A node.js like api in C++

Stefano Cristiano - R&D Director

RECOGNITION ROBOTICS

The Visual Guidance Company

Thanks to the sponsors!

Summary

- Our History
- What are we building
- Why C++ and why node.js
- IO Concurrency
- Event Loop
- Event Loop Integration
- From Node.JS to rrNode
- Examples
- Under the hood

Our History

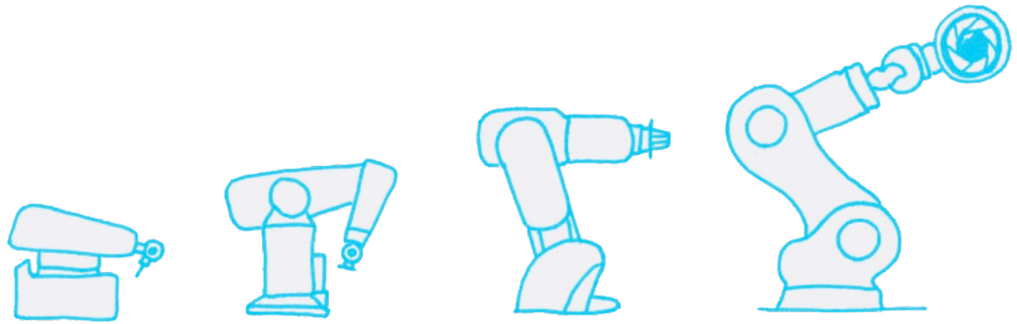
QUICK INTRODUCTION TO RECOGNITION ROBOTICS

Company Mission

Create and provide best robotic guidance products and solutions for automation

History – Recognition Robotics

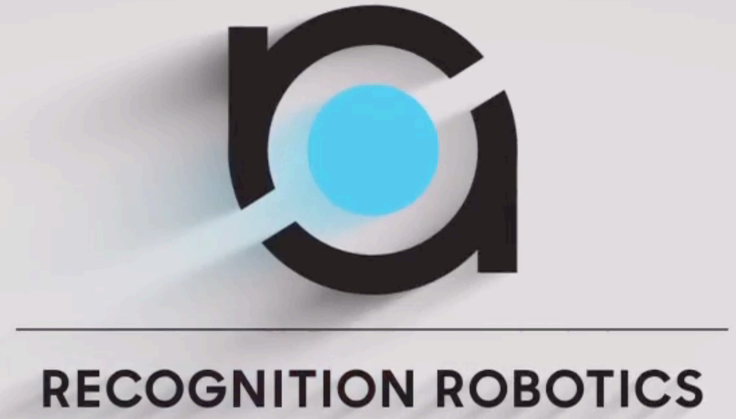
Founded in 2008 to revolutionize machine vision technology with the ultimate goal of advancing toward autonomous, intelligent visual recognition and production.



Visual Guidance. Evolved

WHAT ARE WE BUILDING?

BREAKTHROUGH VISUAL GUIDANCE SOFTWARE

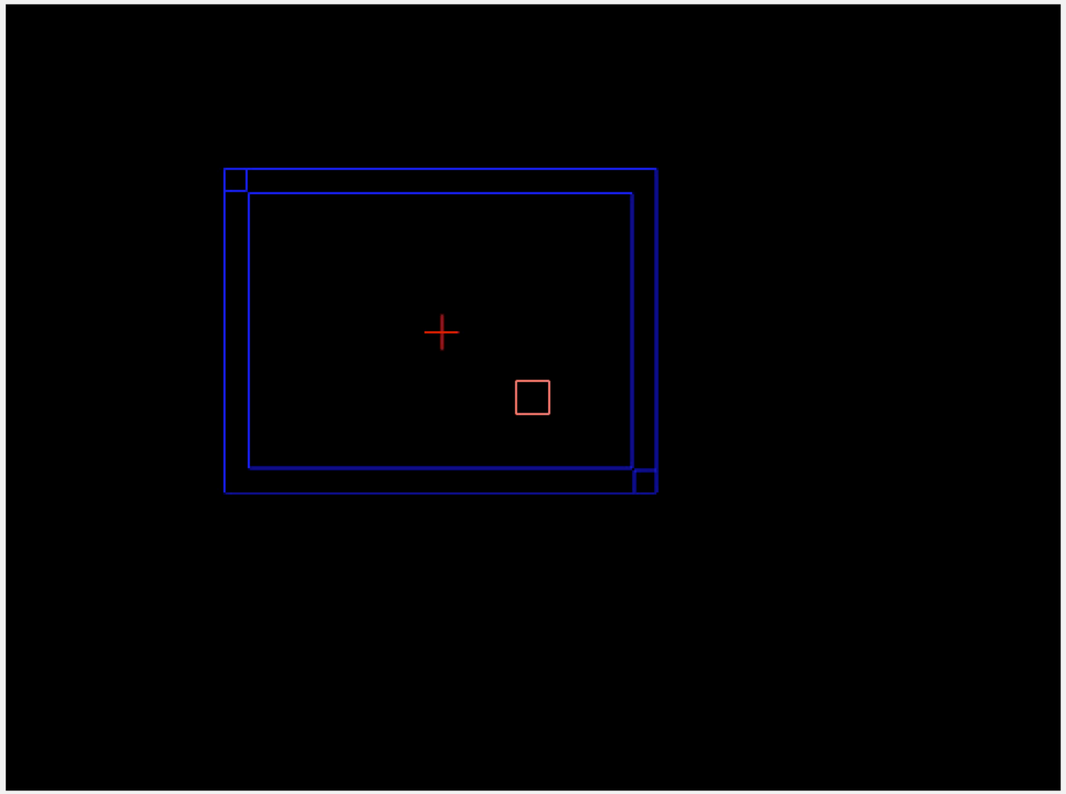

RECOGNITION ROBOTICS

What has all of this to do with C++?

A LOT. REALLY.

CortexRecognition - C:\Cortex\CortexSearch_v4_104-2013\Data\CTX\stefano_glasses.ctx

Log Book | User Login | Smaller Eraser | Bigger Eraser | Camera | Focus | Normal | Exposure | Auto



RECOGNITION ROBOTICS
The Visual Guidance Company

r beye
powered by cortex recognition

VERSION 5.179.0 - Built: Apr 5 2017 at 23:33:55

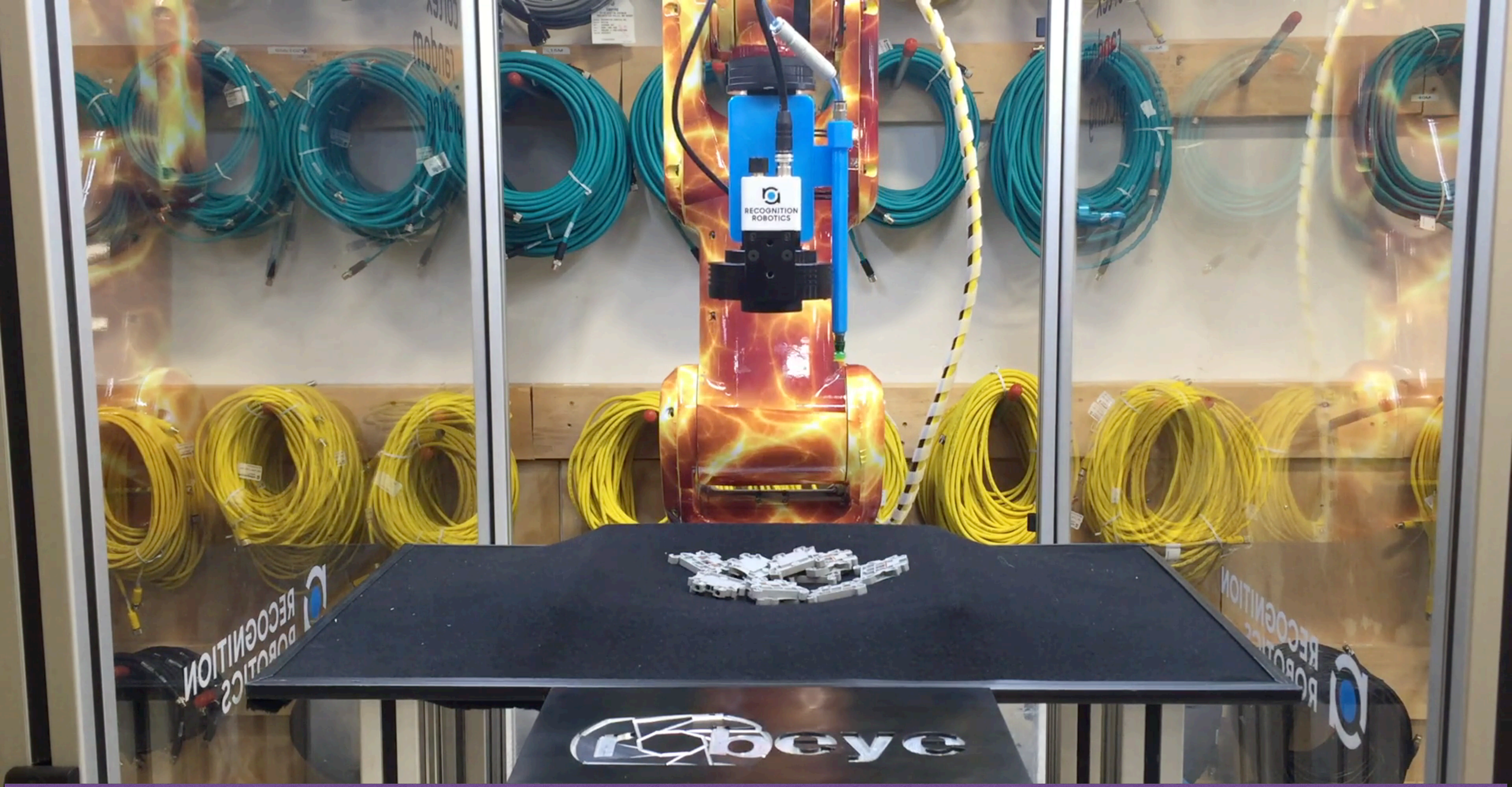
Image Acquisition
Live Image | Load Image | Load Image Sequence | Save Image

Visual Recognition
Teach | Recognize | Continuous Recognition | Recognize Once

Options and Settings

Guidance Data
Match Score: 0 | X : +0.00 mm

THIS IS BUILT 100% IN C++



THIS IS BUILT 100% IN C++

What is in these products

- Computer Vision Algorithms
- Networking protocols
- Robotics simulation in 3D
- Access to third party hardware (cameras, lasers, robots etc.)
- Non trivial User Interfaces
- Remote access to UI
- Extensive Data logging (database)

Our necessities

- Implement fast computer vision algorithms
- Run on embedded devices (low overhead)
- Interface legacy C code / SDK
- Run Multi-Platform
- Create responsive desktop user Interfaces
- Create responsive remote user interfaces
- Implement networking protocols

TODAY'S topic

Why C++ and Why Node.JS

https://pagghiu.github.io/dev/A-Node-Like-Api-for-C++-en/

- Explains some of our reasons why we've gone 100% C++
- Previously our technology stack was:
  - User Interface: HTML/JS
  - Middle layer: C# (.NET)
  - Lower layer: C++ (mainly image processing)
- Main reason we've been switching:
  - Bugs in Mono Framework (back in 2013)
  - Unneeded complexity
  - Difficult to debug end to end
  - Somewhat easy to build inefficient solutions

What is NodeJS

Asynchronous I/O framework using:

- Chrome Javascript engine (V8)
- Cross platform Evented I/O library (libuv)

Used for:

- Mainly server side stuff / backend applications
- Emerging use in Automation and Robotics
- Sometimes using you would never guess

What is libuv

libuv is a multi-platform support library with a focus on asynchronous I/O. It was primarily developed for use by Node.js, but it's also used by Luvit, Julia, pyuv, and others.

- Full-featured event loop backed by epoll, kqueue, IOCP, event ports.
- Asynchronous TCP and UDP sockets
- Asynchronous DNS resolution
- Asynchronous file and file system operations
- File system events
- ANSI escape code controlled TTY
- IPC with socket sharing, using Unix domain sockets or named pipes (Windows)
- Child processes
- Thread pool
- Signal handling
- High resolution clock
- Threading and synchronization primitives

libuv

https://libuv.org

I/O Concurrency

OR WHY WE SHOULD NEVER BLOCK THE MAIN THREAD

Our apps typical latencies

- Regular industrial camera image acquisition (10ms - 30ms)
- Image processing (10ms - 3000ms)
- Networking (anything from 1ms to some seconds, and depending on clients connected)

Latency numbers every programmer should know

```
Latency Comparison Numbers
--------------------------
L1 cache reference                           0.5 ns
Branch mispredict                            5   ns
L2 cache reference                           7   ns
14x L1 cache
Mutex lock/unlock                           25   ns
Main memory reference                      100   ns
20x L2 cache, 200x L1 cache
Compress 1K bytes with Zippy             3,000   ns        3 us
Send 1K bytes over 1 Gbps network       10,000   ns       10 us
Read 4K randomly from SSD*             150,000   ns      150 us
~1GB/sec SSD
Read 1 MB sequentially from memory     250,000   ns      250 us
Round trip within same datacenter      500,000   ns      500 us
Read 1 MB sequentially from SSD*     1,000,000   ns    1,000 us    1 ms
~1GB/sec SSD, 4X memory
Disk seek                           10,000,000   ns   10,000 us   10 ms
20x datacenter roundtrip
Read 1 MB sequentially from disk    20,000,000   ns   20,000 us   20 ms
80x memory, 20X SSD
Send packet CA->Netherlands->CA    150,000,000   ns  150,000 us  150 ms
```

https://gist.github.com/jboner/2841832

Latency numbers every programmer should know

Latency Comparison Numbers						
L1 cache reference	0.5	ns			1	sec
Branch mispredict	5	ns			10	sec
L2 cache reference 14x L1 cache	7	ns			14	sec
Mutex lock/unlock	25	ns			50	sec
Main memory reference 20x L2 cache, 200x L1 cache	100	ns			3:20	min
Compress 1K bytes with Zippy	3,000	ns	3 us		1:40:00	hr
Send 1K bytes over 1 Gbps network	10,000	ns	10 us		5:33:20	hr
Read 4K randomly from SSD* ~1GB/sec SSD	150,000	ns	150 us		3	days
Read 1 MB sequentially from memory	250,000	ns	250 us		5	days
Round trip within same datacenter	500,000	ns	500 us		11	days
Read 1 MB sequentially from SSD* ~1GB/sec SSD, 4X memory	1,000,000	ns	1,000 us	1 ms	23	days
Disk seek 20x datacenter roundtrip	10,000,000	ns	10,000 us	10 ms	231	days
Read 1 MB sequentially from disk 80x memory, 20X SSD	20,000,000	ns	20,000 us	20 ms	462	days
Send packet CA->Netherlands->CA	150,000,000	ns	150,000 us	150 ms	3472	days

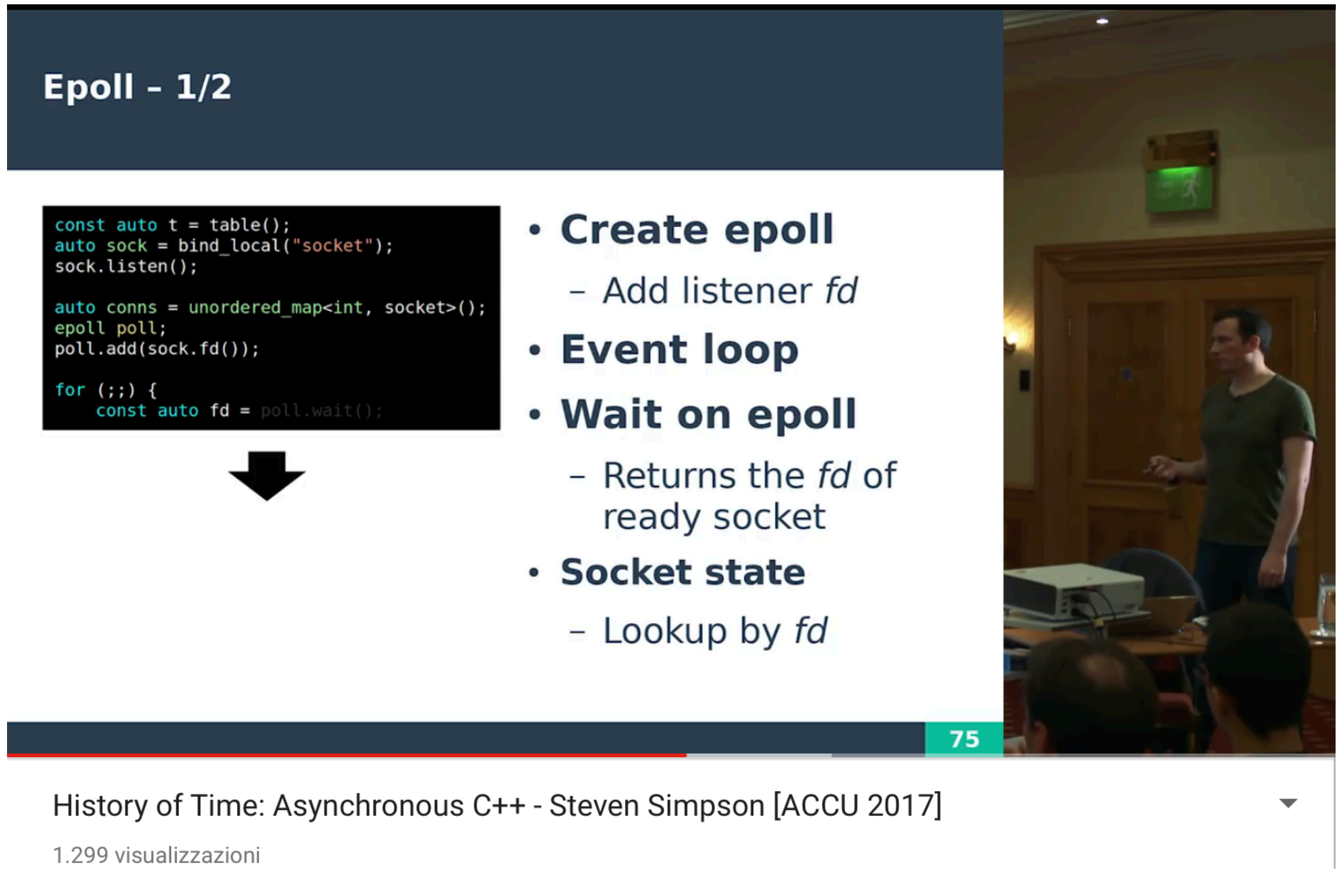
Normalized to 1 sec to get order of magnitude

https://gist.github.com/jboner/2841832

Considerations

- I/O operations are slow
- Potentially long time waiting for a response
- Displaying UI@60 FPS yields a 13ms budget for every frame
- I/O operations in the main thread of an user interface will drop FPS to unacceptable levels
- Must resort to background threads
- Must resort to mutexes for started data race synchronization
- Notoriously hard and error prone

More Info: Asynchronous C++ in networking talk



https://youtu.be/Z8tbjyZFAVQ

- Explains concurrency and event loop networking with callbacks
- Compares with concurrency through threading
- Explains the subtle PRO and CONS of each approach
- Superb talk, recommended

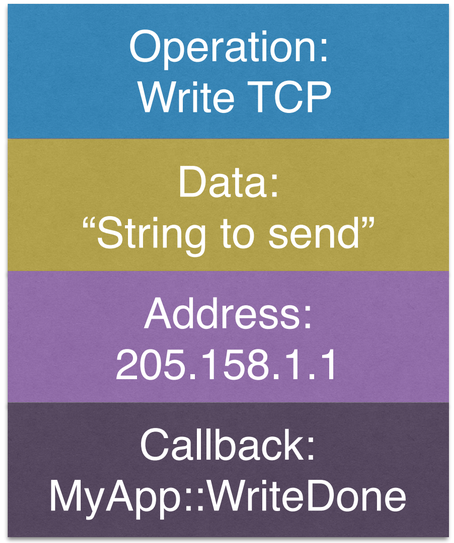
The Event loop

CALLBACKS, CALLBACKS EVERYWHERE

How does an I/O Event loop work - Creation

The application creates a request with:

- Specification of what's needed
  - Example: what file to write
  - Example: what IP address to read from using TCP
- Some meaningful action to be executed after the request will be dispatched
  - Most of the time a callback

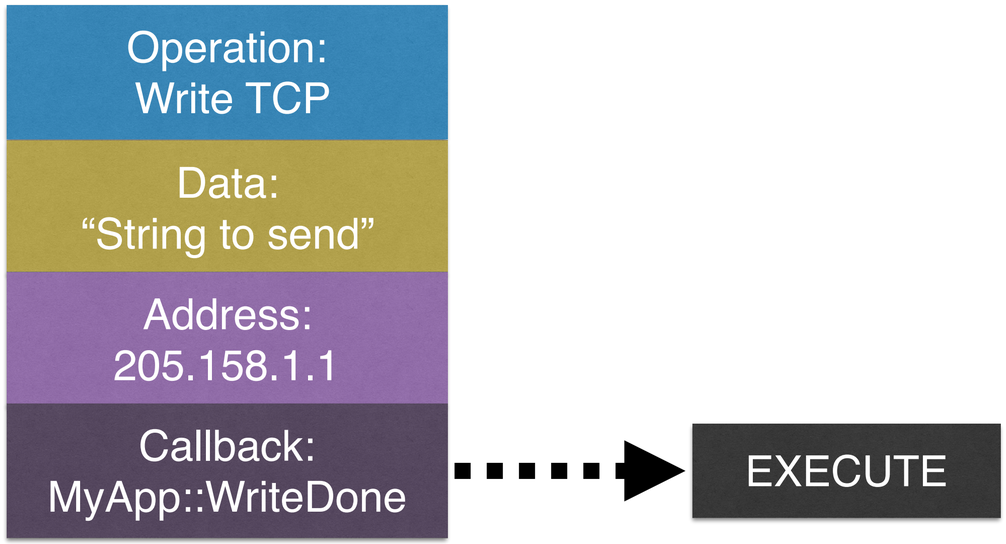


How does an I/O Event loop work - Queuing

GetOverlappedIO
select
epoll
kqueue

- The application:
  - Registers this request into a queue handled by the Operating System
  - Calls a blocking OS kernel function effectively waiting for any of the registered requests to be handled

The I/O Event Loop - callbacks



- Operating System will release/ unblock that blocking call when "something" happens to the list of requests
- The application will then check what request has been handled
- The application will execute callbacks associated with the handled request

The I/O event loop - Advantages

- Simplicity: handle concurrency of many I/O operations within a single thread
- Performance: OS mechanism are highly optimized for handling I/O concurrency
- Handling I/O through an event loop is commonly referred as asynchronous I/O

Event loop integration

HOW DO GUI AND I/O EVENT LOOP TALK TO EACH OTHER

User Interface event loop

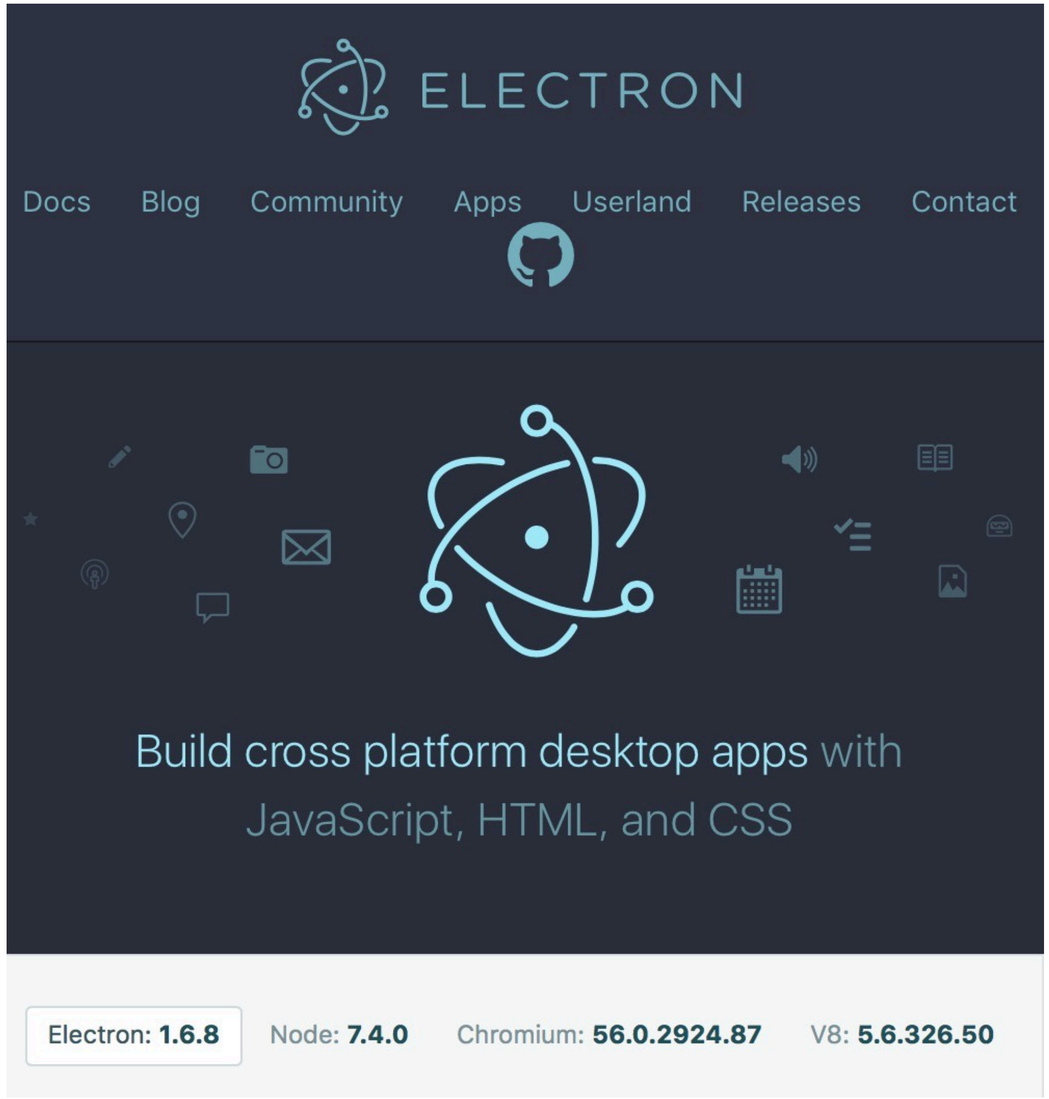
- Most Native UI systems use event loops
- User Program is blocked on a kernel call until "something happens"
- Example:
  - Mouse Move / Click
  - Keyboard event
  - Lost focus
  - etc.
- Details are buried inside implementations of big UI frameworks (Qt, WPF, MFC, Cocoa, GTK or web browsers etc.)

How to Integrate GUI and IO Event Loops

- Run the two event loops in two different threads
  - One of the two event loops will be the main (tipically the GUI) and the other will be secondary (I/O)
  - Pull for events availability in the secondary thread
  - Send a message to the main thread to unblock it
  - The main event loop is unblocked and will process I/O events in its thread

Real world example

- Electron (https://electron.atom.io) is a framework for creating native applications with web tecnologies (Javascript, HTML and CSS)
- Integrates Chromium Event loop (UI) and libuv event loop (I/O)
- https://electron.atom.io/blog/2016/07/28/electron-internals-node-integration



Electron and the event loop - The Secondary I/O Thread

- https://github.com/electron/electron/blob/master/atom/common/node_bindings.cc

```
void NodeBindings::EmbedThreadRunner(void *arg) {
  NodeBindings* self = static_cast<NodeBindings*>(arg);

  while (true) {
    // Wait for the main loop to deal with events.
    uv_sem_wait(&self->embed_sem_);
    if (self->embed_closed_)
      break;

    // Wait for something to happen in uv loop.
    // Note that the PollEvents() is implemented by derived classes, so when
    // this class is being destructed the PollEvents() would not be available
    // anymore. Because of it we must make sure we only invoke PollEvents()
    // when this class is alive.
    self->PollEvents();
    if (self->embed_closed_)
      break;

    // Deal with event in main thread.
    self->WakeupMainThread();
  }
```

Electron and the event loop - Polling Events on Windows

- https://github.com/electron/electron/blob/master/atom/common/node_bindings_win.cc

```
void NodeBindingsWin::PollEvents() {
  // If there are other kinds of events pending, uv_backend_timeout will
  // instruct us not to wait.
  DWORD bytes, timeout;
  ULONG_PTR key;
  OVERLAPPED* overlapped;

  timeout = uv_backend_timeout(uv_loop_);

  GetQueuedCompletionStatus(uv_loop_->iocp,
                            &bytes,
                            &key,
                            &overlapped,
                            timeout);

  // Give the event back so libuv can deal with it.
  if (overlapped != NULL)
    PostQueuedCompletionStatus(uv_loop_->iocp,
                               bytes,
                               key,
                               overlapped);
}
```

Electron and the event loop - Polling Events on Linux

- https://github.com/electron/electron/blob/master/atom/common/node_bindings_linux.cc

```
void NodeBindingsLinux::PollEvents() {
  int timeout = uv_backend_timeout(uv_loop_);

  // Wait for new libuv events.
  int r;
  do {
    struct epoll_event ev;
    r = epoll_wait(epoll_, &ev, 1, timeout);
  } while (r == -1 && errno == EINTR);
}
```

Electron and the event loop - Polling Events on macOS

- https://github.com/electron/electron/blob/master/atom/common/node_bindings_linux.cc

```
void NodeBindingsMac::PollEvents() {
  struct timeval tv;
  int timeout = uv_backend_timeout(uv_loop_);
  if (timeout != -1) {
    tv.tv_sec = timeout / 1000;
    tv.tv_usec = (timeout % 1000) * 1000;
  }

  fd_set readset;
  int fd = uv_backend_fd(uv_loop_);
  FD_ZERO(&readset);
  FD_SET(fd, &readset);

  // Wait for new libuv events.
  int r;
  do {
    r = select(fd + 1, &readset, nullptr, nullptr,
               timeout == -1 ? nullptr : &tv);
  } while (r == -1 && errno == EINTR);
}
```

Electron and the event loop - Waking up main thread and run loop

https://github.com/electron/electron/blob/master/atom/common/node_bindings.cc

```
void NodeBindings::UvRunOnce() {
  node::Environment* env = uv_env();

  // OMISSIS…CHROME SPECIFIC STUFF

  // Deal with uv events.
  int r = uv_run(uv_loop_, UV_RUN_NOWAIT);

  // OMISSIS…CHROME SPECIFIC STUFF

  if (r == 0)
    base::RunLoop().QuitWhenIdle();  // Quit from uv.

  // Tell the worker thread to continue polling.
  uv_sem_post(&embed_sem_);
}

void NodeBindings::WakeupMainThread() {
  DCHECK(task_runner_);
  task_runner_->PostTask(FROM_HERE, base::Bind(&NodeBindings::UvRunOnce,
                                               weak_factory_.GetWeakPtr()));
}
```

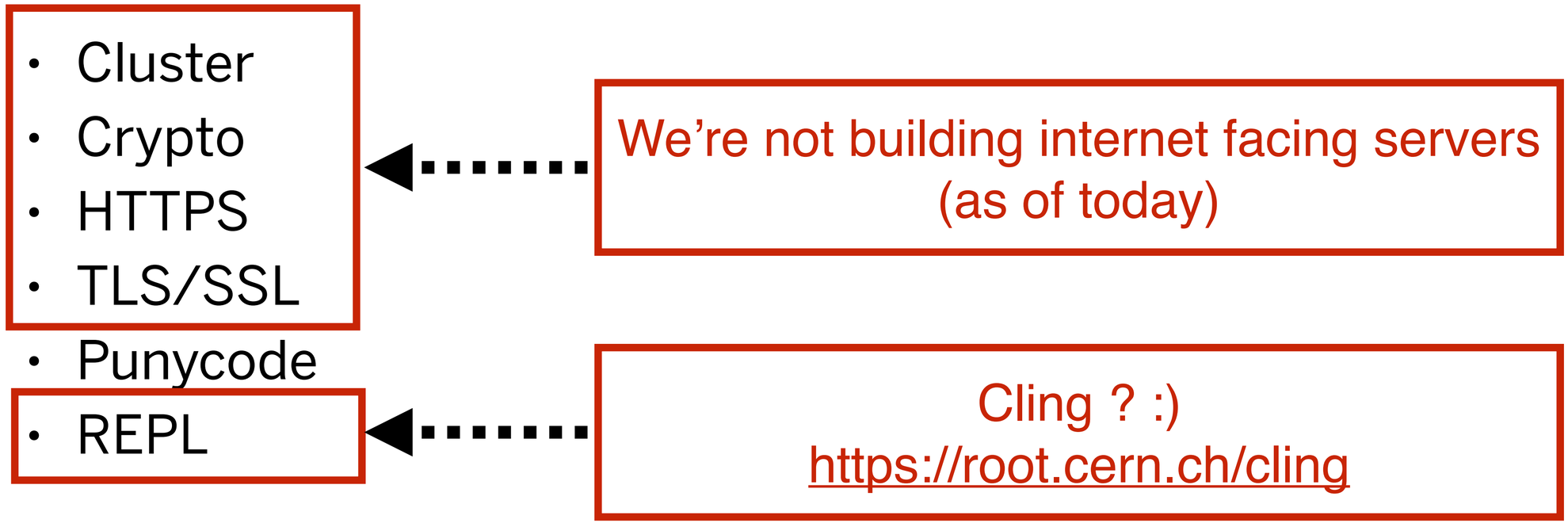
From Node.JS to rrNode

JAVASCRIPT TO C++

Node.JS core modules implemented in the Library

- Buffer
- Child Processes
- Console
- DNS
- Errors
- Events
- File System
- HTTP
- Net
- OS
- Path
- Process
- Query Strings
- Readline
- Stream
- Timers
- TTY
- UDP/Datagram

What is missing



Implemented third party NPM modules

- Websockets
- Serial (RS232)
- Modbus
- Redis client
- SQLite

- It's easy to lookup NPM for some library and use it as inspiration for third party modules

Examples

HOW DOES THE API LOOK LIKE

Examples disclaimers

- Written to fit into a slide not to be clear
- Follow node.js API as much as possible
- “Callback Hell” style
- Production code is more verbose but also more clear
- We desperately need C++ Coroutines in the standard :-|

NodeJS TCP echo server

```
var net = require('net');

var port = 7000;
var server = net.createServer();
server.on('connection', function (socket)
{
  socket.on('data', function(data)
  {
      var value = data.toString('utf8');
      if(value.startsWith("quit"))
      {
          socket.end();
          server.close();
      }
      else
      {
        socket.write(data);
      }
  });
});
server.listen(port,
function(){console.log('Listening at 127.0.0.1:'+port);});
```

rrNode TCP echo server

```
#include "../modules/rrCore/rrCore.h"
#include "../modules/rrNode/rrNode.h"

int main(int argc, const char * argv[])
{
    using namespace rrNode;
    process::setCommandLine(argc, argv);
    loop mainLoop;
    auto server = net::createServer();
    server->onConnect += [=](auto socket) mutable
    {
        socket->onData += [=](auto data) mutable
        {
            if(data.view().startsWith("quit"))
            {
                socket.end();
                server.close();
            }
            else
            {
                socket.write(data);
            }
        };
    };
    const int port = 7000;
    server.listen(port,
    [=]{console::log("Listening at 127.0.0.1:%d",port);});
    mainLoop.run();
    return 0;
}
```

NodeJS echo server - streams

```
var net = require('net');

var port = 7000;
var server = net.createServer();
server.on('connection', function (socket)
{
  socket.pipe(socket);
});
server.listen(port,
function(){console.log('Listening at 127.0.0.1:'+port);});
```

rrNode echo server - streams

```
#include "../modules/rrCore/rrCore.h"
#include "../modules/rrNode/rrNode.h"

int main(int argc, const char * argv[])
{
    using namespace rrNode;
    process::setCommandLine(argc, argv);
    loop mainLoop;
    auto server = net::createServer();
    server->onConnect += [=](auto socket) mutable
    {
        socket.pipe(socket);
    };
    const int port = 7000;
    server.listen(port,
    [=]{console::log("Listening at 127.0.0.1:%d", port);});
    mainLoop.run();
    return 0;
}
```

Writing a TCP echo server in libuv

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "../modules/rrNode/external/libuv/uv.h"
#include "../modules/rrNode/external/external_libuv.c"


uv_loop_t *loop;
struct sockaddr_in addr;

typedef struct {
    uv_write_t req;
    uv_buf_t buf;
} write_req_t;

void free_write_req(uv_write_t *req) {
    write_req_t *wr = (write_req_t*) req;
    free(wr->buf.base);
    free(wr);
}

void alloc_buffer(uv_handle_t *handle, size_t suggested_size, uv_buf_t *buf) {
    buf->base = (char*) malloc(suggested_size);
    buf->len = suggested_size;
}

void echo_write(uv_write_t *req, int status) {
    if (status) {
        fprintf(stderr, "Write error %s\n", uv_strerror(status));
    }
    free_write_req(req);
}

void echo_read(uv_stream_t *client, ssize_t nread, const uv_buf_t *buf) {
    if (nread > 0) {
        write_req_t *req = (write_req_t*) malloc(sizeof(write_req_t));
        req->buf = uv_buf_init(buf->base, (unsigned int)nread);
        uv_write((uv_write_t*) req, client, &req->buf, 1, echo_write);
        return;
    }
    if (nread < 0) {
        if (nread != UV_EOF)
            fprintf(stderr, "Read error %s\n", uv_err_name((int)nread));
        uv_close((uv_handle_t*) client, NULL);
    }

    free(buf->base);
}

void on_new_connection(uv_stream_t *server, int status) {
    if (status < 0) {
        fprintf(stderr, "New connection error %s\n", uv_strerror(status));
        // error!
        return;
    }

    uv_tcp_t *client = (uv_tcp_t*) malloc(sizeof(uv_tcp_t));
    uv_tcp_init(loop, client);
    if (uv_accept(server, (uv_stream_t*) client) == 0) {
        uv_read_start((uv_stream_t*) client, alloc_buffer, echo_read);
    }
    else {
        uv_close((uv_handle_t*) client, NULL);
    }
}

int main() {
    loop = uv_default_loop();

    uv_tcp_t server;
    uv_tcp_init(loop, &server);

    uv_ip4_addr("0.0.0.0", 7000, &addr);

    uv_tcp_bind(&server, (const struct sockaddr*)&addr, 0);
    int r = uv_listen((uv_stream_t*) &server, 128, on_new_connection);
    if (r) {
        fprintf(stderr, "Listen error %s\n", uv_strerror(r));
        return 1;
    }
    return uv_run(loop, UV_RUN_DEFAULT);
}
```

libuv

RECOGNITION
ROBOTICS

The Visual Guidance Company

NodeJS http client

```
var http = require('http');

http.get('http://google.com', (res) => {
    console.log("Method : " + res.req.method);
    console.log("StatusCode:" + res.statusCode);
    console.log("Location:" +
      res.headers["location"]);
    console.log("Content-type: " +
      res.headers["content-type"]);
    var str = ''
    res.on('data', function (chunk) { str += chunk; });
    res.on('end', function () { console.log(str); });
  });
```

rrNode http client

```
#include <memory>
#include "../modules/rrCore/rrCore.h"
#include "../modules/rrNode/rrNode.h"

int main(int argc, const char * argv[])
{
    using namespace rrNode;
    process::setCommandLine(argc, argv);
    loop mainLoop;
    http::GET("http://www.google.com", [](auto res)
    {
        console::log("Method: %s", res.method());
        console::log("StatusCode: %d", res.statusCode());
        console::log("Location: %s",res.header("location"));
        console::log("Content-type: %s",
            res.header("content-type"));
        auto str = std::make_shared<string>();
        res->onData += [=](auto chunk)mutable
        {
            str->append(chunk.view());
        };
        res->onEnd += [=](){ console::log(*str); };
    });
    mainLoop.run();
    return 0;
}
```

NodeJS http server - readFile based

```
http.createServer(requestHandler).listen(3000);
function requestHandler(req, res)
{
  var filePath = __dirname;
  filePath += req.url == "/" ? "/index.html" : req.url;
  if(req.method == "GET")
  {
    var ext = path.extname(filePath);
    var mime = ext ? ext.slice(1) : 'html';
    fs.readFile(filePath, function(err, fileData)
    {
      if(!err)
      {
        res.writeHead(200, {'Content-Type': mimeTypes[mime]});
        res.write(fileData);
      }
      else
      {
        res.writeHead(404, {'Content-Type': 'text/html'});
        res.write("<h1>File not found</h1>");
      };
      res.end();
    });
  }
  else
  {
    res.writeHead(405, {'Content-Type': 'text/html'});
    res.end('<h1>Method not allowed</h1>');
  };
}
```

```
var mimeTypes = {
      "html": "text/html",
      "htm": "text/html",
      "jpeg": "image/jpeg",
      "jpg": "image/jpeg",
      "png": "image/png",
      "js": "text/javascript",
      "css": "text/css"
    };
```

RECOGNITION
ROBOTICS
The Visual Guidance Company

rrNode http server - readFile based

```
int main(int argc, const char * argv[])
{
    using namespace rrNode;
    process::setCommandLine(argc, argv);
    loop defaultLoop;
    auto server = http::Server::create().listen(3000);
    server->onRequest += [=](auto data)
    {
        string filePath = path::dirname(process::execPath);
        filePath += data.request.url() == "/" ? "/index.html" : data.request.url();
        identifier mime(path::extname(filePath).isEmpty() ? "html" : path::extname(filePath).slice(1));
        fs::readFile(filePath, [=,res=data.response](auto fileData) mutable
        {
            if(fileData.isValid())
            {
                res.writeHead(200, arr$("Content-Type", http::mimeTypes(mime)));
                res.write(fileData);
            }
            else
            {
                res.writeHead(404, arr$("Content-Type", "text/html"));
                res.write("<h1>File not found</h1>");
            }
            res.end();
        });
    };
    defaultLoop.run();
    return 0;
}
```

NodeJS http server - stream based

```
var http = require('http'),
path = require('path'),
fs = require('fs');
var mimeTypes = {
      "html": "text/html",
      "htm": "text/html",
      "jpeg": "image/jpeg",
      "jpg": "image/jpeg",
      "png": "image/png",
      "js": "text/javascript",
      "css": "text/css"
    };

http.createServer(requestHandler).listen(3000);
function requestHandler(req, res)
{
  var filePath = __dirname;
  filePath += req.url == "/" ? "/index.html" : req.url;
  var ext = path.extname(filePath);
  var mime = ext ? ext.slice(1) : 'html';
  var fileReadStream = fs.createReadStream(filePath);
  res.writeHead(200,{"Content-type" : mimeTypes[mime]});
  fileReadStream.on('error',function(err)
  {
    res.writeHead(404,{"Content-type" : 'text/html'});
    res.end("<h1>File not found</h1>");
  });
  fileReadStream.pipe(res);
}
```

rrNode http server - stream based

```
#include "../modules/rrCore/rrCore.h"
#include "../modules/rrNode/rrNode.h"

int main(int argc, const char * argv[])
{
    using namespace rrNode;
    process::setCommandLine(argc, argv);
    loop defaultLoop;
    auto server = http::Server::create().listen(3000);
    server->onRequest += [=](auto data)
    {
        string filePath = path::dirname(process::execPath);
        filePath += data.request.url() == "/" ? "/index.html" : data.request.url();
        identifier mime(path::extname(filePath).isEmpty() ? "html" : path::extname(filePath).slice(1));
        auto fileReadStream = fs::createReadStream(filePath);
        data.response.writeHead(200, arr$("Content-Type", http::mimeTypes(mime)));
        fileReadStream->onError += [response=data.response](error) mutable
        {
            // We end up here if the file doesn't exist
            response.writeHead(404, arr$("Content-Type", "text/html"));
            response.end("<h1>File not found</h1>");
        };
        data.response->onUnpipe += bindMem(&fs::readStream::close, fileReadStream);
        fileReadStream->pipe(data.response);
    };
    return defaultLoop.run();
}
```

NodeJS websocket client

```
const WebSocket = require('ws');
const http = require('http');
const server = http.createServer().listen(3000);
const websocketServer = new WebSocket.Server({ server });

websocketServer.on('connection', function connection(ws)
{
  ws.on('message', function incoming(message)
  {
    ws.send('node.js echoing "' + message + '"');
    if(message == "quit")
    {
      ws.close();
      websocketServer.close();
    }
  });
});
```

rrNode websocket client

```
#include "../modules/rrCore/rrCore.h"
#include "../modules/rrNode/rrNode.h"
int main(int argc, const char * argv[])
{
    using namespace rrNode;
    using namespace http;
    process::setCommandLine(argc, argv);
    loop defaultLoop;
    Server websocketServer = http::Server::create().listen(3000);
    websocketServer->onUpgrade += [=](Server::data data) mutable
    {
        auto req = data.request;
        auto sock = data.socket;
        auto ws = websocket::createFromUpgradeRequest(req, sock);
        ws->onMessage += [=](auto msg) mutable
        {
            ws.send("rrNode echoing \"%s\"", msg);
            if(msg == "quit")
            {
                ws.close();
                websocketServer.close();
            }
        };
    };
    return defaultLoop.run();
}
```

NodeJS child-process

```
const childProcess = require('child_process');
const process = require('process');

var opts = {stdio:["pipe","pipe",null]};
var proc = childProcess.spawn("uname", ["-a"], opts);
proc.stdout.pipe(process.stdout);
proc.stderr.pipe(process.stderr);
```

rrNode childProcess

```
#include "../modules/rrCore/rrCore.h"
#include "../modules/rrNode/rrNode.h"

int main(int argc, const char * argv[])
{
    using namespace rrNode;
    process::setCommandLine(argc, argv);
    loop defaultLoop;
    defaultLoop.initStandardIO();
    auto opts = spawnOptions().pipe("stdout").pipe("stderr");
    auto proc = childProcess::spawn("uname", arr$("-a"), opts);
    proc.Stdout().pipe(process::Stdout);
    proc.Stderr().pipe(process::Stderr);
    return defaultLoop.run();
}
```

NodeJS transform stream

```
const childProcess = require('child_process');
const process = require('process');
const Transform = require('stream').Transform;

var encodeBase64 = {
  transform(chunk, encoding, callback) {
    callback(null, chunk.toString('base64'));
  }
};
var toBase64Out = new Transform(encodeBase64);
var toBase64Err = new Transform(encodeBase64);
var opts = {stdio:["pipe", "pipe", null]};
var proc = childProcess.spawn("uname", ["-a"], opts);
proc.stdout.pipe(toBase64Out).pipe(process.stdout);
proc.stderr.pipe(toBase64Err).pipe(process.stderr);
```

rrNode transform stream

```
#include <type_traits>
#include "../modules/rrCore/rrCore.h"
#include "../modules/rrNode/rrNode.h"

int main(int argc, const char * argv[])
{
    using namespace rrNode;
    process::setCommandLine(argc, argv);
    auto encodeBase64 = []( auto& next, auto chunk,
                            auto encodingType, auto done)
    {
        next.push(transforms::base64Encode(chunk));
        done();
    };
    using encoder64 = transformLambdaWrap<decltype(encodeBase64)>;
    encoder64 toBase64Out(std::move(encodeBase64));
    encoder64 toBase64Err(std::move(encodeBase64));
    loop defaultLoop;
    defaultLoop.initStandardIO();
    auto opts = spawnOptions().pipe("stdout").pipe("stderr");
    auto proc = childProcess::spawn("uname", arr$("-a"), opts);
    proc.Stdout().pipeTransform(&toBase64Out).pipe(process::Stdout);
    proc.Stderr().pipeTransform(&toBase64Err).pipe(process::Stderr);
    defaultLoop.run();
    return 0;
}
```

NodeJS timers

```
var timer1Ms = 0;
var timer2Ms = 0;
var timer3Ms = 0;
setInterval(function(){
    timer1Ms += 1000;
    console.log("Timer 1 - after " + timer1Ms + " ms
");
},1000);
setInterval(function(){
    timer2Ms += 500;
    console.log("Timer 2 - after " + timer2Ms + " ms
");
},500);
setInterval(function(){
    timer3Ms += 250;
    console.log("Timer 3 - after " + timer3Ms + " ms
");
},250);
```

rrNode timers

```
#include <AppConfig.h>

#include "../modules/rrCore/rrCore.h"
#include "../modules/rrNode/rrNode.h"

int main(int argc, const char * argv[])
{
    using namespace rrNode;
    process::setCommandLine(argc, argv);
    loop defaultLoop;
    int timer1Ms = 0;
    int timer2Ms = 0;
    int timer3Ms = 0;
    setInterval([&]{
        timer1Ms += 1000;
        console::log("Timer 1 - after %d ms", timer1Ms);}, 1000);
    setInterval([&]{
        timer2Ms += 500;
        console::log("Timer 2 - after %d ms", timer2Ms);}, 500);
    setInterval([&]{
        timer3Ms += 250;
        console::log("Timer 3 - after %d ms", timer3Ms);}, 250);

    return defaultLoop.run();
}
```

Under the hood

THINGS LEARNED

How the library has been written

- First classes have been implemented with own API
- In successive refactoring the API has come closer to Node.JS
- Some classes have been ported 1:1 from source
  - Stream (readable/writable)
  - HTTP

Learnt Lessons when converting

- Never look for 1:1 conversion
- Reinterpret Javascript constructs in the C++ 'way'
- Refactored Javascript doesn't look too far from C++
- Use delegates / function pointers instead of inheritance
- Use the C++ type system
- Limit nameless lambda usage

Memory mangement

- Objects that fire async events:
  - Are allocated on heap
  - GC through reference counting
- Reduce Memory Allocation
  - Fixed Size Delegates (no heap allocation allowed)
  - Fixed Size Vectors (with optional heap allocation)
- Lifetime for some objects is dictated by events
- Event and delegates makes it easy to create cycles
  - Avoid leaking manually emptying all event containers

Future Improvements

- Managing object lifetime using Herb Sutter "Deferred Heap" instead of reference counting
- Make the stream abstraction optional
  - It's great for composing streams
  - It has a performance penalty
- Finish implementing missing modules