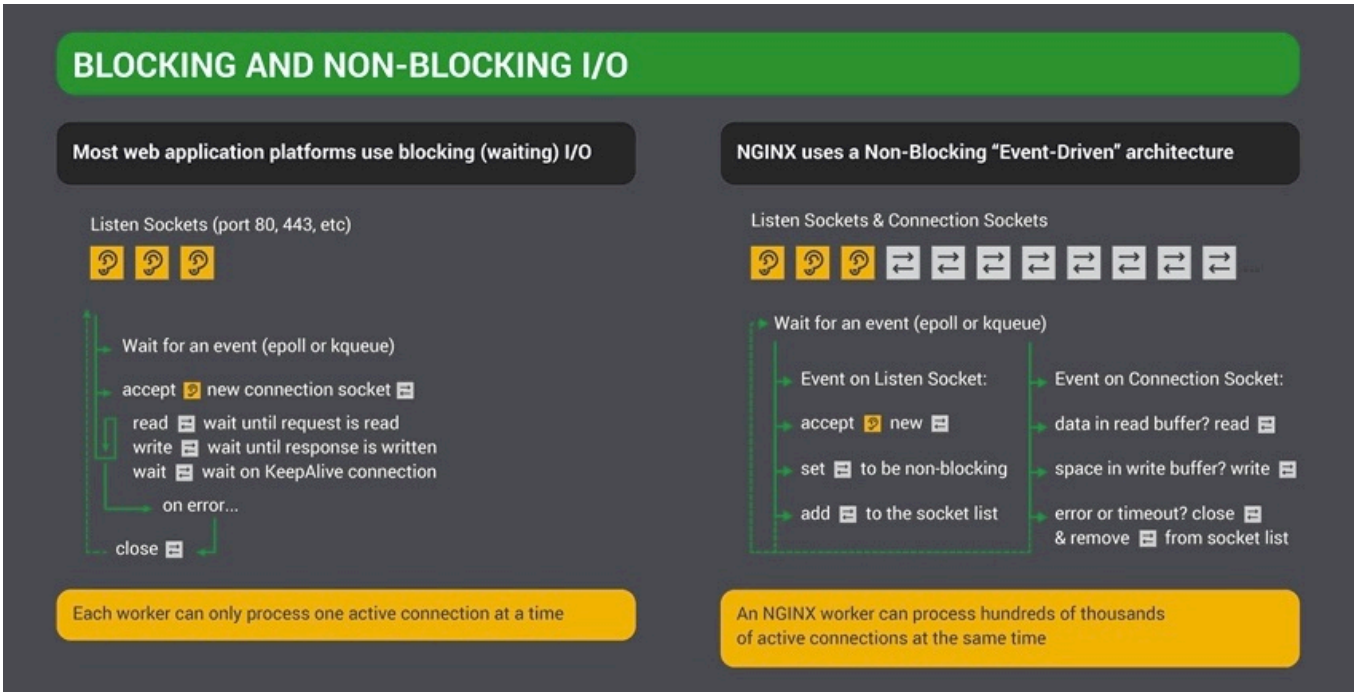
Non-blocking I/O and Event loop

> Non-blocking I/O + Event loop ==> Single Thread + Asynchronous [I/O]

Blocking vs Non-blocking I/O

http://www.wangafu.net/~nickm/libevent-book/01_intro.html

Nginx



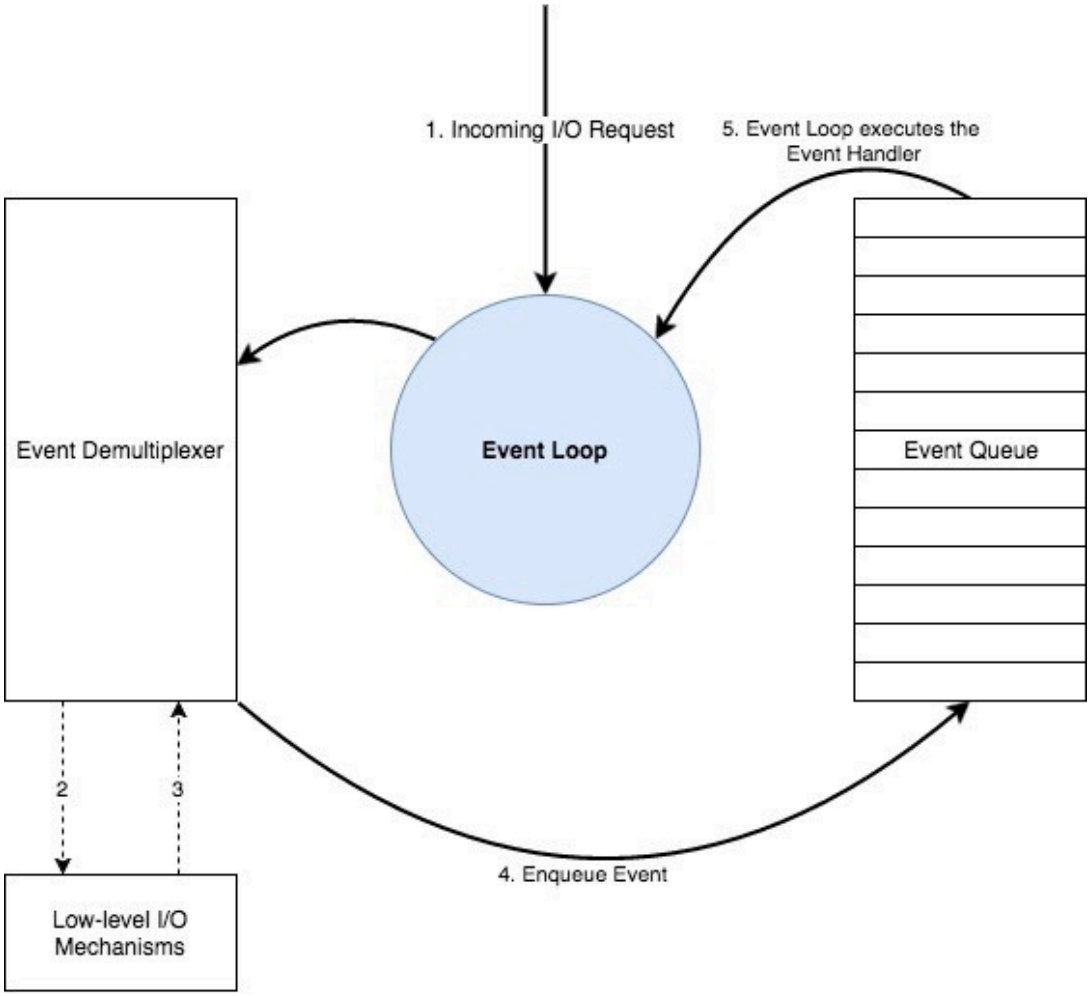
- How Nginx's worker accept connections?

  Multiple processces share a listening socket

- demo: a simple httpserver

Event loop's big picture

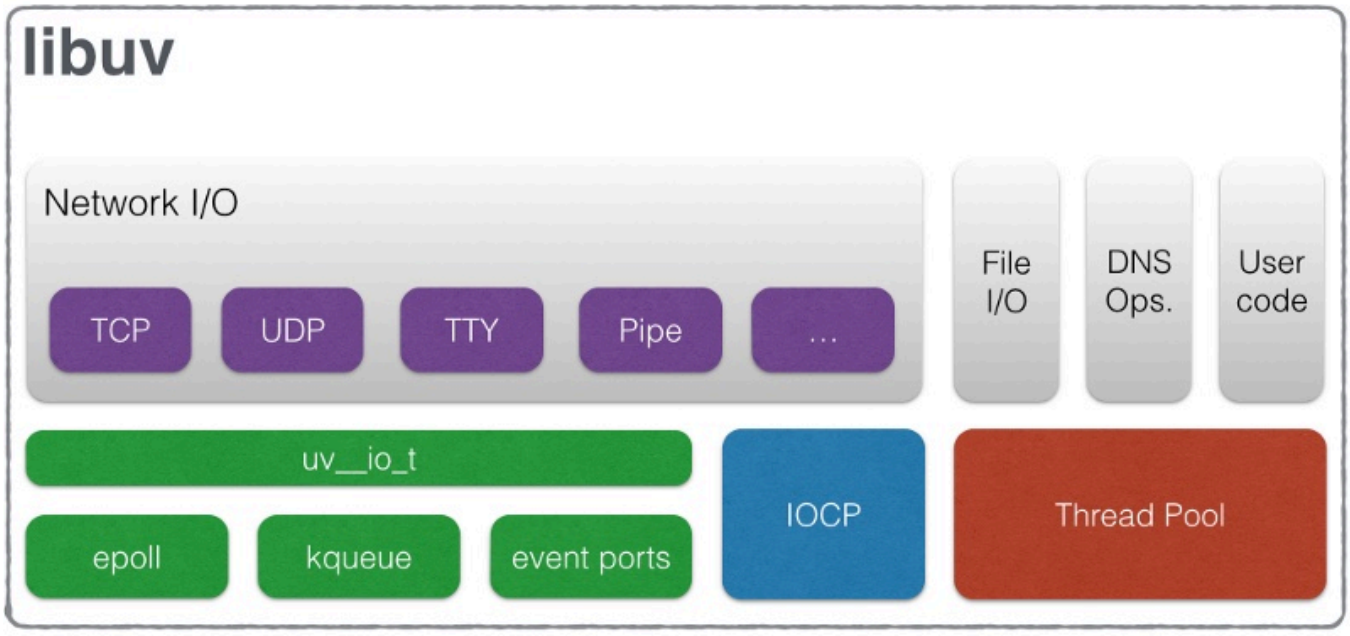
Search "event loop" on Google



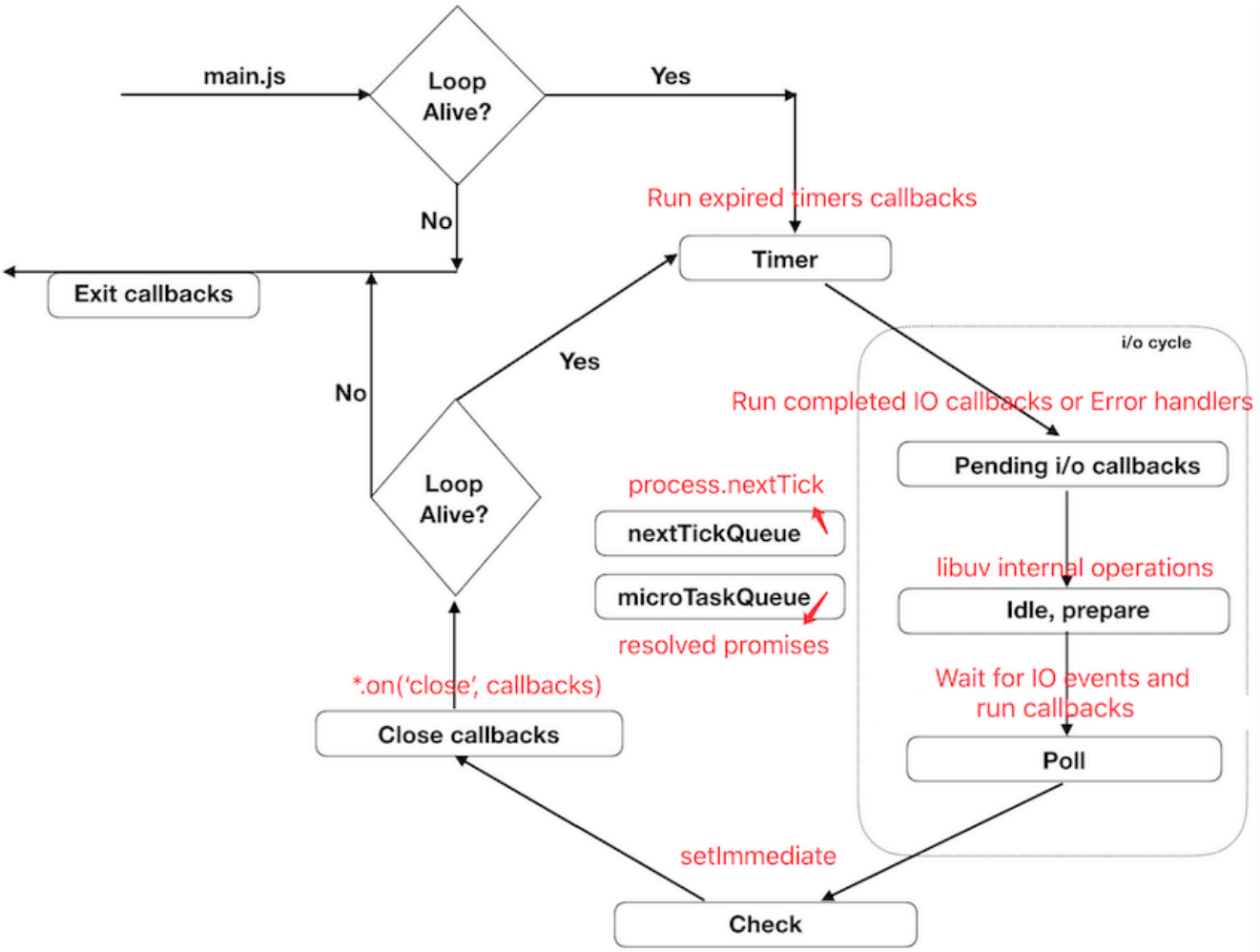
NodeJS event loop

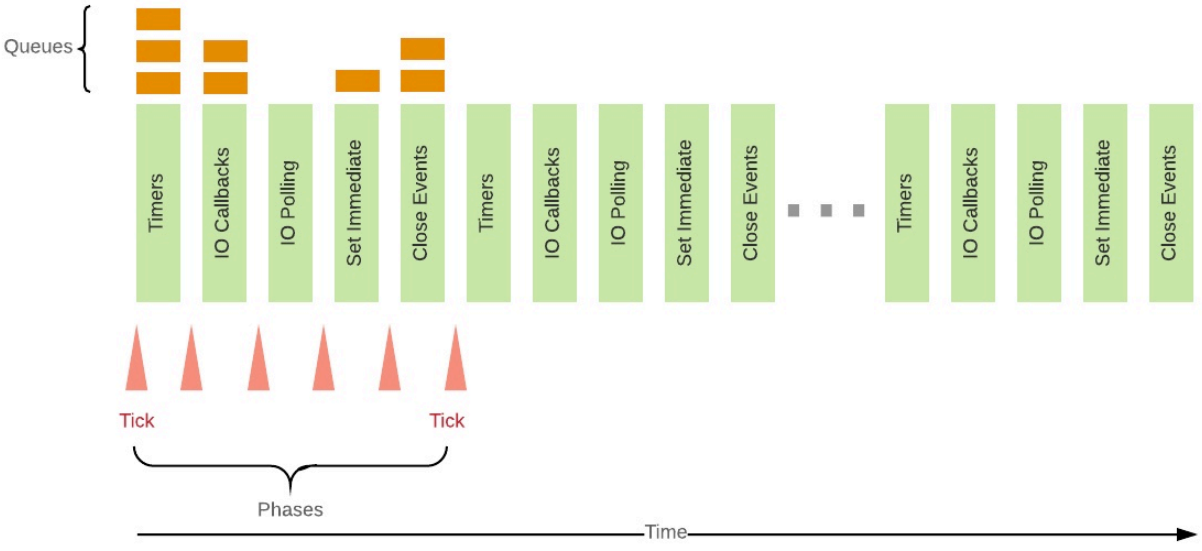
libuv and v8

- v8: For high-performance JavaScript evaluation
- libuv: For Event Loop with Asynchronous I/O



event loop





https://gist.github.com/trevnorris/1f3066ccb0fed9037afa

code snippets

loop apis

https://github.com/libuv/libuv/blob/fab6e64b39f162138fdf9a81765c05490f445f06/src/unix/core.c

```
r = uv__loop_alive(loop);
if (!r)
    uv__update_time(loop);

while (r != 0 && loop->stop_flag == 0) {
    uv__update_time(loop);
    uv__run_timers(loop);
    ran_pending = uv__run_pending(loop);
    uv__run_idle(loop);
    uv__run_prepare(loop);

    timeout = 0;
    if ((mode == UV_RUN_ONCE && !ran_pending) || mode == UV_RUN_DEFAULT)
      timeout = uv_backend_timeout(loop);

    uv__io_poll(loop, timeout);
    uv__run_check(loop);
    uv__run_closing_handles(loop);

    if (mode == UV_RUN_ONCE) {
      uv__update_time(loop);
      uv__run_timers(loop);
    }

    r = uv__loop_alive(loop);
    if (mode == UV_RUN_ONCE || mode == UV_RUN_NOWAIT)
      break;
}

static int uv__loop_alive(const uv_loop_t* loop) {
  return uv__has_active_handles(loop) ||
         uv__has_active_reqs(loop) ||
         loop->closing_handles != NULL;
}

static int uv__run_pending(uv_loop_t* loop) {
  QUEUE* q;
  QUEUE pq;
  uv__io_t* w;

  if (QUEUE_EMPTY(&loop->pending_queue))
    return 0;

  QUEUE_MOVE(&loop->pending_queue, &pq);

  while (!QUEUE_EMPTY(&pq)) {
    q = QUEUE_HEAD(&pq);
    QUEUE_REMOVE(q);
    QUEUE_INIT(q);
    w = QUEUE_DATA(q, uv__io_t, pending_queue);
    w->cb(loop, w, POLLOUT);
  }

  return 1;
}

```

```
57
58 int uv_backend_timeout(const uv_loop_t* loop) {
59   if (loop->stop_flag != 0)
60     return 0;
61
62   if (!uv__has_active_handles(loop) && !uv__has_active_reqs(loop))
63     return 0;
64
65   if (!QUEUE_EMPTY(&loop->idle_handles))
66     return 0;
67
68   if (!QUEUE_EMPTY(&loop->pending_queue))
69     return 0;
70
71   if (loop->closing_handles)
72     return 0;
73
74   return uv__next_timeout(loop);
75 }
76
77 int uv__next_timeout(const uv_loop_t* loop) {
78   const struct heap_node* heap_node;
79   const uv_timer_t* handle;
80   uint64_t diff;
81
82   heap_node = heap_min((const struct heap*) &loop->timer_heap);
83   if (heap_node == NULL)
84     return -1; /* block indefinitely */
85
86   handle = container_of(heap_node, uv_timer_t, heap_node);
87   if (handle->timeout <= loop->time)
88     return 0;
89
90   diff = handle->timeout - loop->time;
91   if (diff > INT_MAX)
92     diff = INT_MAX;
93
94   return diff;
95 }
96
97 void uv__io_poll(uv_loop_t* loop, int timeout) {
98
99   while (!QUEUE_EMPTY(&loop->watcher_queue)) {
100    q = QUEUE_HEAD(&loop->watcher_queue);
101    QUEUE_REMOVE(q);
102    QUEUE_INIT(q);
103
104    w = QUEUE_DATA(q, uv__io_t, watcher_queue);
105    assert(w->pevents != 0);
106    assert(w->fd >= 0);
107    assert(w->fd < (int) loop->nwatchers);
108
109    e.events = w->pevents;
110    e.data = w->fd;
111
112    if (w->events == 0)
```

```
      op = UV__EPOLL_CTL_ADD;
    else
      op = UV__EPOLL_CTL_MOD;

    /* XXX Future optimization: do EPOLL_CTL_MOD lazily if we stop watching
     * events, skip the syscall and squelch the events after epoll_wait().
     */
    if (uv__epoll_ctl(loop->backend_fd, op, w->fd, &e)) {
      if (errno != EEXIST)
        abort();

      assert(op == UV__EPOLL_CTL_ADD);

      /* We've reactivated a file descriptor that's been watched before. */
      if (uv__epoll_ctl(loop->backend_fd, UV__EPOLL_CTL_MOD, w->fd, &e))
        abort();
    }

    w->events = w->pevents;
  }

  for (;;) {
    if (no_epoll_wait || sigmask) {
      nfds = uv__epoll_pwait(loop->backend_fd,
                             events,
                             ARRAY_SIZE(events),
                             timeout,
                             sigmask);
    } else {
      nfds = uv__epoll_wait(loop->backend_fd,
                            events,
                            ARRAY_SIZE(events),
                            timeout);
      if (nfds == -1 && errno == ENOSYS) {
        no_epoll_wait = 1;
        continue;
      }
    }

    for (i = 0; i < nfds; i++) {
      pe = events + i;
      fd = pe->data;

      /* Skip invalidated events, see uv__platform_invalidate_fd */
      if (fd == -1)
        continue;

      assert(fd >= 0);
      assert((unsigned) fd < loop->nwatchers);

      w = loop->watchers[fd];

      if (w == NULL) {
        uv__epoll_ctl(loop->backend_fd, UV__EPOLL_CTL_DEL, fd, pe);
        continue;
      }
```

```
169
170       pe->events &= w->pevents | UV__POLLERR | UV__POLLHUP;
171       if (pe->events == UV__EPOLLERR || pe->events == UV__EPOLLHUP)
172         pe->events |= w->pevents & (UV__EPOLLIN | UV__EPOLLOUT);
173
174       if (pe->events != 0) {
175         w->cb(loop, w, pe->events);
176         nevents++;
177       }
178     }
179     loop->watchers[loop->nwatchers] = NULL;
180     loop->watchers[loop->nwatchers + 1] = NULL;
181
182 }
```

Questions?

- What's difference between pending IO callbacks and callbacks in poll phase?
- What kind of callback will be push into pending queue?

```
1  Note: uv__io_feed() is the only function to insert onto pending_queue.
2
3  Note: The following use uv__io_feed():
4
5  - uv_pipe_connect(), but only in the case of an error.
6
7  - uv__write_req_finish(), part of stream.c
8
9  - uv_tcp_connect(), but only in the case of an error.
10
11 - uv__udp_send_msg(), for all sent messages.
```

Best references

- Morning Keynote- Everything You Need to Know About Node.js Event Loop - Bert Belder, IBM
- NodeConf EU | A deep dive into libuv - Saul Ibarra Coretge
- Handling IO — NodeJS Event Loop Part 4
- Node.js event loop workflow & lifecycle in low level
- http://docs.libuv.org/en/v1.x/loop.html
- libuv design overview