Node.js

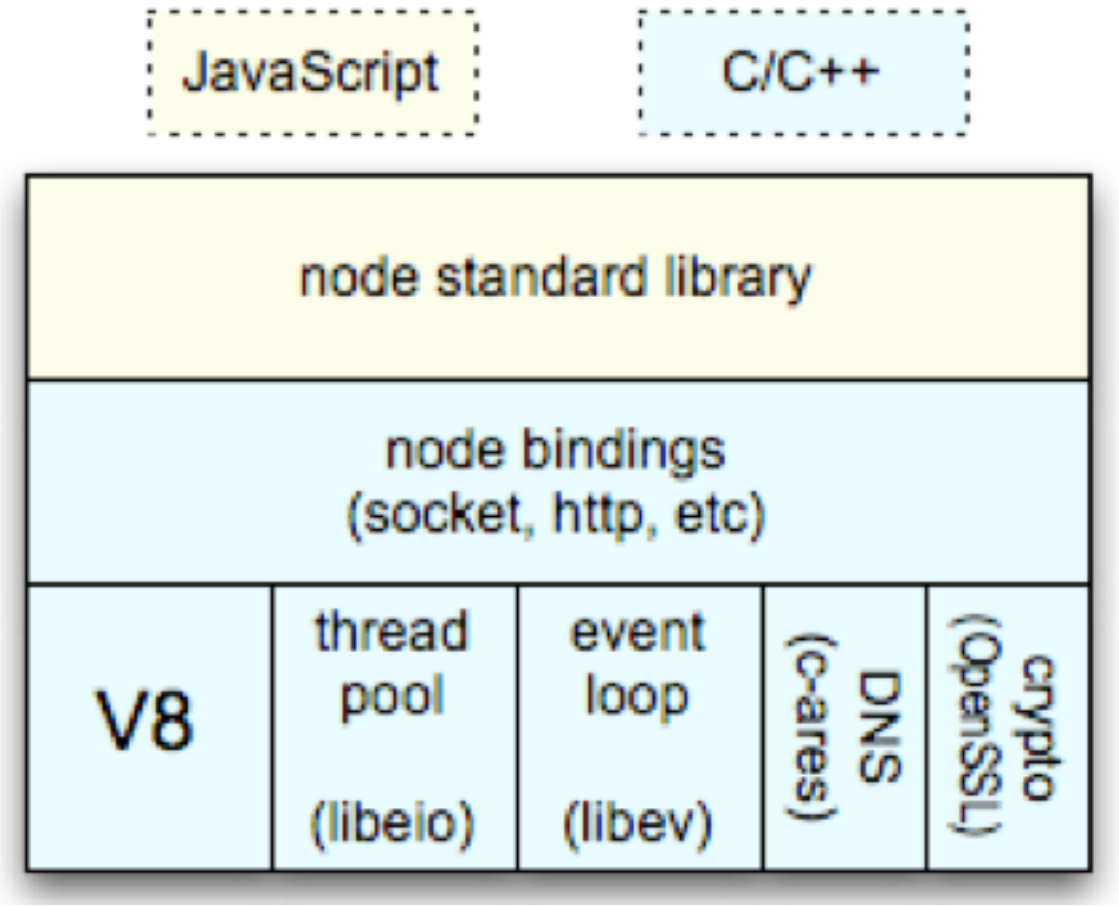
Architecture

Introduction

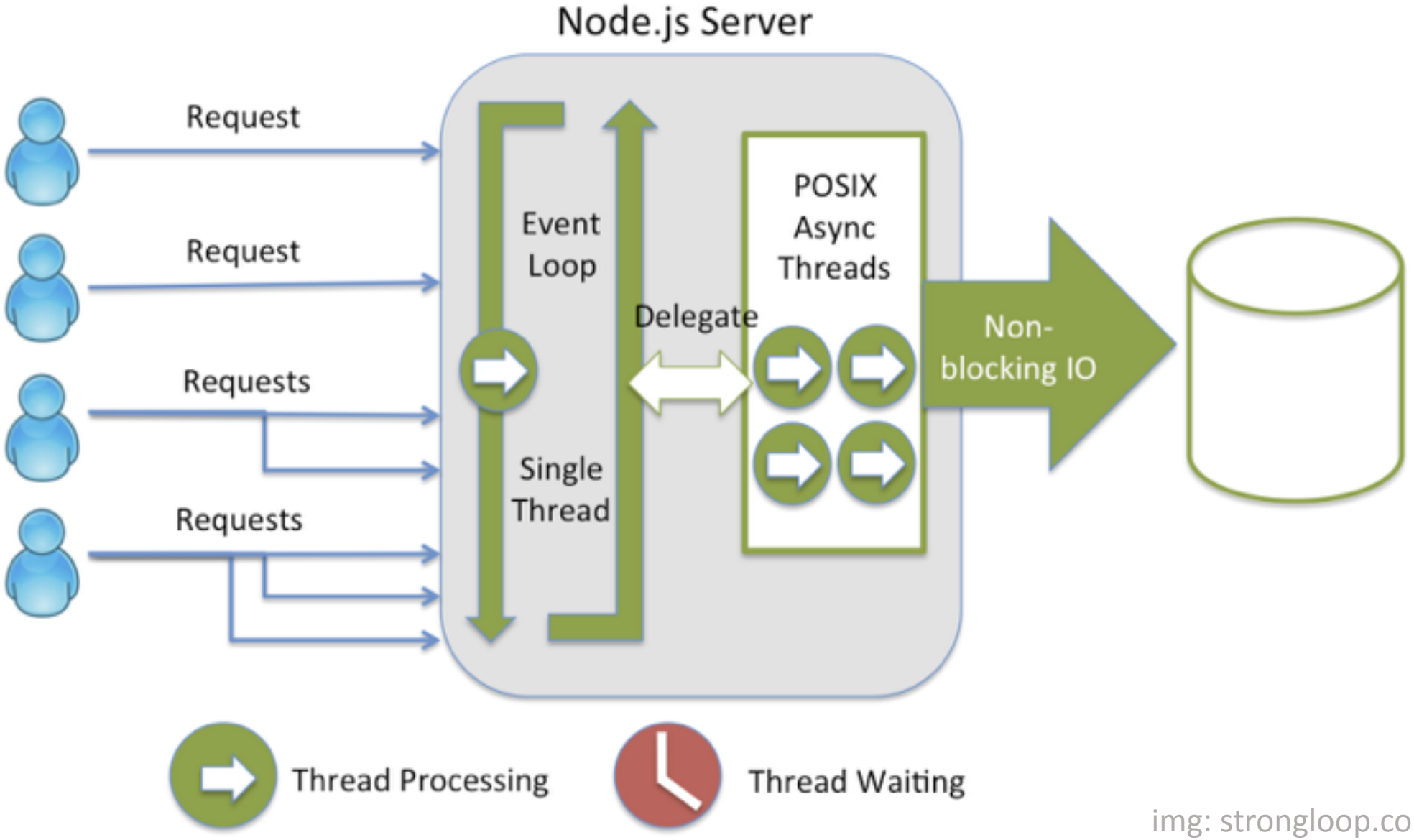
- Node.js is a JavaScript Runtime
- You run JavaScript without Browsers, HTMLs <script … />
- Built on Asynchronous I/O Design Patterns
- Callback for every I/O
- Mostly written in C++
- Google V8 Engine for JavaScript
- Single Thread for JavaScript
- Multiple threads for background Async workers

Node.js

- **libev** is the event loop which actually runs internally in node.js to perform simple event loop operations
- **libeio** is a library to perform input output asynchronously. It handles files, data handlers, sockets etc
- **libuv** is an abstraction layer on the top of libeio , libev, c-ares (for DNS) and iocp (for windows asyncronous-io). LibUv performs, maintains and manages all the io and events in the event pool

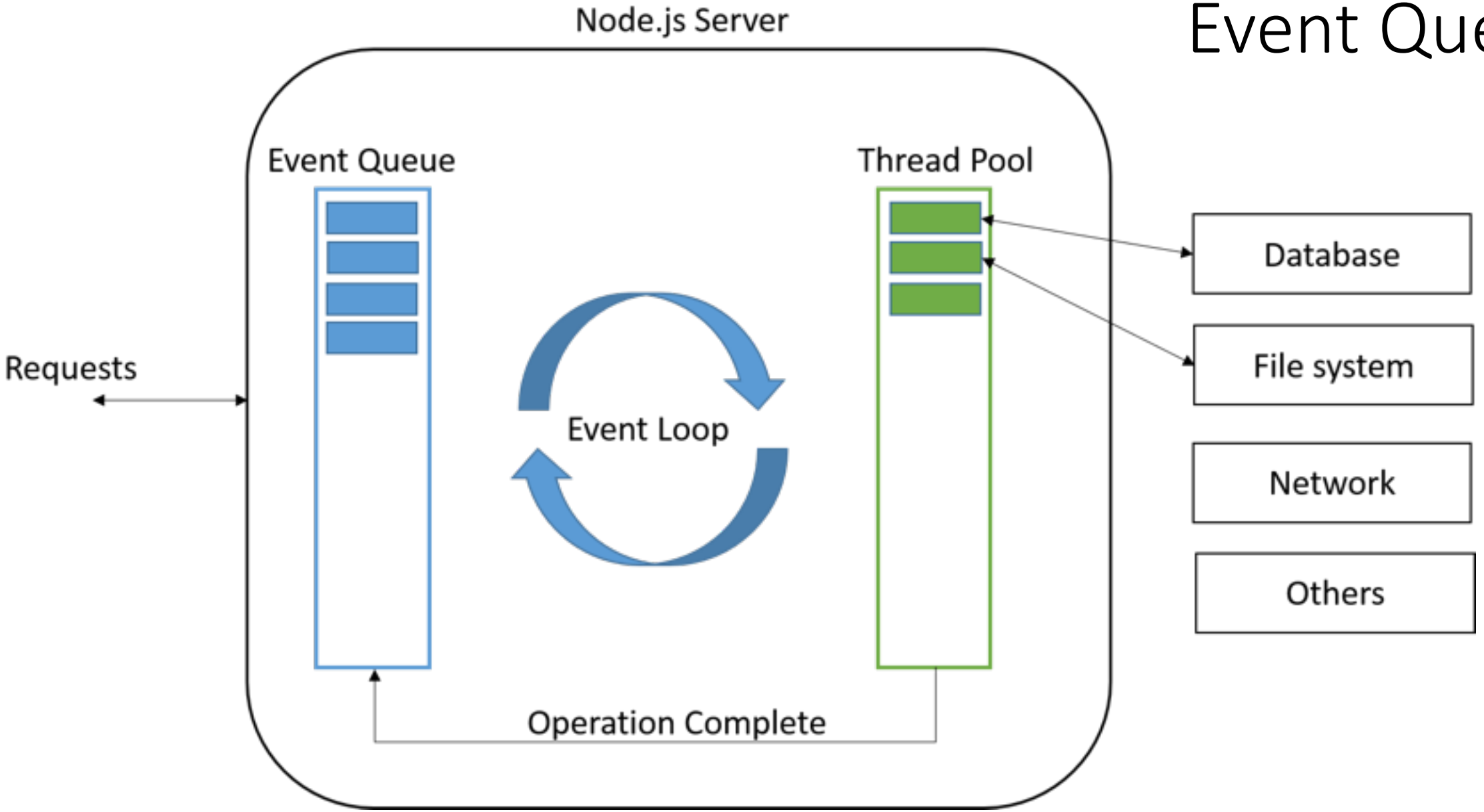


Node.js Event Loop

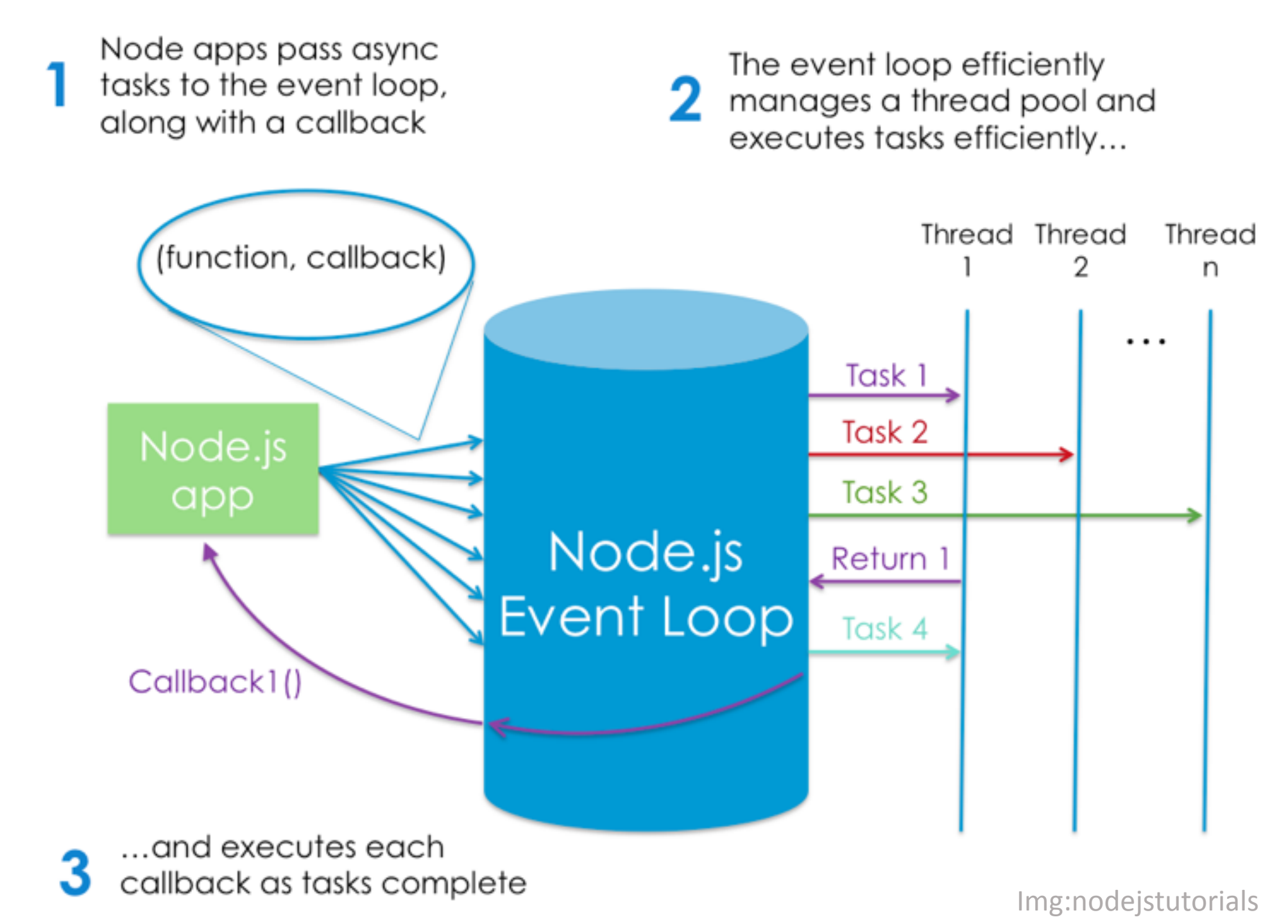


img: strongloop.com

Event Queue



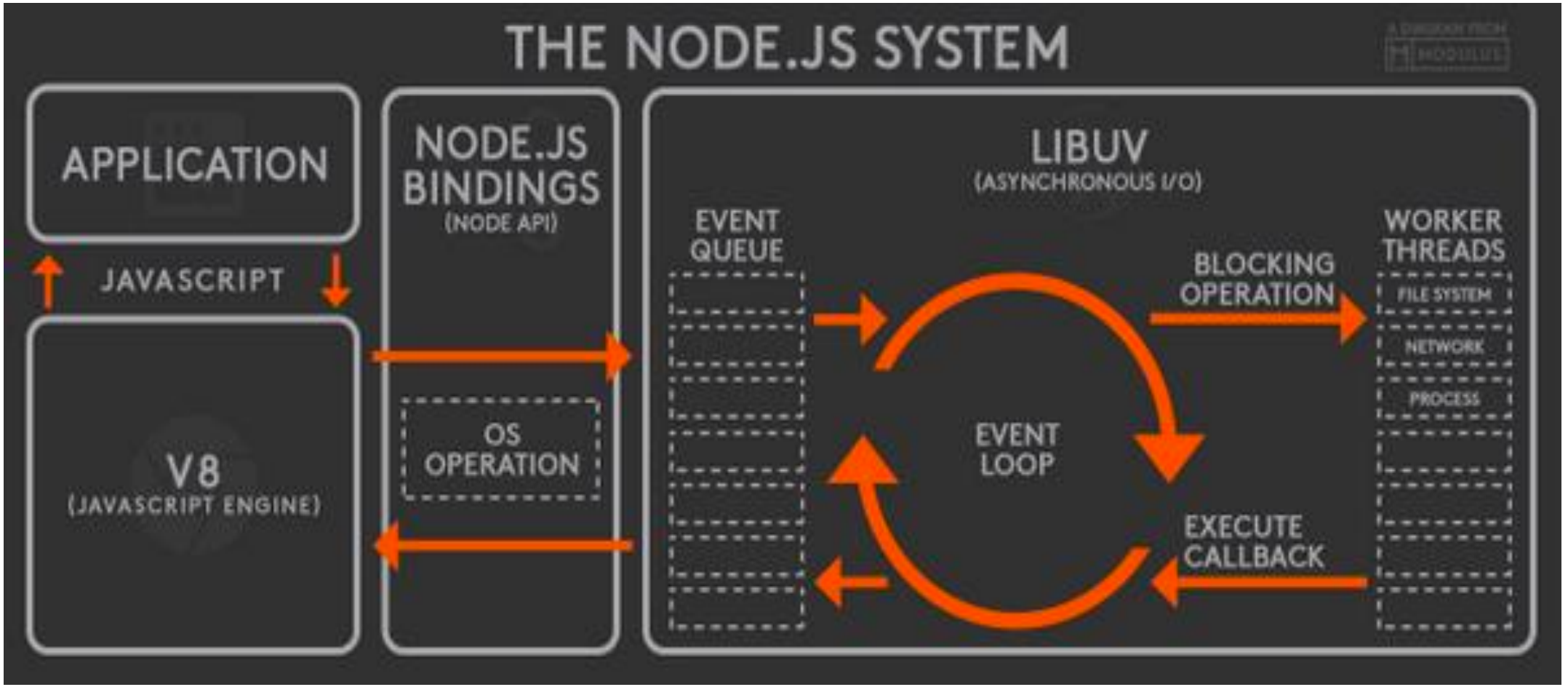
img: abdelraoof.com



Img:nodejstutorials

Node.js still uses Threads for workers
You can scale as many threads by configuring Environment variable

Yet Another Block Diagram to remember



img: @BusyRich

Google V8

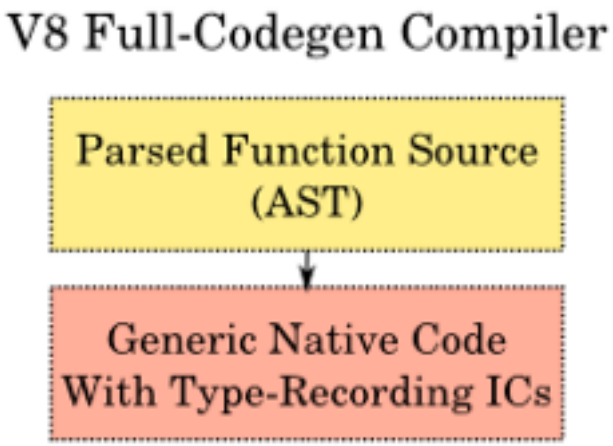
- High performance Java Script Engine
- Open Source
- Developed, Maintained by Google
- Written in C++
- Used in Billions of Devices, Chrome, Chromium browsers
- V8 compile JavaScript (yes, doesn’t interpret line by line)

V8 Compiler

- Generate Executable Machine Code on the fly, not Byte Code or Intermediate Code (Supports ARM, MIPS, IA32, x86_64)
- Profiler to identify hotspot, often used code at runtime
- Full-CodeGen & Crankshaft compilers
- Full-codegen - Simple, quick compiler and compile code pretty fast that produces simple (slow) machine code
- Crankshaft - Highly optimized code, runs really fast. Works on identified hotspot similar to Java HotSpot. Compile slower, generate fast, optimized machine code

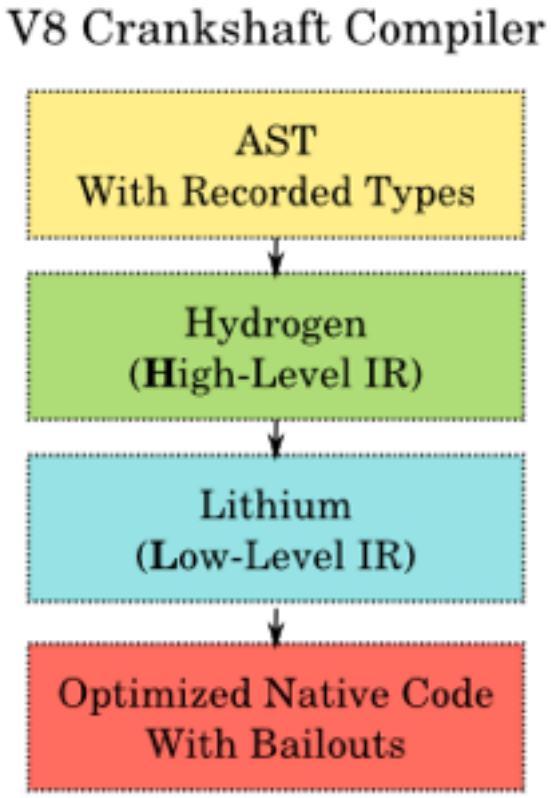
Full-Codegen

- Parse JS, converts to Machine Code
- No intermediate code generation => no interpreter
- Goal: Run the code as soon as possible
- Keeps the local variables on stack and hash not on registers
- Emit call to MacroAssembler



V8-Crankshaft Compiler

- Optimize most used functions called “hotspot” with Type Feedback
- Converts Parsed JS AST into Hydrogen high-level intermediate representation, constant folding, value numbering, method inlining
- Low - Level, further optimized, virtual registers
- Removes boxing/unboxing operations, reduces a lot of instructions



Node.js callback

```
fs.rename('/tmp/hello',  '/tmp/world', function(err) {
    if (err) throw err;
   fs.stat('/tmp/world', function(err, stats) {
     if (err) throw err;
     console.log(JSON.stringify(stats));
  });
})
```

JavaScript code is always executed in sequential steps In single thread.

fs.rename and fs.stat shall call Background worker threads to Rename the file and get the file Descriptors

Blocking Vs Non-Blocking I/O

```
//Traditional Blocking I/O
var result = db.query("select * from orders");
//wait for result!
doSomethingWith(result);
//this statement is blocked
doSomethingWithOutResult();
```

```
// Non-traditional, Node.js style Non-blocking I/O
 // notice: Callback

db.query("select * from orders", function (err, result) {
    doSomethingWith(result);
    //wait for result to be processed
}); //executes without any delay!

doSomethingWithOutResult();
```

What if I need multiple Threads for my JavaScript?

- Cluster https://nodejs.org/api/cluster.html
- Cluster work with master and worker processes
- Communicate via IPC, handles are passed between master and workers
- No in-memory access, no state mechanism
- Uses kernel internals to share the socket descriptors across process

What can be done with Node.js

- Write JavaScript Applications for Desktop Application
- Server Application (HTTP, REST APIs, Web Application, Web Socket, MQTT, any networking related)
- Write Command Line Interface tools (CLI) like Grunt, Build System

Where not to use Node.js

- Rich computing Environment like Analytics
- Machine Learning
- Anything that involves many sequential, computation heavy instructions that is not suitable for single threads
- Still you can extend Node.js with C++ to make use of heavy computing needs

Node Cli

- To start node cli, run
- > node

- To run JavaScript, use

- > node filename.js

Environment Variable

```
console.log(process.env);

console.log(process.env.PATH);
```

Command Line Arguments

```
//all variables
console.log(process.argv);

// print process.argv
process.argv.forEach(function (val, index, array) {
  console.log(index + ': ' + val);
});
```

process.nextTick

- process.nextTick (function() {}) - this queue the callback, on the list, execute on next event

Thread Pool Size

- By default, it uses 4 threads
- To increase, Set Environment variable UV_THREADPOOL_SIZE=64
- http://docs.libuv.org/en/v1.x/threadpool.html