NodeJS

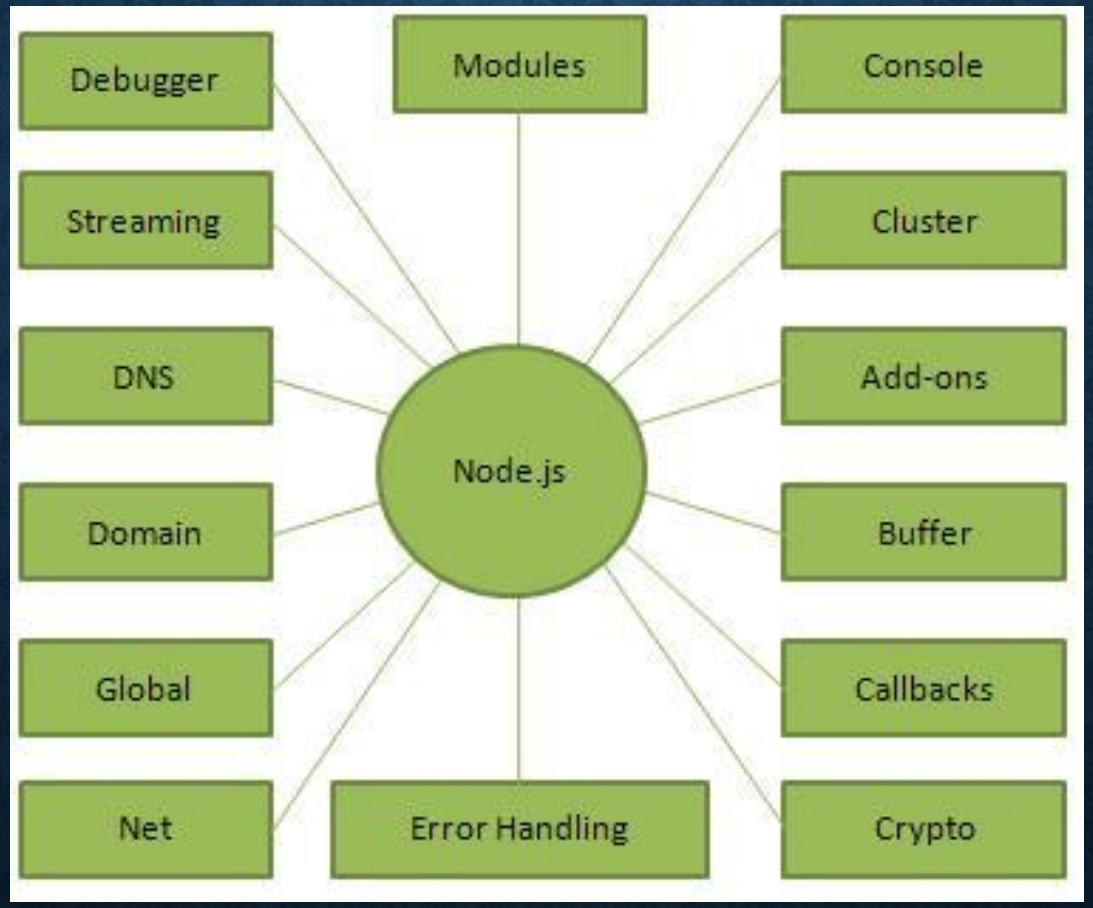
The Basics

COS216
AVINASH SINGH
DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF PRETORIA

NODEJS - OVERVIEW

- Server-Side JavaScript
  - Built on Google's V8 Engine (Open-source, High-performance, Web Assembly engine)
- Initial release in 2009, by Ryan Dahl
- Fast and Scalable
- Open source
- Cross Platform

NODEJS - OVERVIEW



NODEJS - FEATURES

- Asynchronous and Event Driven
- Very Fast
- Single Threaded
- Highly Scalable
- No Buffering
- Generate Dynamic Content

NODEJS – WHY NOT SINGLE THREAD?

- Harder to maintain threads (programming issue)
- Shared storage and locks
- Race conditions
- Deadlocks
- Context switching costs

NODEJS – WHY SINGLE THREAD?

- Faster Asynchronous processing
- Increase in performance
- Scalability
- Memory efficient
- Non-blocking (if implemented correctly)

NODEJS – HOW IT FITS IN?

- Unlike Apache (webserver), which just provides hosting capabilities (no programming or logic), NodeJS is tightly integrated providing a webserver and logic/programming options.
- Unlike PHP, coding an API on node is much more faster and powerful. Since there is additional features and functionality Node provides opposed to PHP
- Many websites and web applications are moving towards NodeJS

NODEJS – WHAT MAKES IT SUPERIOR?

- Advantages

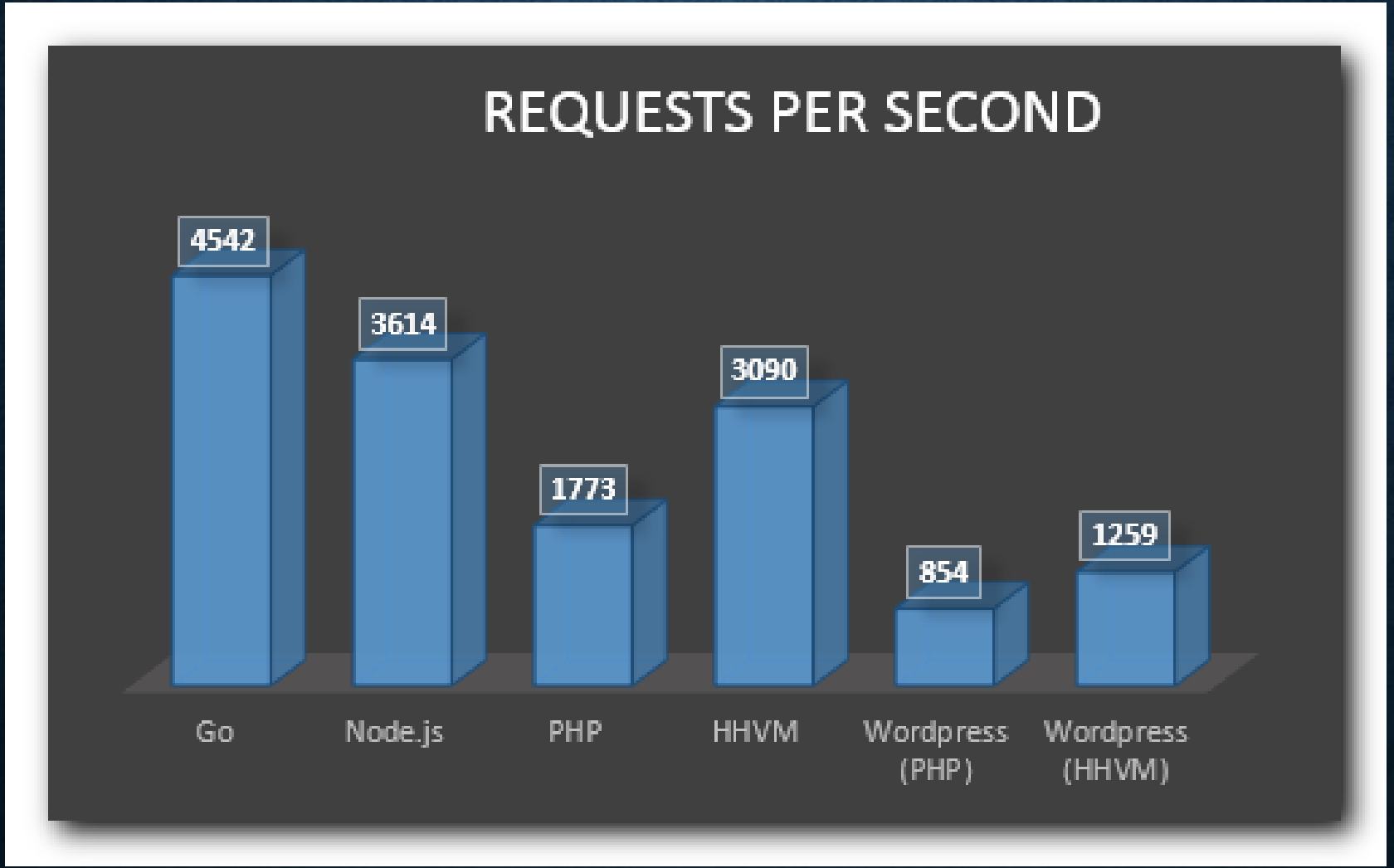
NodeJS	Apache/PHP
Separation of concerns	Deep code base
Lightweight	SQL
More features	Handle larger loads
Database Interfaces	Many libraries and Frameworks
JSON	
Speed and Scalability	
Single programming language (although there is support to convert between other languages)	

NODEJS – WHAT MAKES IT RESTRICTIVE?

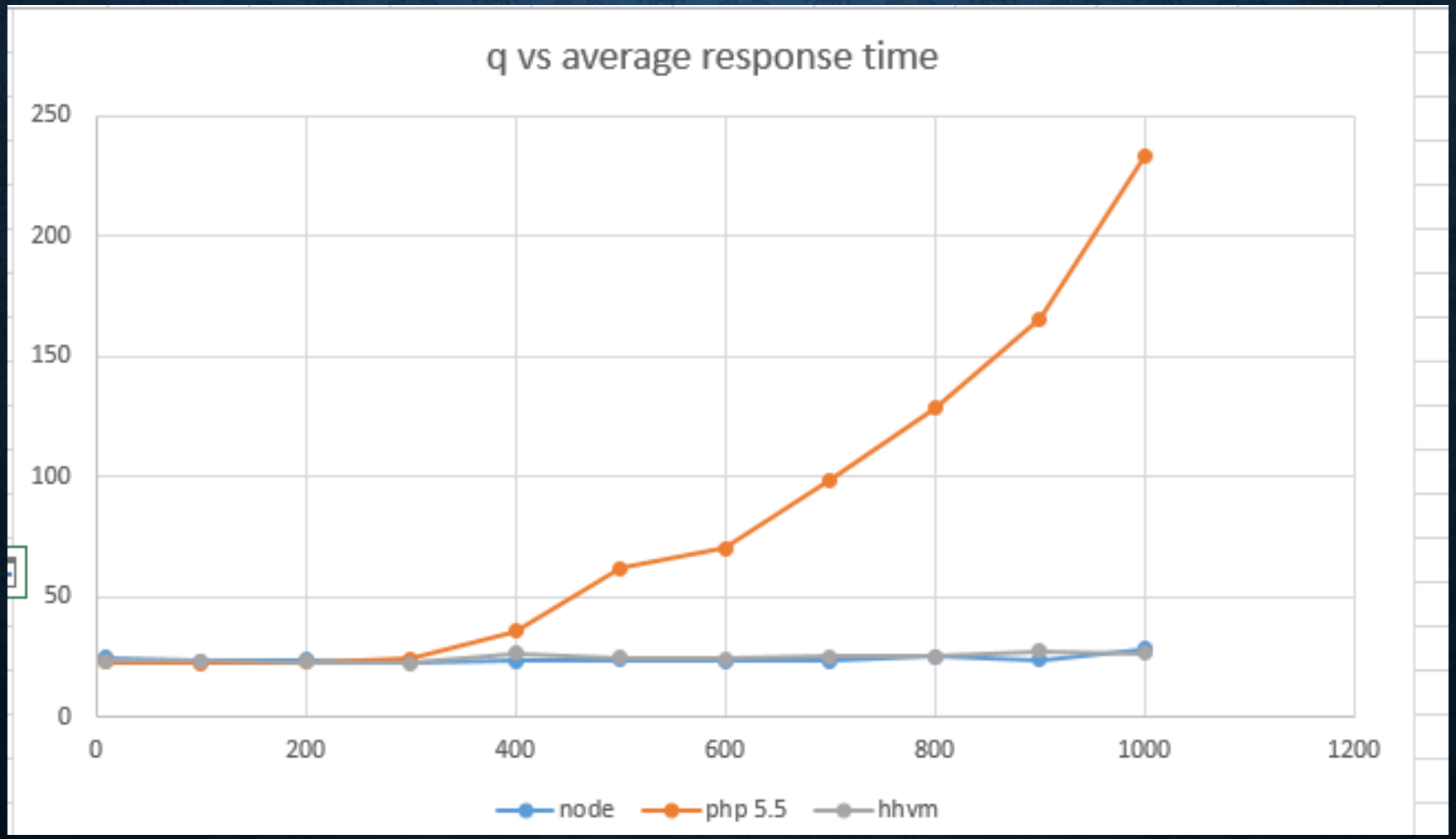
- Disadvantages

NodeJS	Apache/PHP
Unstable API	Slower
Lack of Robust Libraries/Modules	Not easy to scale
Not suitable for large and complex web applications	Not suitable for large and complex web applications (*)
Async (harder to code, and maintain)	Security
Complexity of closures	Harder to program / make APIs
	Poor error handling
	Weak type

NODEJS – SOME PERFORMANCE STATS?



NODEJS – SOME PERFORMANCE STATS?



NODEJS - COMPARISON

- **A common task for a web server can be to open a file on the server and return the content to the client.**

PHP or ASP	NodeJS
Sends the task to the computer's file system.	Sends the task to the computer's file system.
Waits while the file system opens and reads the file.	Ready to handle the next request.
Returns the content to the client.	When the file system has opened and read the file, the server returns the content to the client.
Ready to handle the next request.	

NODEJS - SETUP

- Download at https://nodejs.org/en/
- Run using the Node interpreter
- Code is written in a .js file

NODEJS - REPL

- **Read** - Reads user's input, parses the input into JavaScript data-structure, and stores in memory.
- **Eval** - Takes and evaluates the data structure.
- **Print** - Prints the result.
- **Loop** - Loops the above command until the user presses ctrl-c twice

NODEJS – API FUNCTIONS

- Blocking Functions (Synchronous execution)
- Non-blocking Functions (Asynchronous execution)

NODEJS - BLOCKING

```
// blocking I/O + threads
var urls = db.query("select * from urls"); // wait
urls.each(function (url) {
  var page = http.get(url); // wait
  save(page); // wait
});
```

NODEJS – CALLBACKS (NON-BLOCKING)

- Callback is an asynchronous (non-blocking) version of a function
- Called once a given task has finished
- This ensures that there is no blocking (remember single thread?)

- One major drawback to Callbacks is when heavily nested, it becomes hard to read and maintain. This is known as Callback Hell.
  - How can it be solved?

NODEJS – CALLBACKS (NON-BLOCKING)

```
// non-blocking I/O + event loop
db.query("select * from urls", function (urls) {
  urls.each(function (url) {
    http.get(url, function (page) {
      save(page);
    });
  });
});
```

NODEJS – MAIN MODULES

- OS
- Path
- Net
- DNS
- Domain

NODEJS – MODULES

- You can create your own modules in NodeJS using the **exports** keyword.
- Works very much the same as JavaScript.

- Example: module that creates and returns a date.

```
exports.myDateTime = function () {
  return Date();
};
```

NODEJS – NPM

- NodeJS has several modules/libraries that you can use.
- These libraries can be downloaded using NPM (Node package manager).

  npm install –g package_name *// the –g flag means install it globally*
- Once a required module/library is installed, you will need to import it into your script using the **"require()" function.**
- This simply loads the library into a variable for usage.

- Example:

  var fs = require("fs"); *// this loads the file system library into fs*

NODEJS – EXAMPLE

- Below is a example of how you would read the contents of a file and print it to the screen.

```
var fs = require("fs");
var data = fs.readFileSync("input.txt");

console.log(data.toString());
console.log("End of file");
```

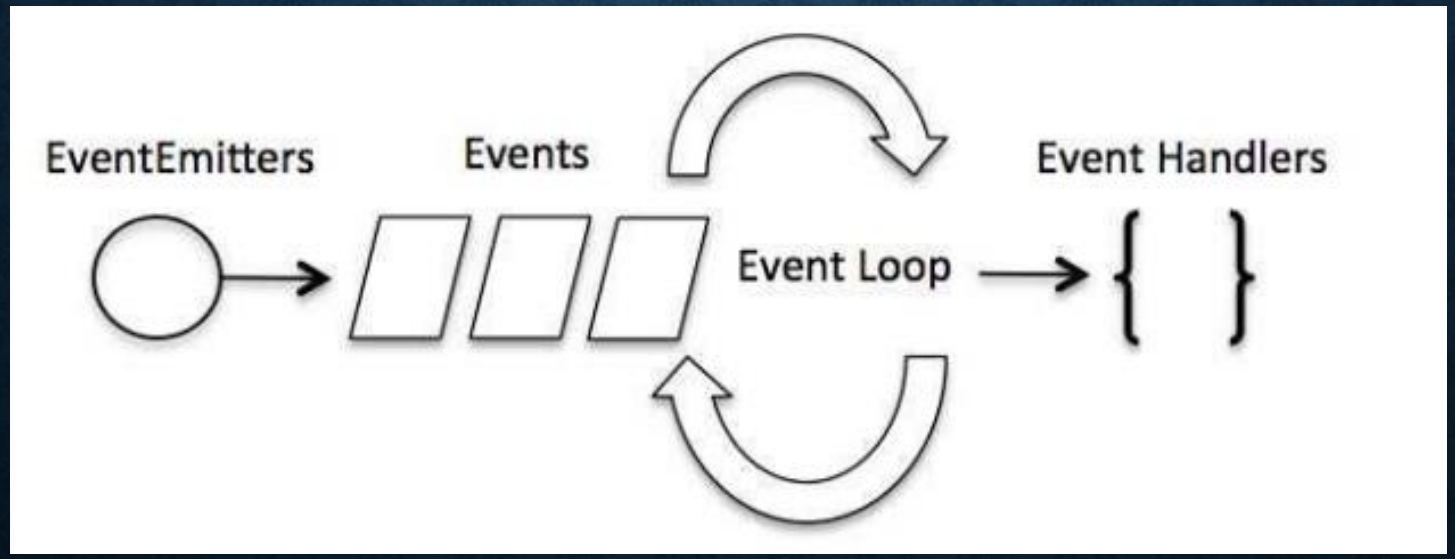
NODEJS – EXAMPLE (NON-BLOCKING)

- Below is a example of how you would read the contents of a file and print it to the screen in a non-blocking way.

```
var fs = require("fs");
fs.readFile('input.txt', function (err, data)
{
    if (err)
        return console.error(err);
    console.log(data.toString());
});
console.log("End of file");
```

NODEJS – EVENT LOOP

Node makes use of the Observer design pattern, since the thread keeps watch of when a task gets completed and fires the corresponding event signalling the event-listener function to execute.



NODEJS – EVENT EXAMPLE

```
// Import events module
var events = require('events'); // Create an eventEmitter object var
eventEmitter = new events.EventEmitter(); // Create an event handler var
connectHandler = function connected() {
    console.log('connection successful.'); // Fire the data_received event
    eventEmitter.emit('data_received');
}
// Bind the connection event with the handler
eventEmitter.on('connection', connectHandler);
// Bind the data_received event with the anonymous function
eventEmitter.on('data_received', function(){
console.log('data received successfully.');
});
// Fire the connection event
eventEmitter.emit('connection');
```

NODEJS – EVENTS MODULE METHODS

- addListener(event, listener)
- on(event, listener)
- once(event, listener)
- removeListener(event, listener)
- removeAllListeners([event])
- setMaxListeners(n)
- listeners(event)
- emit(event, [arg1], [arg2], [...])

NODEJS – BUFFERS

- Node provides a buffer class which provides instances to store raw data that corresponds to a raw memory allocation outside the v8 heap.
- There are many ways to create a buffer in node:
  - var buf = new Buffer(100);
  - var buf = new Buffer([10, 20, 30, 40, 50]);
  - var buf = new Buffer("Simply Easy Learning", "utf-8");

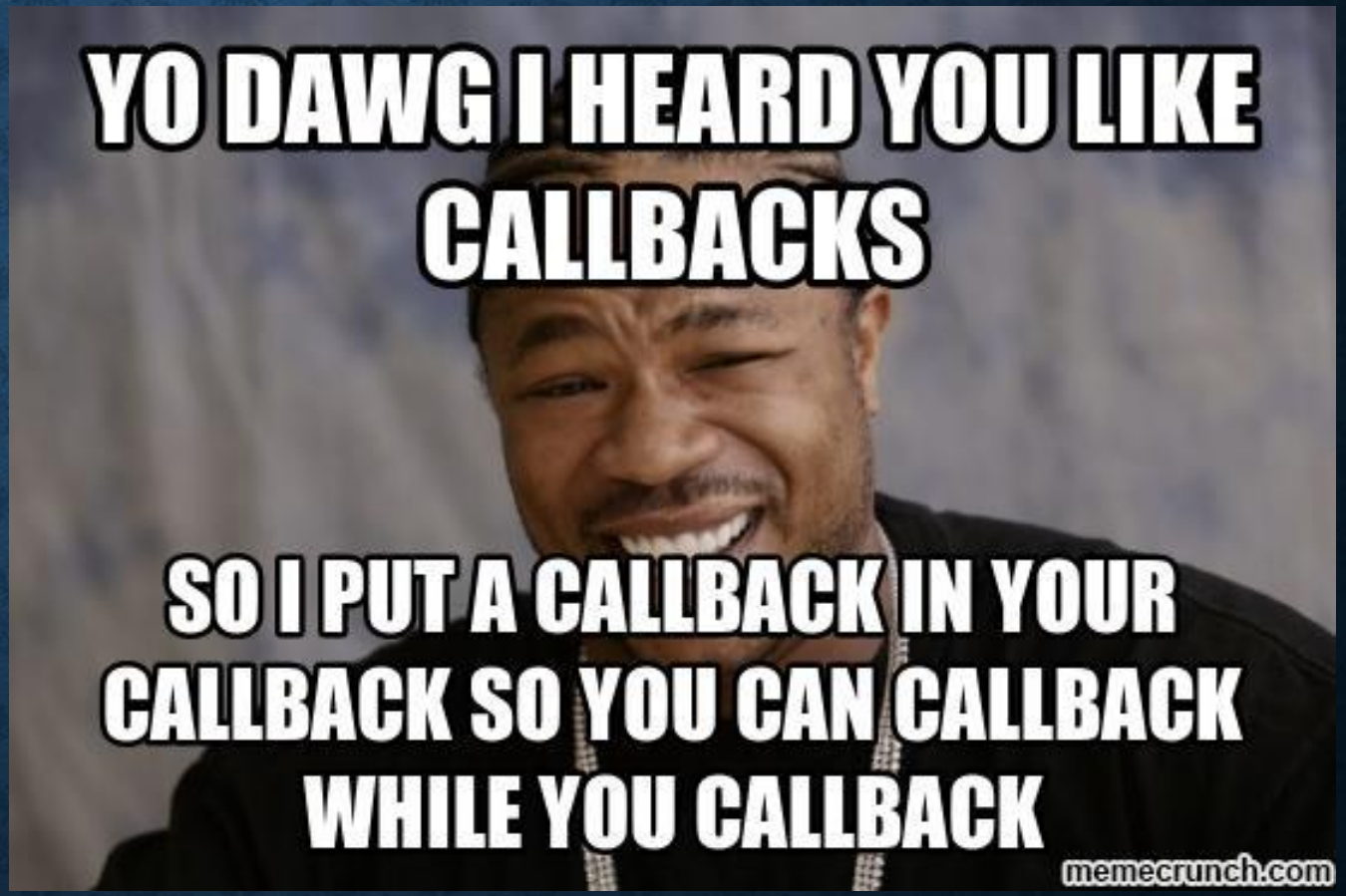
- Buffers have a few features that makes it easier to work with such as converting a buffer to JSON, concatenation of buffers, etc.
- Full list can be found https://nodejs.org/api/buffer.html

NODEJS – STREAMS

- Streams are objects that supports both reading and writing in a continuous way.
- 4 types:
  - Readable
  - Writeable
  - Duplex – Both read and write
  - Transform – a duplex where output is computed based on input (logic)

NODEJS – GLOBAL OBJECTS

- Available in all modules (Global)
- Can be used directly
- Some examples (*note: double underscore (__)*):
  - __filename
  - __dirname
  - setTimeout(callback, milliseconds) // Like how JS does it
  - clearTimeout(timer)
- Other common object are “console” and “process”


YO DAWG I HEARD YOU LIKE CALLBACKS
SO I PUT A CALLBACK IN YOUR CALLBACK SO YOU CAN CALLBACK WHILE YOU CALLBACK
memecrunch.com