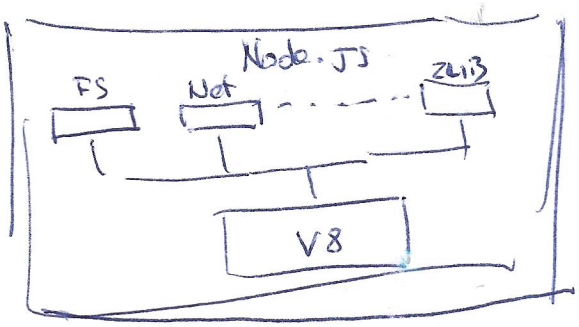
Node.js expose low level I/O → Fs - threads - Network



JavaScript Interpreter — is more like a JIT Compiler.

We are more interested in the fact that V8 provides a C++ API into the JavaScript that is executing. This API allows us to write C++ code that can be loaded by Node.js and this C++ code become fully callable from Javascript

The API allows us to hold resources allocated by JavaScript in C++ and even call JavaScript functions from C++.

Three main components

1. C++ that holds our C++ addon
2. Binding.gyp
3. JavaScript file that use the AddOns

Hello World Example

Registering the Module

Get a reference to the V8 execution context

JavaScript Memory Allocation

V8 JavaScript code Runtime - libuv Event loop implementation

Isolate is an independent instance of the V8 runtime.

Execution context | Memory management/gc | Heap / Pool of memory

It's not true that we want build native Addons for speed there are some tasks that perform well in JavaScript.

All that said, C++ should deliver at least a 2x runtime improvement over JavaScript for most CPU-bound operations or tasks.

First reason CPU-heavy task to complete in a Node.js Application

Second reason You already have some C/C++ code that you want to integrate in your Node.js application.

Migrate over C++ code that is already important to your applica than or integrate it through native addons.

Performance yes but you pay attention to the cost of data copying.

We always copy data out of V8 before utilizing fully.

Node.js and V8 Engine

How is this done? What is providing this as bridge between these two very different languages?

Google V8 JavaScript library

↓

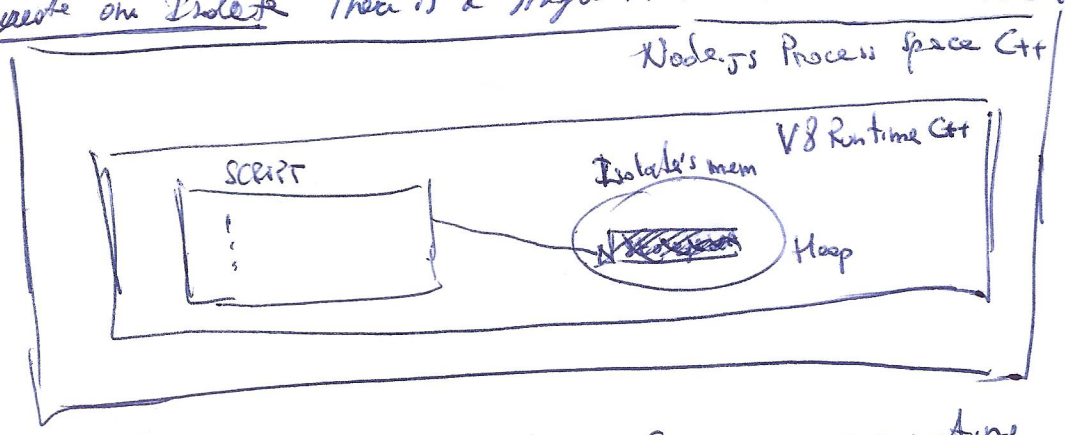
C++ Library which provides a high performance execution environment for JavaScript code.

Node.js is just the glue and support code around a few key components.

1. Google V8 JavaScript Engine
2. C++ modules for low-level I/O
3. Event loop

Isolate is an independent instance of the V8 runtime. / GC / memory management / Execution Context / Context flow

Node creates one Isolate. There is a single Thread has access to Isolate



Context
↓
Global Object in the Heap of the ISOLATE

V8 not only needs to know the references originating from JavaScript but also in the addons.

Handle → Object that represents the existence of such a reference in C++.

Local → Allocated by C++ call stack

Process to exchange variable value between JavaScript and C++. Primitive and non Primitive Data Type.

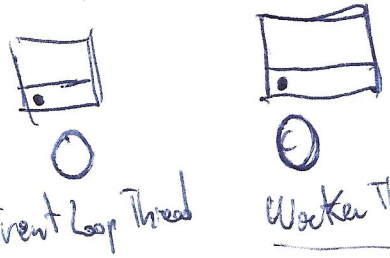
Best Error handling

Asynchronous with Worker Thread

2 Threads

- Event Loop Thread
  - Executing JavaScript code
  - We stay in this thread when we cross over in to C++ addon
  - This is the thread that we don't want to stall by doing heavy calculations
- Worker thread managed by libuv

Each thread has its own stack



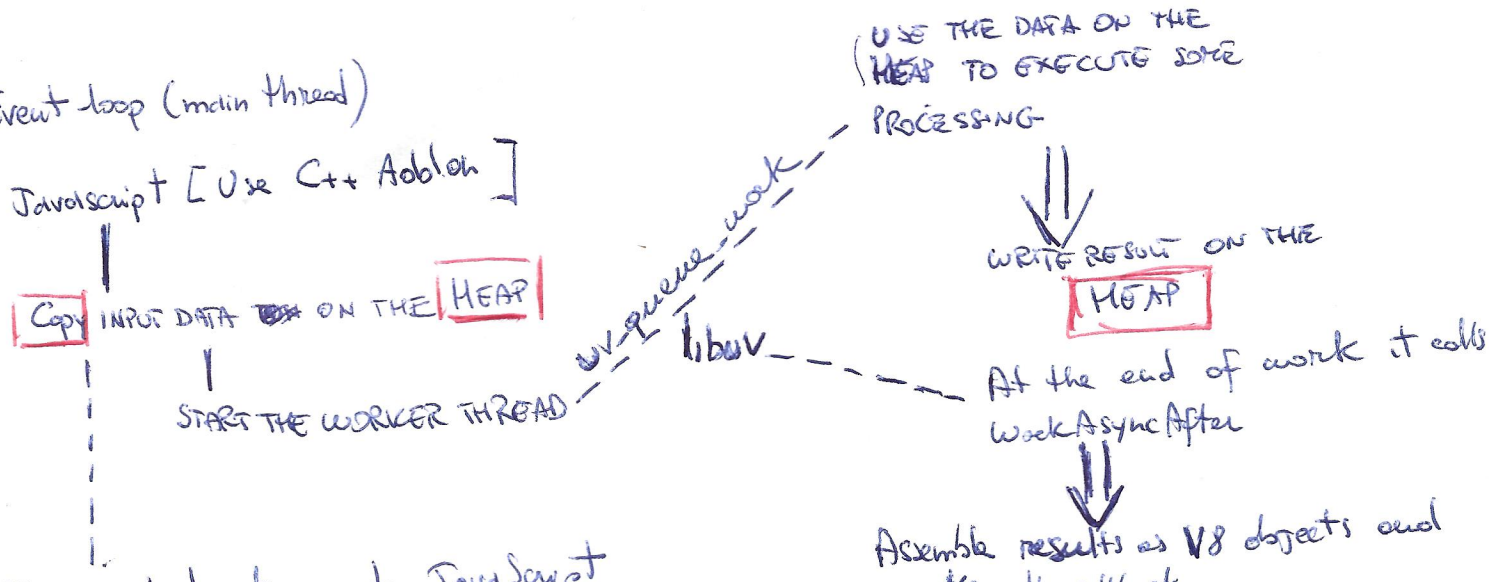
It is not possible to share stack variables between the event loop thread and worker threads

But these two threads share the heap and we are going to put our input and output data.

Javascript is implicitly single threaded V8 is built around the notion that data within Javascript is strictly accessible by one thread at a time.

YOU CAN'T ACCESS V8 MEMORY OUTSIDE THE EVENT-LOOP'S THREAD

To avoid the violation of this rules you need to create a copy of the input/output data.



Worker DATA

```
struct WorkerData {
    uv_work_t request;
    Persistent<Function> callback;
    ...
}
```

→ It will be kept in the scope by V8 and not garbage collected.

Copying input and output data is being done in the event-loop if it takes long time we're blocking the event-loop - In that case we have two problems:

1. Copying data might be a waste of memory
2. Copying data might take long, which ties up the Node.js event-loop

---

~~NAN~~

~~NAN provides a set of macros and utilities that achieve two goals:~~

Source code has been written it must be compiled into the binary ***.node

binding.gyp → Describe the build configuration of the module using JSON-like format

This file is used by node-gyp a tool written specifically to compile Node.js Addons

node-gyp configure → Generate project build files for the current platform
node-gyp build

build/Release → ***.node

Use bindings to import the module.

Direct use of Node.js and V8 API

V8 API can and has, changed dramatically from one V8 release to the next.

NAN provide a set of tools that addon developer are recommended to use to keep the compatibility between the past and the future.

API for building Native Addons independent from the underlying JavaScript Runtime (ie V8) This API will be the Application Binary Interface (ABI) stable across different version of Node.js.

N-API is independent from the underlying Javascript runtime (ex V8)

N-API calls Return STATUS CODE => napi_status

This return value is passed via out parameter

napi_value — ABSTRACT ALL JS VALUE

In case of error additional info can be obtained calling napi_get_last_error_info

What is providing this an bridge between these two very different languages?

Google V8 JavaScript Library

↓

V8 OpenSource JavaScript execution engine. V8 is a C++ library which provides a high performance execution environment for JavaScript code

V8 provides C++ API into JavaScript that is executing.

⇓

C++ that can be ~~loaded~~ from ~~JS~~ by Node.JS and this code become fully callable from ~~your~~ your JavaScript code.

To hold resources allocated by JavaScript in C++ and even call Javascript function.

Initialization code ~~&~~ macro that creates code that will register a module has need .. ~~any~~ as you forget. When the module is required it ~~ensure that~~ ensure to call Init method

---

Pairing parameter from JavaScript to C++

Inspecting ~~the~~ argument before using ~~the~~ them.

Callback

cb -> Call(Null(isolate) , 0 , {})

- Null(isolate) → this
- 0 → Number of parameter
- {} → collection of parameters

It's possible to shield yourself from API changes using Native Abstraction for Node.JS

JavaScript ⟶ C++ Pass data as a single copy

1) JS/V8 → C++
2) Execute the processing / calculations
3) C++ ⟶ JS/V8

Any existing code C++ can be used without. much precautions.

Copying data take times

Integration Pattern: Data Transferring

We are doubling memory consumption and its also costing us processing time to do all this.

The event loop implemented by a library called libuv should never stall on I/O or heavy CPU because, in theory, there is likely other work to be done.

You use C++ to perform long running task where C++ will not pause from JavaScript

Keeping those goals in mind, the locker and unlocker approach fork, inexorably. The worker thread can only access V8 data when the event loop is sleeping.

Where is that data coming from?

Is it coming from a database? Retrieve it in C++ not in JS
Is it coming from a file? Read it from C++

~~Object Wrap API~~

If your code doesn't sit somewhere already - JavaScript code is actually building it incrementally over time, the another option is to store the data in the C++ addon directly. As you build data instead of creating JavaScript arrays, objects make calls into the C++ addon to copy data incrementally in to C++ data structure that will remain in the scope:

ObjectWrap —> Use C++ Object directly from JavaScript
Buffer object —> This technique works because Buffer object allocate data outside of V8 so it could be accessible from the threads.

Object Wrapping

There is a way to create the bridge between JavaScript and your addon but you have to create a new kind of ~~prototype~~ object

node::ObjectWrap

It contains all the right tools to allow us to connect JS code with C++ code.

Classes that extend ObjectWrap can be instantiated from JavaScript using the new operator and their methods can be directly invoked from JavaScript

The wrap word refer to the way to group methods and state. There isn't nothing magically.

Neither constructor or destructor is public. They will not called directly from outside (much less from JavaScript)

Init has the responsibility to add all class functions that will be called from JavaScript to the exports object.

Init is called by InitAll which is the entry point to our addon and it's called as soon as our addon is required.

The new C++ object then get's some Node.JS sprinkled into is by calling it's Wrap method

Handle the constructor because we can create functions

object with the new operator and simply calling function

NODE_SET_PROTOTYPE_METHOD to add a prototype method on object

SetInternalFieldCount instructs V8 to allocate internal storage slots for every instance created using template.

It's not Node.js that defining the API for which we interoperate with JavaScript. The V8 API has changed over time and there is no guarantee it won't again.

Sometime there will be some breaking changes introduced in each new release.

Many applications are still running on Node.js v0.10, v0.11, v4.0 & 5.0

There is a real need for some form of abstraction, so that we might target a more stable and clean API.

NAN (Native abstraction for Node.js) provides a set of macros and utilities that achieve two goals:

1) Maintaining a stable API across all Node.js versions
2) Providing a set of utilities to help simplify some of the most difficult parts of Node.js and V8 APIs

NAN is not a high level API, but it we have a small utility to simplify ~~addon development~~ asynchronous addon development.

Man module's header files living on the build paths when we compile our C++ addon.

Maybe Type Concept

Maybe Types may or not hold values when we call ToLocalChecked() the value will be converted to the real Local

~~NAN abstracts~~

NAN doesn't provide a way to cast arrays, probably because the V8 API has never changed about that.

One area where NAN goes a bit beyond simple macros and API redefinition is when dealing with callbacks and asynchronous workers.

Callback object is a NAN class that wraps a standard V8::Function the most important thing is that it protects ~~function, it~~ wrapped function from garbage collection.

NAN offers several utility classes that abstract the concept of moving data between the event loop and worker threads. Nan::AsyncWorker

First step is to create a class that inherits from AsyncWorker and implement Execute() method.
Generally input will be saved on fields for your worker thread.
After the execution that happens on background thread a HandleOKCallback or HandleErrorCallback will be executed on the Event Loop.

NAN provide a utility method called AsyncQueueWorker which accepts a AsyncBase ker and queues it's execution up such that Execute is running on a worker thread.

This frees us from dealing directly with libuv.

The asynchronous version of an addon is for important in that the heavy CPU computation being done for large N values won't tie up the event loop from working on other things.

AsyncProgressWorker builds out a convenient pattern for sending progress updates back to JavaScript during asynchronous execution.

ObjectWrap allow us to pass complete C++ classes to JavaScript. We decorate them with Node/V8 boilerplate code that leaves us susceptible to version issues. ObjectWrap API in fact has been changed through several versions of Node.JS—

NAN aims to never fundamentally alter the programming model around C++ addons, only to provide a thin layer around V8 API to ensure backward and forwards compatibility.

Use NAN whereever possible, it's simply and safer than trying to keep up with versions of V8 yourself.

Alternatives to Addons

Automation ⟶ call your C++ as a standalone app in a child process.

Shared Library ⟶ pack your C++ code to a shared library and call those routines directly from javascript

FFI (Foreign Function Interface)

Streaming between Node and C++

Sometimes we use C++ to do some heavy task that actually generates partial results over time.

All partial results were returned immediately through the invoking of a JS callback

In this case we will use a ~~bean~~ series of callbacks to build an EventEmitter Interface -

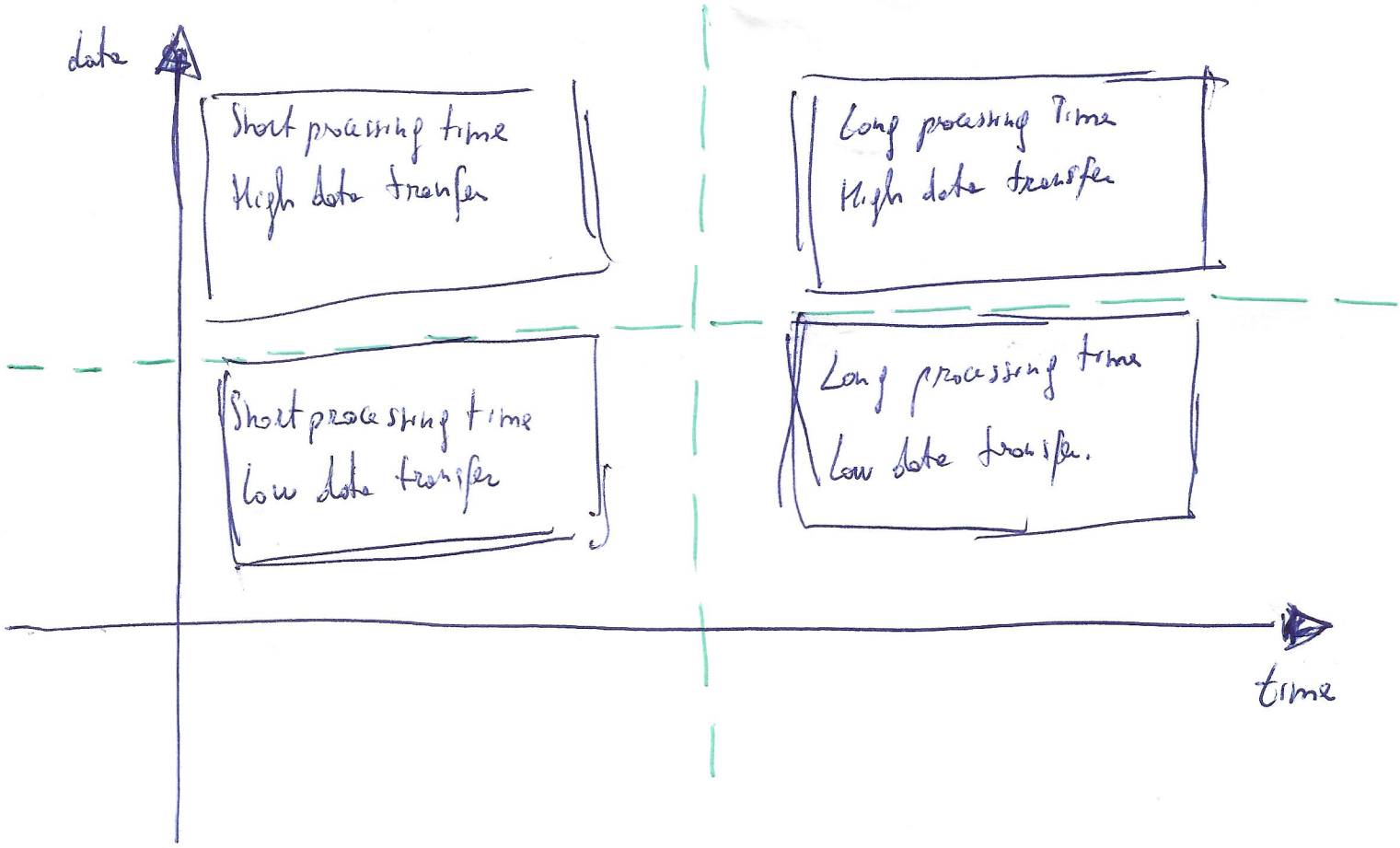
Event - Based Addons

complete progress error

streaming_worker.h

We can use JSON in our Addons

Streaming C++ output



We can categorize our native Add-Ons using two axes

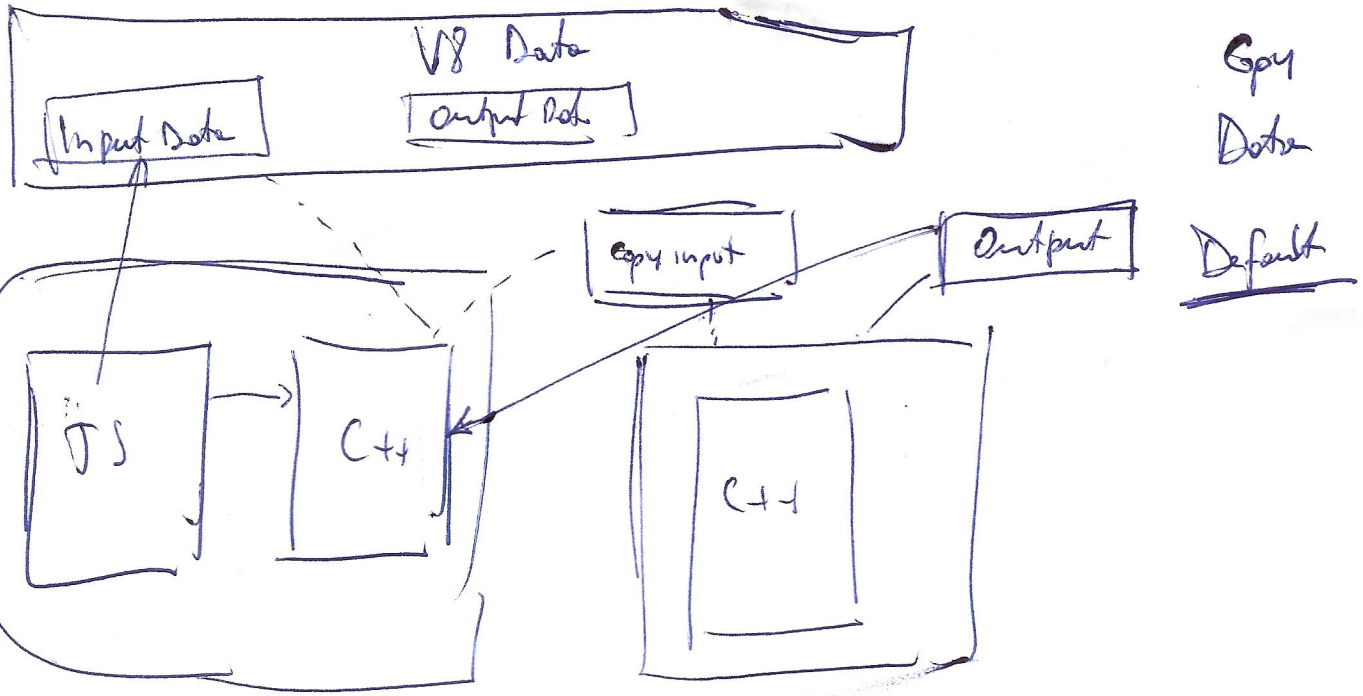
time → Processing time we will spend in the C++ code

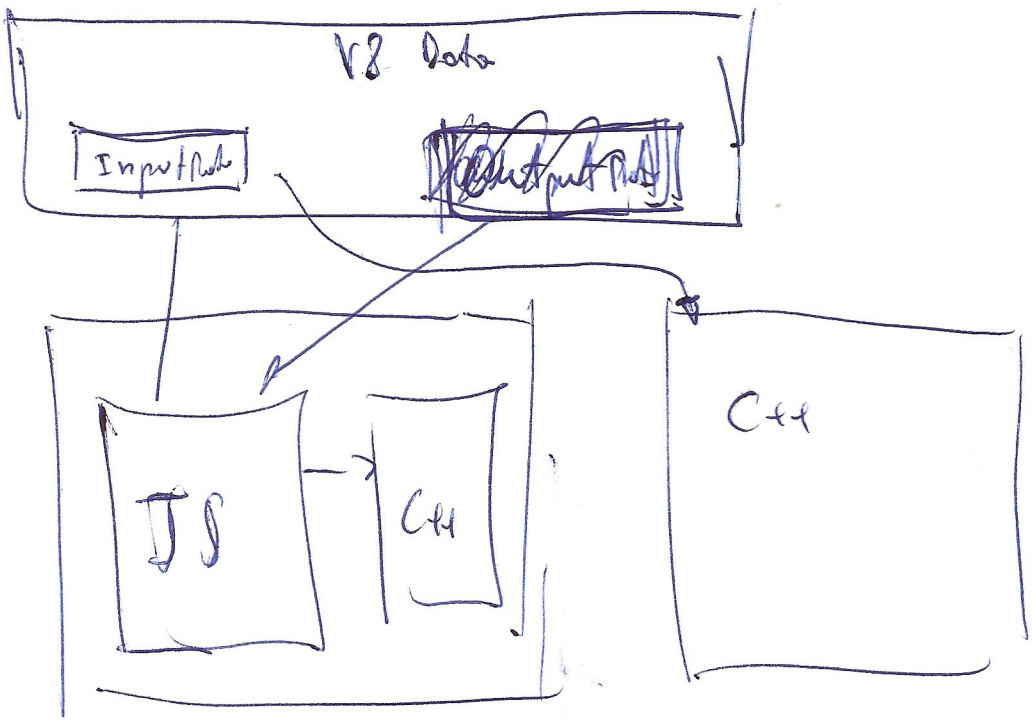
data → The amount of data flowing between C++ and JavaScript

Short processing time → your add-on can possibly be synchronous. C++ code running on Node.js event loop

Long processing time → your add-on have to be can be asynchronous.

JavaScript primitives are immutable, storage cells associated with primitive JavaScript variables cannot be altered by a C++ addon. The primitive JavaScript variables can be reassigned to new storage cells created by C++ but this means changing data always results in new memory allocation.





Instances of the Buffer class are similar to arrays of integers but correspond to fixed-sized, raw memory allocations outside the V8 heap.

You can't ignore the costs of copying data between V8 storage cells and C++ variables. If you aren't careful you can easily kill the performance boost you might have thought you were getting by dropping into C++ to perform your work.

Buffers offer a way to work with the same data in both JavaScript and C++ thus avoiding the need to create copies.