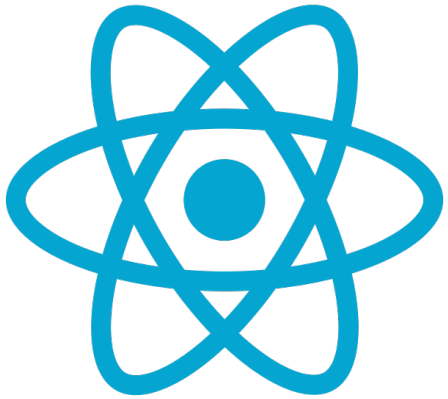
Getting the Most out of JavaScript…

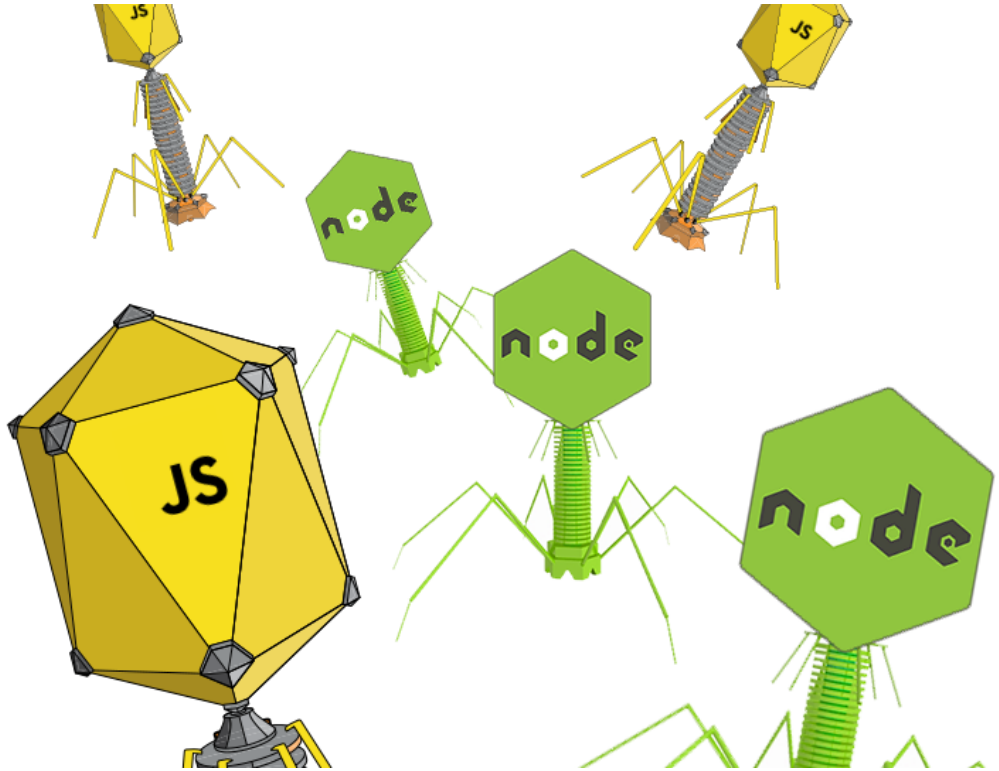
… Without using JavaScript

Dave Ramirez
Application Development Specialist
Thomas Jefferson University & Jefferson Health
@DICEGRP

People are using JavaScript for more things than anyone ever wanted to...



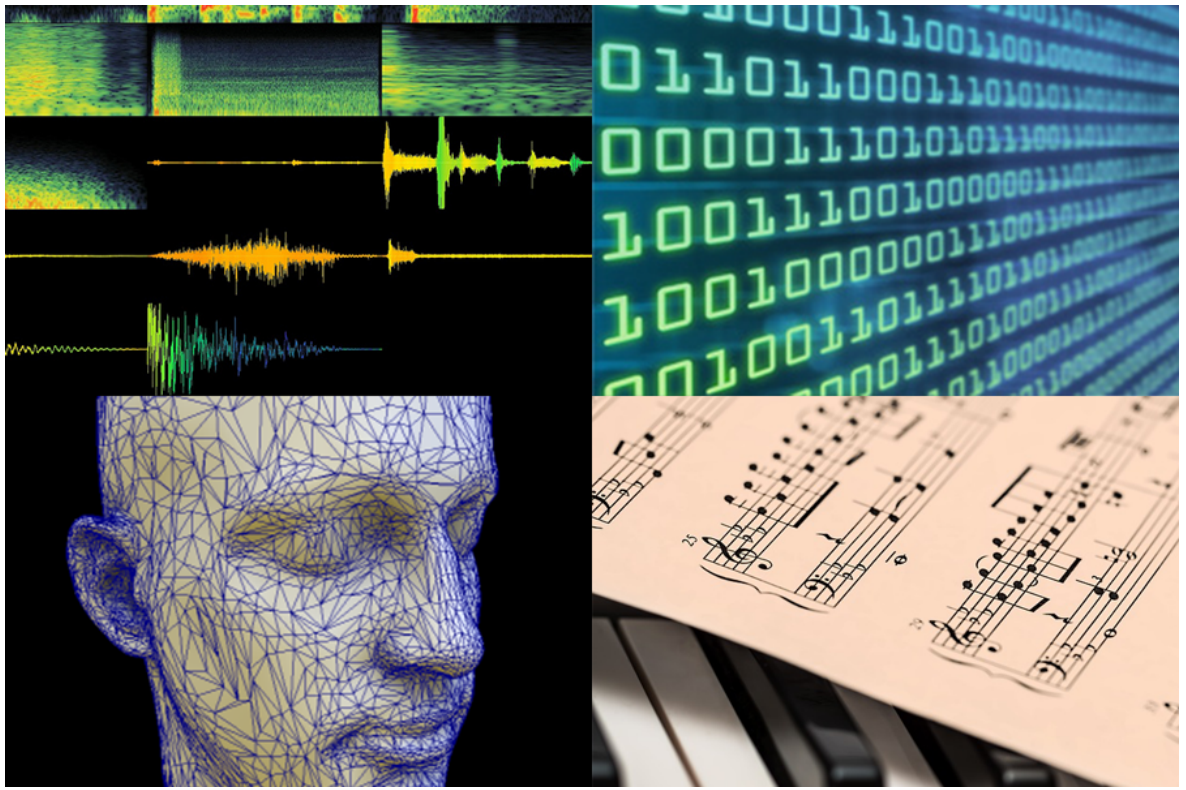
JavaScript is a virus...



... and it CANNOT. BE. STOPPED.

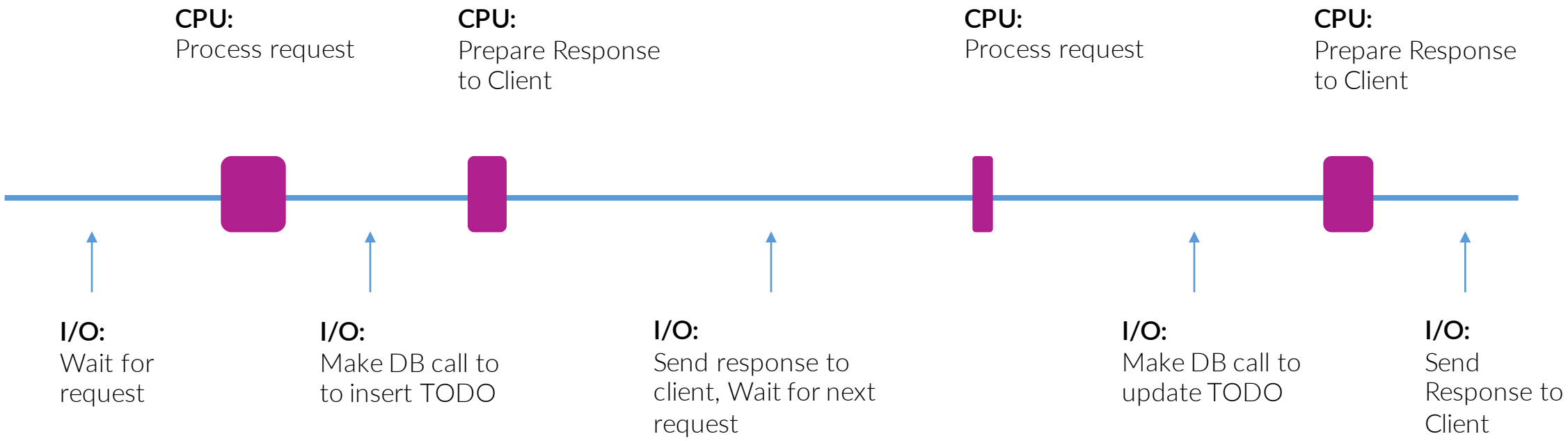
But it can’t do everything!

And what it’s NOT good at tends to be pretty cool…



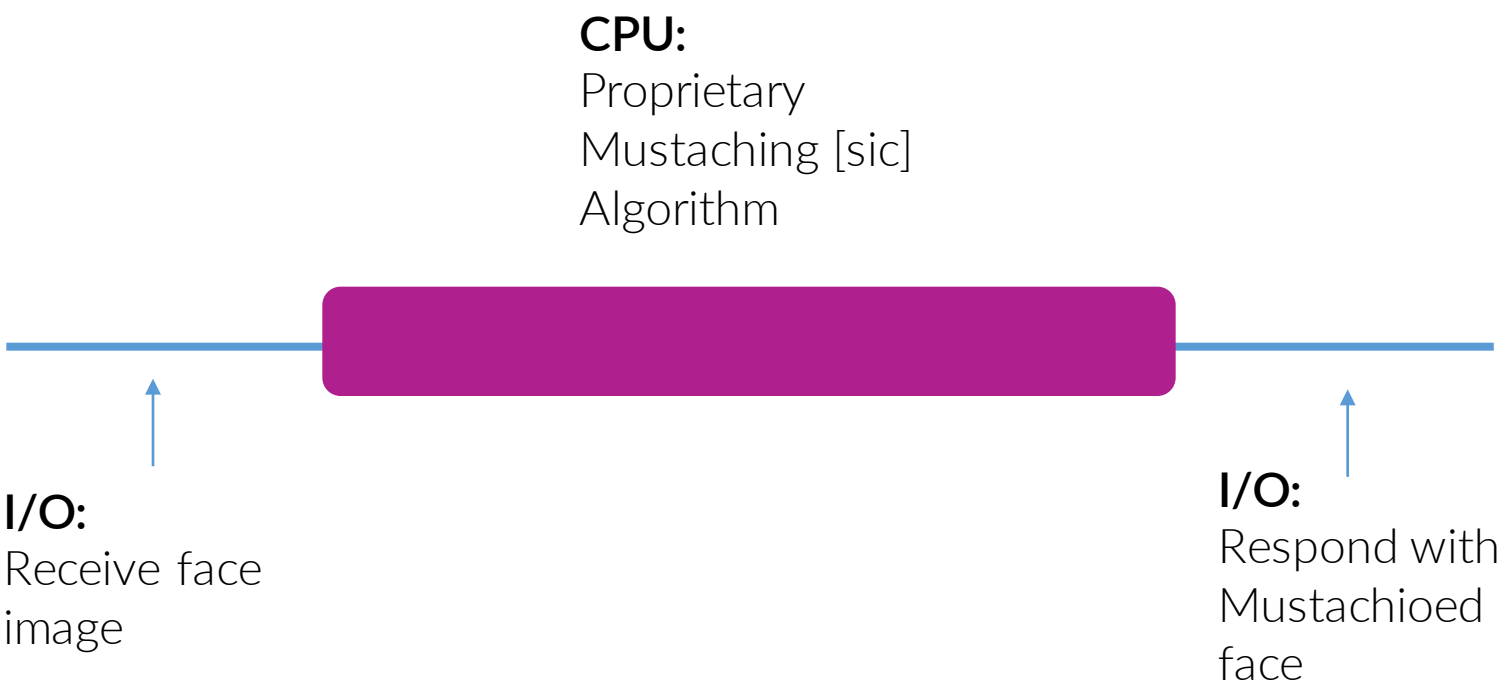
Tasks that require little IO but BIG processing power

- JavaScript is well suited to I/O bound applications
- Example: **TODO app**



Tasks that require little IO but BIG processing power

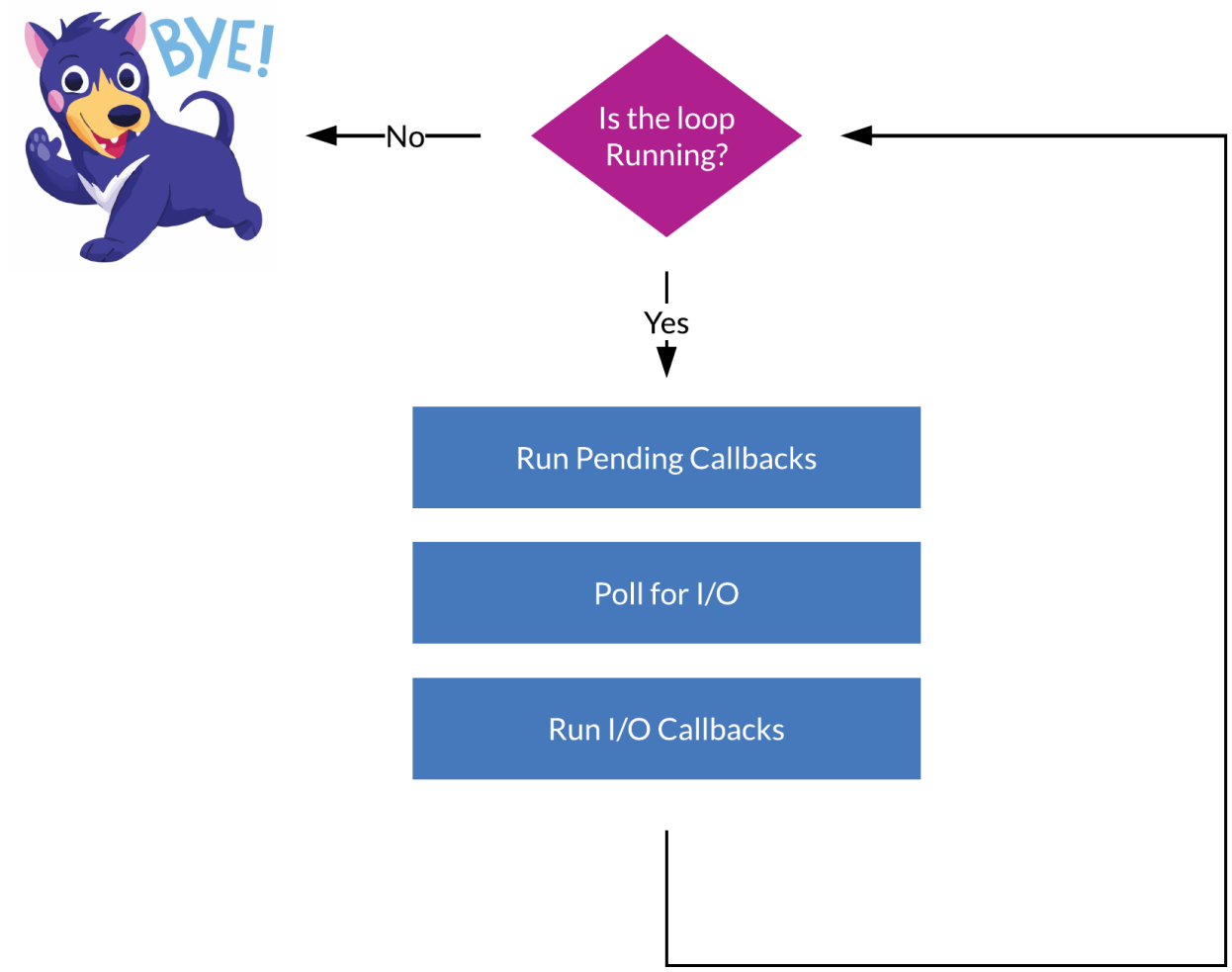
- JavaScript **struggles** with CPU bound applications
- Example: **Electron Mustache app**



libuv

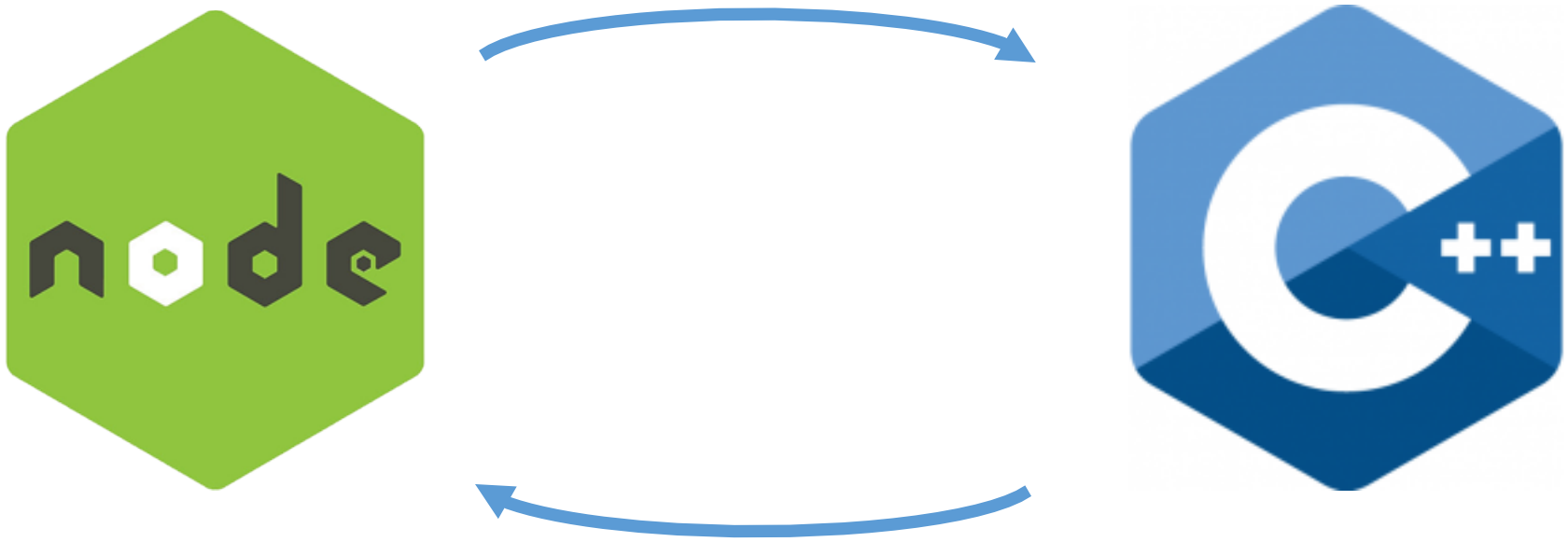
libuv in a Nutshell

(This is a gross simplification)



So how do I add mustaches to my electron app?

We can write a native NodeJS addon in C++!



Binding.cpp

```
#include <nan.h>
#include "worker.h"

NAN_METHOD(RenderMustache) {
  if (info.Length() < 2) {
    Nan::ThrowTypeError("Wrong number of arguments");
    return;
  }

  if (!info[0]->IsObject() || !info[0]->IsObject()) {
    Nan::ThrowTypeError("Wrong arguments");
    return;
  }

  if (!info[1]->IsFunction() || !info[1]->IsFunction()) {
    Nan::ThrowTypeError("Wrong arguments");
    return;
  }

  v8::Local<v8::Object> bufferObj = info[0]->ToObject();
  char* bufferData = node::Buffer::Data(bufferObj);
  size_t bufferLength = node::Buffer::Length(bufferObj);

  v8::Local<v8::Function> cb = info[1].As<v8::Function>();

  const unsigned argc = 1;
  v8::Local<v8::Value> argv[argc] = { bufferObj };

  Nan::Callback *callback = new Nan::Callback(cb);
  Nan::AsyncQueueWorker(new MustacheWorker(callback, bufferData, bufferLength));
}

NAN_MODULE_INIT(Init) {
    Nan::Set(target, Nan::New<v8::String>("renderMustache").ToLocalChecked(),
        Nan::GetFunction(Nan::New<v8::FunctionTemplate>(RenderMustache)).ToLocalChecked());
}

NODE_MODULE(binding, Init)
```


```
NAN_METHOD(RenderMustache) {
  if (info.Length() < 2) {
    Nan::ThrowTypeError("Wrong number of arguments");
    return;
  }

  if (!info[0]->IsObject() || !info[0]->IsObject()) {
    Nan::ThrowTypeError("Wrong arguments");
    return;
  }

  if (!info[1]->IsFunction() || !info[1]->IsFunction()) {
    Nan::ThrowTypeError("Wrong arguments");
    return;
  }
```

Binding.cpp

```
#include <nan.h>
#include "worker.h"

NAN_METHOD(RenderMustache) {
  if (info.Length() < 2) {
    Nan::ThrowTypeError("Wrong number of arguments");
    return;
  }

  if (!info[0]->IsObject() || !info[0]->IsObject()) {
    Nan::ThrowTypeError("Wrong arguments");
    return;
  }

  if (!info[1]->IsFunction() || !info[1]->IsFunction()) {
    Nan::ThrowTypeError("Wrong arguments");
    return;
  }

  v8::Local<v8::Object> bufferObj = info[0]->ToObject();
  char* bufferData = node::Buffer::Data(bufferObj);
  size_t bufferLength = node::Buffer::Length(bufferObj);

  v8::Local<v8::Function> cb = info[1].As<v8::Function>();

  const unsigned argc = 1;
  v8::Local<v8::Value> argv[argc] = { bufferObj };

  Nan::Callback *callback = new Nan::Callback(cb);
  Nan::AsyncQueueWorker(new MustacheWorker(callback, bufferData, bufferLength));
}

NAN_MODULE_INIT(Init) {
    Nan::Set(target, Nan::New<v8::String>("renderMustache").ToLocalChecked(),
        Nan::GetFunction(Nan::New<v8::FunctionTemplate>(RenderMustache)).ToLocalChecked());
}

NODE_MODULE(binding, Init)
```

```
v8::Local<v8::Object> bufferObj = info[0]->ToObject();
char* bufferData = node::Buffer::Data(bufferObj);
size_t bufferLength = node::Buffer::Length(bufferObj);

v8::Local<v8::Function> cb = info[1].As<v8::Function>();
```

Binding.cpp

```cpp
#include <nan.h>
#include "worker.h"

NAN_METHOD(RenderMustache) {
  if (info.Length() < 2) {
    Nan::ThrowTypeError("Wrong number of arguments");
    return;
  }

  if (!info[0]->IsObject() || !info[0]->IsObject()) {
    Nan::ThrowTypeError("Wrong arguments");
    return;
  }

  if (!info[1]->IsFunction() || !info[1]->IsFunction()) {
    Nan::ThrowTypeError("Wrong arguments");
    return;
  }

  v8::Local<v8::Object> bufferObj = info[0]->ToObject();
  char* bufferData = node::Buffer::Data(bufferObj);
  size_t bufferLength = node::Buffer::Length(bufferObj);

  v8::Local<v8::Function> cb = info[1].As<v8::Function>();

  const unsigned argc = 1;
  v8::Local<v8::Value> argv[argc] = { bufferObj };

  Nan::Callback *callback = new Nan::Callback(cb);
  Nan::AsyncQueueWorker(new MustacheWorker(callback, bufferData, bufferLength));
}

NAN_MODULE_INIT(Init) {
    Nan::Set(target, Nan::New<v8::String>("renderMustache").ToLocalChecked(),
        Nan::GetFunction(Nan::New<v8::FunctionTemplate>(RenderMustache)).ToLocalChecked());
}

NODE_MODULE(binding, Init)
```

```cpp
const unsigned argc = 1;
v8::Local<v8::Value> argv[argc] = { bufferObj };

Nan::Callback *callback = new Nan::Callback(cb);
Nan::AsyncQueueWorker(
  new MustacheWorker(callback, bufferData, bufferLength)
);
```

Binding.cpp

```
#include <nan.h>
#include "worker.h"

NAN_METHOD(RenderMustache) {
  if (info.Length() < 2) {
    Nan::ThrowTypeError("Wrong number of arguments");
    return;
  }

  if (!info[0]->IsObject() || !info[0]->IsObject()) {
    Nan::ThrowTypeError("Wrong arguments");
    return;
  }

  if (!info[1]->IsFunction() || !info[1]->IsFunction()) {
    Nan::ThrowTypeError("Wrong arguments");
    return;
  }

  v8::Local<v8::Object> bufferObj = info[0]->ToObject();
  char* bufferData = node::Buffer::Data(bufferObj);
  size_t bufferLength = node::Buffer::Length(bufferObj);

  v8::Local<v8::Function> cb = info[1].As<v8::Function>();

  const unsigned argc = 1;
  v8::Local<v8::Value> argv[argc] = { bufferObj };

  Nan::Callback *callback = new Nan::Callback(cb);
  Nan::AsyncQueueWorker(new MustacheWorker(callback, bufferData, bufferLength));
}

NAN_MODULE_INIT(Init) {
    Nan::Set(target, Nan::New<v8::String>("renderMustache").ToLocalChecked(),
        Nan::GetFunction(Nan::New<v8::FunctionTemplate>(RenderMustache)).ToLocalChecked());
}

NODE_MODULE(binding, Init)
```

```
NAN_MODULE_INIT(Init) {
    Nan::Set(target,
        Nan::New<v8::String>("renderMustache").ToLocalChecked(),
        Nan::GetFunction(
          Nan::New<v8::FunctionTemplate>(RenderMustache)
        ).ToLocalChecked()
    );
}

NODE_MODULE(binding, Init)
```

MustacheWorker.cpp

```
#include <nan.h>
#include <SnapChat/mustache.h>

class MustacheWorker : public Nan::AsyncWorker {
    public:
        MustacheWorker(
          Nan::Callback *callback,
          char *bufferData,
          size_t bufferLength
        )
        : Nan::AsyncWorker(callback),
          bufferData(bufferData),
          bufferLength(bufferLength) {}
        ~MustacheWorker() {}

        void Execute () {
          // this function executes in the worker thread
          SnapChat::Mustache::Render(bufferData);
        }

        void HandleOKCallback () {
            // we are now back in the event loop's thread
            v8::Local<v8::Value> argv[] = {
              Nan::NewBuffer(bufferData, bufferLength).ToLocalChecked()
            };
            callback->Call(1, argv);
        }

    private:
        char *bufferData;
        size_t bufferLength;
};
```


```
class MustacheWorker : public Nan::AsyncWorker {
    public:
        MustacheWorker(
          Nan::Callback *callback,
          char *bufferData,
          size_t bufferLength
        )
        : Nan::AsyncWorker(callback),
          bufferData(bufferData),
          bufferLength(bufferLength) {}
        ~MustacheWorker() {}
```

MustacheWorker.cpp

```
#include <nan.h>
#include <SnapChat/mustache.h>

class MustacheWorker : public Nan::AsyncWorker {
    public:
        MustacheWorker(
          Nan::Callback *callback,
          char *bufferData,
          size_t bufferLength
        )
        : Nan::AsyncWorker(callback),
          bufferData(bufferData),
          bufferLength(bufferLength) {}
        ~MustacheWorker() {}

        void Execute () {
          // this function executes in the worker thread
          SnapChat::Mustache::Render(bufferData);
        }

        void HandleOKCallback () {
            // we are now back in the event loop's thread
            v8::Local<v8::Value> argv[] = {
              Nan::NewBuffer(bufferData, bufferLength).ToLocalChecked()
            };
            callback->Call(1, argv);
        }

    private:
        char *bufferData;
        size_t bufferLength;
};
```


```
void Execute () {
  // this function executes in the worker thread
  SnapChat::Mustache::Render(bufferData);
}
```

MustacheWorker.cpp

```
#include <nan.h>
#include <SnapChat/mustache.h>

class MustacheWorker : public Nan::AsyncWorker {
    public:
        MustacheWorker(
          Nan::Callback *callback,
          char *bufferData,
          size_t bufferLength
        )
        : Nan::AsyncWorker(callback),
          bufferData(bufferData),
          bufferLength(bufferLength) {}
        ~MustacheWorker() {}

        void Execute () {
          // this function executes in the worker thread
          SnapChat::Mustache::Render(bufferData);
        }

        void HandleOKCallback () {
            // we are now back in the event loop's thread
            v8::Local<v8::Value> argv[] = {
              Nan::NewBuffer(bufferData, bufferLength).ToLocalChecked()
            };
            callback->Call(1, argv);
        }

    private:
        char *bufferData;
        size_t bufferLength;
};
```

```
void HandleOKCallback () {
    // we are now back in the event loop's thread
    v8::Local<v8::Value> argv[] = {
      Nan::NewBuffer(bufferData, bufferLength).ToLocalChecked()
    };
    callback->Call(1, argv);
}
```

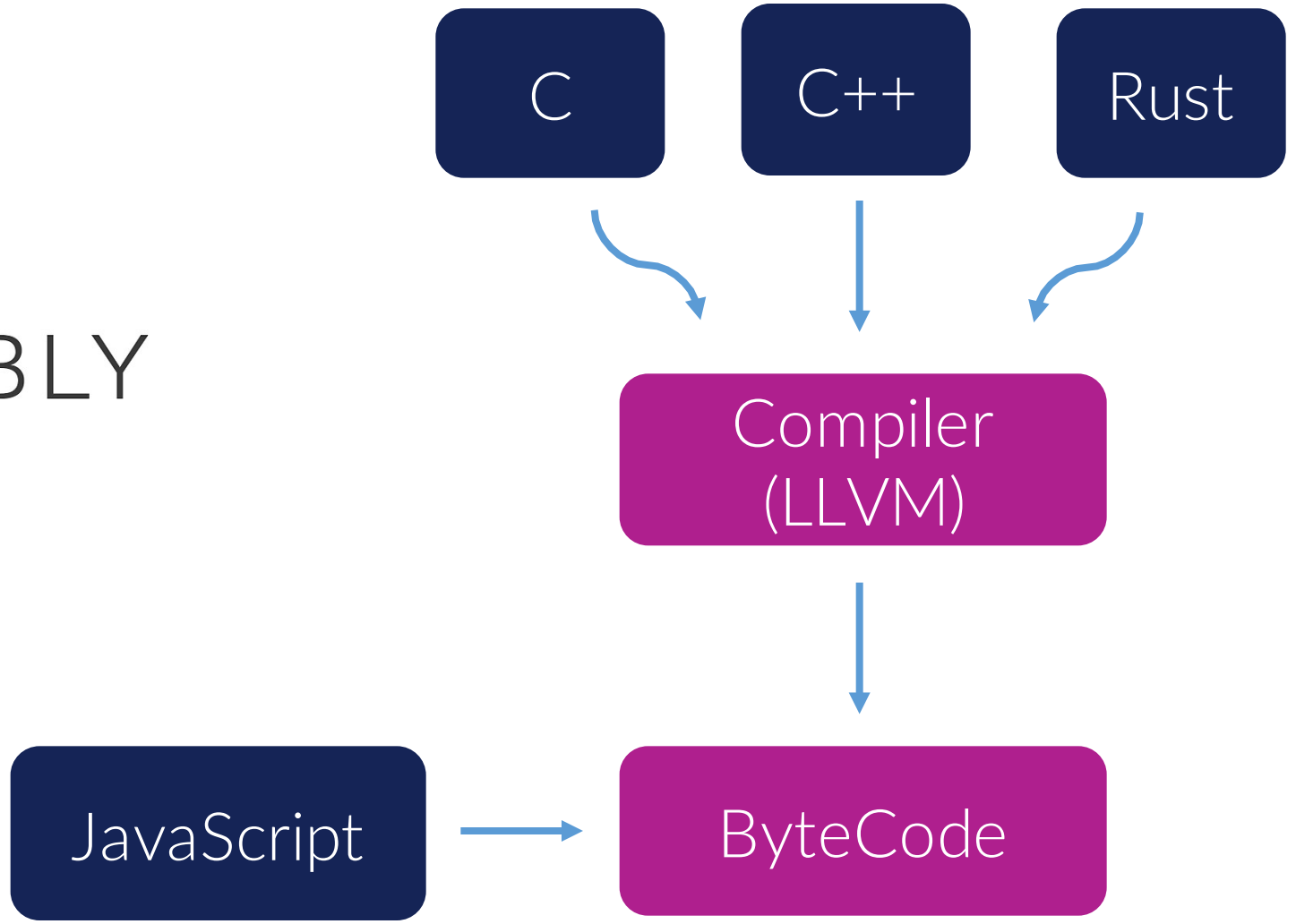
index.js

```
const mustache = require('./build/Release/binding.node')
const fs = require('fs')

fs.readFile('face.jpg', renderMustache);

function renderMustache(err, buffer) {
  if (err) throw err;
  mustache.renderMustache(buffer, mustacheRendered)
}

function mustacheRendered(data) {
  console.log('success!')
}
```

At the moment, the method for implementing Addons is rather complicated...

- Actual Quote from NodeJS Official Documentation


WA
WEBASSEMBLY


C
C++
Rust
Compiler
(LLVM)
JavaScript
ByteCode

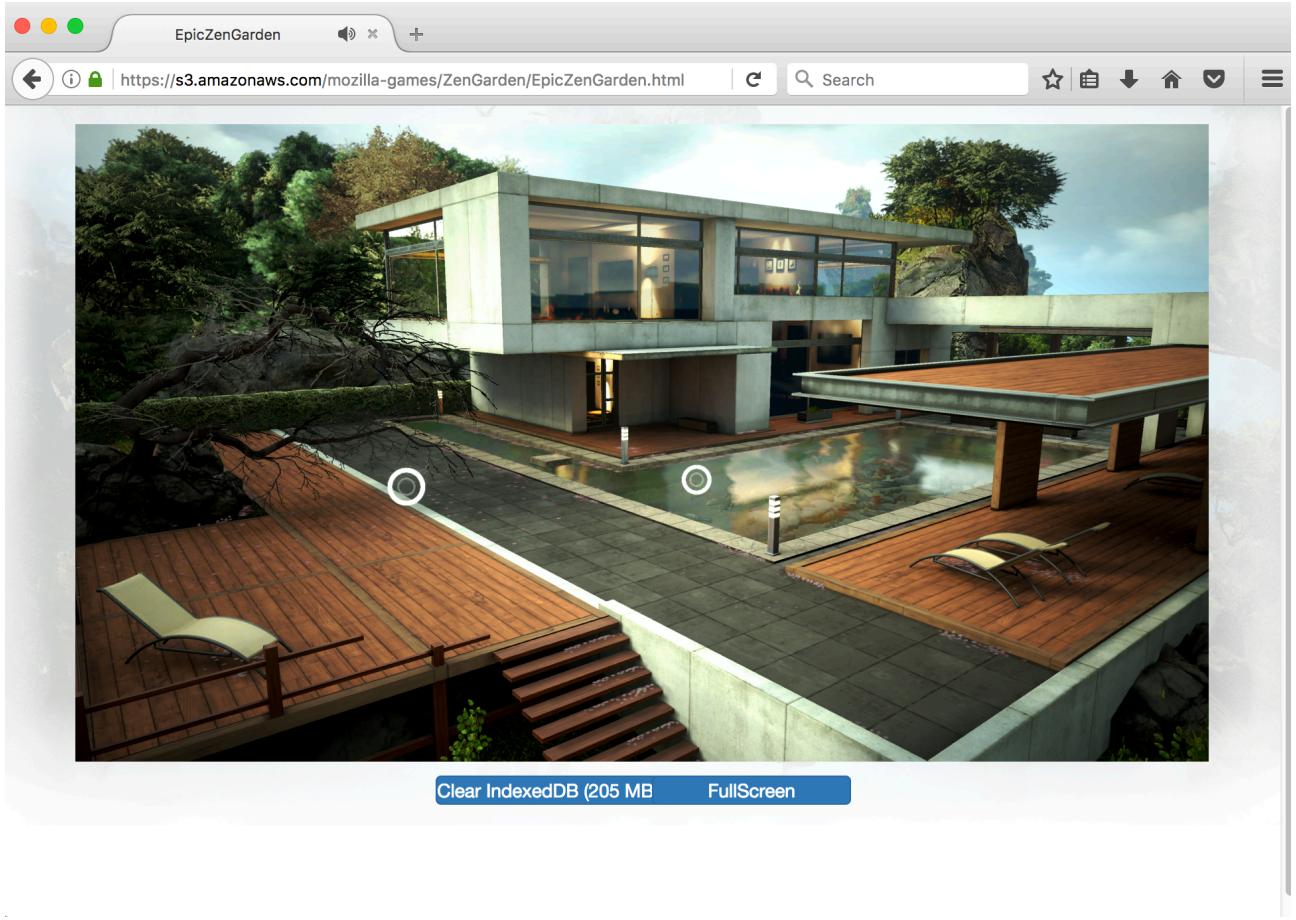
What’s the catch?

- Browser Sandbox
- Virtualization overhead
- Tooling is still young

Planned, but not in MVP:

- No DOM integration (yet)
- Thread Support (yet)

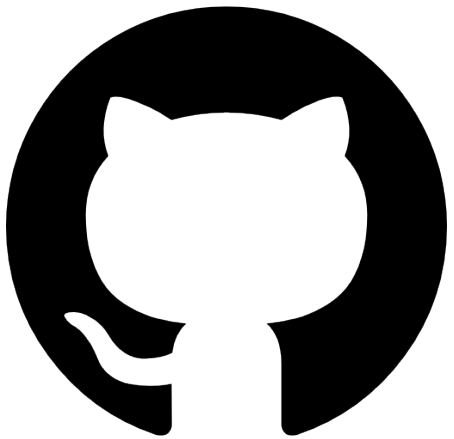
Its pretty cool…



Can I use it yet?

Thank You

@daveramirez



/ramirezd42