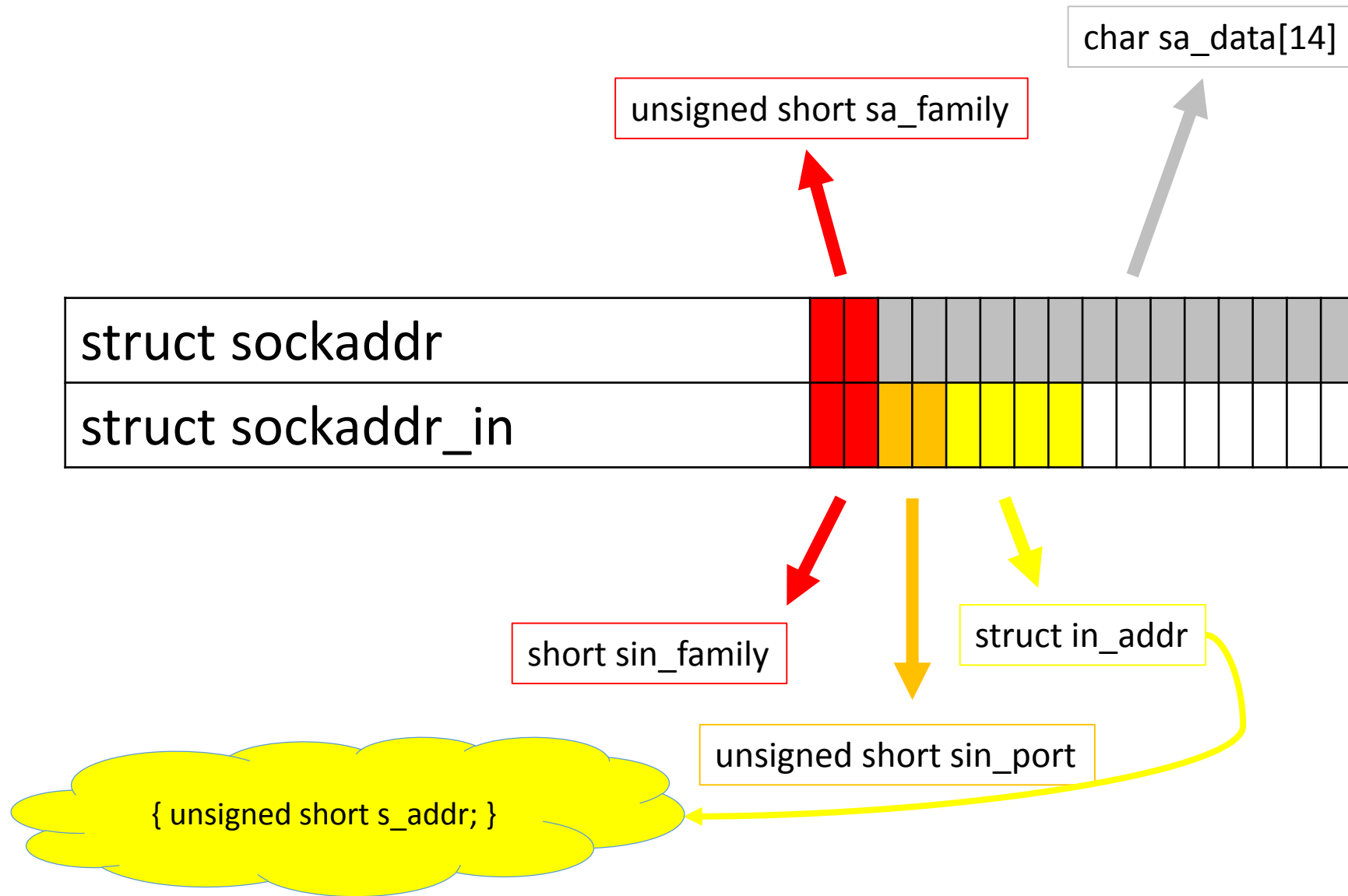
Сокеты Беркли



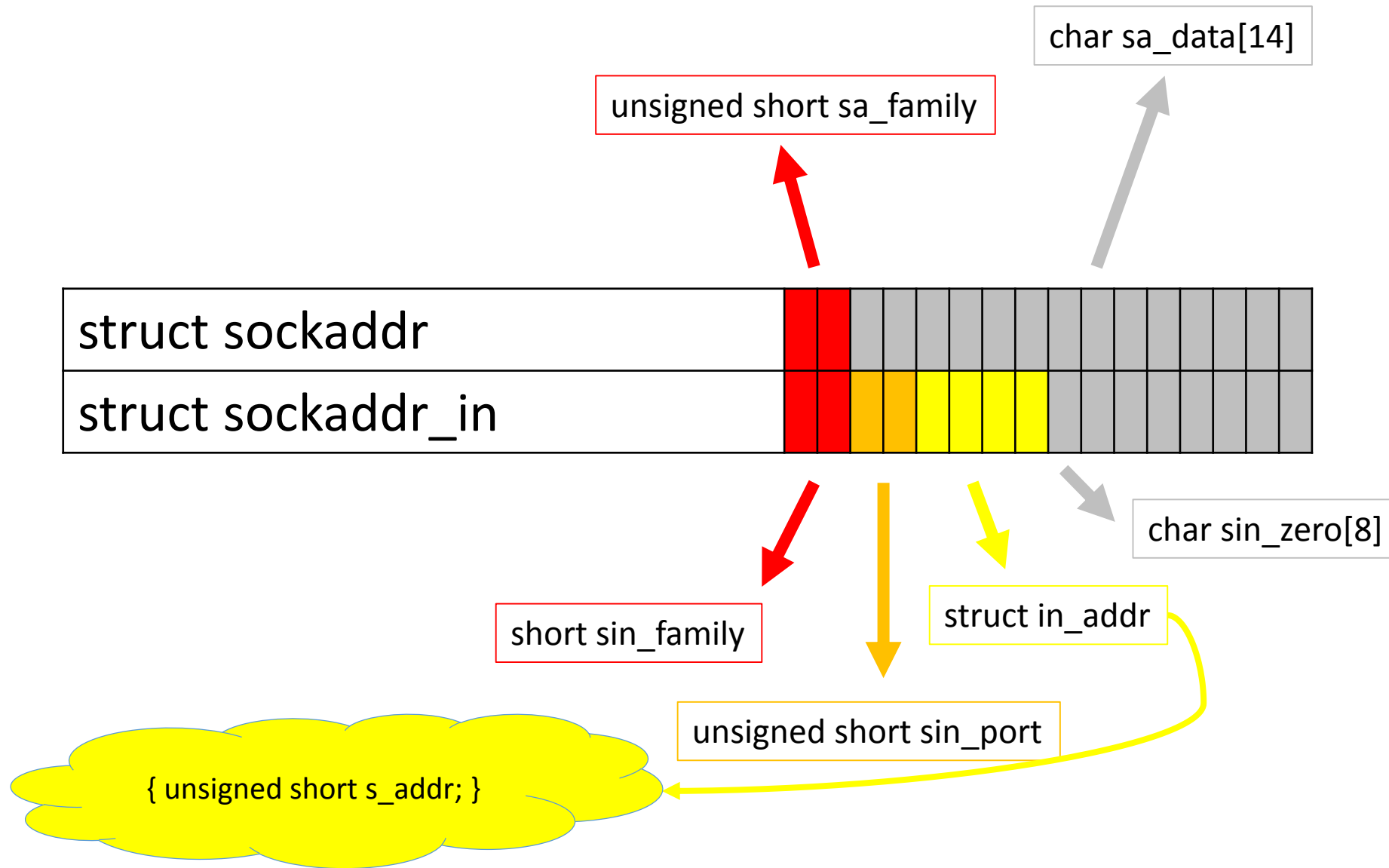
Сокеты Беркли



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));`

IPv4

Заполнение структуры sockaddr_in

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

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;
```

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);
```

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);
```

IPv4

Заполнение структуры sockaddr_in

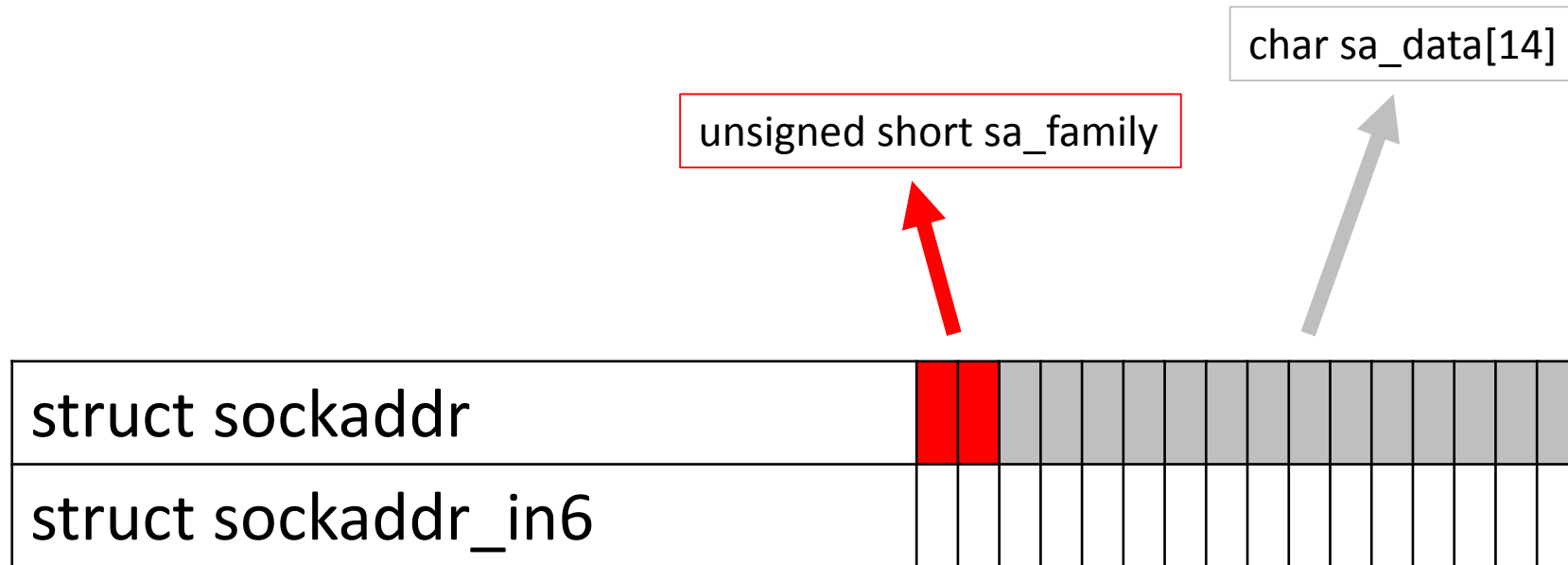
```
1. struct sockaddr_in SockAddr;
2. // memset(&SockAddr, 0, sizeof(SockAddr));
3. bzero(&SockAddr, sizeof(SockAddr));
4. SockAddr.sin_family = AF_INET;
5. SockAddr.sin_port = htons(12345);
6. // SockAddr.sin_addr.s_addr = htonl(INADDR_ANY);
7. // SockAddr.sin_addr.s_addr = htonl(INADDR_LOOPBACK);
8. SockAddr.sin_addr.s_addr = inet_addr("0.0.0.0");
```

IPv4

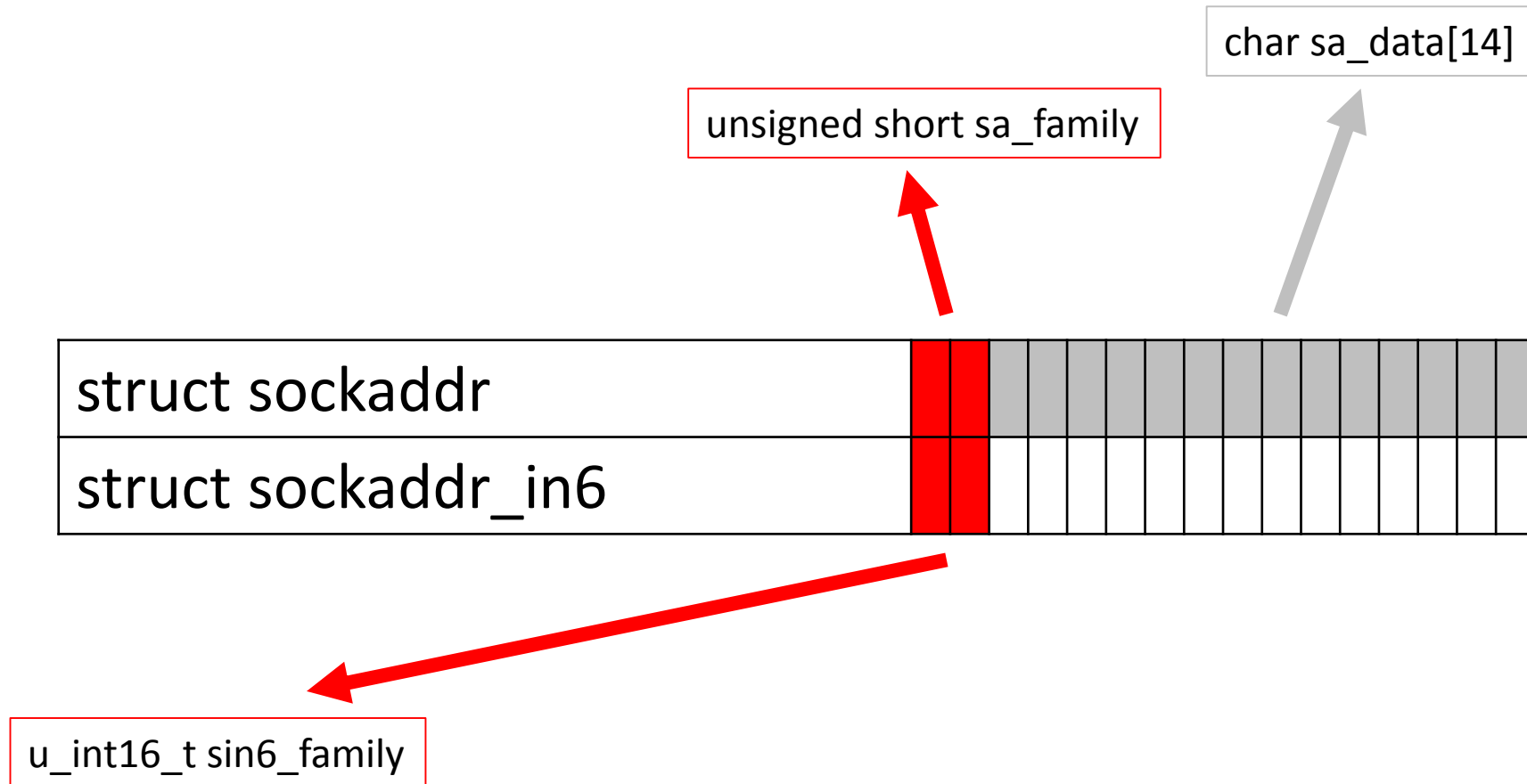
Заполнение структуры sockaddr_in

```
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("0.0.0.0");
9.  struct hostent * hp = gethostbyname("example.org");
10. bcopy(hp->h_addr, &(SockAddr.sin_addr.s_addr), hp->h_length);
```

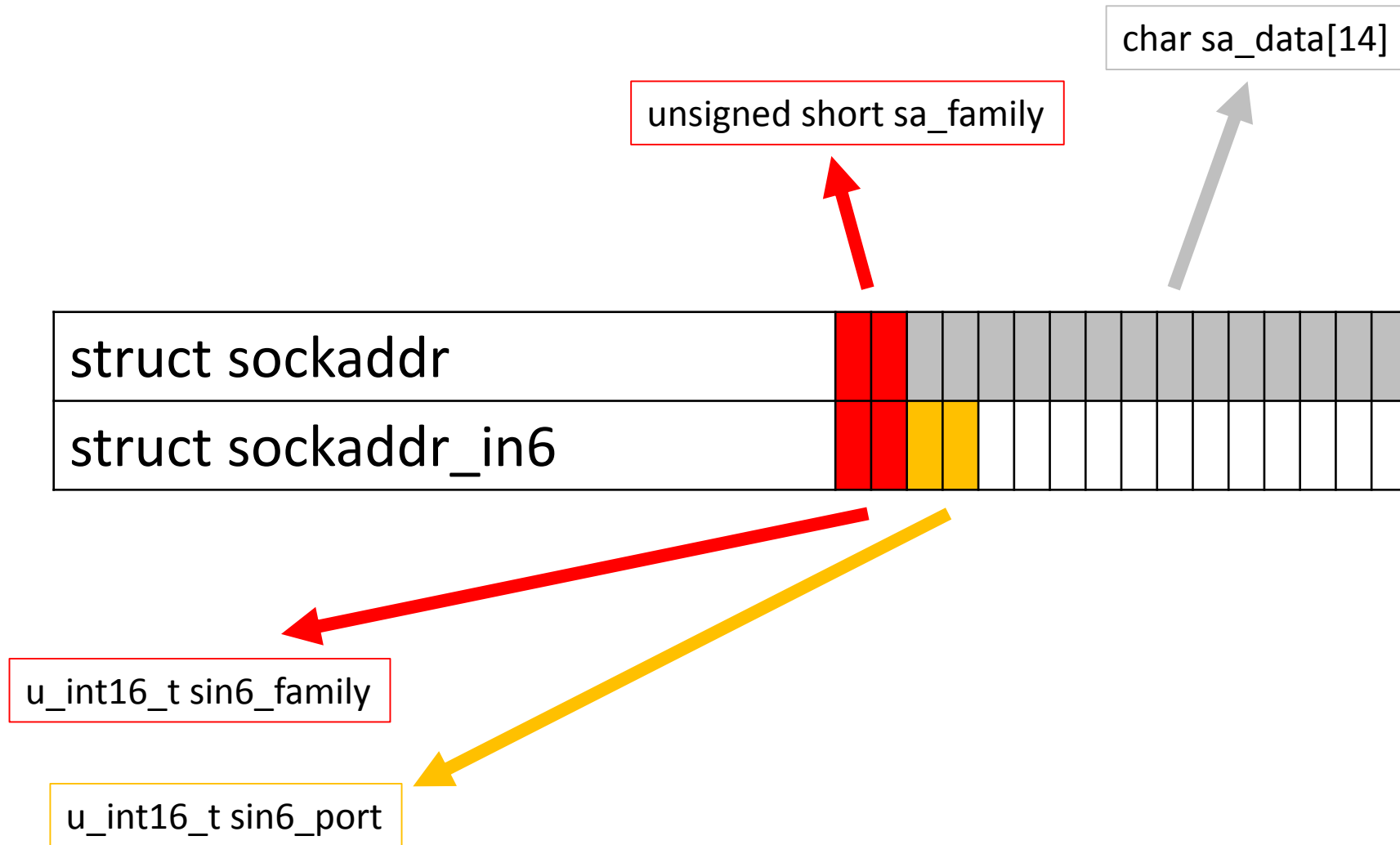
IPv6



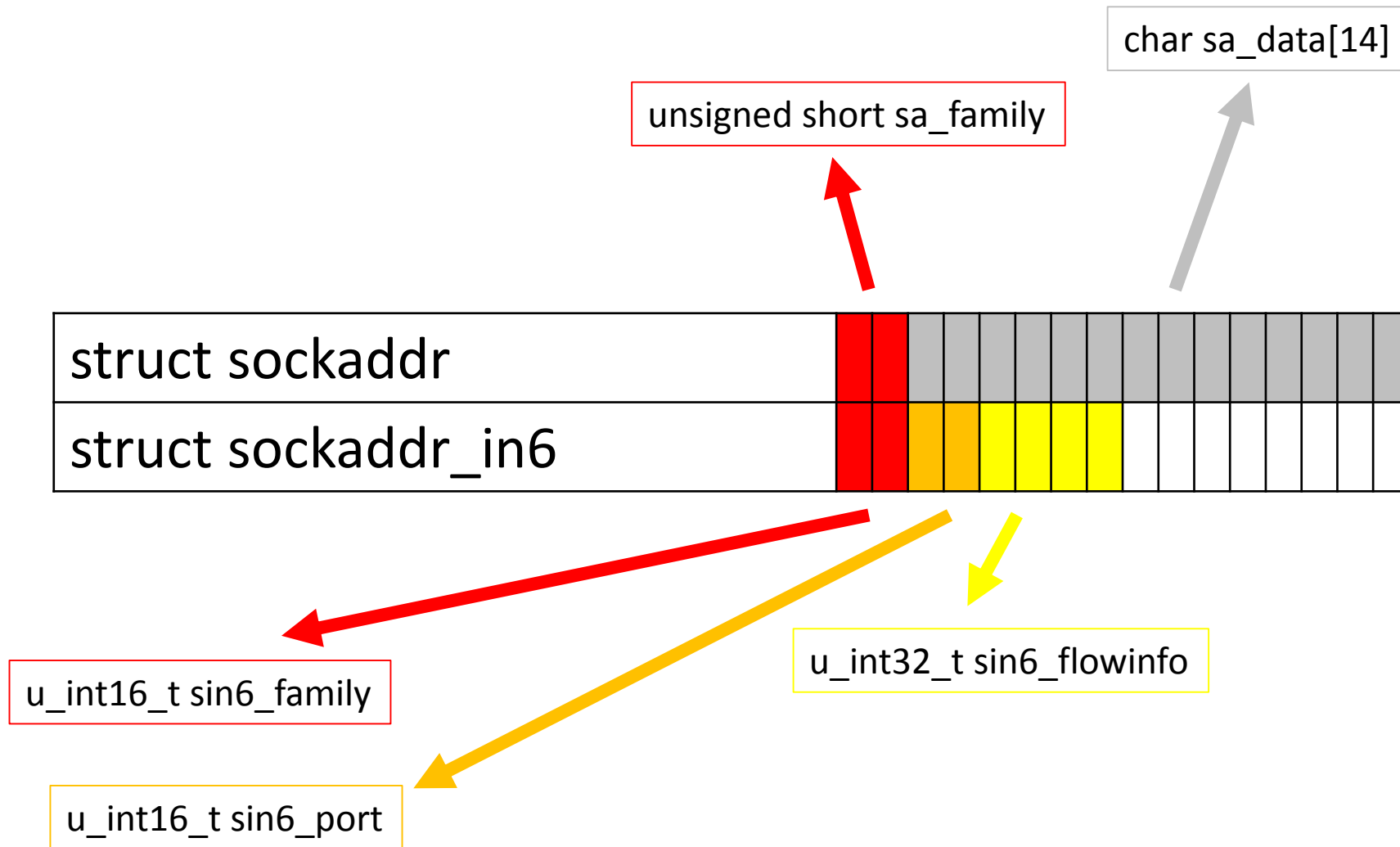
IPv6



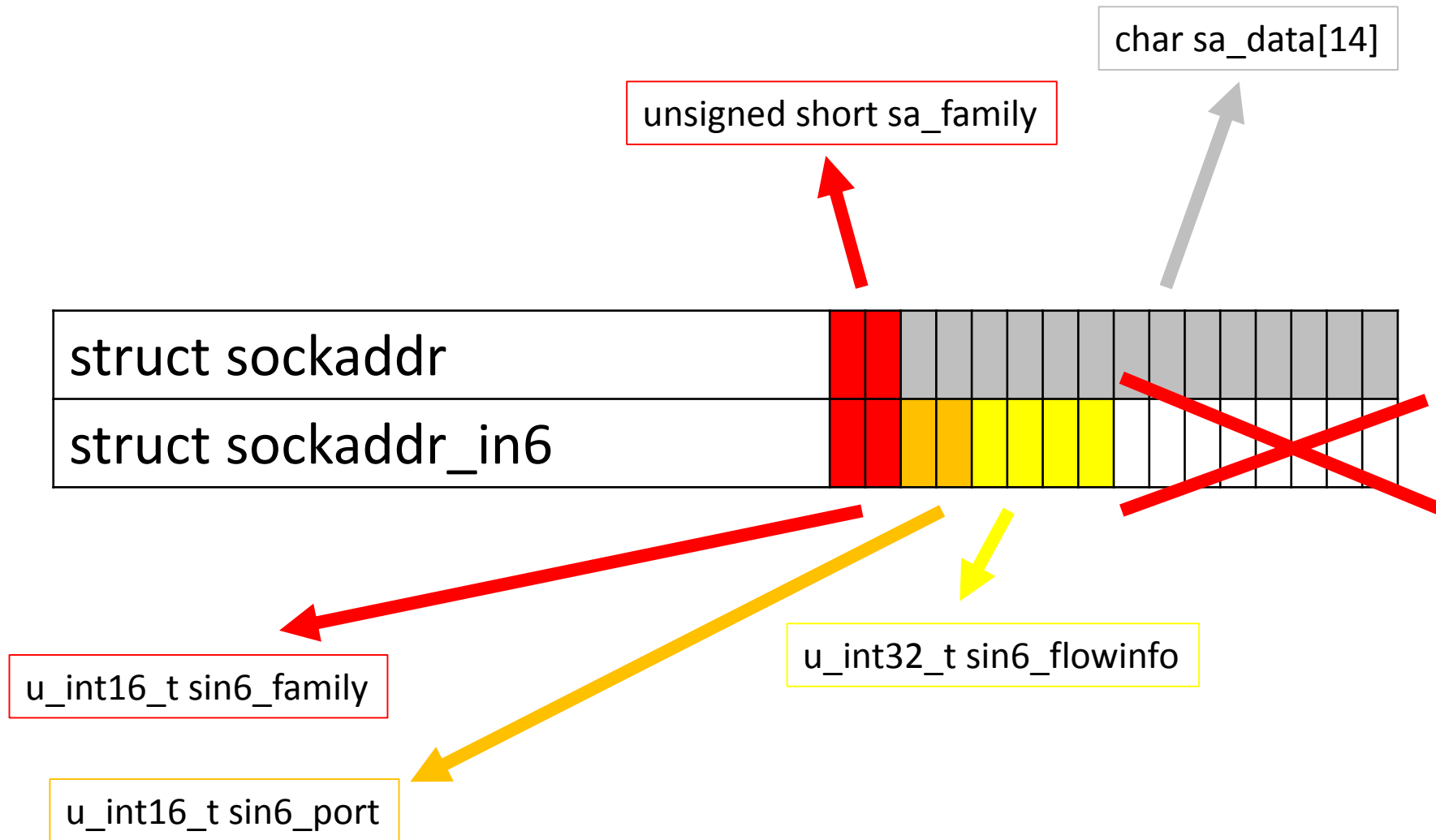
IPv6



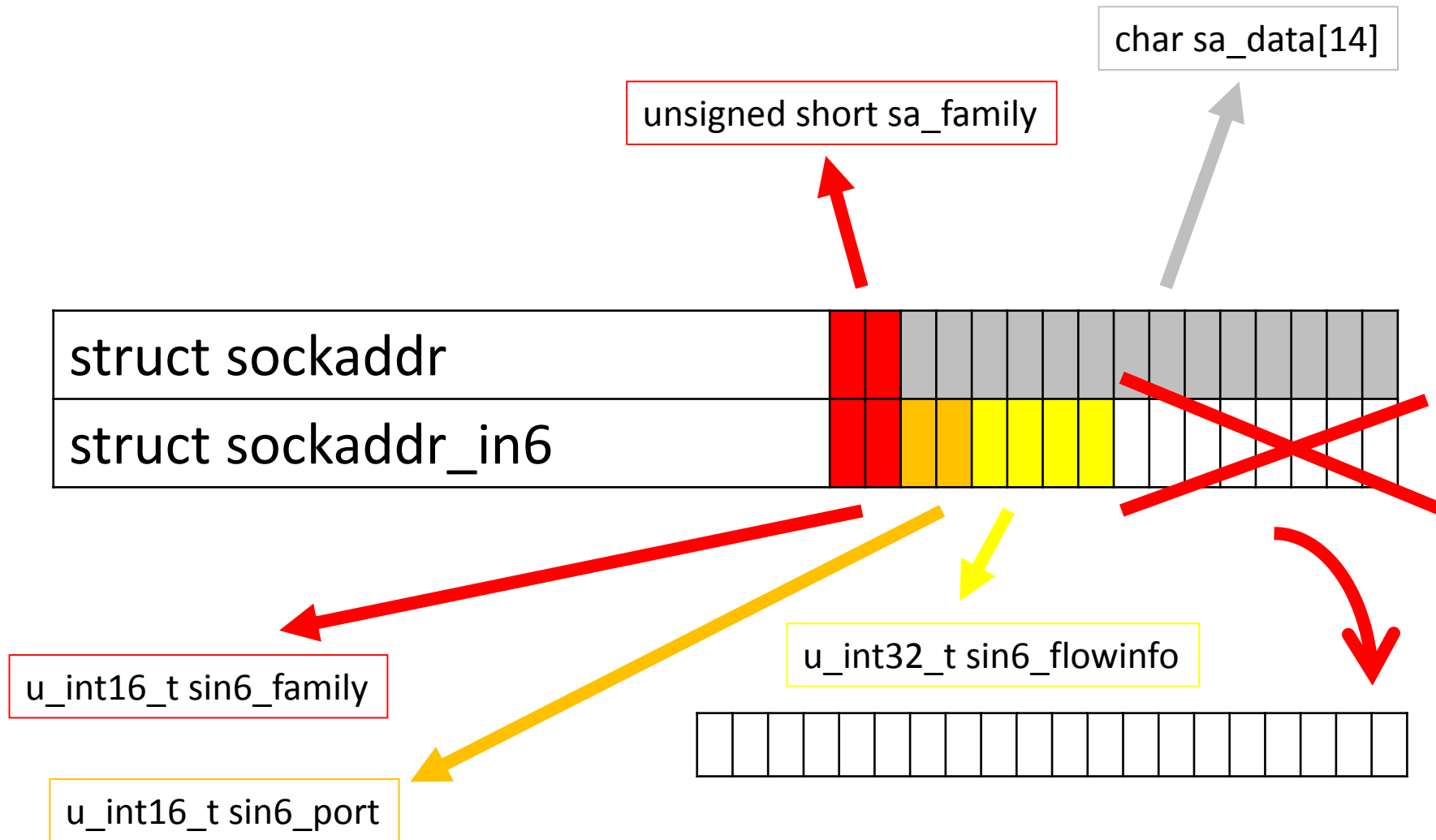
IPv6



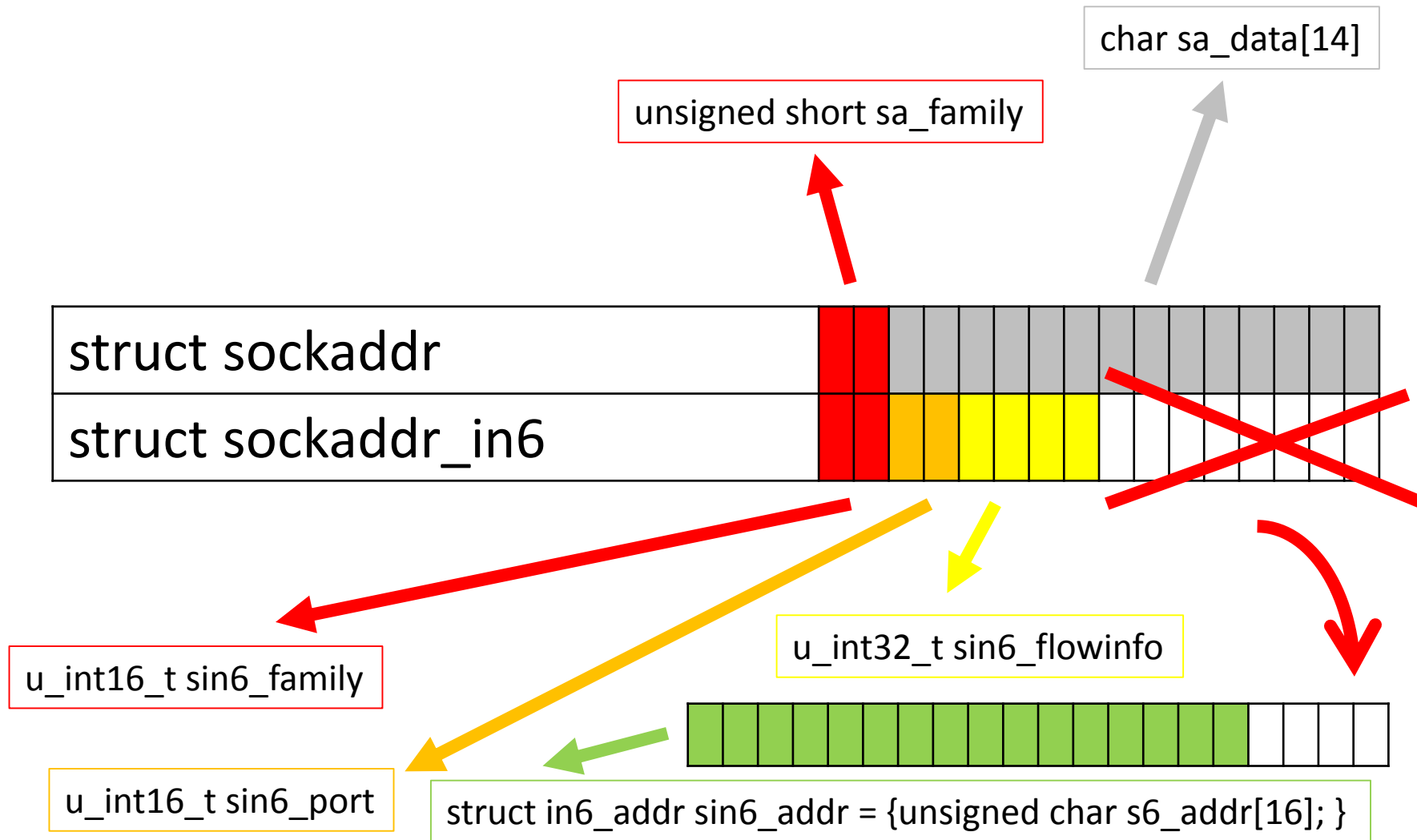
IPv6



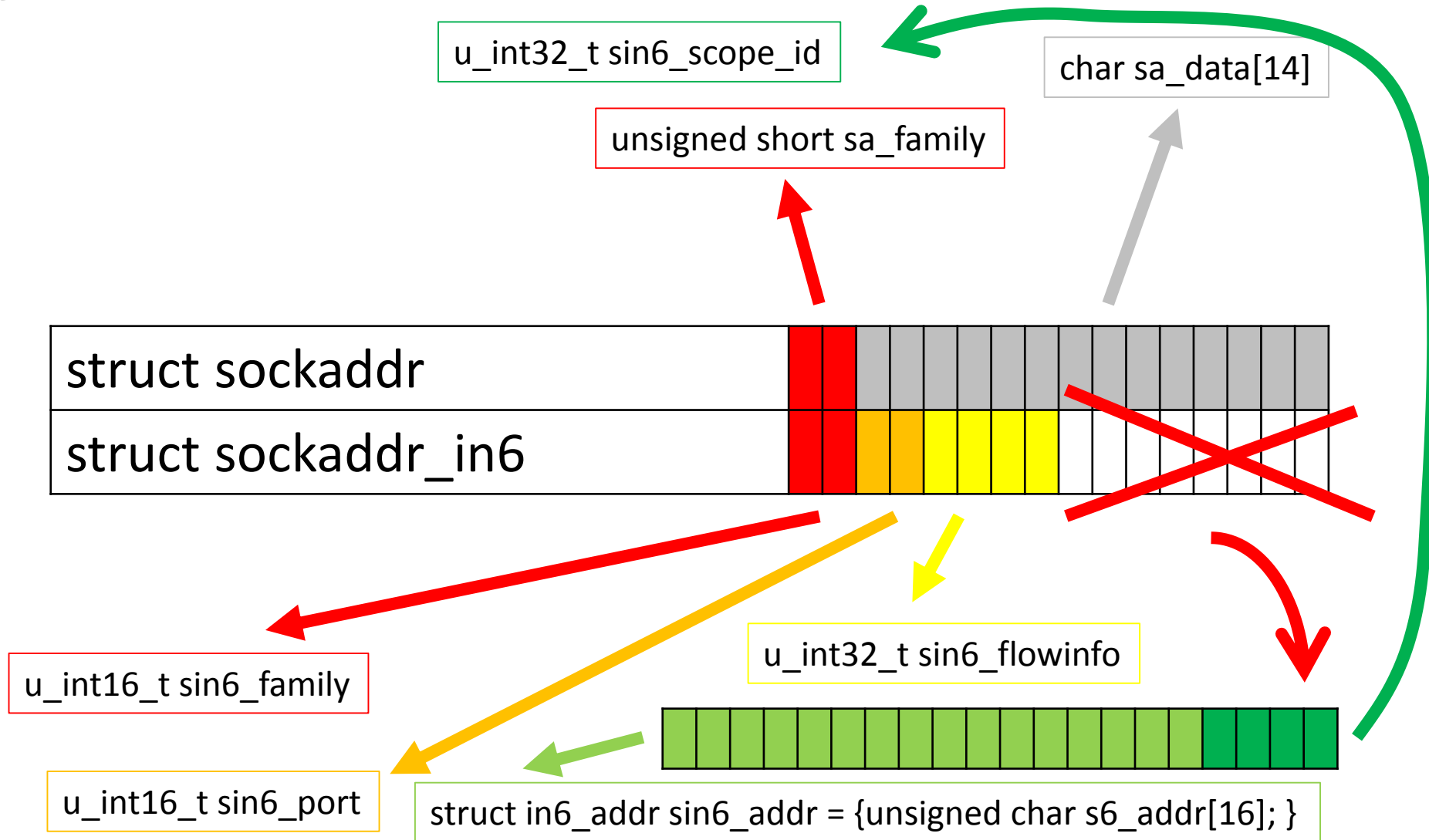
IPv6



IPv6



IPv6



Заполнение структуры sockaddr_in6

```
1.  SockAddr.sin_addr.s_addr = inet_addr("0.0.0.0");
```

Заполнение структуры sockaddr_in6

1. ~~SocketAddr.sin_addr.s_addr = inet_addr("0.0.0.0");~~
2. `inet_pton(AF_INET, "0.0.0.0", &(SocketAddr.sin_addr));`

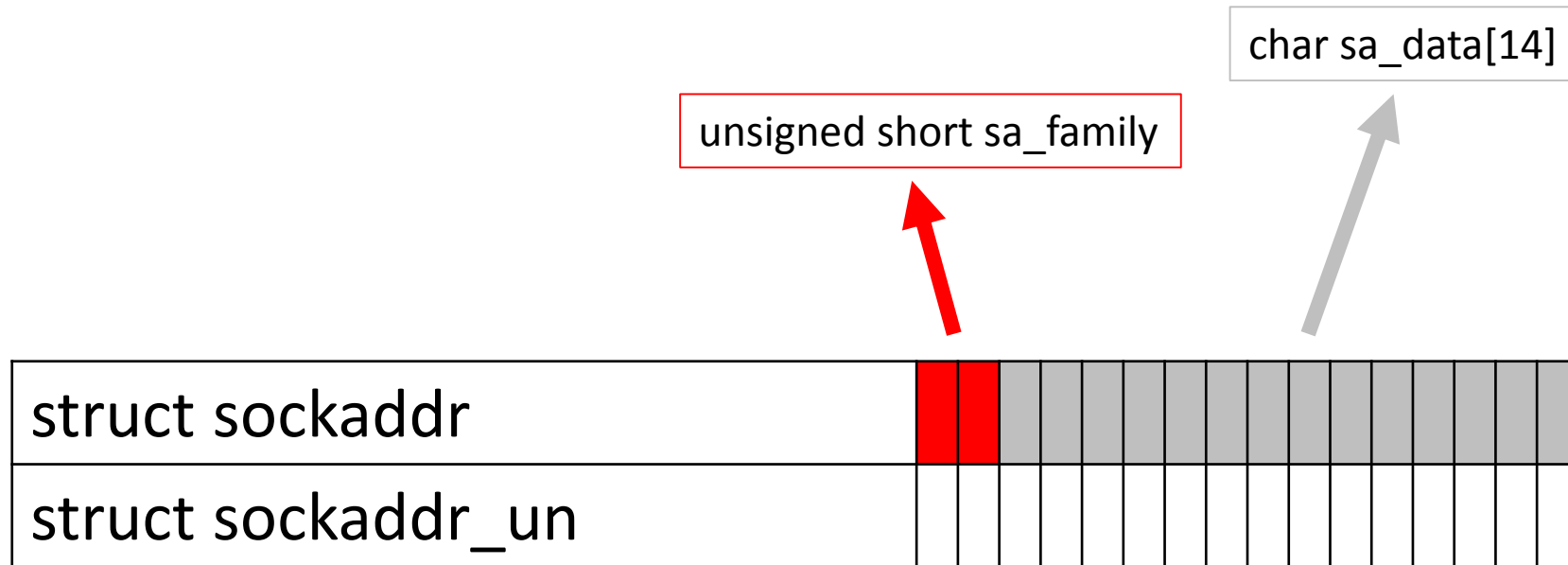
Заполнение структуры sockaddr_in6

1. ~~SocketAddr.sin_addr.s_addr = inet_addr("0.0.0.0");~~
2. `inet_pton(AF_INET, "0.0.0.0", &(SocketAddr.sin_addr));`
3. `inet_pton(AF_INET6, "2001:db8:8714:3a90::12", &(SocketAddr6.sin6_addr));`

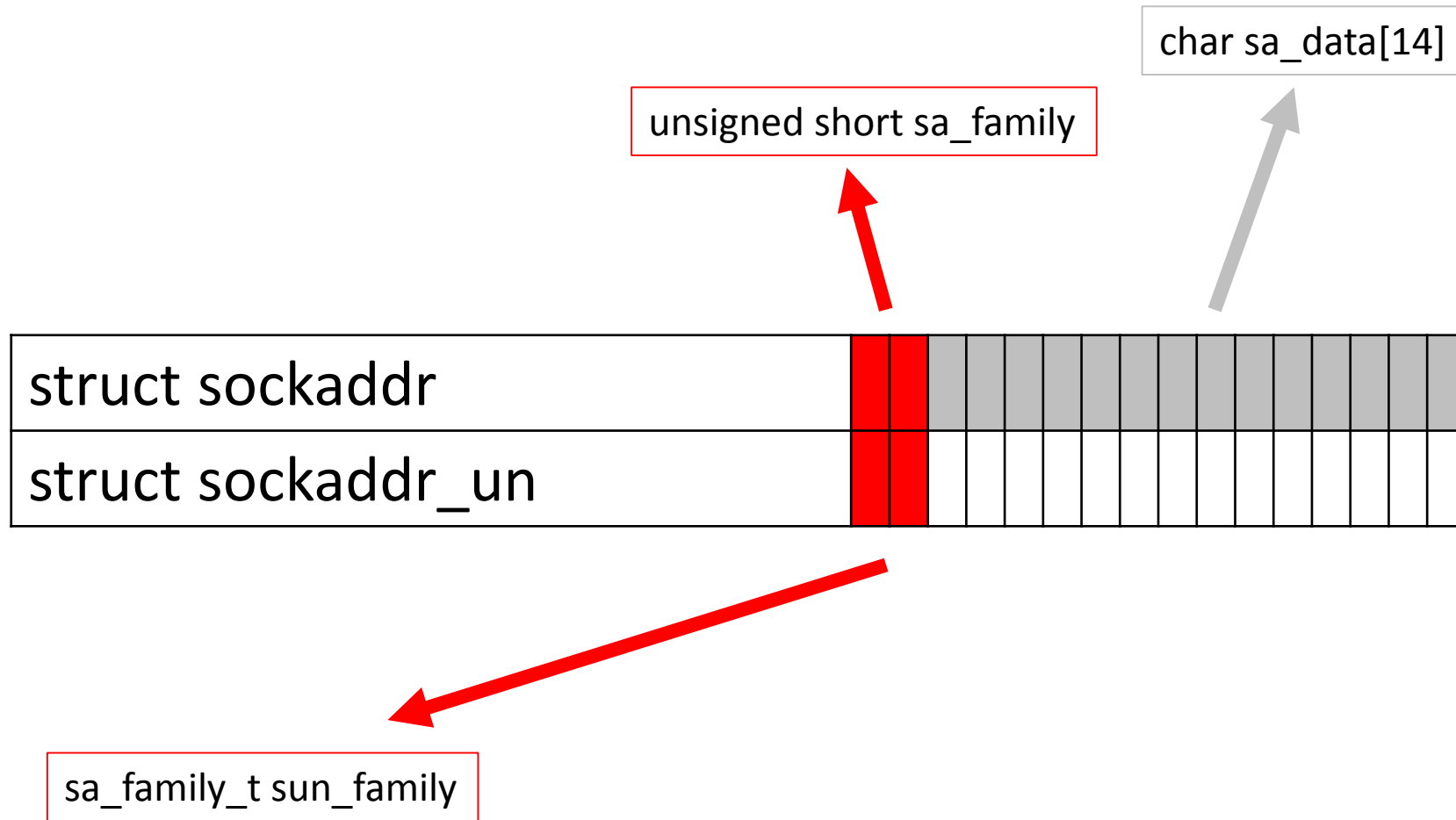
Заполнение структуры sockaddr_un

```
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);
```

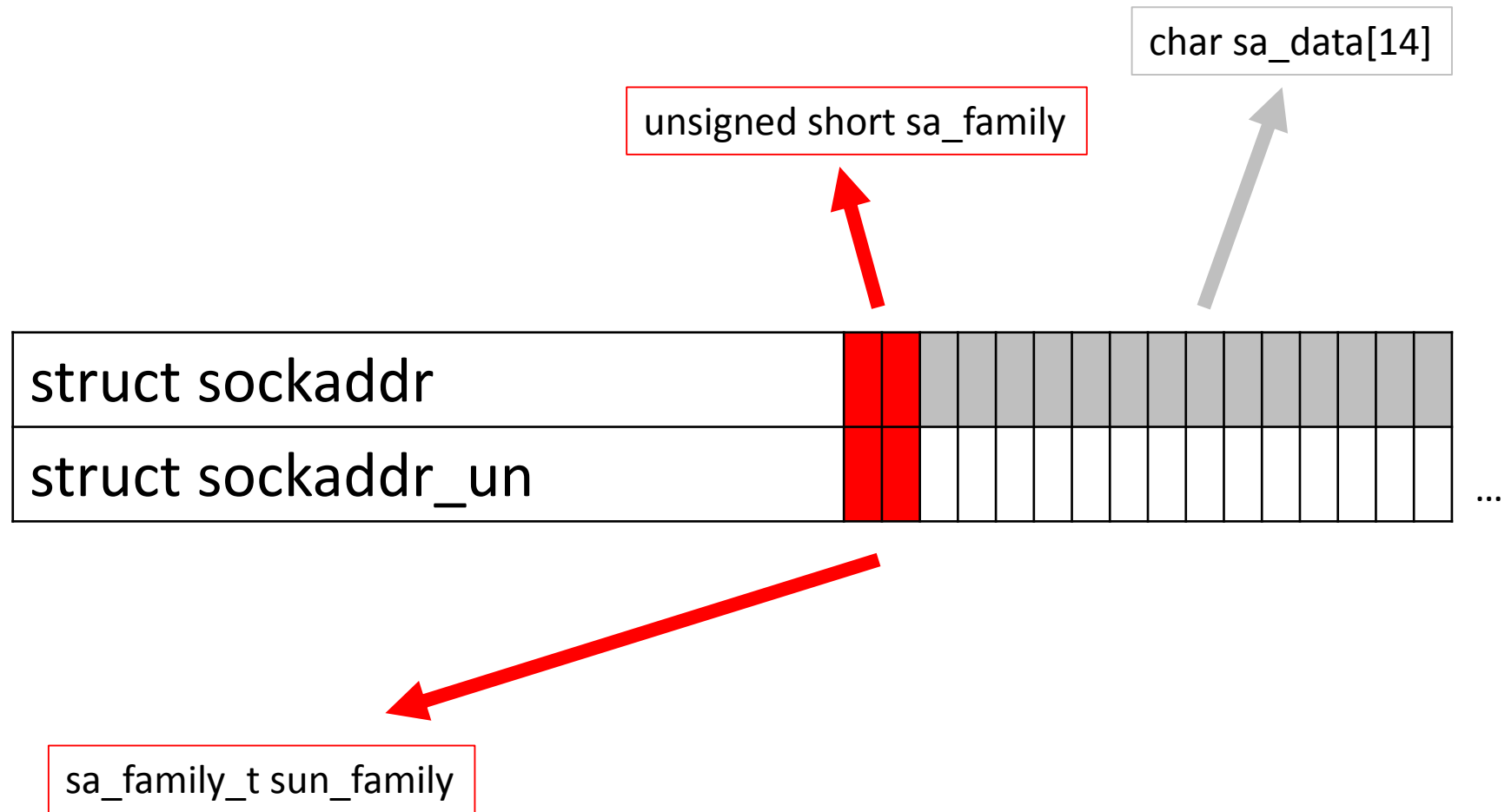
UDS



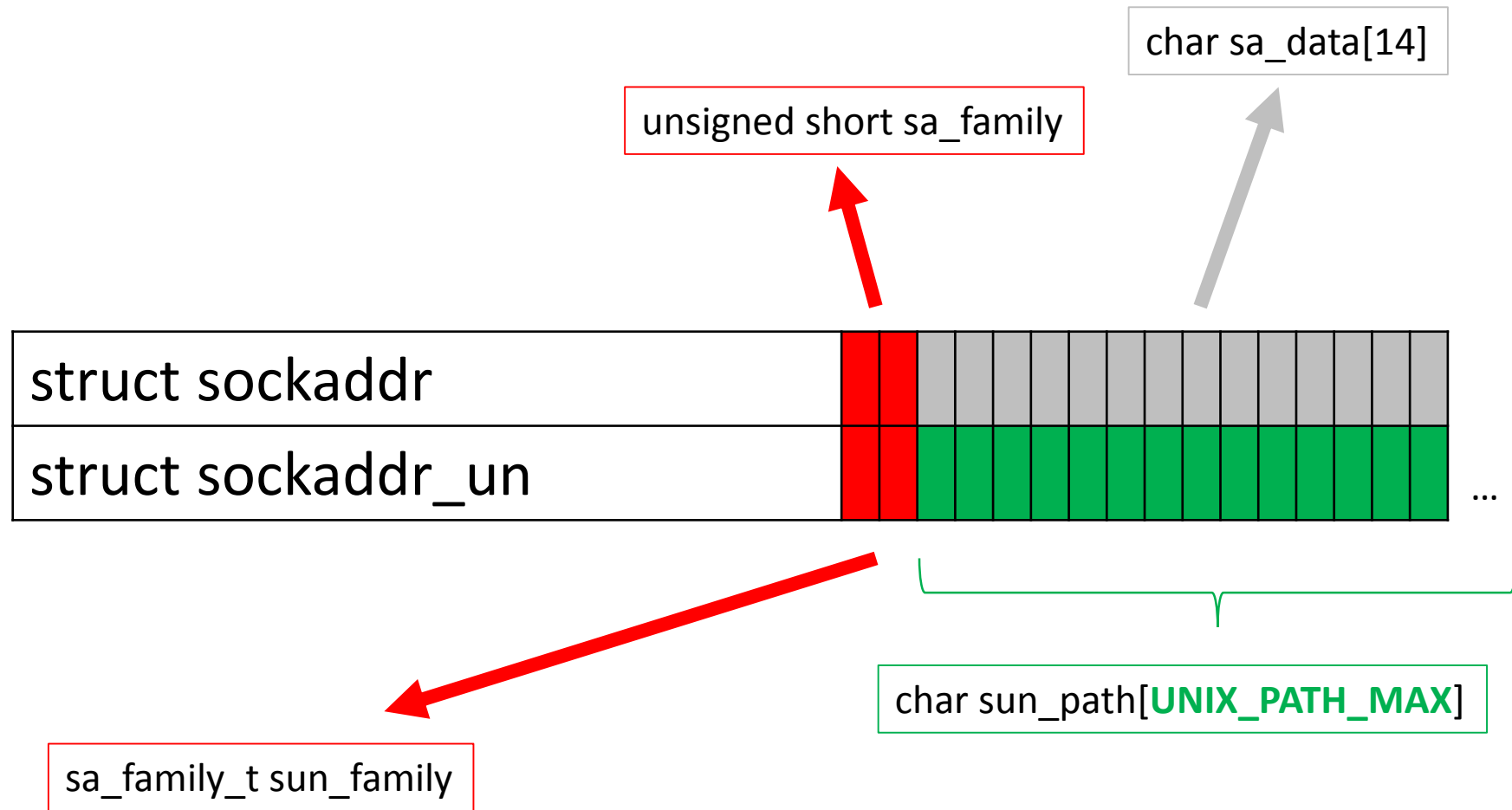
UDS



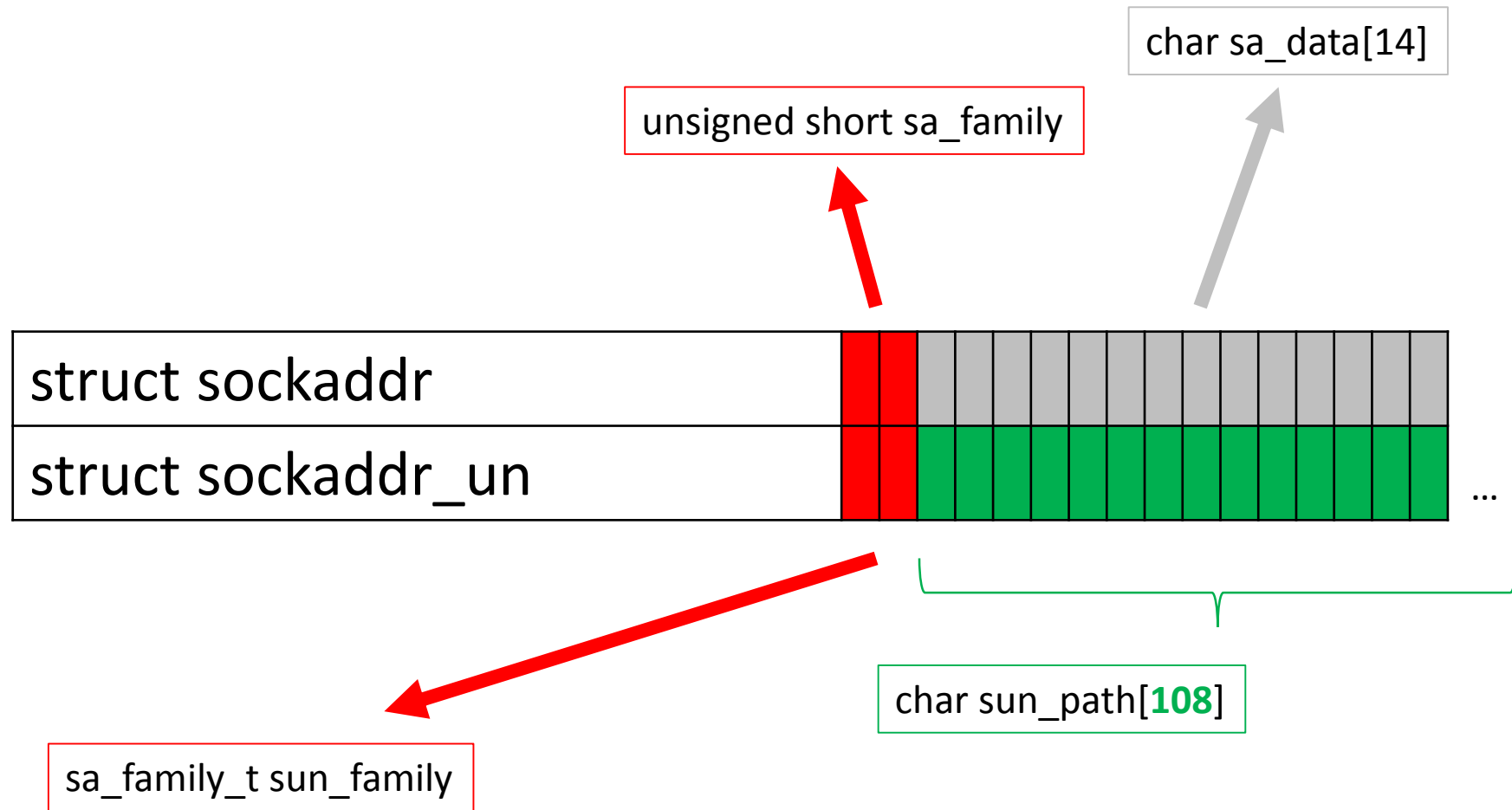
UDS



UDS



UDS



Сокеты Беркли

listener

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

Сокеты Беркли

slave socket

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

Сокеты Беркли

slave socket

```
1.  int SlaveSocket = accept(MasterSocket, 0, 0);  
  
2.  int SlaveSocket = accept(MasterSocket,  
3.                          struct sockaddr *addr,  
4.                          socklen_t *addrlen);
```

Сокеты Беркли

TCP-сервер

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

Сокеты Беркли

ТСР-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.      int SlaveSocket = accept(MasterSocket, 0, 0);
10.     // ...
11. }
```

Сокеты Беркли

ТСР-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

закрыть сокет?

1. `shutdown(ClientSocket, SHUT_RDWR);`
2. `shutdown(SlaveSocket, SHUT_RDWR);`
3. **SHUT_RDWR**
4. **SHUT_RD**
5. **SHUT_WR**
6. `close(ClientSocket);`
7. `close(SlaveSocket);`

Сокеты Беркли

Передавать данные?

1. `ssize_t read(int fd, void *buf, size_t count);`
2. `ssize_t write(int fd, const void *buf, size_t count);`

Сокеты Беркли

Передавать данные?

1. ~~`ssize_t read(int fd, void *buf, size_t count);`~~
2. ~~`ssize_t write(int fd, const void *buf, size_t count);`~~
3. `ssize_t recv(int s, void *buf, size_t len, int flags);`
4. `ssize_t send(int s, const void *buf, size_t len, int flags);`

Сокеты Беркли

Передавать данные?

1. ~~`ssize_t read(int fd, void *buf, size_t count);`~~
2. ~~`ssize_t write(int fd, const void *buf, size_t count);`~~
3. `ssize_t recv(int s, void *buf, size_t len, int flags);`
4. `ssize_t send(int s, const void *buf, size_t len, int flags);`

`MSG_NOSIGNAL`

```
signal(SIGPIPE, SIG_IGN);
```

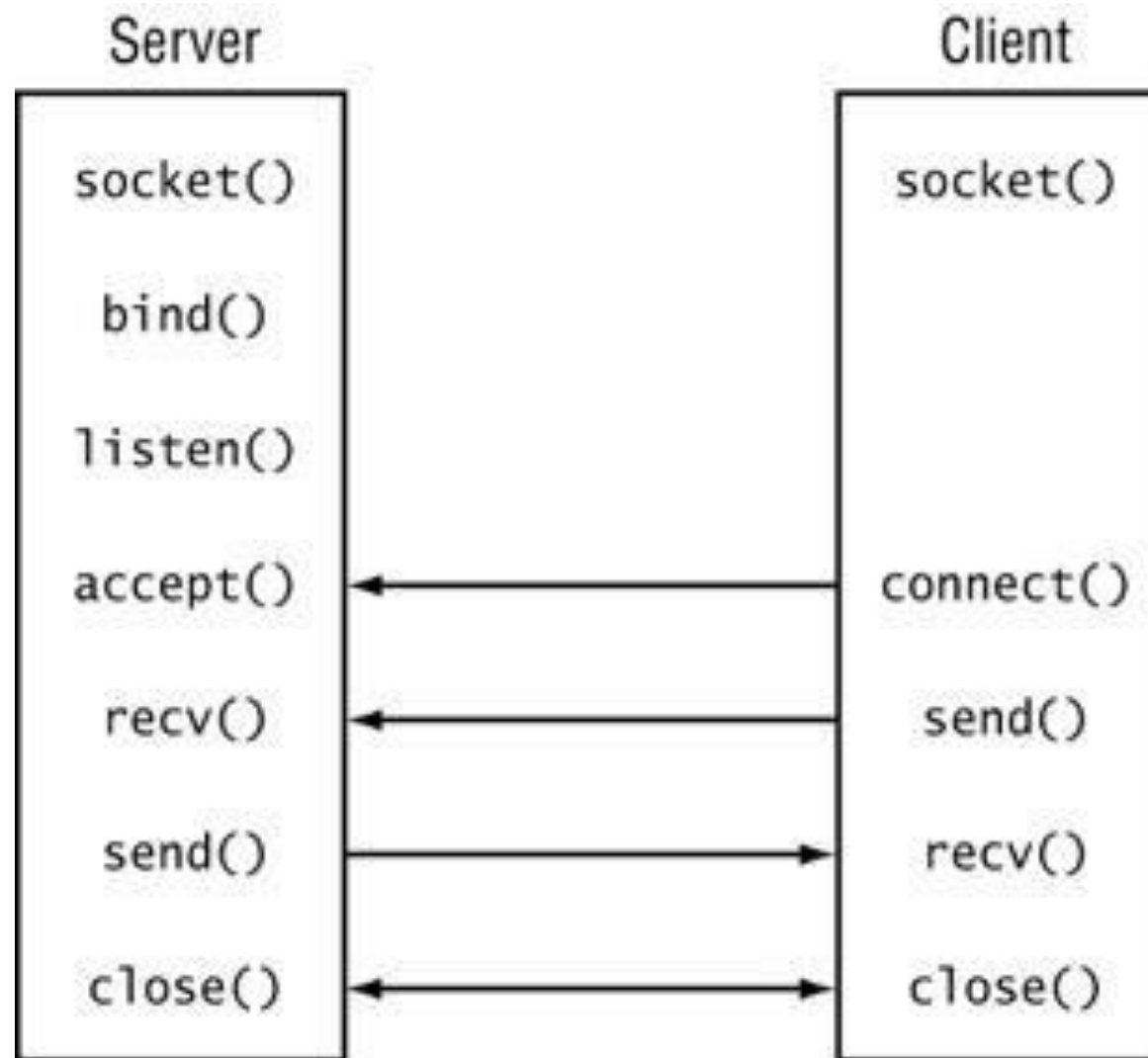
Сокеты Беркли

Передавать данные по UDP?

1. `ssize_t sendto(int s,
 const void *buf,
 size_t len,
 int flags,
 const struct sockaddr *to,
 socklen_t tolen);`
2. `ssize_t recvfrom(int s,
 void *buf,
 size_t len,
 int flags,
 struct sockaddr *from,
 socklen_t *fromlen);`

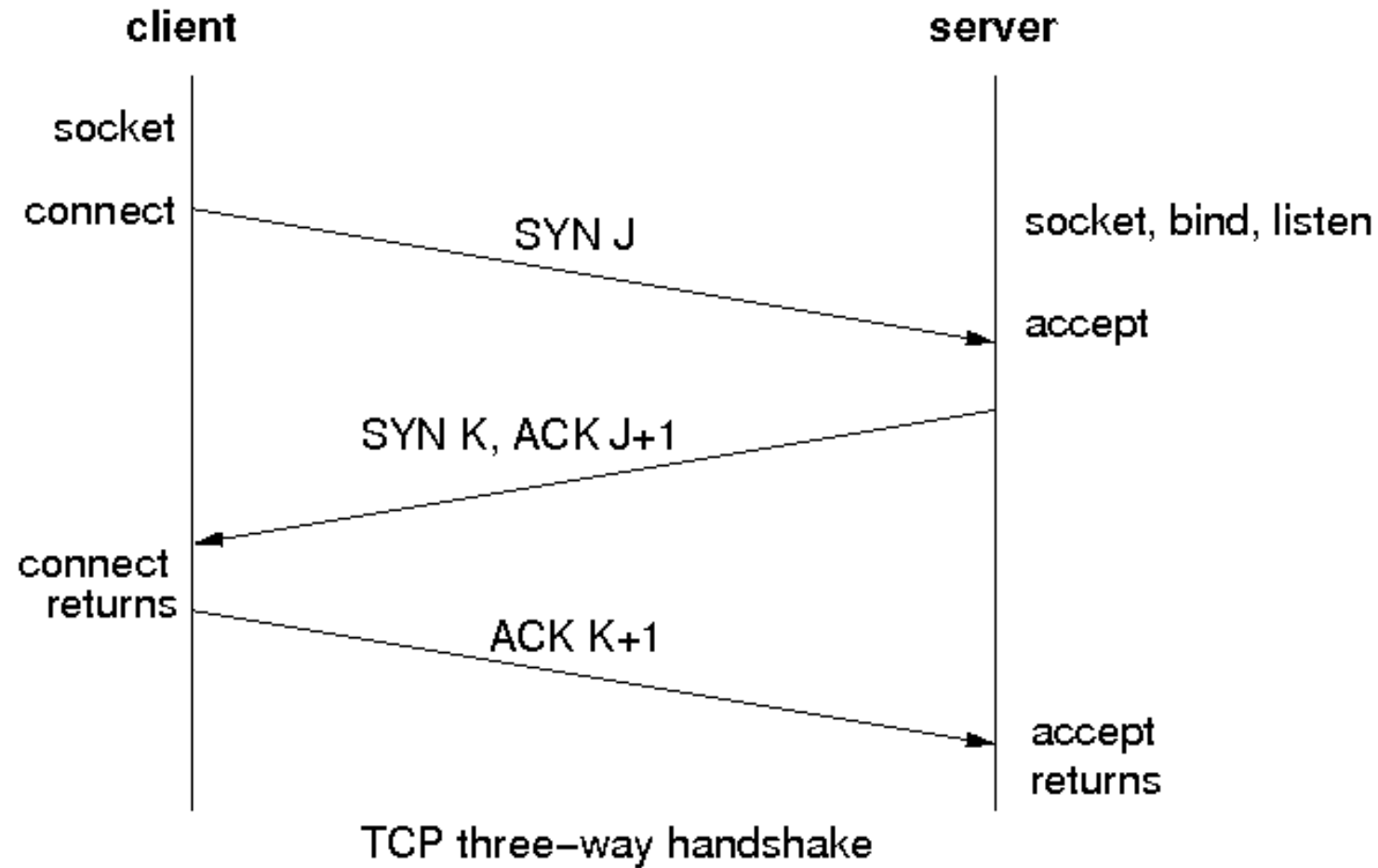
Сокеты Беркли

TCP



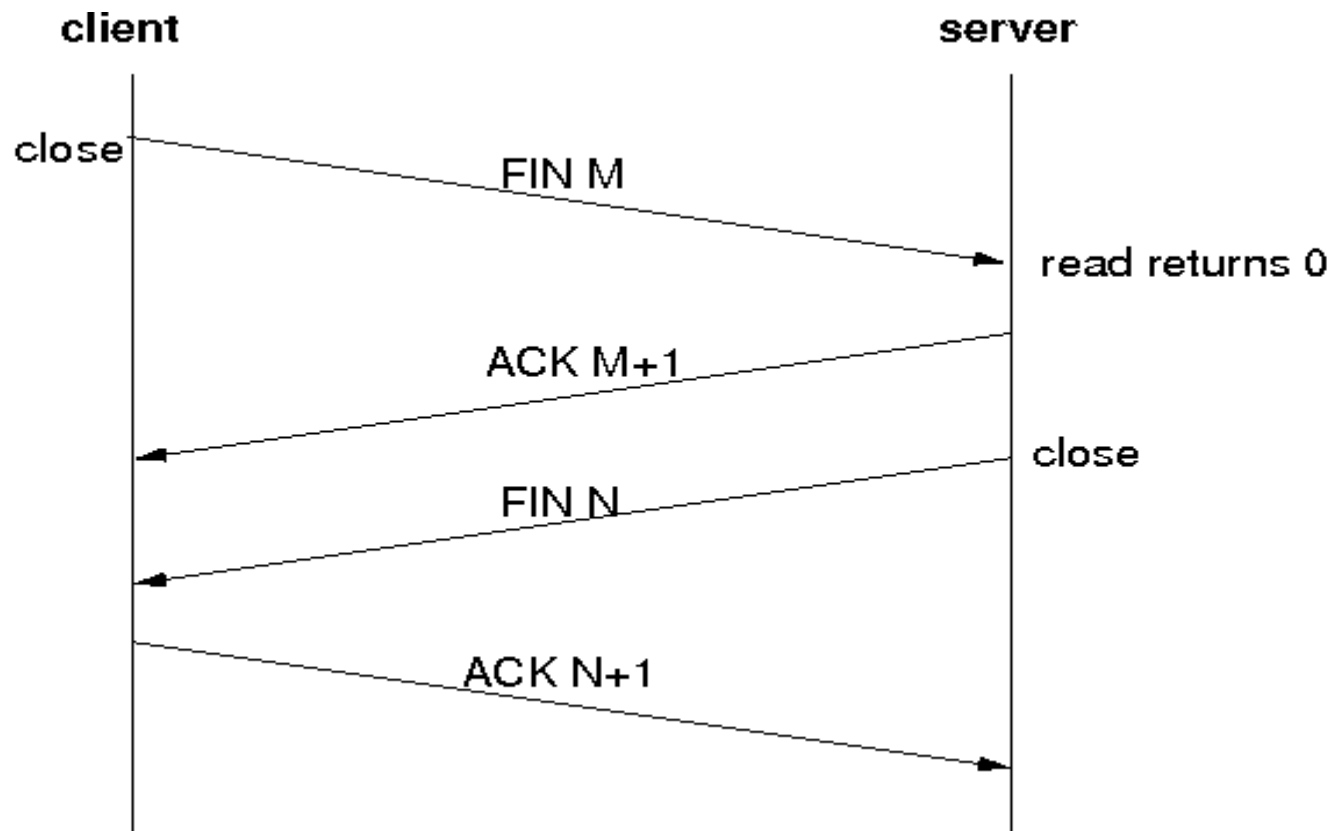
Сокеты Беркли

SYN-foot



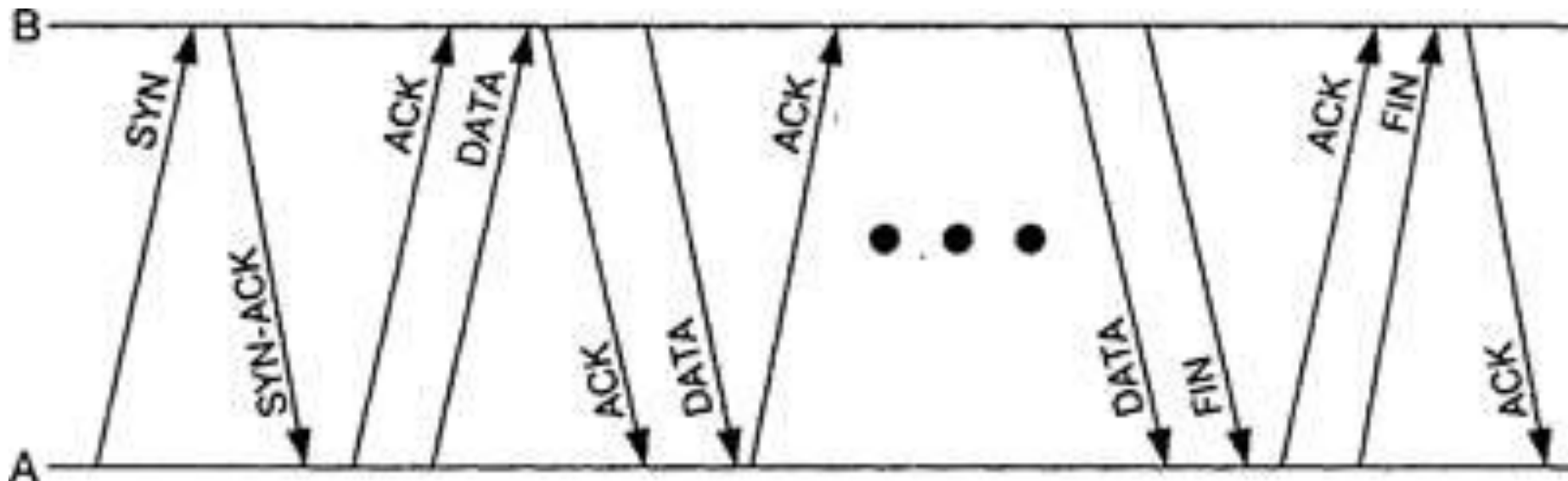
Сокеты Беркли

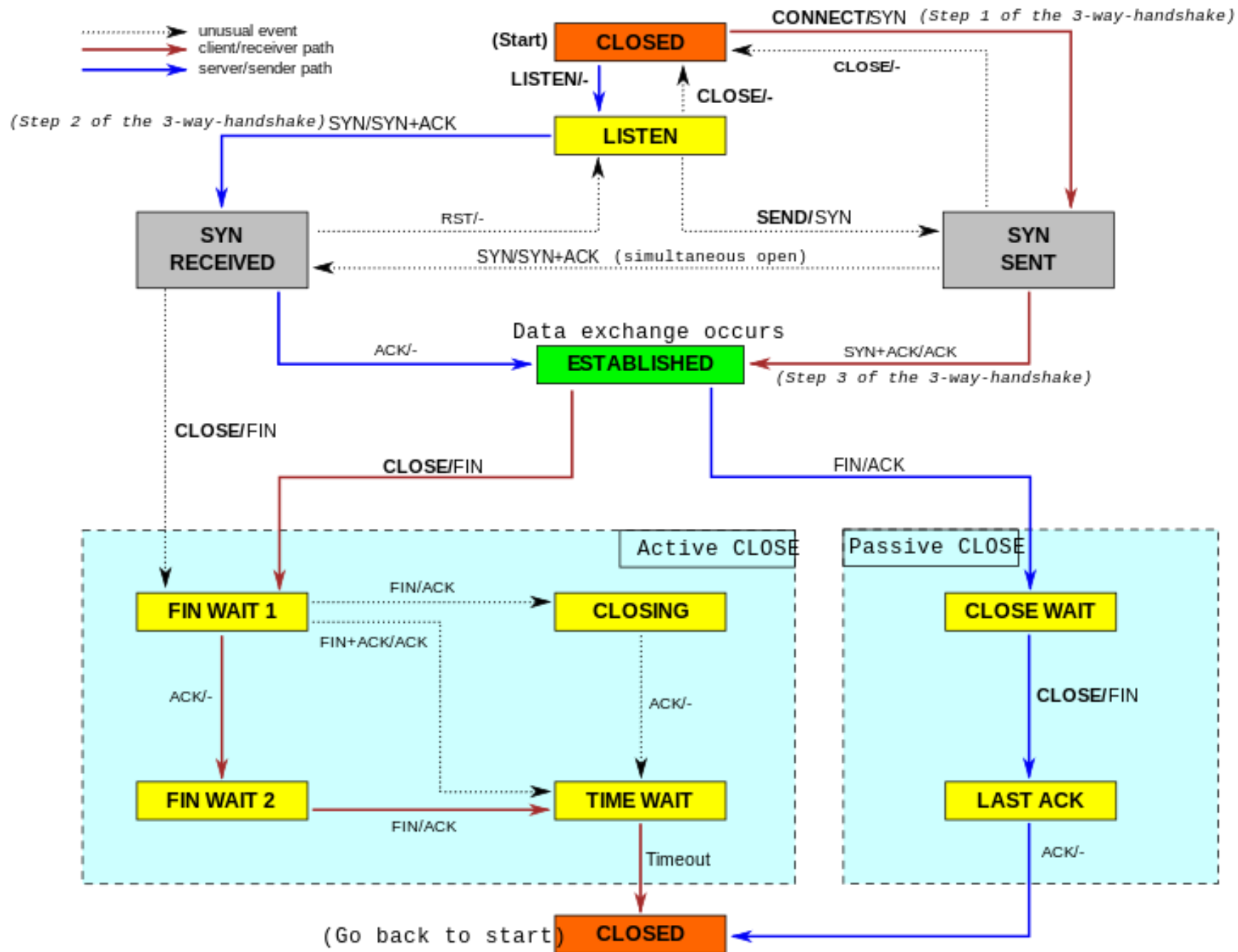
завершение соединения



Сокеты Беркли

жизнь соединения





Сокеты Беркли

неблокирующий сокет

```
1.  int set_nonblock(int fd) {
2.      int flags;
3.      #if defined(O_NONBLOCK)
4.          if (-1 == (flags = fcntl(fd, F_GETFL, 0)))
5.              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

fd	fd	fd	fd	fd	fd	fd	fd	fd	fd
----	----	----	----	----	----	----	----	----	----

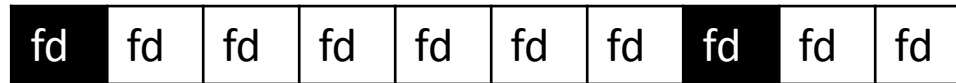
Мультиплексирование

fd	fd	fd	fd	fd	fd	fd	fd	fd	fd
----	----	----	----	----	----	----	----	----	----

Мультиплексирование

fd	fd	fd	fd	fd	fd	fd	fd	fd	fd
----	----	----	----	----	----	----	----	----	----

Мультиплексирование



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 = slaves.begin(); Iter != slaves.end(); ++Iter)
5.      FD_SET(*Iter, &Set);
```

Мультиплексирование

Работа с select

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

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

Мультиплексирование

Работа с select

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

6.      int Max = std::max(MasterSocket, *std::max_element(slaves.begin(), slaves.end()));
7.      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 = slaves.begin(); Iter != slaves.end(); ++Iter)
5.      FD_SET(*Iter, &Set);

6.  while (true) {
7.      int Max = std::max(MasterSocket, *std::max_element(slaves.begin(), slaves.end()));
8.      select(Max+1, &Set, NULL, NULL, NULL);

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

Мультиплексирование

Работа с select

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

6.  while (true) {
7.      int Max = std::max(MasterSocket, *std::max_element(slaves.begin(), slaves.end()));
8.      select(Max+1, &Set, NULL, NULL, NULL);

9.      if (FD_ISSET(MasterSocket, &Set)) {
10.         int SlaveSocket = accept(MasterSocket, 0, 0);
11.         slaves.insert(SlaveSocket);
12.     }
13.     for (auto Iter = slaves.begin(); Iter != slaves.end(); Iter++)
14.         if (FD_ISSET(*Iter, &Set)) { /* ... */ }
15. }
```

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

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

Мультиплексирование

Работа с poll

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

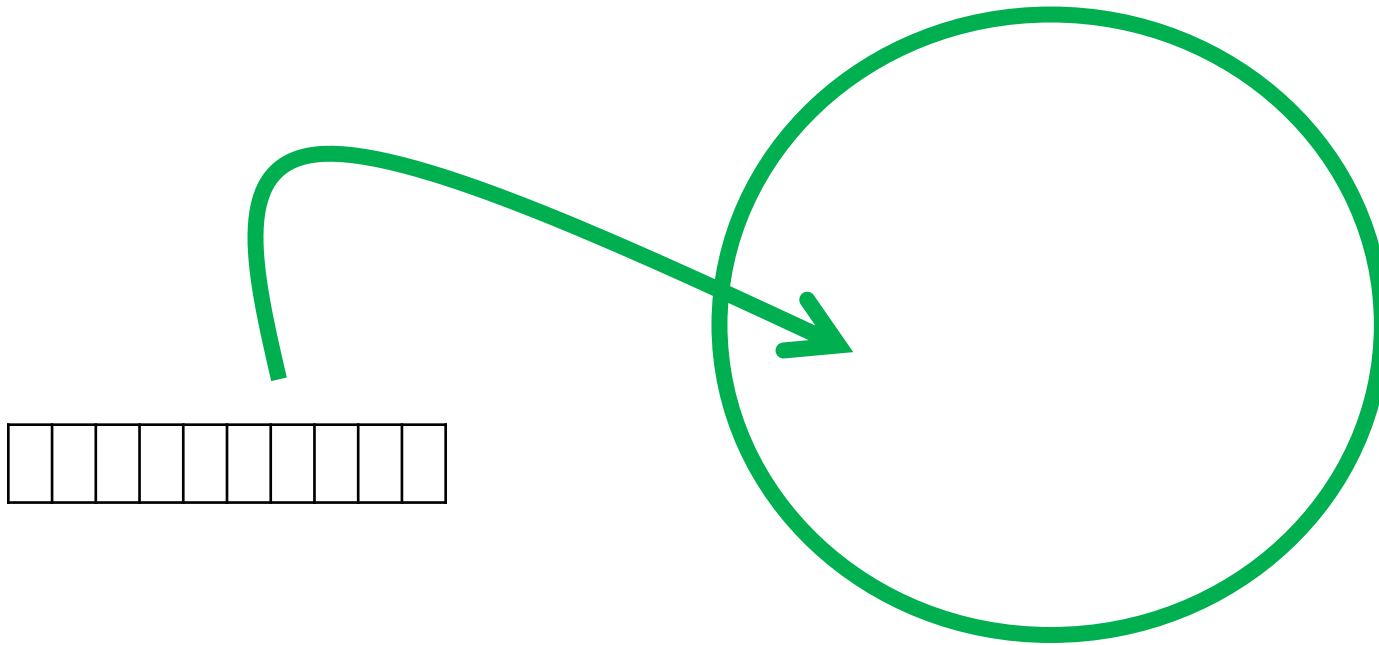
4.  unsigned Index = 1;
5.  for (auto Iter = SlaveSockets.begin(); Iter != SlaveSockets.end(); Iter++) {
6.      Set[Index].fd = *Iter;
7.      Set[Index].events = POLLIN;
8.      Index++;
9.  }
10. unsigned SetSize = 1 + SlaveSockets.size();

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

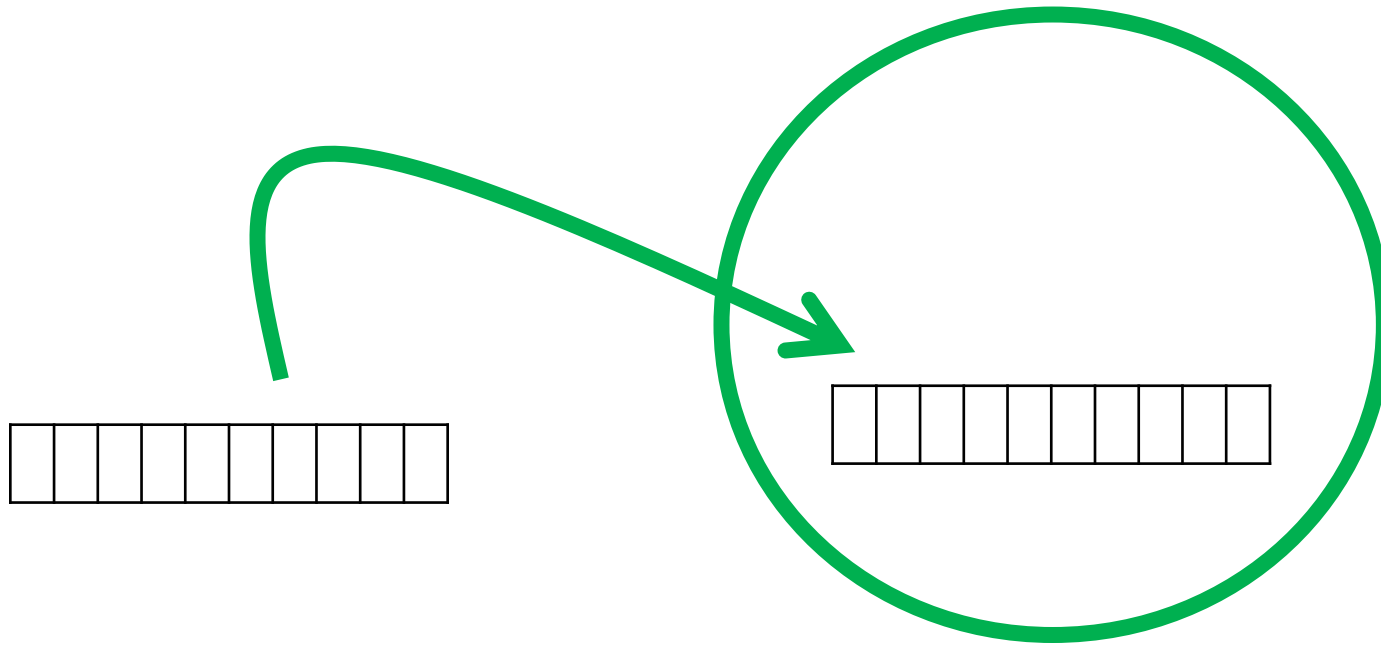
12. for (unsigned i = 0; i < SetSize; i++) {
13.     if (Set[i].events & POLLIN) {
14.         /* ... */
15.     }
16. }
```

C10k problem

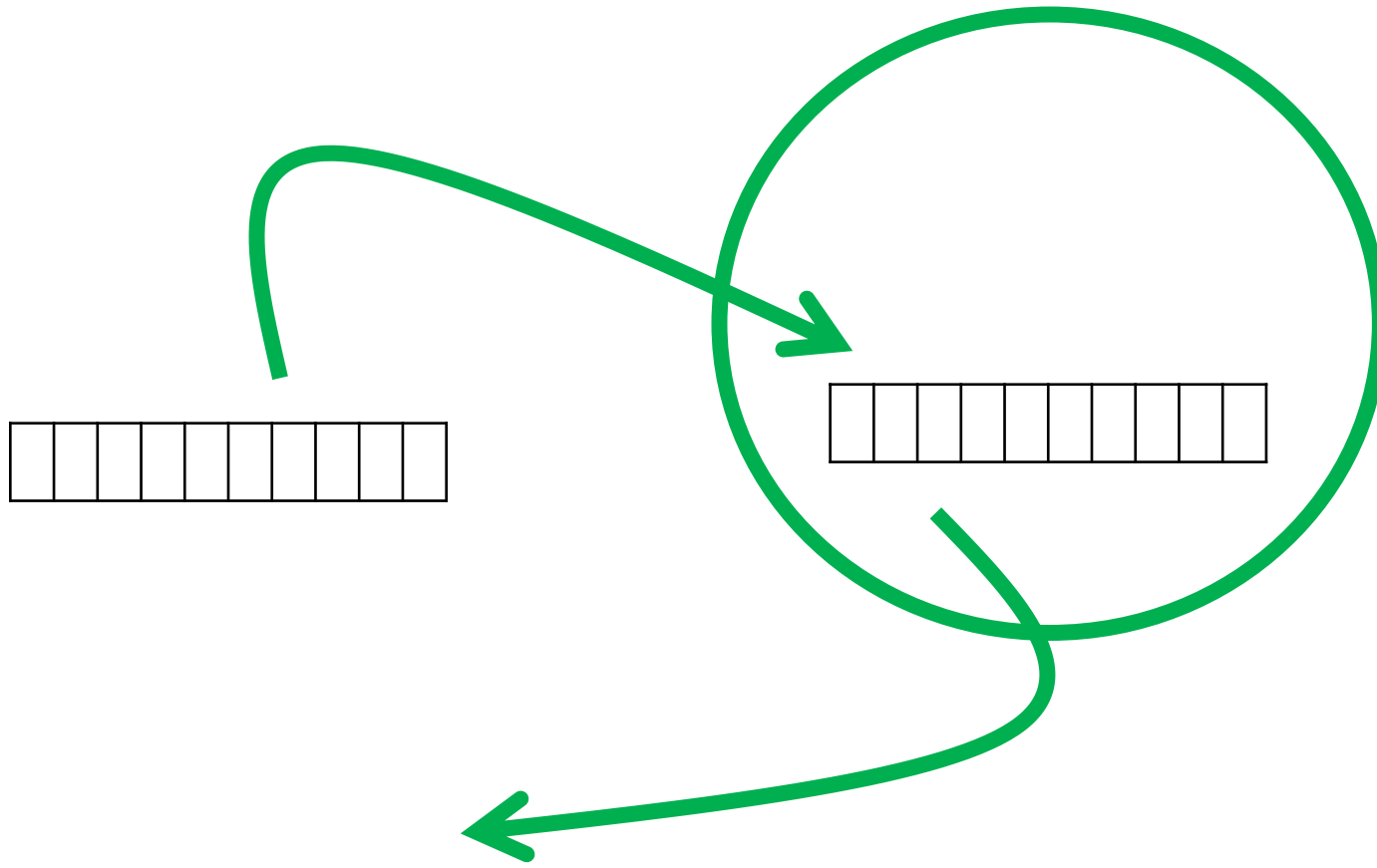
C10k problem



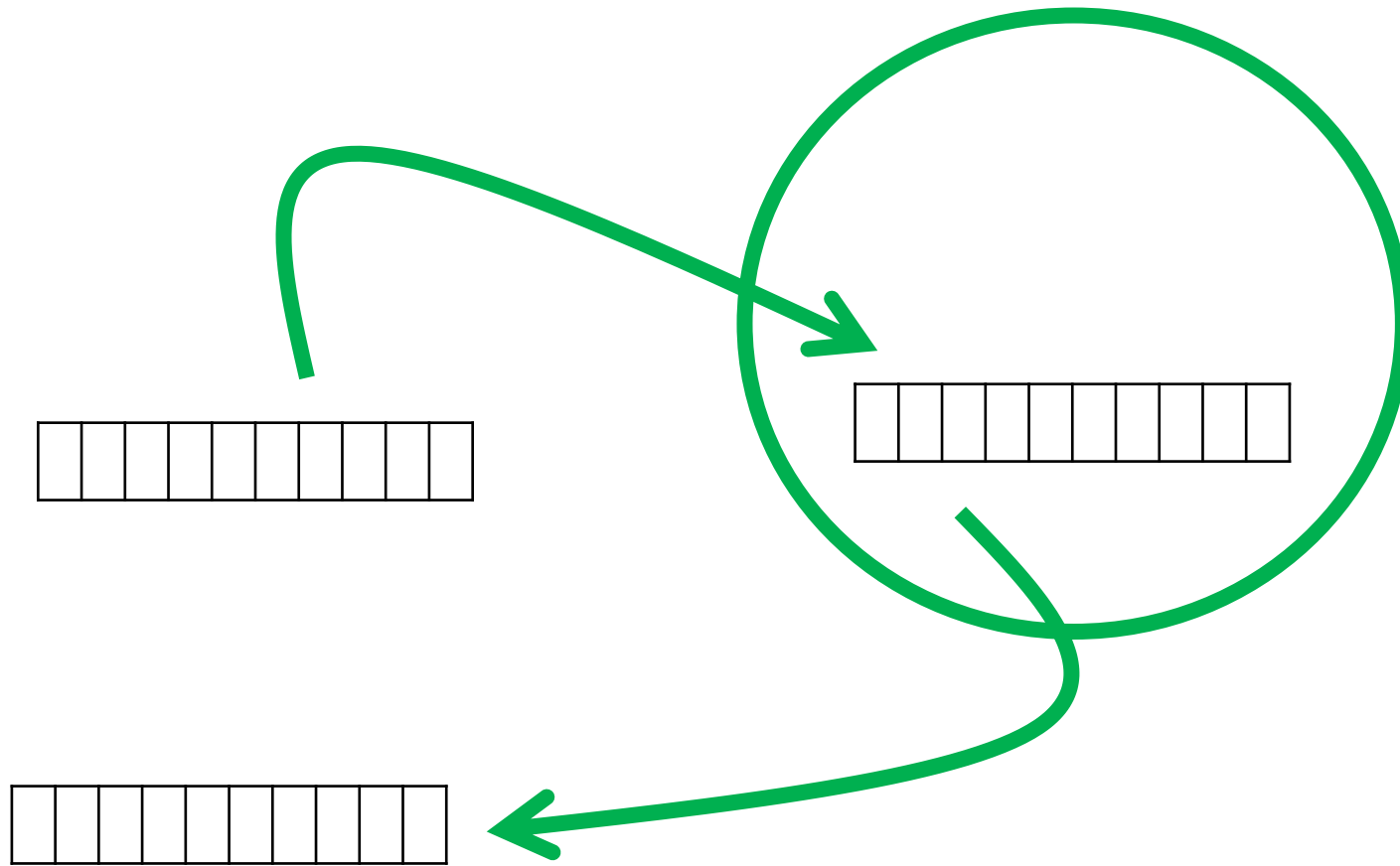
C10k problem



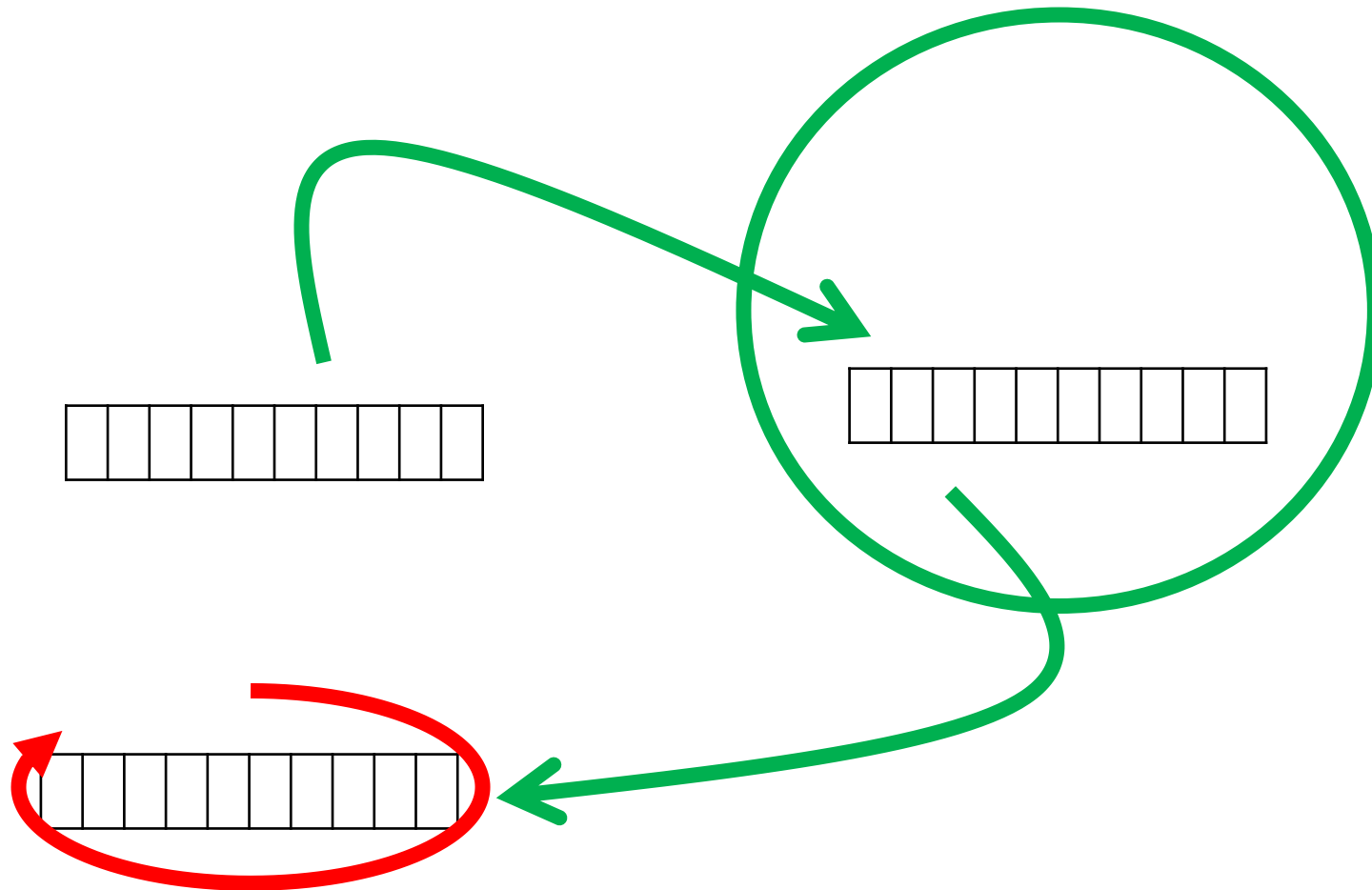
C10k problem



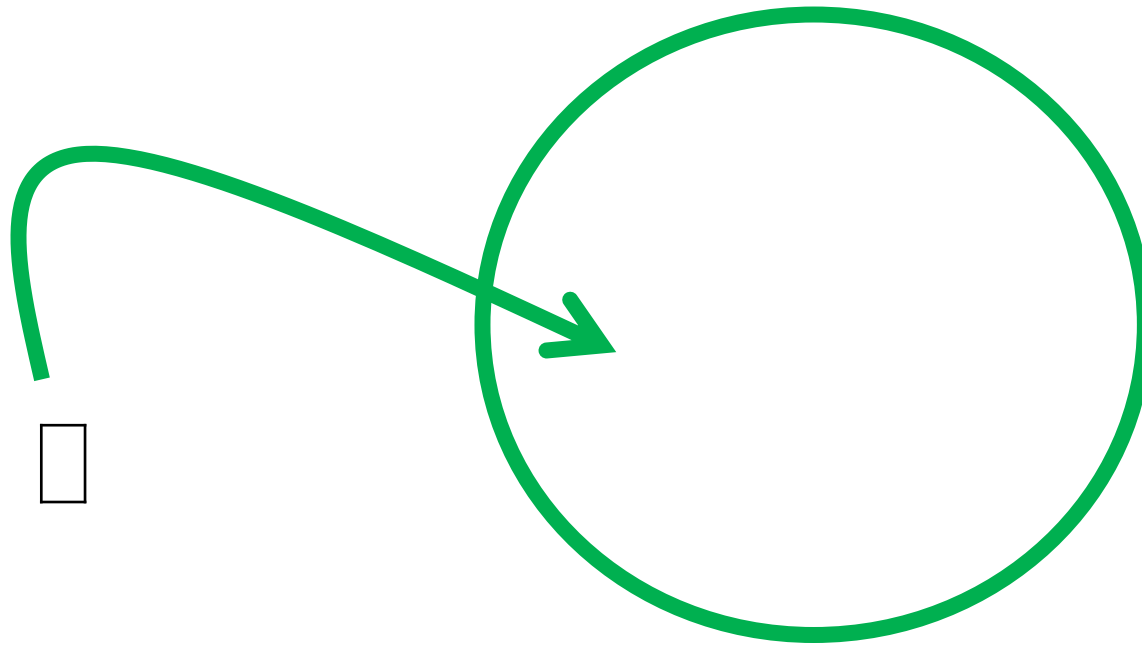
C10k problem



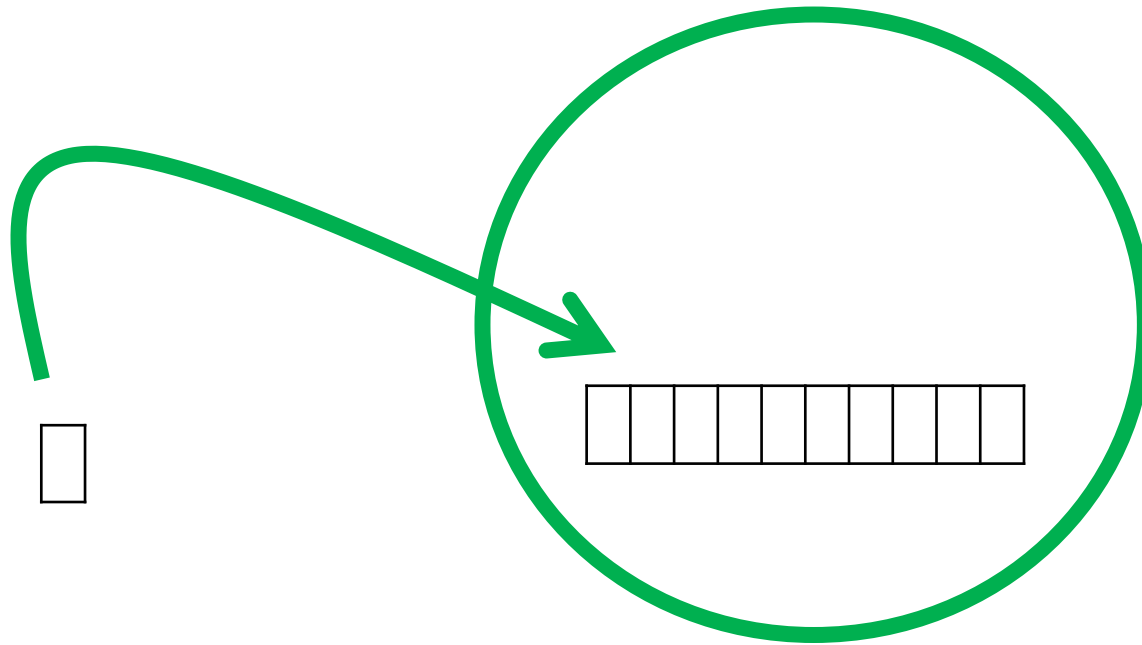
C10k problem



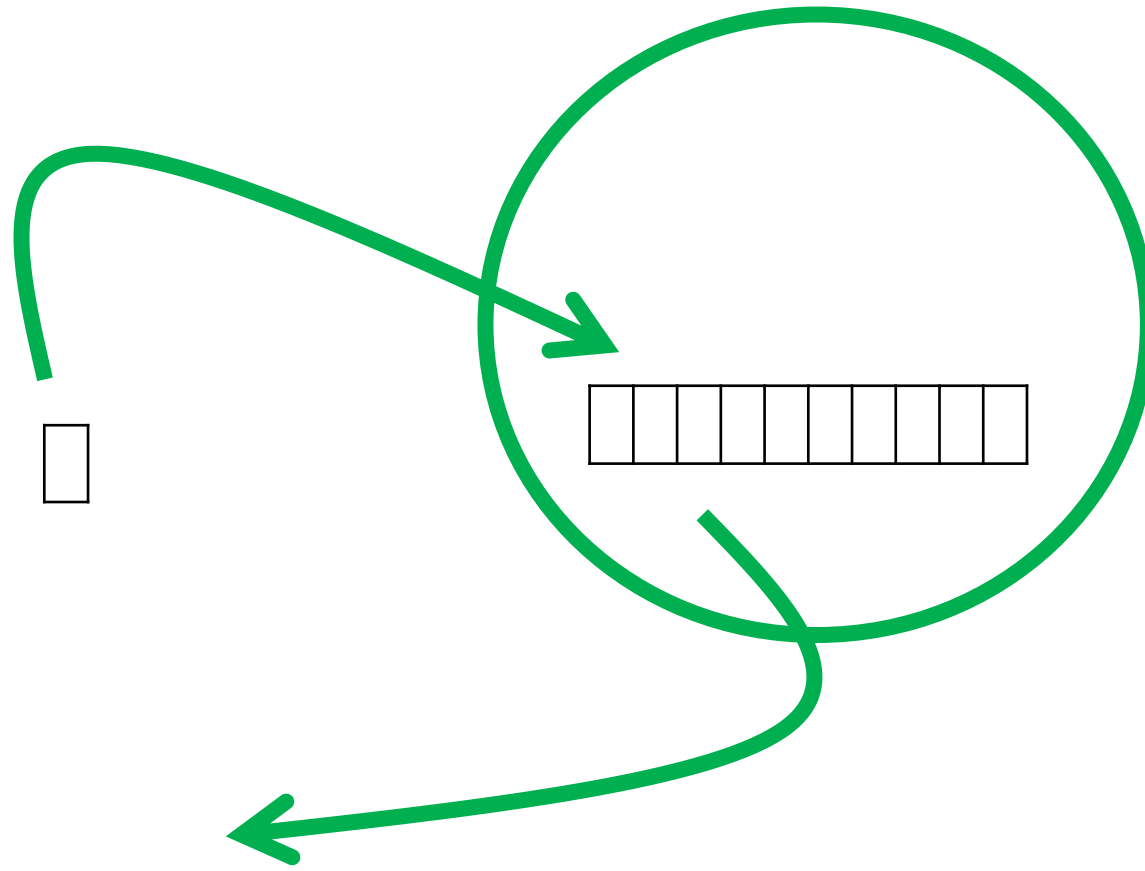
C10k problem



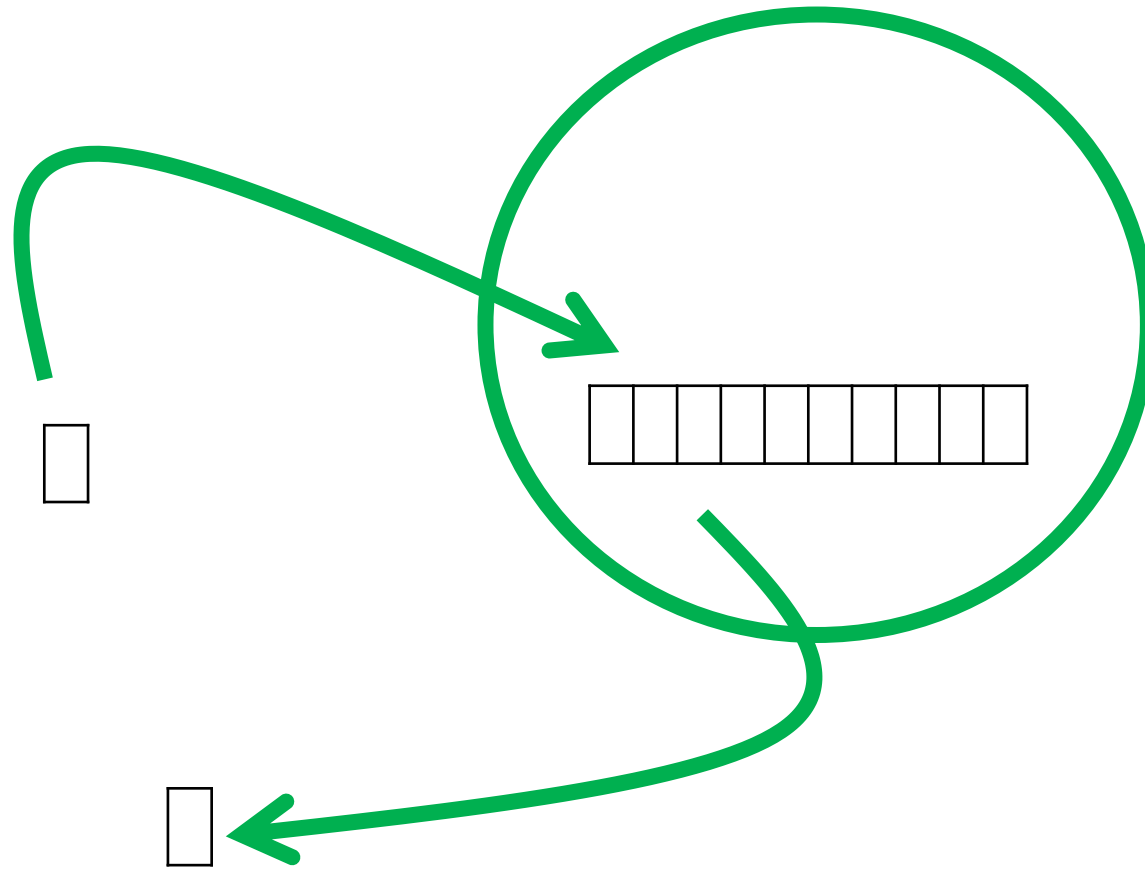
C10k problem



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

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

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

11. }
```

Мультиплексирование

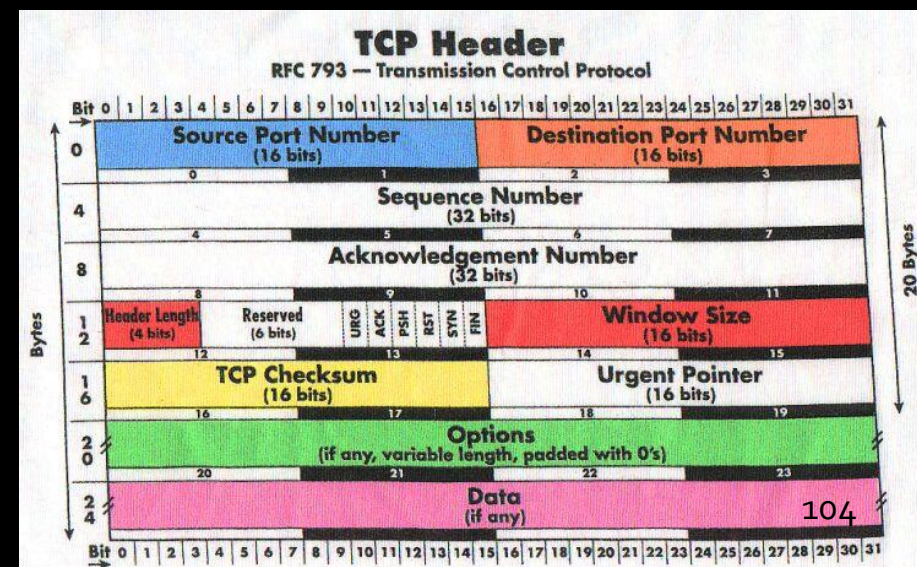
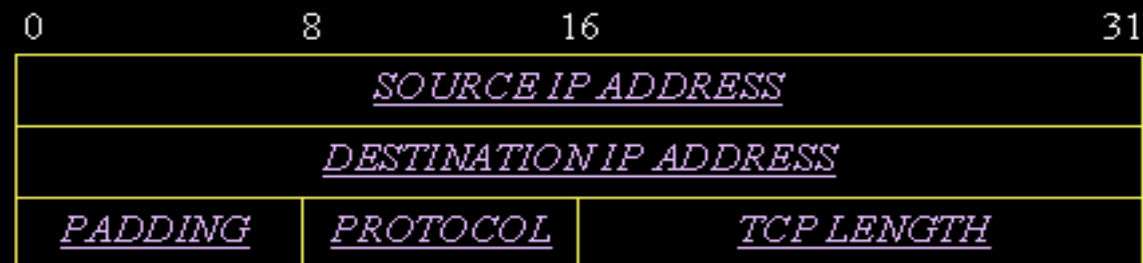
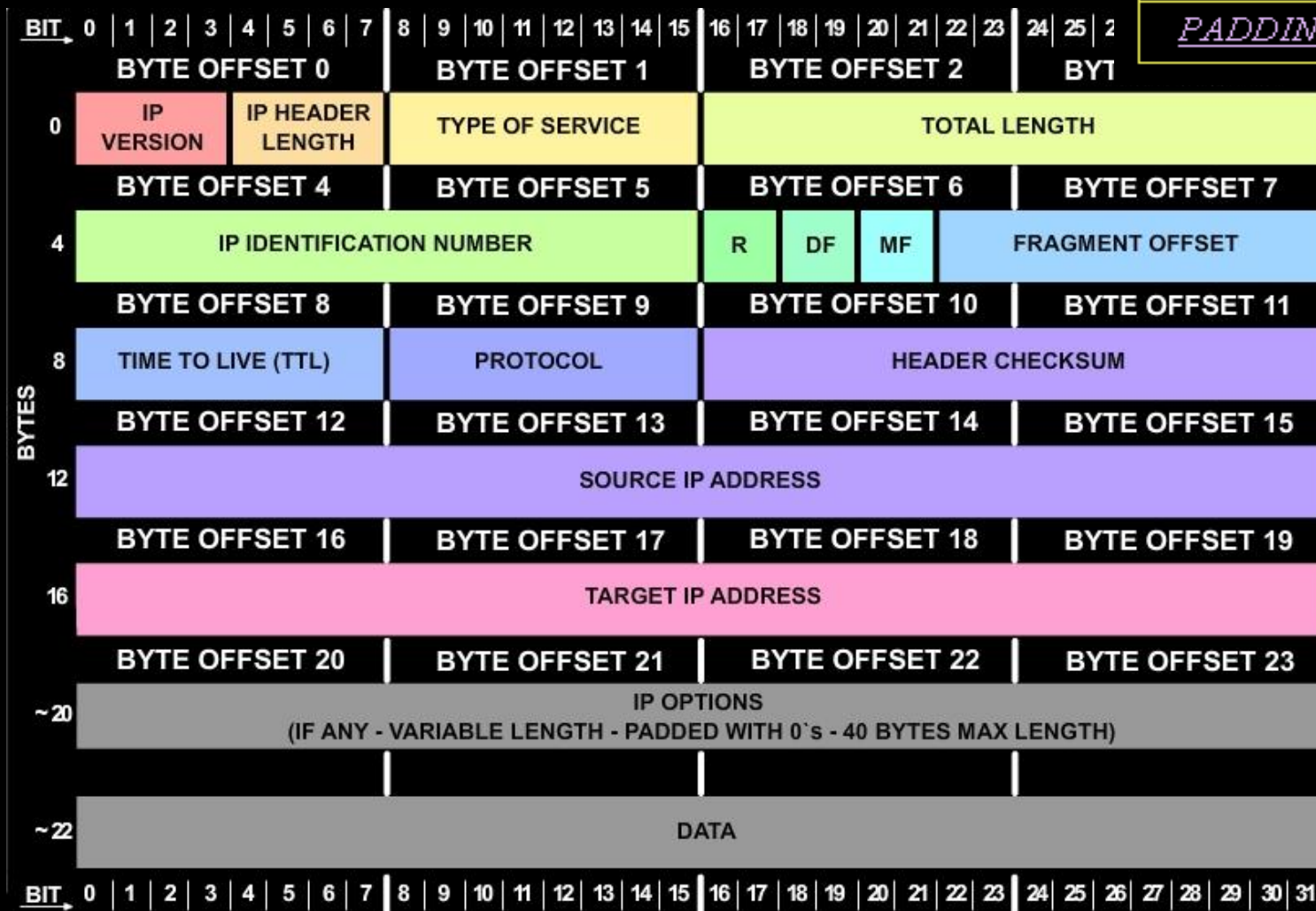
Работа с **kqueue**

```
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.      bzero(&KEvent, sizeof(KEvent));
8.      kevent(KQueue, NULL, 0, &KEvent, 1, NULL);
9.      if (KEvent.filter == EVFILT_READ)
10.     { /* ... */ }
11.     if (KEvent.ident == MasterSocket) {
12.         int SlaveSocket = accept(MasterSocket, 0, 0);
13.         bzero(&KEvent, sizeof(KEvent));
14.         EV_SET(&KEvent, SlaveSocket, EVFILT_READ, EV_ADD, 0, 0, 0);
15.         kevent(KQueue, &KEvent, 1, NULL, 0, NULL);
16.     } else
17.     { /* ... */ }
18. }
```

Raw-sockets

TCP Pseudo-Header:



Raw-сокеты

raw-socket

```
1.  int RAWSocket = socket(AF_INET, SOCK_RAW, IPPROTO_RAW);  
2.  int RAWSocket = socket(AF_INET, SOCK_RAW, IPPROTO_TCP);  
  
3.  int tmp = 1;  
4.  setsockopt(sock, 0, IP_HDRINCL, &tmp, sizeof(tmp));  
  
5.  int RAWSocket = socket(PF_PACKET, SOCK_RAW, <protocol>);
```

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

Литература

У. Стивенс. UNIX. Разработка сетевых приложений.

W. Richard Stevens. UNIX Network Programming

