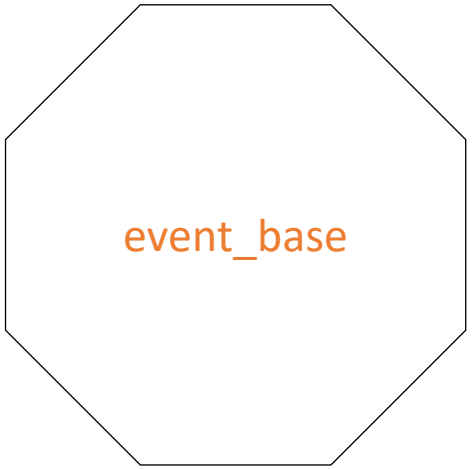
Курс «Проектирование больших систем на языке C++»

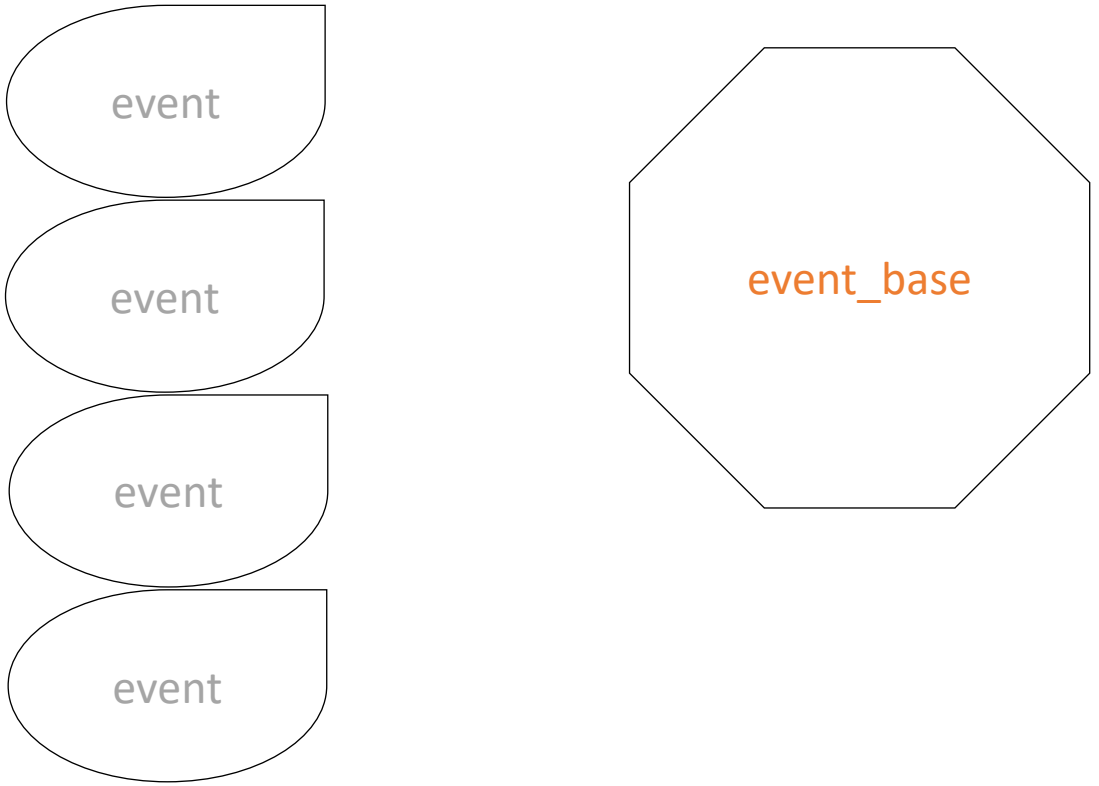
Лекция

Библиотеки сетевого взаимодействия

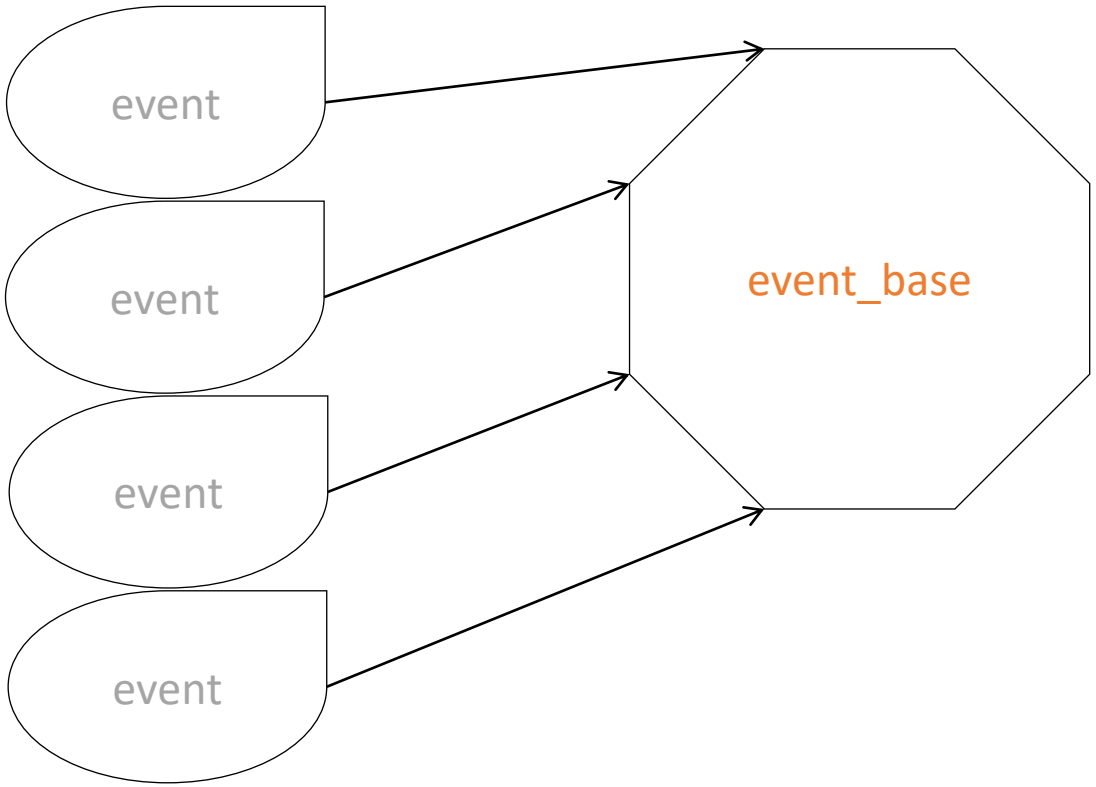
libevent



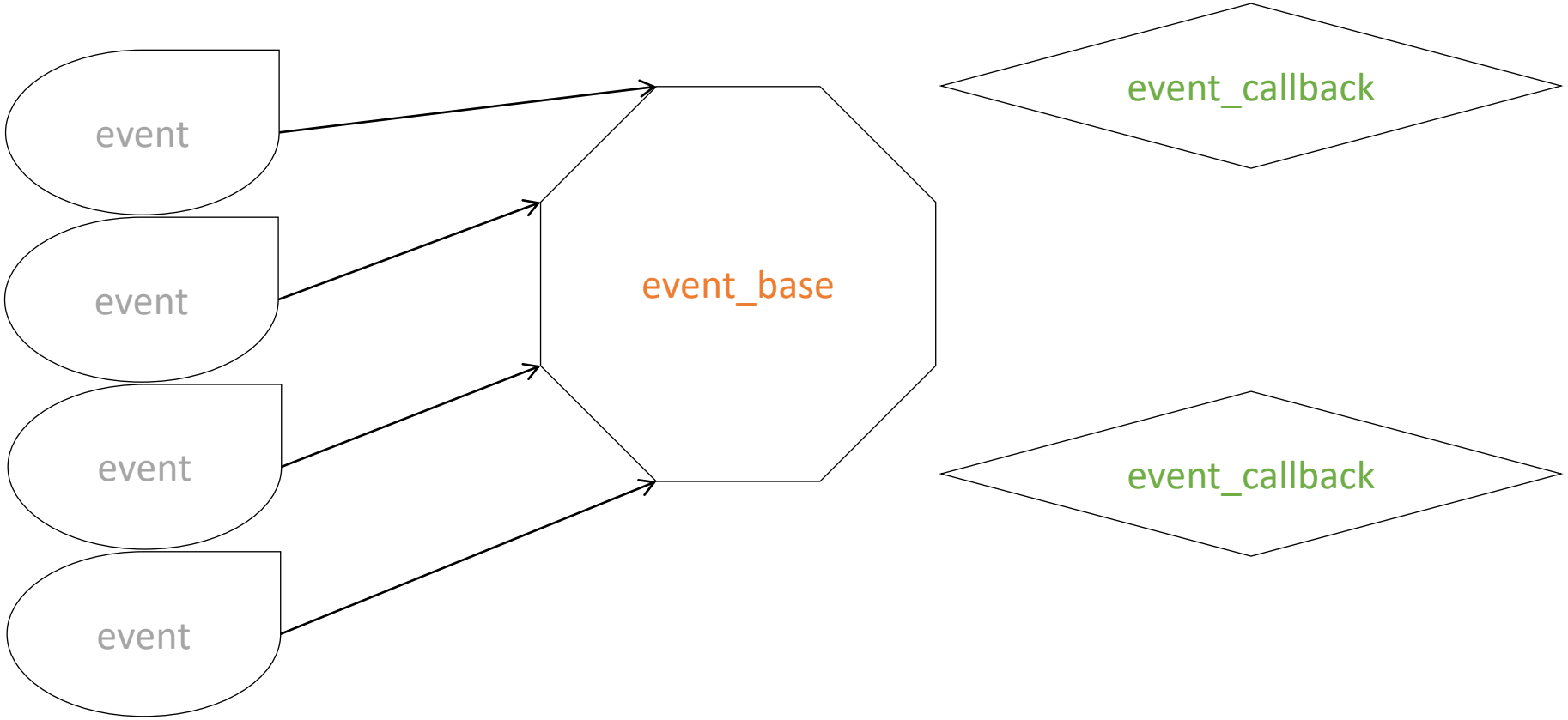
libevent



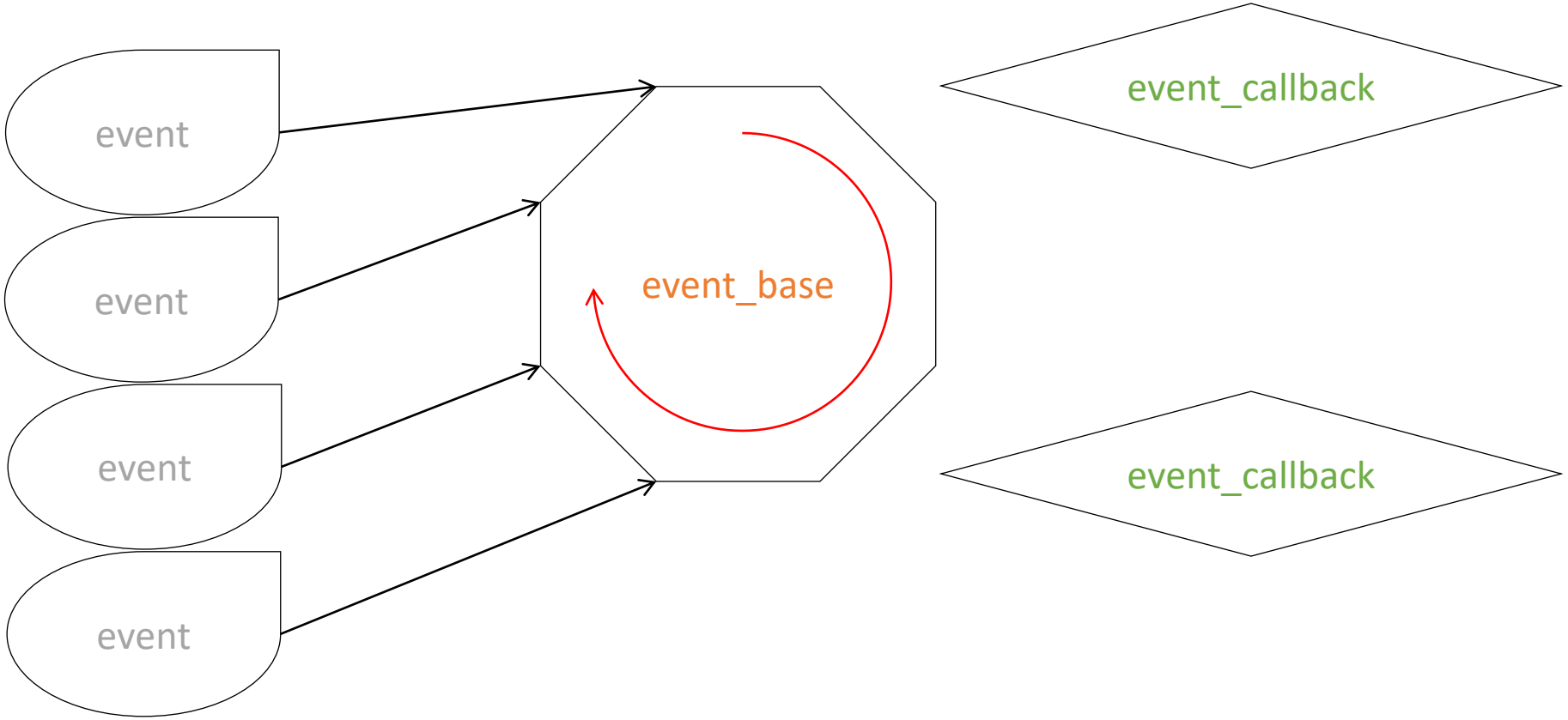
libevent



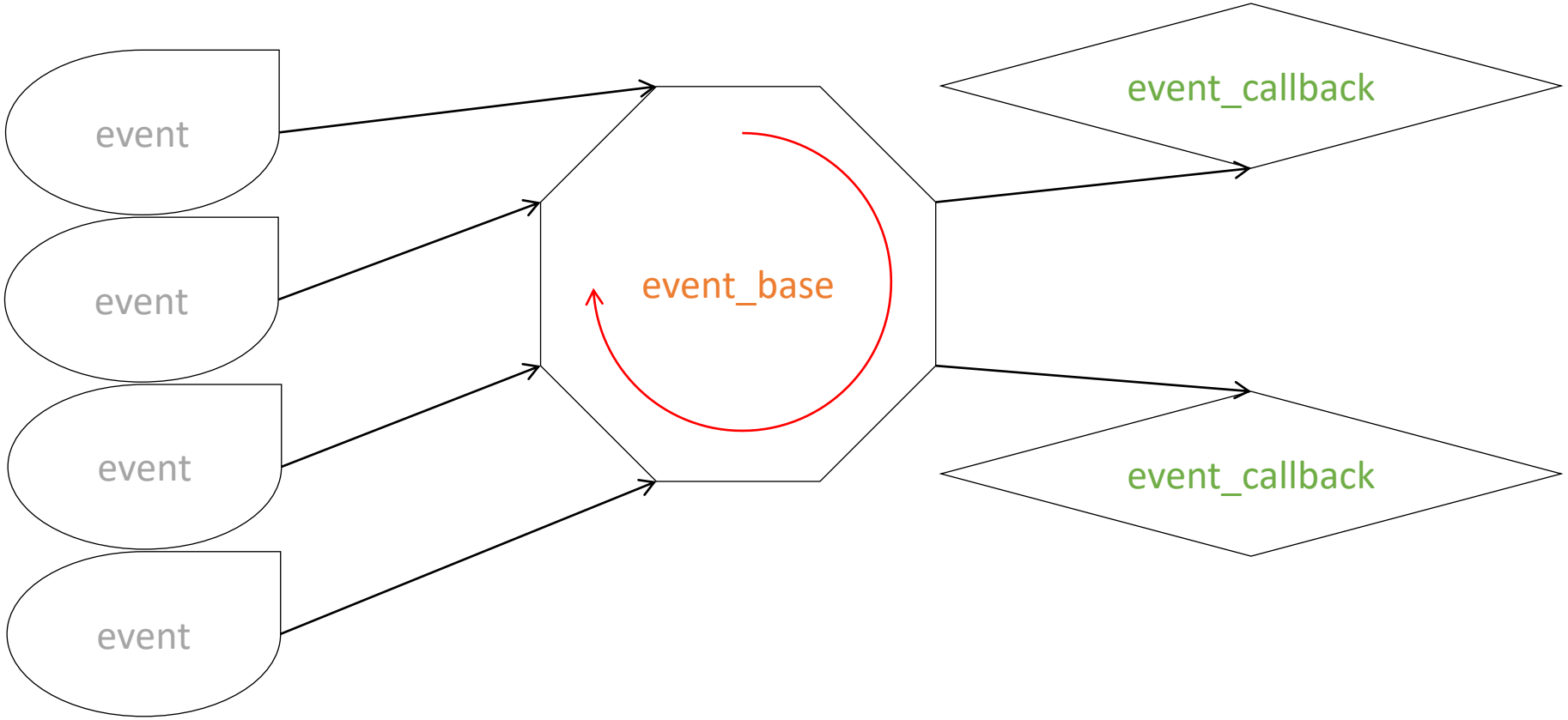
libevent


event
event
event
event
event_base
event_callback
event_callback

libevent


event
event
event
event
event_base
event_callback
event_callback

libevent


event
event
event
event
event_base
event_callback
event_callback

libevent

event_base

```
1.    // without config
2.    struct event_base *event_base_new(void);
```

libevent

event_base

```
1.  // without config
2.  struct event_base *event_base_new(void);

3.  // with config
4.  struct event_config *event_config_new(void);
5.  struct event_base *event_base_new_with_config(const struct event_config *cfg);
6.  void event_config_free(struct event_config *cfg);
```

libevent

event_base

```
1.  // without config
2.  struct event_base *event_base_new(void);

3.  // with config
4.  struct event_config *event_config_new(void);
5.  struct event_base *event_base_new_with_config(const struct event_config *cfg);
6.  void event_config_free(struct event_config *cfg);

7.  // deallocating
8.  void event_base_free(struct event_base *base);
```

libevent

event_base

```
1.    EVLOOP_ONCE
2.    EVLOOP_NONBLOCK
3.    EVLOOP_NO_EXIT_ON_EMPTY
4.    int event_base_loop(struct event_base*, int flags);
```

libevent

event_base

```
1.  EVLOOP_ONCE
2.  EVLOOP_NONBLOCK
3.  EVLOOP_NO_EXIT_ON_EMPTY
4.  int event_base_loop(struct event_base*, int flags);

5.  // No flags
6.  int event_base_dispatch(struct event_base*);
```

libevent

event_base

```
1.  EVLOOP_ONCE
2.  EVLOOP_NONBLOCK
3.  EVLOOP_NO_EXIT_ON_EMPTY
4.  int event_base_loop(struct event_base*, int flags);

5.  // No flags
6.  int event_base_dispatch(struct event_base*);

7.  // Stop
8.  int event_base_loopexit(struct event_base*, const struct timeval*);
9.  int event_base_loopbreak(struct event_base*);
```

libevent

event_base

```
1.   EVLOOP_ONCE
2.   EVLOOP_NONBLOCK
3.   EVLOOP_NO_EXIT_ON_EMPTY
4.   int event_base_loop(struct event_base*, int flags);

5.   // No flags
6.   int event_base_dispatch(struct event_base*);

7.   // Stop
8.   int event_base_loopexit(struct event_base*, const struct timeval*);
9.   int event_base_loopbreak(struct event_base*);

10.  // Is stopping?
11.  int event_base_got_exit(struct event_base*);
12.  int event_base_got_break(struct event_base*);
```

libevent

event

```
1.   EV_TIMEOUT
2.   EV_WRITE
3.   EV_SIGNAL

4.   EV_PERSIST
5.   EV_ET

6.   typedef void (*event_callback_fn)(evutil_socket_t,
7.                                      short event,
8.                                      void *data);

9.   struct event *event_new(struct event_base*,
10.                          evutil_socket_t fd,
11.                          short what,
12.                          event_callback_fn cb,
13.                          void *arg);
14.  void event_free(struct event*);
```

libevent

Пример описания события

```
1.   void cb_func(evutil_socket_t fd, short what, void *arg)
2.   {
3.       const char *data = arg;
4.       printf("Got an event on socket %d:%s%s%s%s [%s]",
5.              (int)fd,
6.              (what & EV_TIMEOUT)  ? " timeout" : "",
7.              (what & EV_WRITE)    ? " write"   : "",
8.              (what & EV_READ)     ? " read"    : "",
9.              (what & EV_SIGNAL)   ? " signal"  : "",
10.             data);
11.  }
```

libevent

Пример работы с событиями

```
1.   void main_loop(evutil_socket_t fd1, evutil_socket_t fd2)
2.   {
3.       struct event *ev1, *ev2;
4.       struct timeval five_seconds = { 5, 0 };
5.       struct event_base *base = event_base_new();

6.       // Caller has already set up fd1, fd2 somehow, and make them nonblocking
7.       ev1 = event_new(base, fd1, EV_TIMEOUT | EV_READ | EV_PERSIST, cb_func, "Reading ev");
8.       ev2 = event_new(base, fd2 ,EV_WRITE | EV_PERSIST, cb_func, "Writing event");

9.       event_add(ev1, &five_seconds);
10.      event_add(ev2, NULL);
11.      event_base_dispatch(base);
12.  }
```

libevent

Пример работы с event_self_cbarg

```
1.  static int n_calls = 0;
2.  void cb_func(evutil_socket_t fd, short what, void *arg)
3.  {
4.      struct event *me = arg;
5.      printf("cb_func called %d times so far.\n", ++n_calls);
6.      if (n_calls > 100)
7.          event_del(me);
8.  }

9.  void run(struct event_base *base)
10. {
11.     struct timeval one_sec = { 1, 0 };
12.     struct event *ev;
13.     ev = event_new(base, -1, EV_PERSIST, cb_func, event_self_cbarg());
14.     event_add(ev, &one_sec);
15.     event_base_dispatch(base);
16. }
```

libevent

Timeout-only events

```
1.  #define evtimer_new(base, callback, arg) \
2.      event_new((base), -1, 0, (callback), (arg))
3.  #define evtimer_add(ev, tv) \
4.      event_add((ev),(tv))
5.  #define evtimer_del(ev) \
6.      event_del(ev)
7.  #define evtimer_pending(ev, tv_out) \
8.      event_pending((ev), EV_TIMEOUT, (tv_out))
9.  #define evtimer_assign(event, base, callback, arg)
10.     event_assign(event, base, -1, 0, callback, arg)
```

libevent

Signal events

```
1.  #define evsignal_new(base, signum, cb, arg) \
2.      event_new(base, signum, EV_SIGNAL|EV_PERSIST, cb, arg)
3.  #define evsignal_add(ev, tv) \
4.      event_add((ev),(tv))
5.  #define evsignal_del(ev) \
6.      event_del(ev)
7.  #define evsignal_pending(ev, what, tv_out) \
8.      event_pending((ev), (what), (tv_out))
9.  #define evsignal_assign(event, base, signum, callback, arg)
10.     event_assign(event, base, signum, EV_SIGNAL|EV_PERSIST, callback, arg)
```

libevent

event_assign

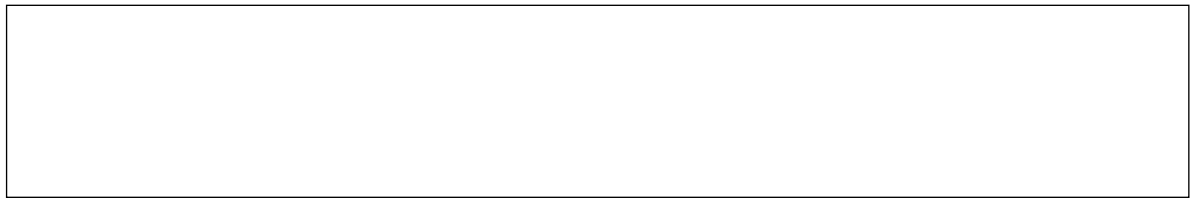
```
1.   int event_assign(struct event *event,
2.                    struct event_base *base,
3.                    evutil_socket_t fd,
4.                    short what,
5.                    void (*callback)(evutil_socket_t, short, void*),
6.                    void *arg);
```

libevent

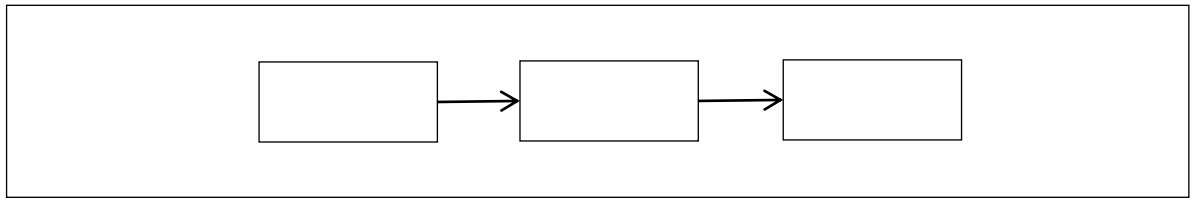
event_pair

```
1.   struct event_pair {
2.       evutil_socket_t fd;
3.       struct event read_event;
4.       struct event write_event;
5.   };
6.   void readcb(evutil_socket_t, short, void*);
7.   void writecb(evutil_socket_t, short, void*);

8.   struct event_pair *event_pair_new(struct event_base *base, evutil_socket_t fd) {
9.       struct event_pair *p = malloc(sizeof(struct event_pair));
10.      if (!p)
11.          return NULL;
12.      p->fd = fd;
13.      event_assign(&p->read_event,  base, fd, EV_READ  | EV_PERSIST, readcb,  p);
14.      event_assign(&p->write_event, base, fd, EV_WRITE | EV_PERSIST, writecb, p);
15.      return p;
16.  }
```

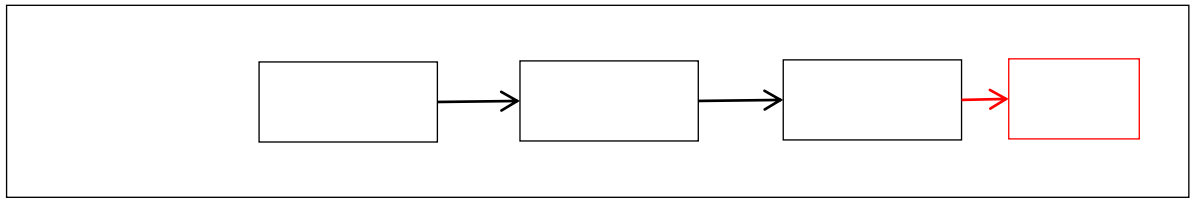
evbuffer



evbuffer

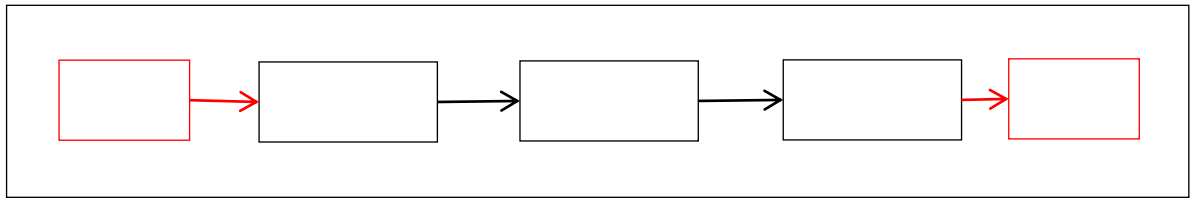


evbuffer



evbuffer_add
evbuffer_add_printf
evbuffer_add_vprintf
evbuffer_add_reference
evbuffer_add_buffer
evbuffer_add_buffer_reference
evbuffer_add_file
evbuffer_add_file_segment
evbuffer_read

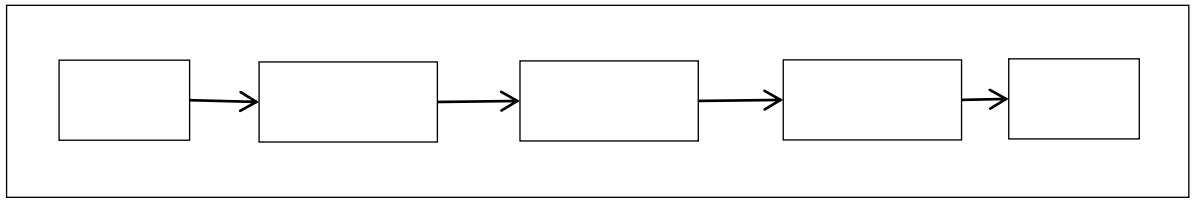
evbuffer



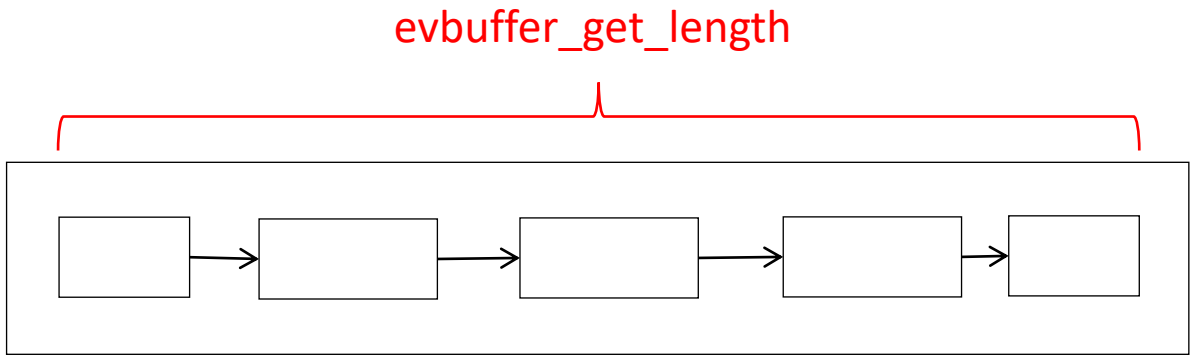
evbuffer_prepend
evbuffer_prepend_buffer

evbuffer_add
evbuffer_add_printf
evbuffer_add_vprintf
evbuffer_add_reference
evbuffer_add_buffer
evbuffer_add_buffer_reference
evbuffer_add_file
evbuffer_add_file_segment
evbuffer_read

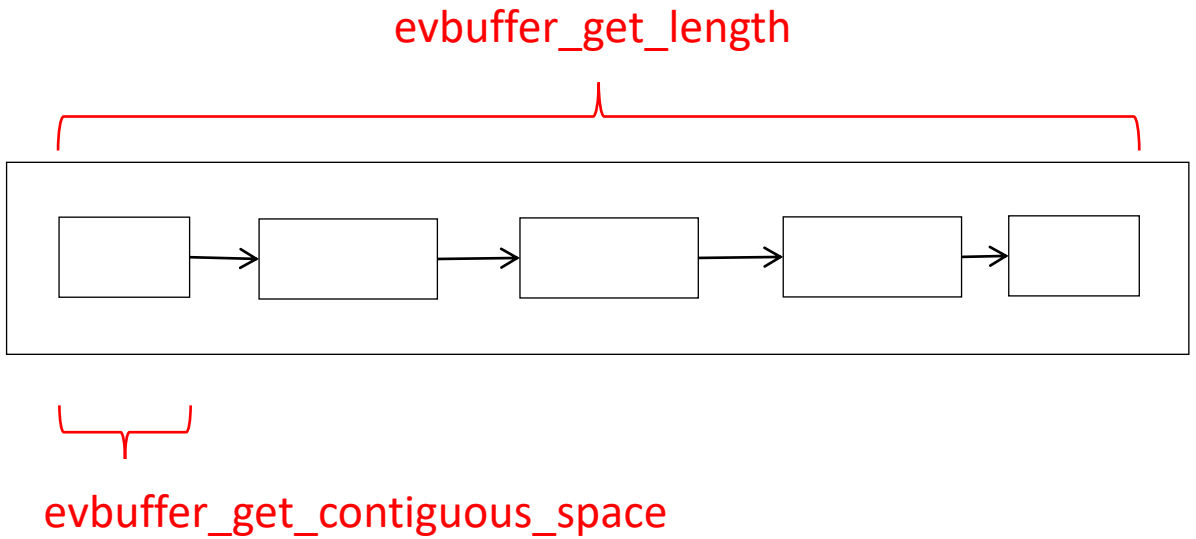
evbuffer



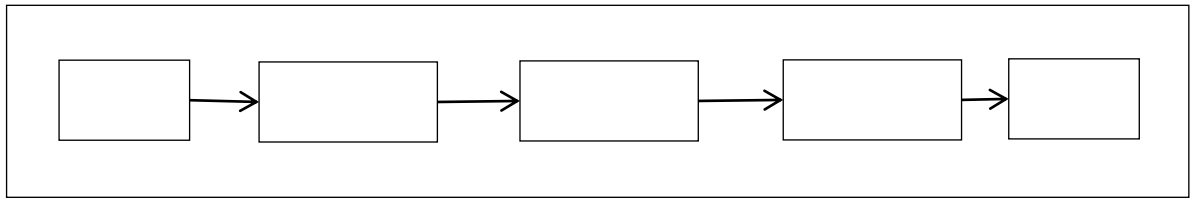
evbuffer



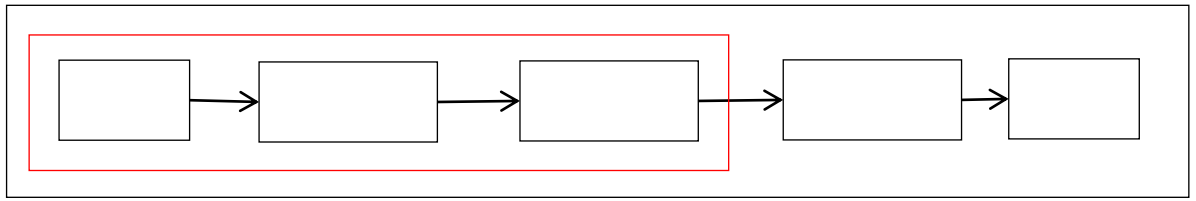
evbuffer



evbuffer

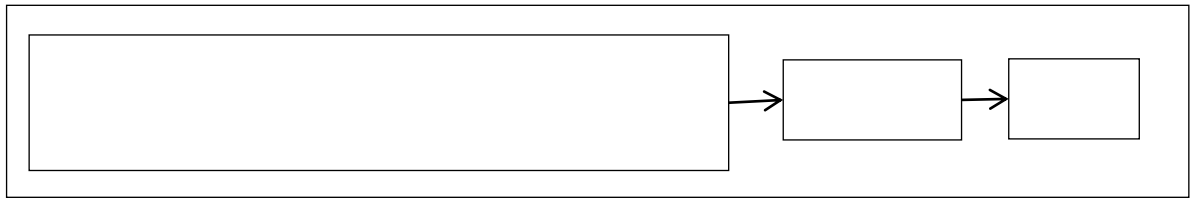


evbuffer

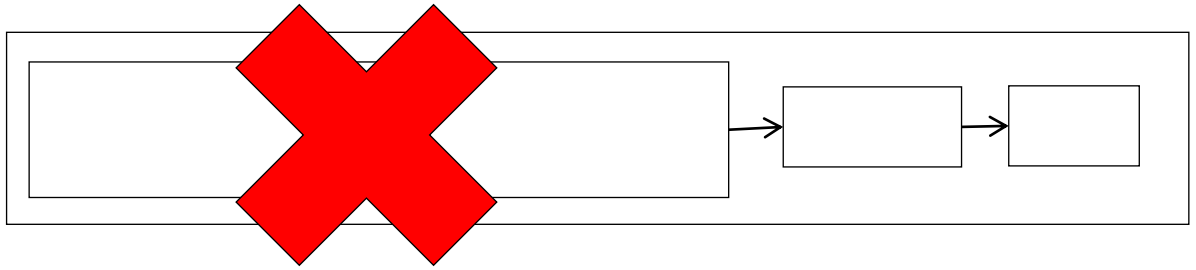


evbuffer_pullup

evbuffer

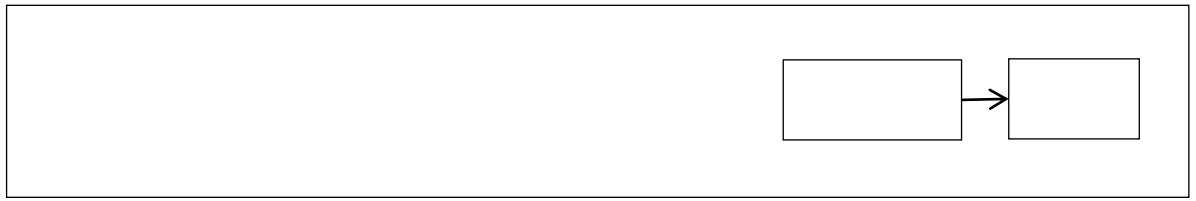


evbuffer

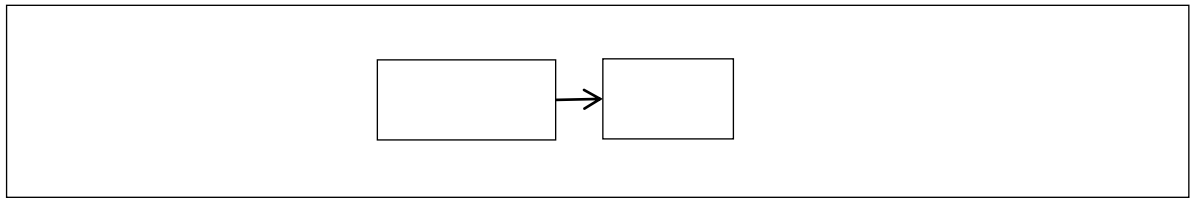


evbuffer_drain
evbuffer_remove
evbuffer_remove_buffer

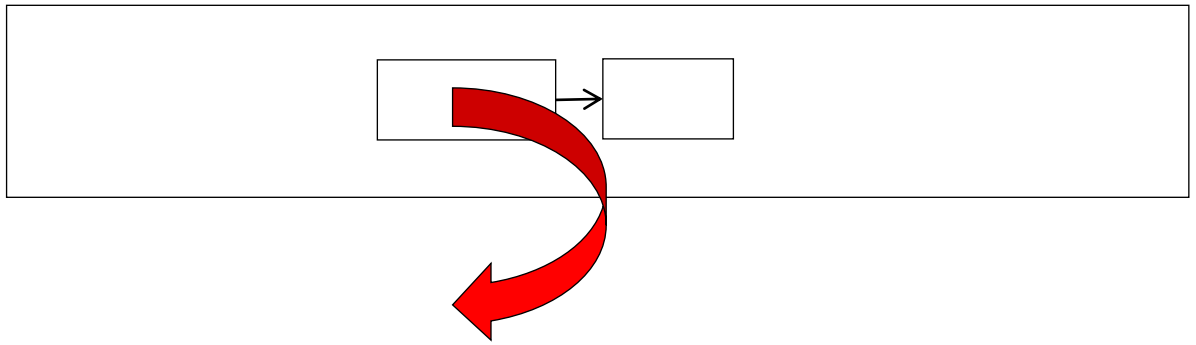
evbuffer



evbuffer

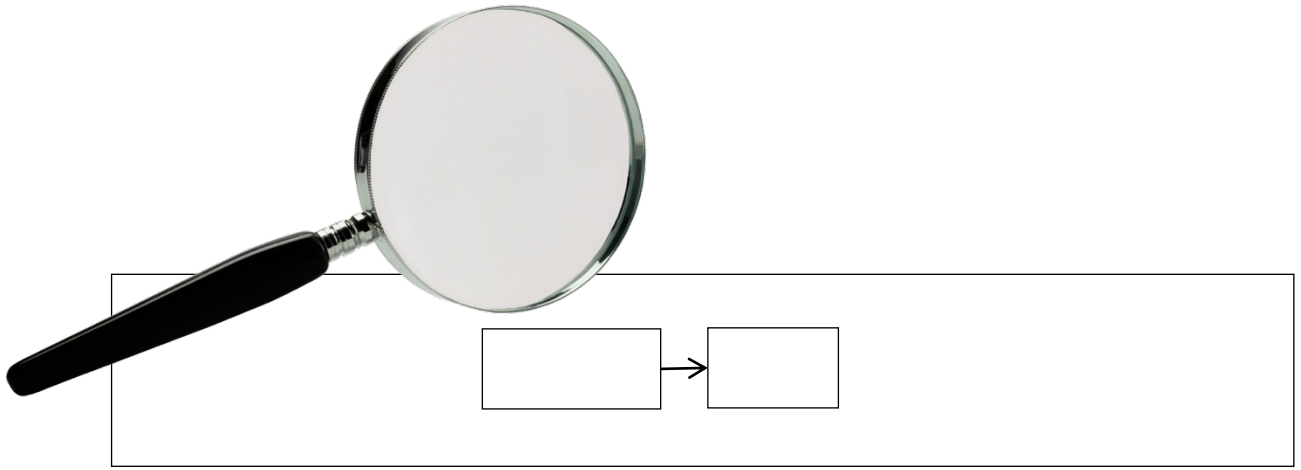


evbuffer



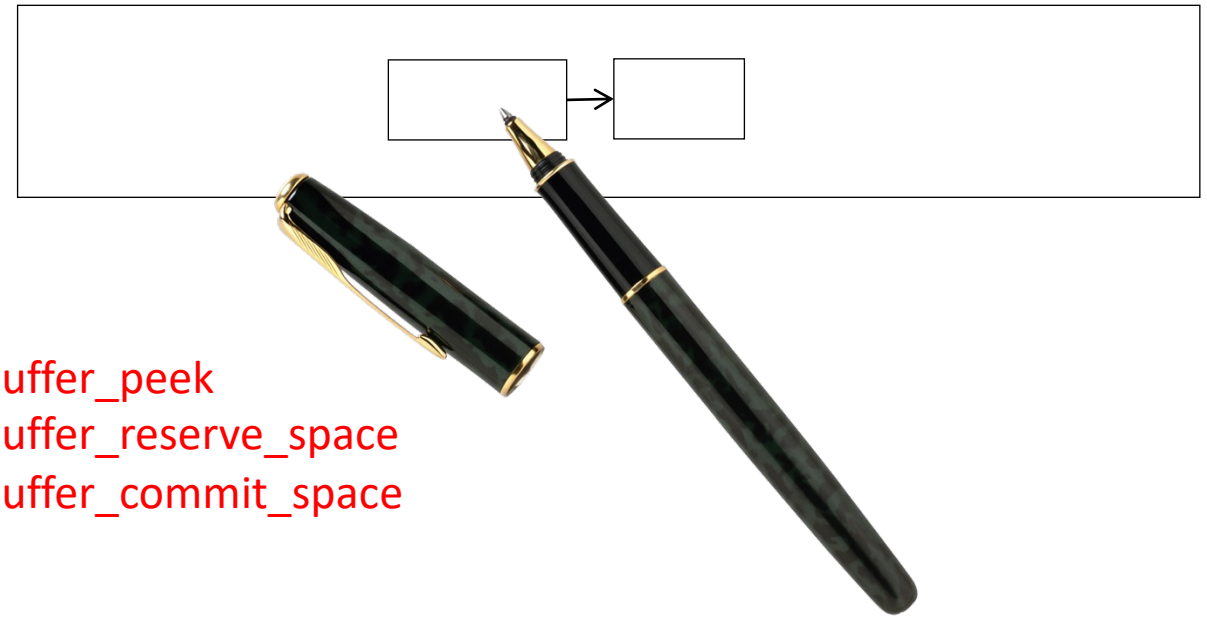
evbuffer_copyout
evbuffer_copyout_from
evbuffer_write
evbuffer_write_atmost

evbuffer



evbuffer_search
evbuffer_search_range
evbuffer_search_eol

evbuffer



evbuffer_peek
evbuffer_reserve_space
evbuffer_commit_space

evbuffer

evbuffer

```
1.    struct evbuffer *evbuffer_new(void);
2.    void evbuffer_free(struct evbuffer *buf);
```

evbuffer

evbuffer

```
1.  struct evbuffer *evbuffer_new(void);
2.  void evbuffer_free(struct evbuffer *buf);

3.  // Thread safety
4.  int evbuffer_enable_locking(struct evbuffer *buf, void *lock);
5.  void evbuffer_lock(struct evbuffer *buf);
6.  void evbuffer_unlock(struct evbuffer *buf);
```

bufferevents

Socket-based bufferevent

```
1.  struct bufferevent *bufferevent_socket_new(struct event_base *base,
2.                                             evutil_socket_t fd,
3.                                             enum bufferevent_options options);
4.  void bufferevent_free(struct bufferevent *bev);

5.  int bufferevent_socket_connect(struct bufferevent *bev,
6.                                 struct sockaddr *address,
7.                                 int addrlen);

8.  int bufferevent_socket_connect_hostname(struct bufferevent *bev,
9.                                          struct evdns_base *dns_base,
10.                                         int family, const char *hostname, int port);

11. int bufferevent_socket_get_dns_error(struct bufferevent *bev);
```

bufferevents

Socket-based bufferevent

```
1.    BEV_EVENT_READING
2.    BEV_EVENT_WRITING
3.    BEV_EVENT_ERROR
4.    BEV_EVENT_TIMEOUT
5.    BEV_EVENT_EOF
6.    BEV_EVENT_CONNECTED
```

bufferevents

Socket-based bufferevent

```
1.  typedef void (*bufferevent_data_cb)(struct bufferevent *bev, void *ctx);
2.  typedef void (*bufferevent_event_cb)(struct bufferevent *bev,
3.                                       short events,
4.                                       void *ctx);

5.  void bufferevent_setcb(struct bufferevent *bufev,
6.                         bufferevent_data_cb readcb,
7.                         bufferevent_data_cb writecb,
8.                         bufferevent_event_cb eventcb,
9.                         void *cbarg);
10. void bufferevent_getcb(struct bufferevent *bufev,
11.                        bufferevent_data_cb *readcb_ptr,
12.                        bufferevent_data_cb *writecb_ptr,
13.                        bufferevent_event_cb *eventcb_ptr,
14.                        void **cbarg_ptr);

15. void bufferevent_enable(struct bufferevent *bufev, short events);
16. void bufferevent_disable(struct bufferevent *bufev, short events);
17. short bufferevent_get_enabled(struct bufferevent *bufev);
```

bufferevents

Socket-based bufferevent

```
1.   struct bufferevent *bufferevent_socket_new(struct event_base *base,
2.                                              evutil_socket_t fd,
3.                                              enum bufferevent_options options);
4.   void bufferevent_free(struct bufferevent *bev);
```

bufferevents

Socket-based bufferevent

```
1.  struct evbuffer *bufferevent_get_input(struct bufferevent *bufev);
2.  struct evbuffer *bufferevent_get_output(struct bufferevent *bufev);

3.  int bufferevent_write(struct bufferevent *bufev, const void *data, size_t size);
4.  int bufferevent_write_buffer(struct bufferevent *bufev, struct evbuffer *buf);

5.  size_t bufferevent_read(struct bufferevent *bufev, void *data, size_t size);
6.  int bufferevent_read_buffer(struct bufferevent *bufev, struct evbuffer *buf);
```

bufferevents

Socket-based bufferevent

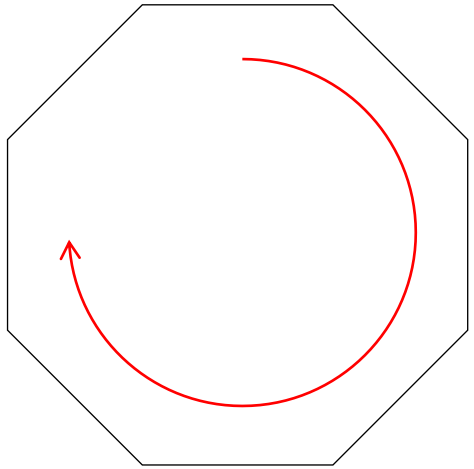
```
1.    void bufferevent_set_timeouts(struct bufferevent *bufev,
                                    const struct timeval *timeout_read,
                                    const struct timeval *timeout_write);

2.    int bufferevent_flush(struct bufferevent *bufev,
                            short iotype,
                            enum bufferevent_flush_mode state);
```

bufferevents

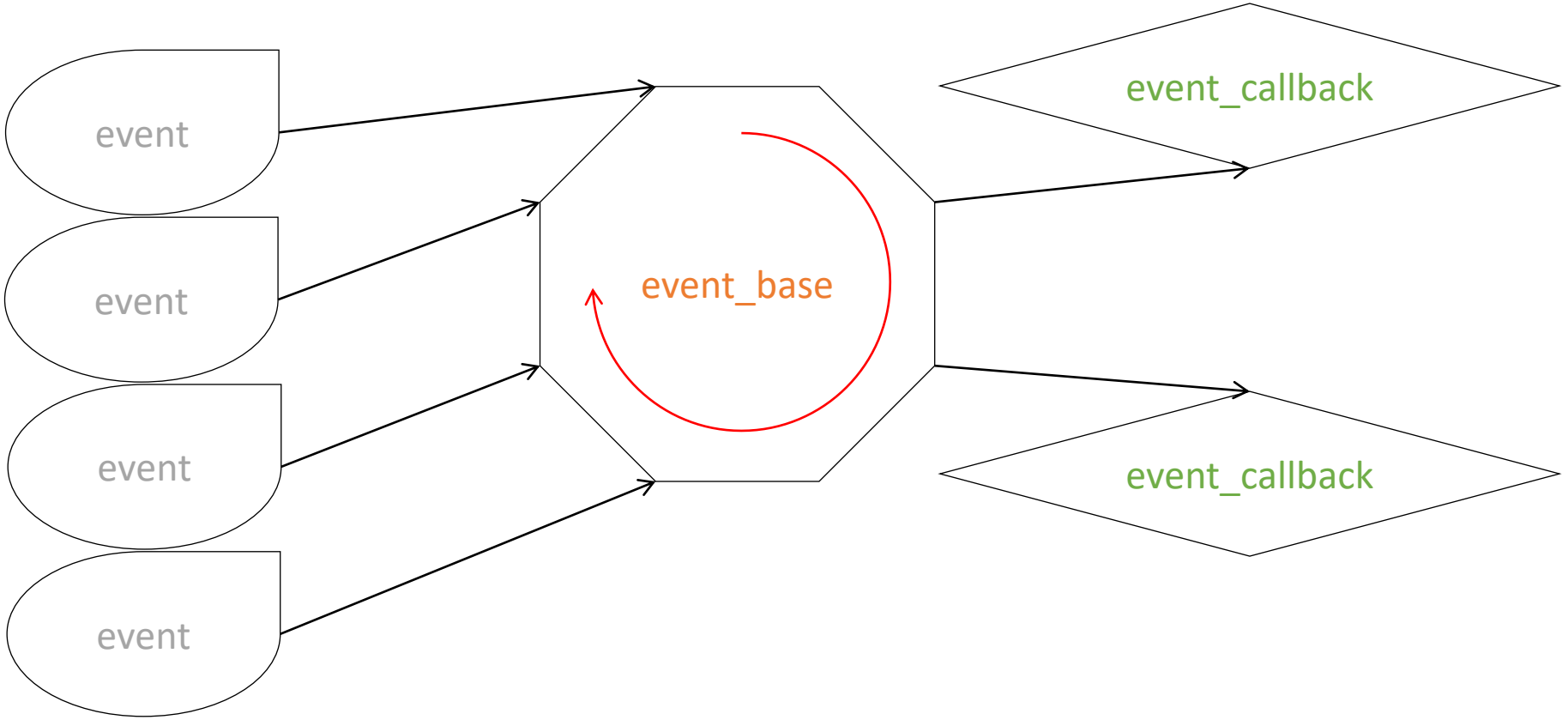
listener

```
1.   struct evconnlistener *evconnlistener_new(struct event_base *base, evconnlistener_cb cb,
2.                                             void *arg, unsigned flags, int backlog,
3.                                             evutil_socket_t fd);

4.   struct evconnlistener *evconnlistener_new_bind(struct event_base *base, evconnlistener_cb cb,
5.                                                  void * arg, unsigned flags, int backlog,
6.                                                  const struct sockaddr *sa, int sa_len);

7.   void evconnlistener_free(struct evconnlistener*);
8.   void evconnlistener_set_error_cb(struct evconnlistener*, evconnlistener_cb cb);

9.   typedef void (*evconnlistener_cb)(struct evconnlistener *listener,
10.                                    evutil_socket_t MasterSock,
11.                                    struct sockaddr *addr_client,
12.                                    int addr_client_len,
13.                                    void *arg);
```

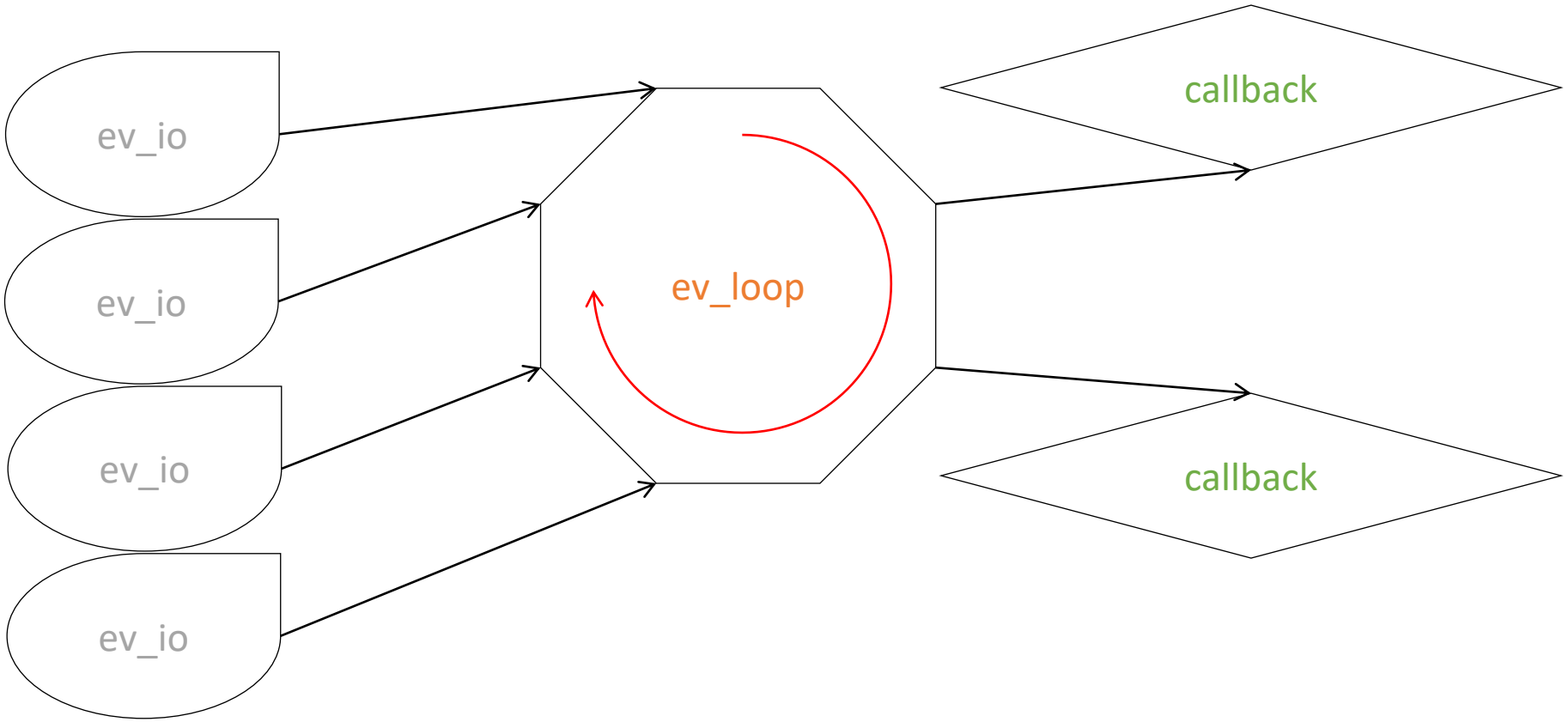
libev



libevent


event
event
event
event
event_base
event_callback
event_callback

libev



libev

loop

```
1.  // главный цикл сообщений
2.  struct ev_loop *ev_default_loop(unsigned flags);
3.  void ev_default_fork(void);
4.  void ev_default_destroy(void);

5.  // дополнительные циклы сообщений
6.  struct ev_loop *ev_loop_new(unsigned flags);
7.  ev_loop_destroy(struct ev_loop *loop);
8.  ev_loop_fork(struct ev_loop *loop); // копии циклов сообщений

9.  EVLOOP_NONBLOCK и EVLOOP_ONESHOT

10. // проверки
11. int ev_is_default_loop(struct ev_loop *loop);
12. unsigned ev_loop_count(struct ev_loop *loop);

13. // запустить/остановить
14. ev_loop(struct ev_loop *loop, int flags);
15. ev_unloop(struct ev_loop *loop, int how);
```

libev

Anatomy of a watcher

```
1.  static void my_cb(struct ev_loop *loop, struct ev_io *w, int revents) {
2.      ev_io_stop(w);
3.      ev_unloop(loop, EVUNLOOP_ALL);
4.  }

5.  struct ev_loop *loop = ev_default_loop(0);
6.  struct ev_io stdin_watcher; //ввод с клавиатуры
7.  ev_init(&stdin_watcher, my_cb);
8.  ev_io_set(&stdin_watcher, STDIN_FILENO, EV_READ);
9.  ev_io_start(loop, &stdin_watcher);
10. ev_loop(loop, 0);
```

libev

Anatomy of a watcher

```
1.  static void my_cb(struct ev_loop *loop, struct ev_io *w, int revents) {
2.      ev_io_stop(w);
3.      ev_unloop(loop, EVUNLOOP_ALL);
4.  }

5.  struct ev_loop *loop = ev_default_loop(0);
6.  struct ev_io stdin_watcher;
7.  ev_init(&stdin_watcher, my_cb);
8.  ev_io_set(&stdin_watcher, STDIN_FILENO, EV_READ);
9.  ev_io_start(loop, &stdin_watcher);
10. ev_loop(loop, 0);
```

libev

Anatomy of a watcher

```
1.   static void my_cb(struct ev_loop *loop, struct ev_io *w, int revents) {
2.       ev_io_stop(w);
3.       ev_unloop(loop, EVUNLOOP_ALL);
4.   }

5.   struct ev_loop *loop = ev_default_loop(0);
6.   struct ev_io stdin_watcher;
7.   ev_init(&stdin_watcher, my_cb);
8.   ev_io_set(&stdin_watcher, STDIN_FILENO, EV_READ);
9.   ev_io_start(loop, &stdin_watcher);
10.  ev_loop(loop, 0);
```

libev

Associating custom data with a watcher

```
1.   struct my_io {
2.       struct ev_io io;
3.       int otherfd;
4.       void *somedata;
5.       struct whatever *mostinteresting;
6.   }
```

libev

Associating custom data with a watcher

```
1.   struct my_io {
2.       struct ev_io io;
3.       int otherfd;
4.       void *somedata;
5.       struct whatever *mostinteresting;
6.   }

7.   static void my_cb(struct ev_loop *loop, struct ev_io *watcher, int revents) {
8.       struct my_io *w = (struct my_io*)watcher;
9.       /* ... */
10.  }
```

libev

Associating custom data with a watcher

```
1.   struct my_biggy {
2.       int some_data;
3.       ev_timer t1;
4.       ev_timer t2;
5.   }
```

libev

Associating custom data with a watcher

```
1.  struct my_biggy {
2.      int some_data;
3.      ev_timer t1;
4.      ev_timer t2;
5.  }

6.  static void t1_cb(struct ev_loop *loop, struct ev_timer *w, int revents) {
7.      struct my_biggy big = (struct my_biggy*)(((char*)w) - offsetof(struct my_biggy, t1));
8.      // ...
9.  }

10. static void t2_cb(struct ev_loop *loop, struct ev_timer *w, int revents) {
11.     struct my_biggy big = (struct my_biggy*)(((char*)w) - offsetof(struct my_biggy, t2));
12.     // ...
13. }
```

libev

ev_io

```
1.  static void stdin_readable_cb(struct ev_loop *loop, struct ev_io *w, int revents) {
2.      ev_io_stop(loop, w);
3.      // .. read from stdin here (or from w->fd) and haqndle any I/O errors
4.  }

5.  // ...

6.  struct ev_loop *loop = ev_default_init(0);
7.  struct ev_io stdin_readable;
8.  ev_io_init(&stdin_readable, stdin_readable_cb, STDIN_FILENO, EV_READ);
9.  ev_io_start(loop, &stdin_readable);
10. ev_loop(loop, 0);
```

libev

ev_timer

```
1.  static void one_minute_cb(struct ev_loop *loop, struct ev_timer *w, int revents) {
2.      ... one minute over, w is actually stopped right here
3.  }

4.  struct ev_timer mytimer;
5.  ev_timer_init(&mytimer, one_minute_cb, 60. /*after*/, 0. /*repeat*/);
6.  ev_timer_start(loop, &mytimer);
```

libev

ev_periodic

```
1.  static void timeout_cb(struct ev_loop *loop, struct ev_timer *w, int revents) {
2.      ... ten seconds without any activity
3.  }

4.  struct ev_timer mytimer;
5.  ev_timer_init(&mytimer,timeout_cb, 0., 10.); //note, only repeat used
6.  ev_timer_again(&mytimer); //start timer
7.  ev_loop(loop, 0);
8.  //and in some piece of code that gets executed on any "activity":
9.  //reset the timeout to start ticking again at 10 seconds
10. ev_timer_again(&mytimer);
```

libev

ev_signal

```
1.  static void sigint_cb(struct ev_loop *loop, struct ev_signal *w, int revents) {
2.      ev_unloop(loop, EVUNLOOP_ALL);
3.  }

4.  struct ev_signal signal_watcher;
5.  ev_signal_init(&signal_watcher, sigint_cb, SIGINT);
6.  ev_signal_start(loop, &sigint_cb);
```

libev

ev_child

```
1.  ev_child cw;
2.  static void child_cb(struct ev_loop *loop, struct ev_child *w, int revents) {
3.      ev_child_stop(loop, w);
4.      printf("process %d exited with status %x\n", w->rpid, w->rstatus);
5.  }

6.  pid_t pid = fork();
7.  if (pid < 0) { //error else if (pid == 0)
8.      // the forked child executes here exit(1);
9.  } else {
10.     ev_child_init(&cw, child_cb, pid, 0);
11.     ev_child_start(ev_default_loop(0), &cw);
12. }
```

libev

ev_stat

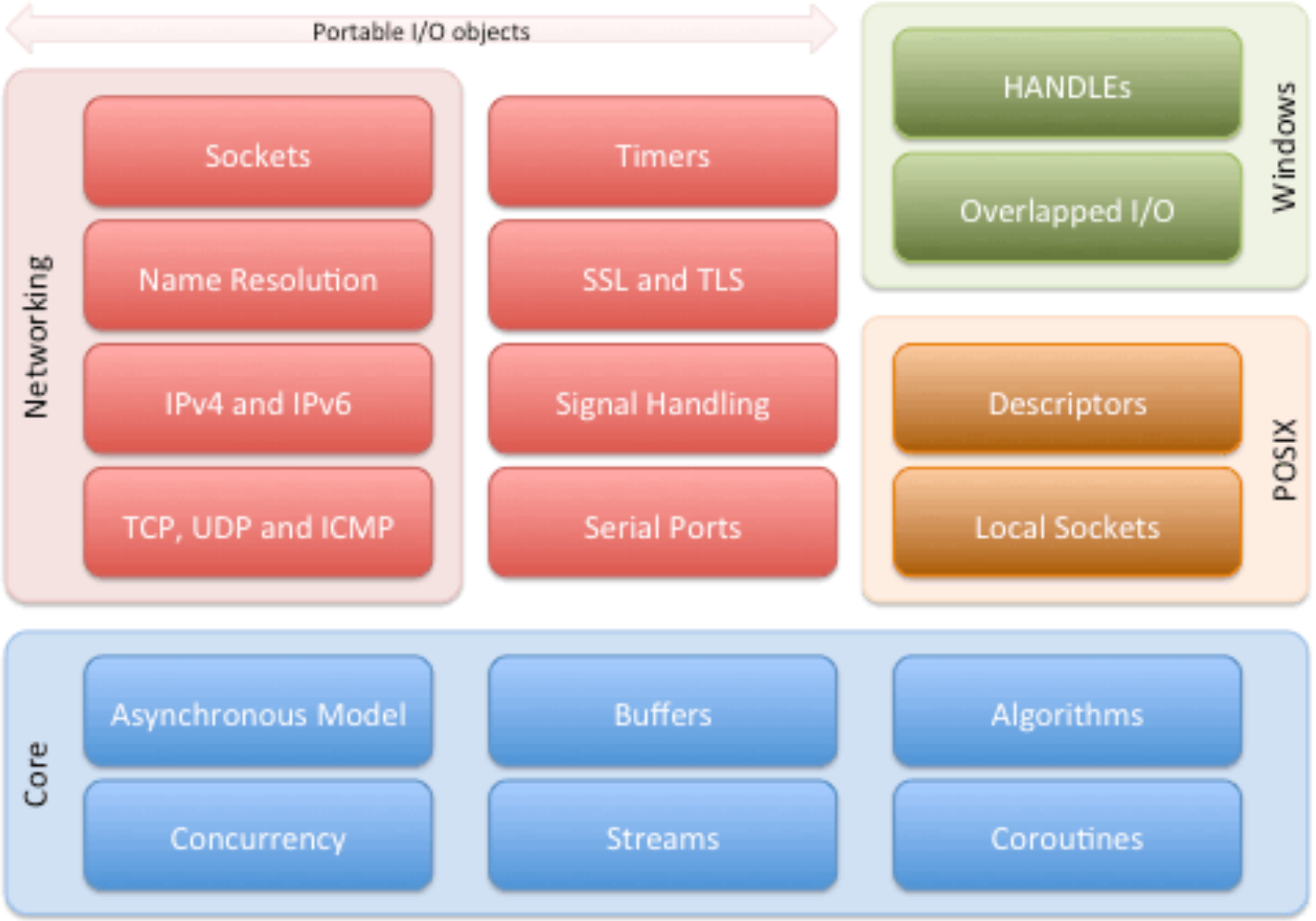
```
1.  static void passwd_cb(struct ev_loop *loop, ev_stat *w, int revents) {
2.      if (w->attr.st_nlink) {
3.          printf("passwd current size %ld\n",  (long)w->attr.st_size);
4.          printf("passwd current atime %ld\n", (long)w->attr.st_atime);
5.          printf("passwd current mtime %ld\n", (long)w->attr.st_mtime);
6.      } else { //shall not abuse printf for puts
7.          puts ("wow, /etc/passwd is not there, expect problems. “
8.                "if this is windows, they already arrived\n");
9.      }
10. }

11. ev_stat passwd;
12. ev_stat_init(&passwd, passwd_cb, "/etc/passwd", 0.);
13. ev_stat_start(loop, &passwd);
```

libev

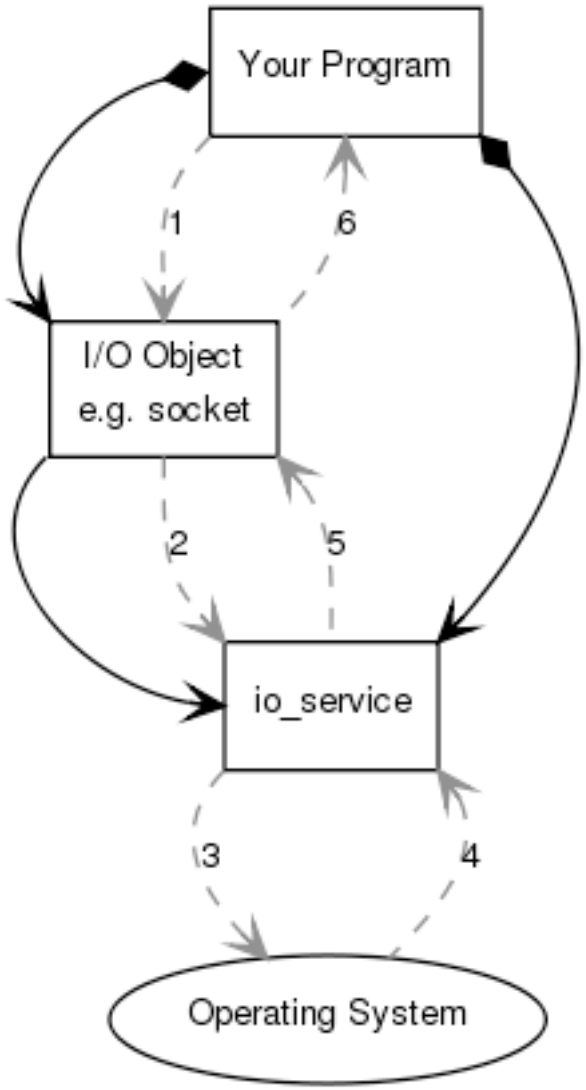
ev_idle

```
1.  static void idle_cb(struct ev_loop *loop, struct ev_idle *w, int revents) {
2.      free(w); // now do something you wanted to do
3.               // when the program has no longer anything immediate to do.
4.  }

5.  struct ev_idle *idle_watcher = malloc(sizeof(struct ev_idle));
6.  ev_idle_init(idle_watcher, idle_cb);
7.  ev_idle_start(loop, idle_cb);
```

boost::asio


Portable I/O objects
Networking
Sockets
Name Resolution
IPv4 and IPv6
TCP, UDP and ICMP
Timers
SSL and TLS
Signal Handling
Serial Ports
Windows
HANDLEs
Overlapped I/O
POSIX
Descriptors
Local Sockets
Core
Asynchronous Model
Concurrency
Buffers
Streams
Algorithms
Coroutines

boost::asio

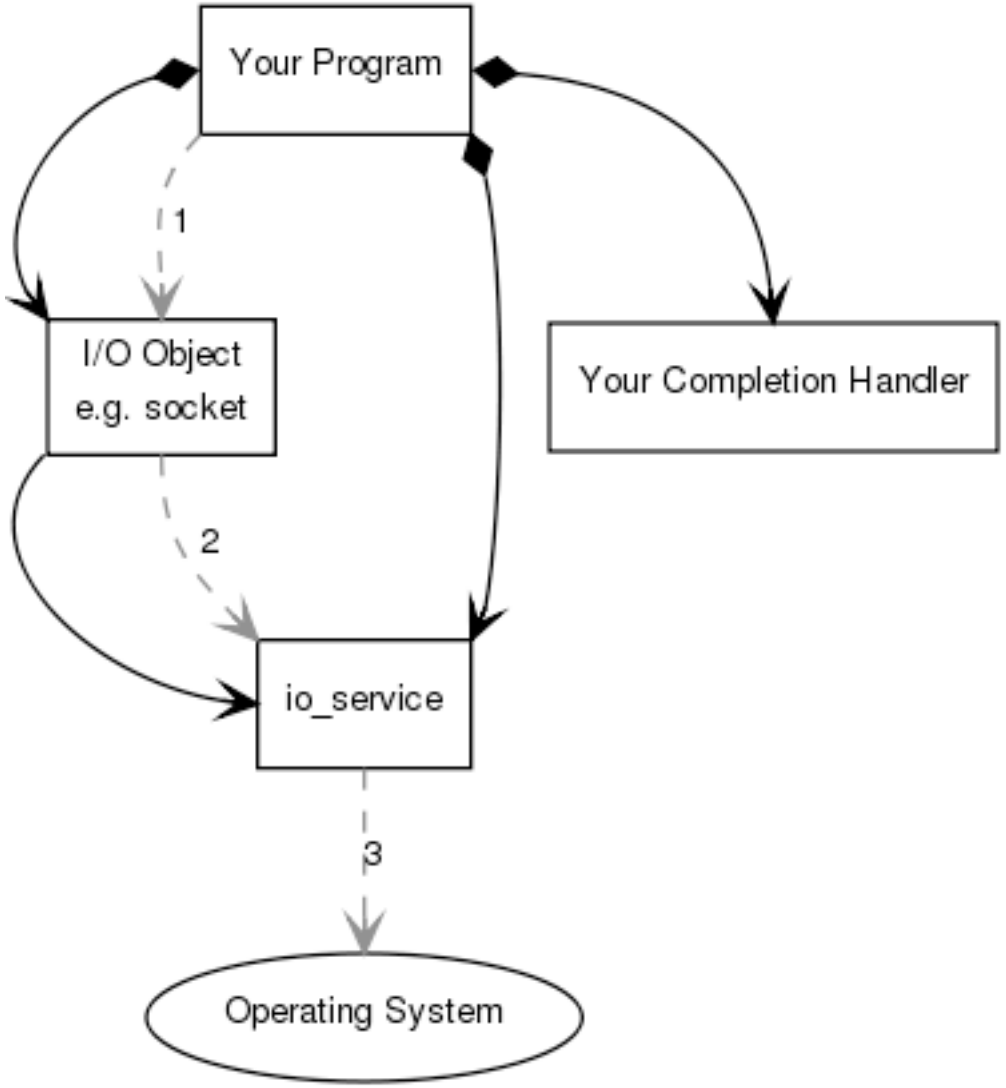

Your Program
1
6
I/O Object
e.g. socket
2
5
io_service
3
4
Operating System

boost::asio

Синхронная работа

```
1.  boost::asio::io_service io_service;
2.  boost::asio::ip::tcp::socket socket(io_service);

3.  boost::system::error_code ec;
4.  socket.connect(server_endpoint, ec);
```

boost::asio

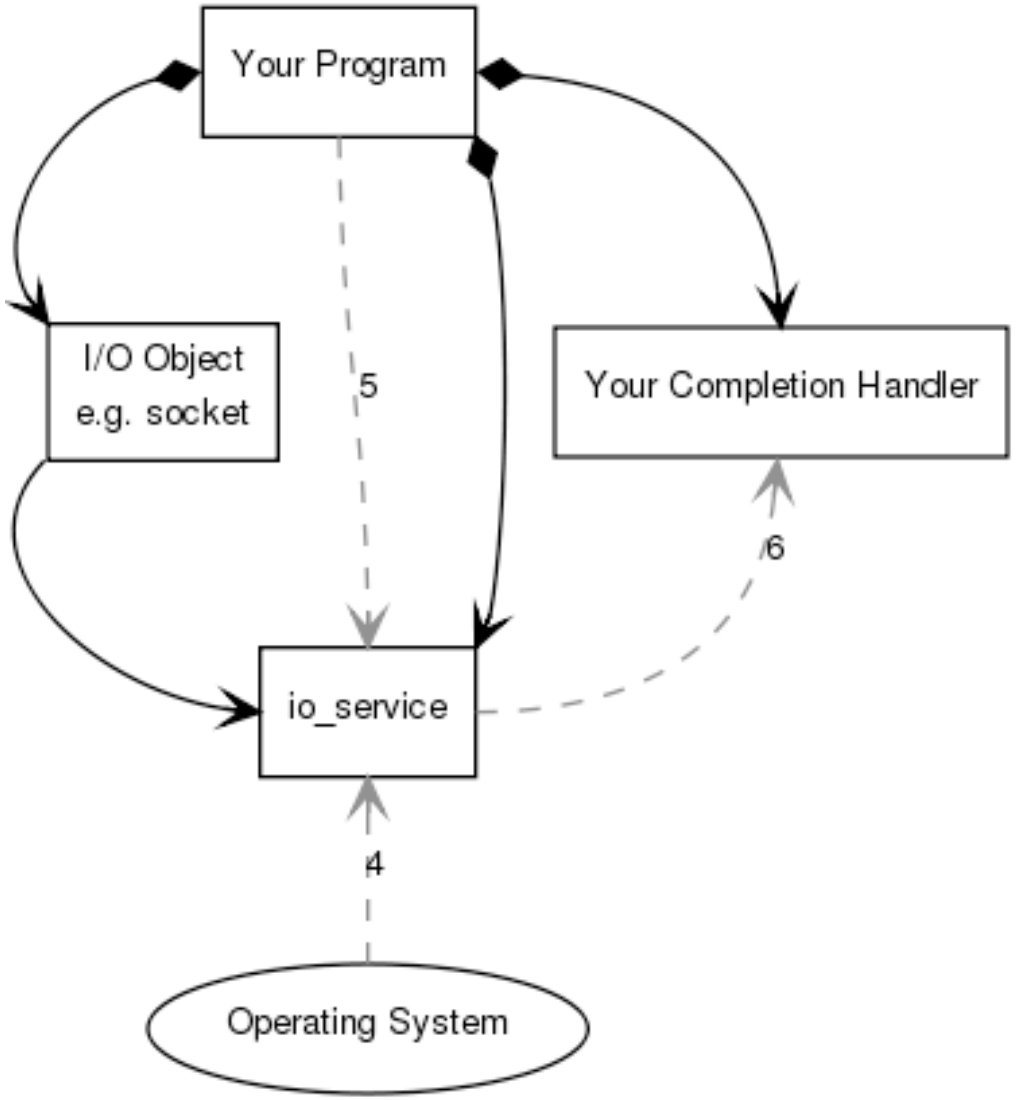

Your Program
1
I/O Object
e.g. socket
Your Completion Handler
2
io_service
3
Operating System

boost::asio

Асинхронная работа

```
1.  void your_completion_handler(const boost::system::error_code &ec) {
2.      // ...
3.  }

4.  boost::asio::io_service io_service;
5.  boost::asio::ip::tcp::socket socket(io_service);
6.  socket.async_connect(endpoint, your_completion_handler);

7.  io_service.run();
```

boost::asio


Your Program
I/O Object
e.g. socket
5
Your Completion Handler
6
io_service
4
Operating System

boost::asio

std::bind *или* boost::bind

```
1.  double my_divide(double x, double y) {
2.      return x / y;
3.  }

4.  int main()
5.  {
6.      using namespace std::placeholders;
7.      auto fn_five = std::bind(my_divide, 10, 2);
8.      auto fn_half = std::bind(my_divide, _1, 2);
9.      auto fn_invert = std::bind(my_divide, _2, _1);
10.     auto fn_rounding = std::bind<int>(my_divide, _1, _2);
11. }
```

boost::asio

std::bind или boost::bind

```
1.  double my_divide(double x, double y) {
2.      return x / y;
3.  }

4.  int main()
5.  {
6.      using namespace std::placeholders;
7.      auto fn_five = std::bind(my_divide, 10, 2);
8.      auto fn_half = std::bind(my_divide, _1, 2);
9.      auto fn_invert = std::bind(my_divide, _2, _1);
10.     auto fn_rounding = std::bind<int>(my_divide, _1, _2);
11. }
```

boost::asio

std::bind или boost::bind

```
1.  double my_divide(double x, double y) {
2.      return x / y;
3.  }

4.  int main()
5.  {
6.      using namespace std::placeholders;
7.      auto fn_five = std::bind(my_divide, 10, 2);             // returns 10/2
8.      auto fn_half = std::bind(my_divide, _1, 2);             // returns x/2
9.      auto fn_invert = std::bind(my_divide, _2, _1);          // returns y/x
10.     auto fn_rounding = std::bind<int>(my_divide, _1, _2);   // returns int(x/y)
11. }
```

boost::asio

TCP

```
1.  boost::asio::ip::tcp::endpoint endpoint(boost::asio::ip::udp::v4("0.0.0.0"), 12345);
2.  boost::asio::ip::tcp::acceptor acceptor(io_service, endpoint);
3.  boost::asio::ip::tcp::socket socket(io_service);
4.  acceptor.accept(socket);
```

boost::asio

UDP

```
1.  boost::asio::ip::udp::endpoint endpoint(boost::asio::ip::udp::v4(), 12345);
2.  boost::asio::ip::udp::socket socket(io_service, endpoint);
```

boost::asio

```
1   #include <cstdlib>
2   #include <iostream>
3   #include <boost/bind.hpp>
4   #include <boost/asio.hpp>
5
6   using boost::asio::ip::tcp;
7
8   class session {
9   private:
10      tcp::socket socket_;
11      enum { max_length = 1024 };
12      char data_[max_length];
13  public:
14      session(boost::asio::io_service &io_service) : socket_(io_service) {}
15      tcp::socket &socket() { return socket_; }
16      void start() {
17          socket_.async_read_some(
18              boost::asio::buffer(data_, max_length),
19              boost::bind(&session::handle_read,
20                  this,
21                  boost::asio::placeholders::error,
22                  boost::asio::placeholders::bytes_transferred));
23      };
24  };
```

boost::asio

```
26   void handle_read(const boost::system::error_code &error,
27                    size_t bytes_transferred) {
28       if (!error) {
29       boost::asio::async_write(socket_,
30           boost::asio::buffer(data_, bytes_transferred),
31           boost::bind(&session::handle_write,
32           this,
33           boost::asio::placeholders::error));
34       } else {
35       delete this;
36       }
37   }
38
39   void handle_write(const boost::system::error_code &error) {
40       if (!error) {
41       socket_.async_read_some(
42           boost::asio::buffer(data_, max_length),
43           boost::bind(&session::handle_read, this,
44           boost::asio::placeholders::error,
45           boost::asio::placeholders::bytes_transferred));
46       } else {
47       delete this;
48       }
49   }
```

boost::asio

```
51  class server {
52  private:
53      boost::asio::io_service &io_service_;
54      tcp::acceptor acceptor_;
55  public:
56      server(boost::asio::io_service &io_service, short port):
57          io_service_(io_service),
58          acceptor_(io_service, tcp::endpoint(tcp::v4(), port)) {
59          session *new_session = new session(io_service_);
60          acceptor_.async_accept(new_session->socket(),
61              boost::bind(&server::handle_accept, this, new_session,
62              boost::asio::placeholders::error));
63      }
64
65      void handle_accept(session *new_session,
66                         const boost::system::error_code &error) {
67          if (!error) {
68              new_session->start();
69              new_session = new session(io_service_);
70              acceptor_.async_accept(new_session->socket(),
71                  boost::bind(&server::handle_accept, this,
72                  new_session,
73                  boost::asio::placeholders::error));
74          } else {
75              delete new_session;
76          }
77      }
78  };
```

boost::asio

```
80    int main(int argc, char **argv) {
81        boost::asio::io_service io_service;
82        using namespace std;
83        server s(io_service, atoi(argv[1]));
84        io_service.run();
85        return 0;
86    }
```

boost::asio

DNS

```
1.  boost::asio::ip::tcp::socket socket(io_service);
2.  boost::asio::ip::tcp::resolver resolver(io_service);
3.  boost::asio::ip::tcp::resolver::query query("www.boost.org", "http");
4.  boost::asio::ip::tcp::resolver::iterator iter = resolver.resolve(query);
5.  boost::asio::ip::tcp::resolver::iterator end; //end marker

6.  while (iter != end) {
7.      boost::asio::ip::tcp::endpoint endpoint = *iter;
8.      ++iter;
9.      std::cout << endpoint << std::endl;
10. }
```

boost::asio

TCP-Клиент

```
1.  boost::asio::ip::tcp::socket socket(io_service);
2.  boost::asio::ip::tcp::resolver resolver(io_service);
3.  boost::asio::ip::tcp::resolver::query query("www.boost.org", "http");
4.  boost::asio::ip::tcp::resolver::iterator iter = resolver.resolve(query);
5.  // Синхронно
6.  boost::asio::connect(socket, iter);

7.  // Асинхронно
8.  boost::asio::async_connect(socket, iter,
9.      boost::bind(&client::handle_connect, this,
10.         boost::asio::placeholders::error));

11. void client::handle_connect(const error_code& error) {
12.     if (!error) {
13.         // ... start read or write operations
14.     } else {
15.         // ... handle error
16.     }
17. }
```

boost::asio

Поток

```
1.  boost::asio::ip::tcp::iostream stream;
2.  stream.expires_from_now(boost::posix_time::seconds(60));
3.  stream.connect("www.boost.org", "http");
4.  stream << "GET /LICENSE_1_0.txt HTTP/1.0\r\n";
5.  stream << "Host: www.boost.org\r\n";
6.  stream << "Accept: */*\r\n";
7.  stream << "Connection: close\r\n\r\n";
8.  stream.flush();
9.  std::cout << stream.rdbuf();
```

Документация

- **libevent** – http://www.wangafu.net/~nickm/libevent-book/TOC.html
- **libev** – http://pod.tst.eu/http://cvs.schmorp.de/libev/ev.pod
- **boost::asio** – https://www.boost.org/doc/libs/1_74_0/doc/html/boost_asio.html

- https://github.com/tarantool/tarantool/blob/master/third_party/libev/ev.h