

# WEEK 6: NODE.JS



*With exercise and slides support from Hugh Kennedy*

# **DAY 1: NODE 101**

# INSTALLING NODE

Mac/Win <http://nodejs.org/download>

"Other" <http://git.io/wUzU4g>

# WHAT IS NODE.JS?

Javascript Runtime built on Chrome's V8 engine  
(it is **not** a framework)

- Written in C/C++
- Evented, non-blocking I/O

# PREFECT FOR:

- Data (I/O) Intensive Applications
- Mobile Backend (Push notifications)
- Real-time Streaming Applications (chat, games, etc.)
- API 'glue' (connect many services together)
- **NOT** CPU intensive applications

# ORIGIN STORY

*Turns out, a lot of the frameworks were designed in a way that they made the assumption a request -> response is something that happens instantaneously and that your entire web development experience should be abstracted as a function. You get a request, you return a response. That is the extent of your context.*

-- Ryan Dahl (creator of Node.js)

*Node was originally born out of this problem - how can you handle two things at the same time? Non-blocking sockets is one way. Node is more or less the idea, or exploring the idea: what if everything was non-blocking? What if you never waited for any IO to happen?*

original presentation



history lesson



# BUT WHAT /S NODE?!??!

*Node.js® is a platform built on Chrome's JavaScript runtime for easily building fast, scalable network applications. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices.*

**WAIT, WHAT?**







®

open source  
initiative



**Paul Irish**

@paul\_irish



Following

SimCity's UI is completely written in JavaScript running on V8 & WebKit:  
[twitter.com/MaxisScott/sta...](https://twitter.com/MaxisScott/status/258143111111111111)  
[news.ycombinator.com/item?id=5359143](http://news.ycombinator.com/item?id=5359143)



RETWEETS

708

FAVORITES

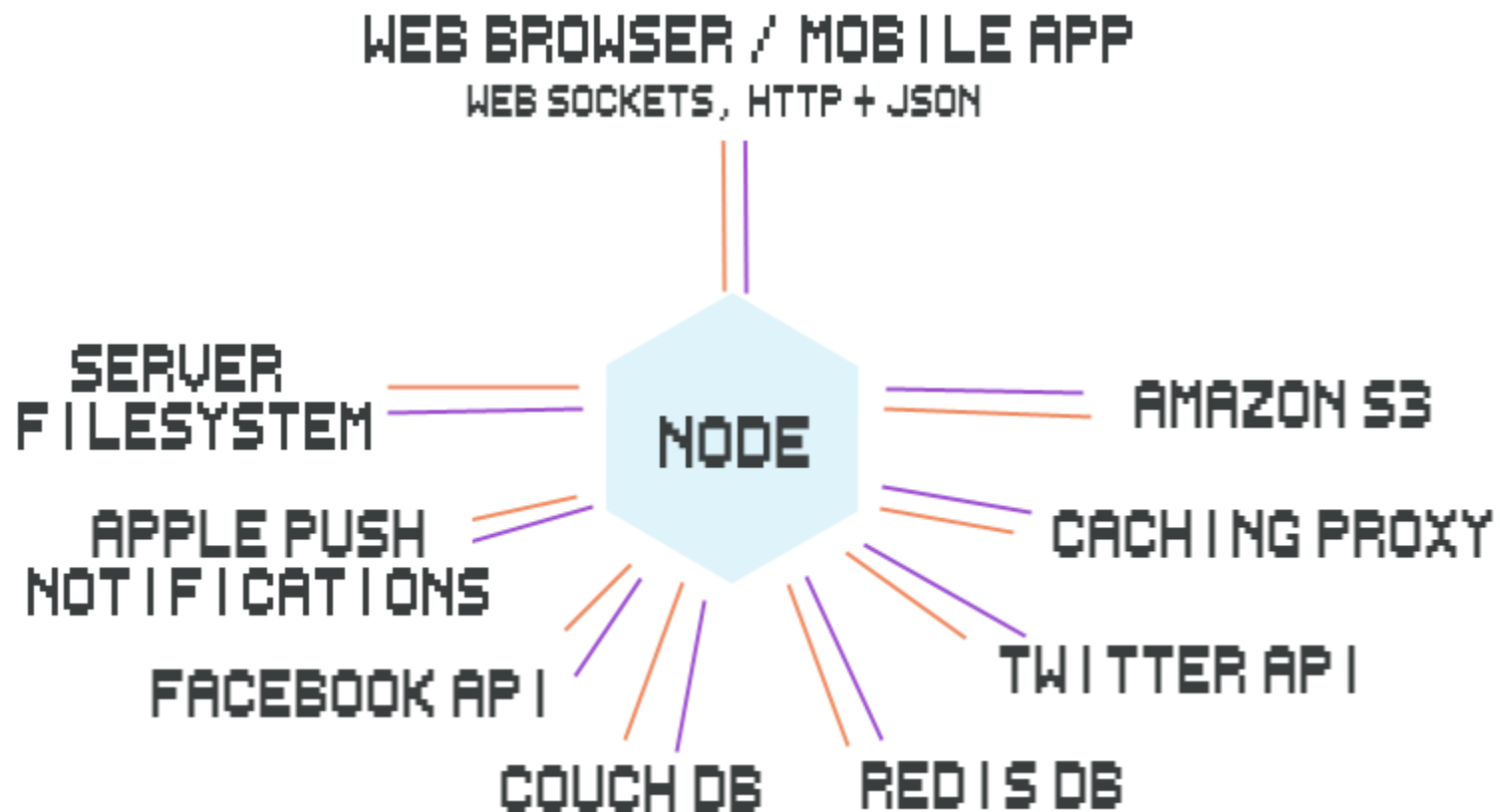
314



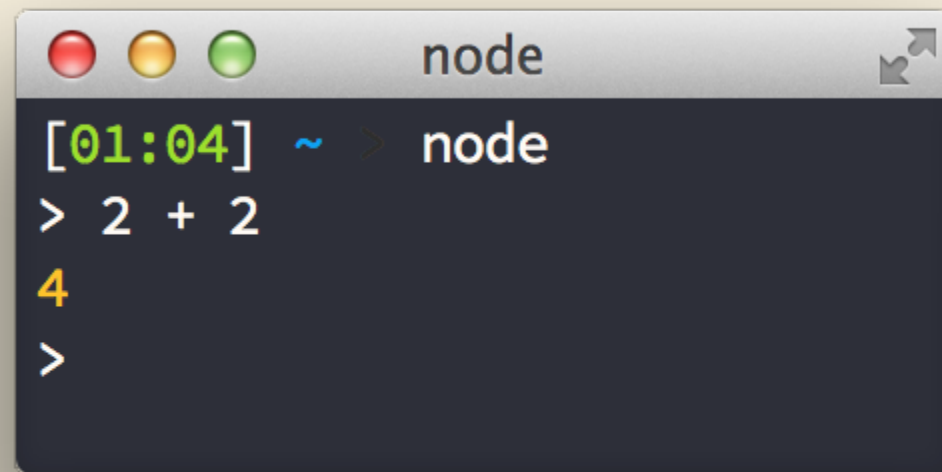
3:25 PM - 12 Mar 2013

**NODE IS ESSENTIALLY V8, WITH SOME ADDITIONS SO THAT YOU  
CAN USE IT TO WRITE SERVERS.**

# INPUT/OUTPUT







```
[01:04] ~ > node  
> 2 + 2  
4  
>
```

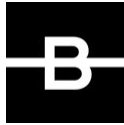
A terminal window titled "node" with standard macOS window controls (red, yellow, green buttons) and a maximize button. The terminal content shows a timestamped prompt "[01:04] ~", a command prompt ">", the command "node", and a subsequent calculation "2 + 2" resulting in "4". The prompt ">" is repeated on the next line.

# **SID LEE DASHBOARD**

# WHO USES NODE?

- PayPal
- NASA
- Mapbox
- Yahoo
- Uber
- Walmart
- Facebook

**DO NOT TOUCH**



## EELS 3D projection mapping multiplayer game

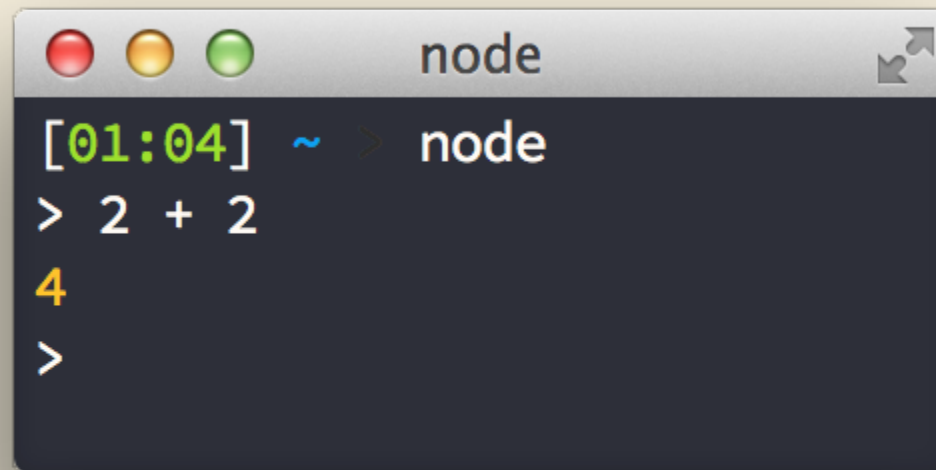
from **B-Reel** PRO

02:08



] **HD**

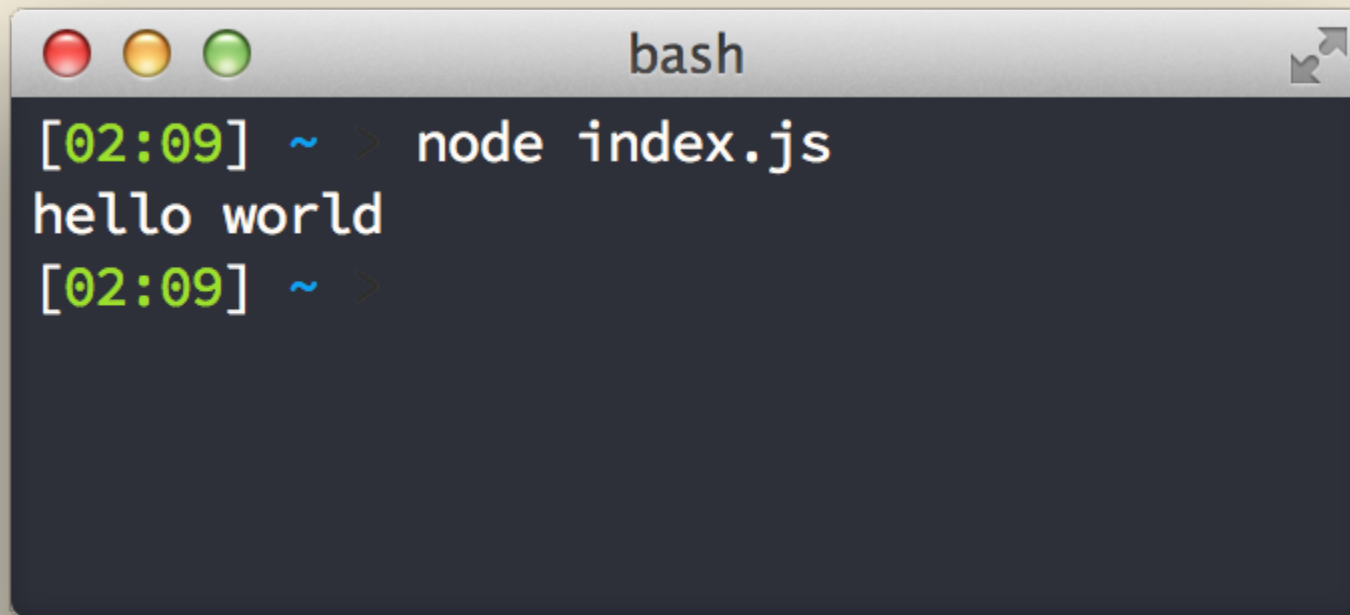
# RUNNING NODE



A terminal window titled "node" with standard macOS window controls (red, yellow, green buttons). The terminal shows a prompt where the user has entered "node". The prompt is "[01:04] ~ >". The user has entered "2 + 2" and the terminal has outputted "4". The prompt is now ">".

```
[01:04] ~ > node  
> 2 + 2  
4  
>
```

```
// index.js  
console.log('hello world')
```



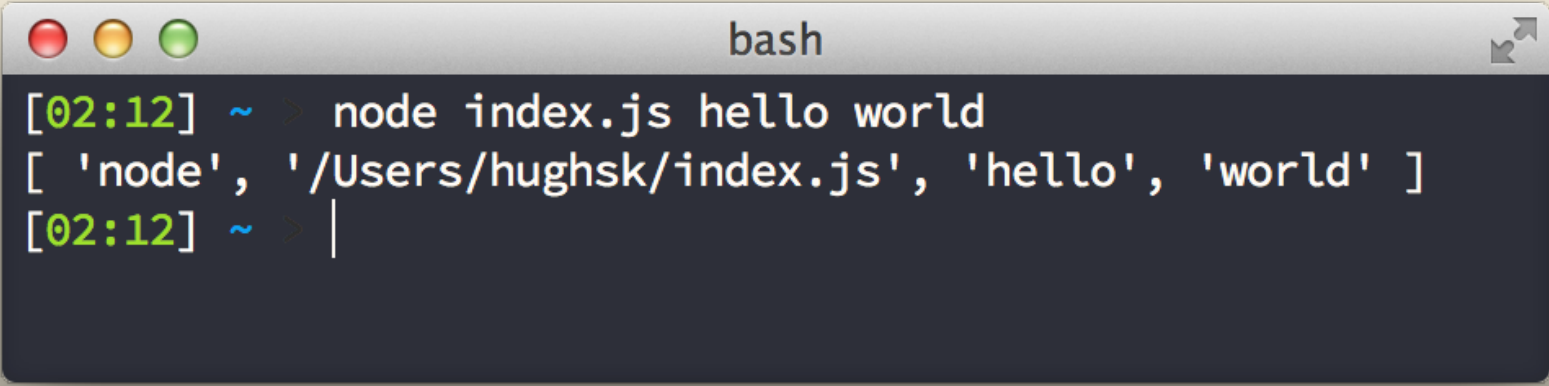
A terminal window titled "bash" with standard macOS window controls (red, yellow, green buttons) and a maximize button. The terminal has a dark blue background and displays the following text:

```
[02:09] ~ > node index.js  
hello world  
[02:09] ~ >
```



# PROCESS.ARGV

```
// index.js  
console.log(process.argv)
```



```
bash
[02:12] ~ > node index.js hello world
[ 'node', '/Users/hughsk/index.js', 'hello', 'world' ]
[02:12] ~ > |
```

A terminal window titled "bash" with standard macOS window controls (red, yellow, green buttons) and a close button. The terminal shows a command prompt where the user has entered "node index.js hello world". The output is an array: ["node", "/Users/hughsk/index.js", "hello", "world"]. The prompt is now waiting for the next input, indicated by a vertical bar.



**NPM MAKES NODE GREAT**

**ANYONE CAN INSTALL**

**ANYONE CAN PUBLISH**

**>125,000 PACKAGES**

**>200 PACKAGES/DAY**



- Server Frameworks
- Browser Libraries
- Build Tools
- Web APIs
- Protocols
- Computational Geometry
- Image/Audio Processing
- 3D Graphics
- Scientific Computing
- Databases
- Statistical Analysis

```
$ npm install stats-lite
```

```
$ ls -l node_modules  
stats-lite/
```

```
// index.js
var stats = require('stats-lite')
var stdev = stats.stdev([1, 5, 6, 1, 2, 0])

console.log(stdev)
```

```
$ node index.js  
2.217355782608345
```

# ASSIGNMENT #0

## YOUR FIRST NODE PROGRAM

- It should take two numbers on the command line as arguments
- multiply them together
- log the result out to the console.

**WHY NODE?**

**NODE IS GOOD AT ASYNCHRONOUS  
PROGRAMMING**



# BLOCKING I/O

```
var data = fs.readFileSync('index.js', 'utf8')  
  
console.log(data)
```

Class	Operation	Time cost
Memory	L1 cache reference:	1 ns
	L2 cache reference:	4 ns
	Main memory reference:	100 ns
I/O	SSD random-read:	16,000 ns
	Round-trip in same datacenter:	500,000 ns
	Physical disk seek:	4,000,000 ns
	Round-trip from US to EU:	150,000,000 ns

**NON-BLOCKING I/O**

```
fs.readFile('index.js', 'utf8', function(err, data) {  
  if (err) throw err  
  console.log(data)  
})  
  
console.log('this happens straight away!')
```

# EXAMPLES

- Interactive Sites
- Real time communication
- Games!

**GAMES**

# WORDSQUARED

Word<sup>2</sup> TaraHarkon Logout ★ 40 +

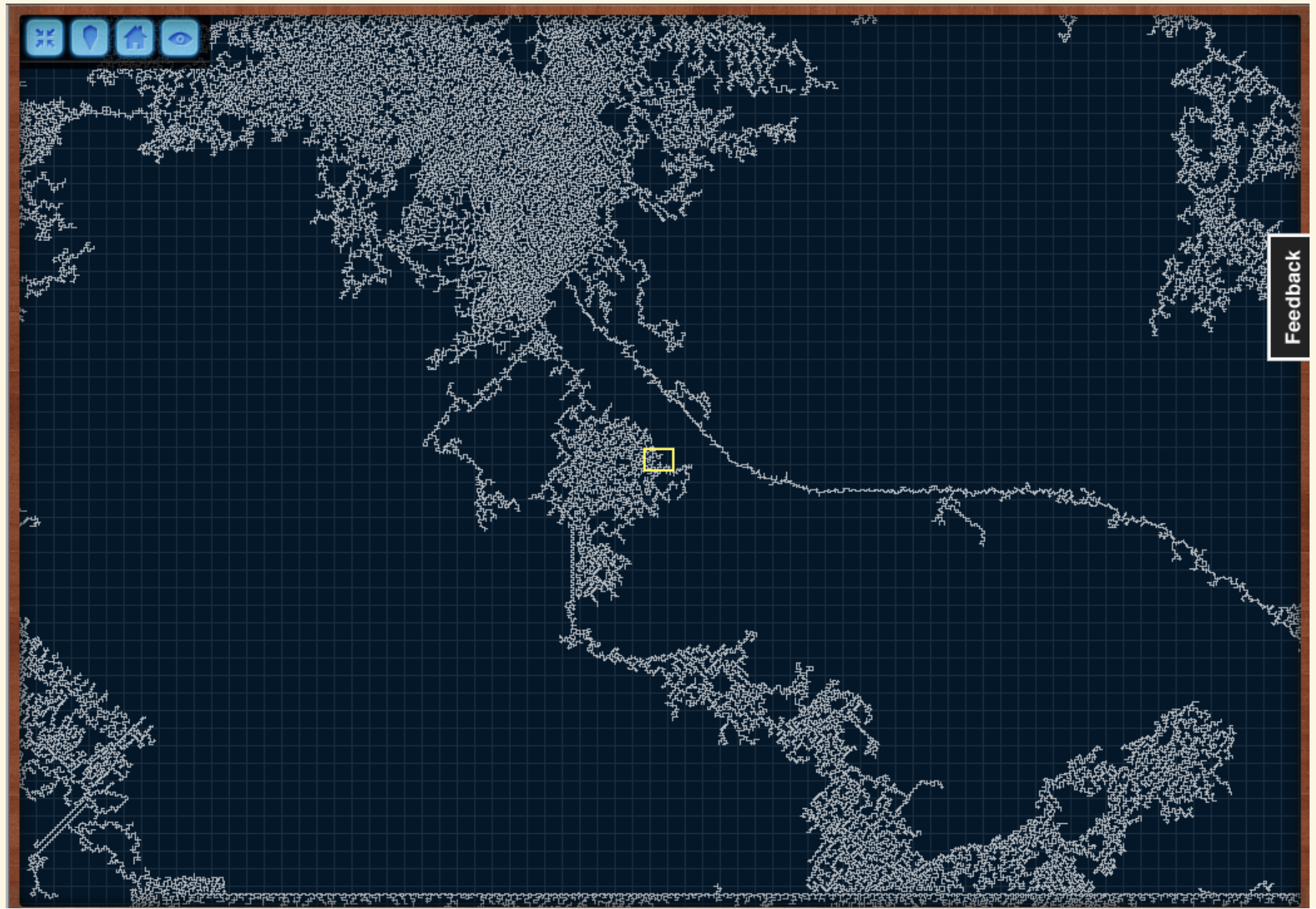
Like us! Tweet 6,544 Like 11k 12

Feedback

Level 1 12 / 60 until next level

Shuffle Swap Tiles Teleport

# WORDSQUARED





# BROWSER QUEST



source code on Github

# EVENTED? SAY WHAT?!?

Imagine a food cart... your favorite food cart! On a nice little  
Wednesday



# WAITING IS HARD IN JAVASCRIPT!

```
# Ruby

print("Hello")

sleep(5)

print("world!")
```

```
// node.js

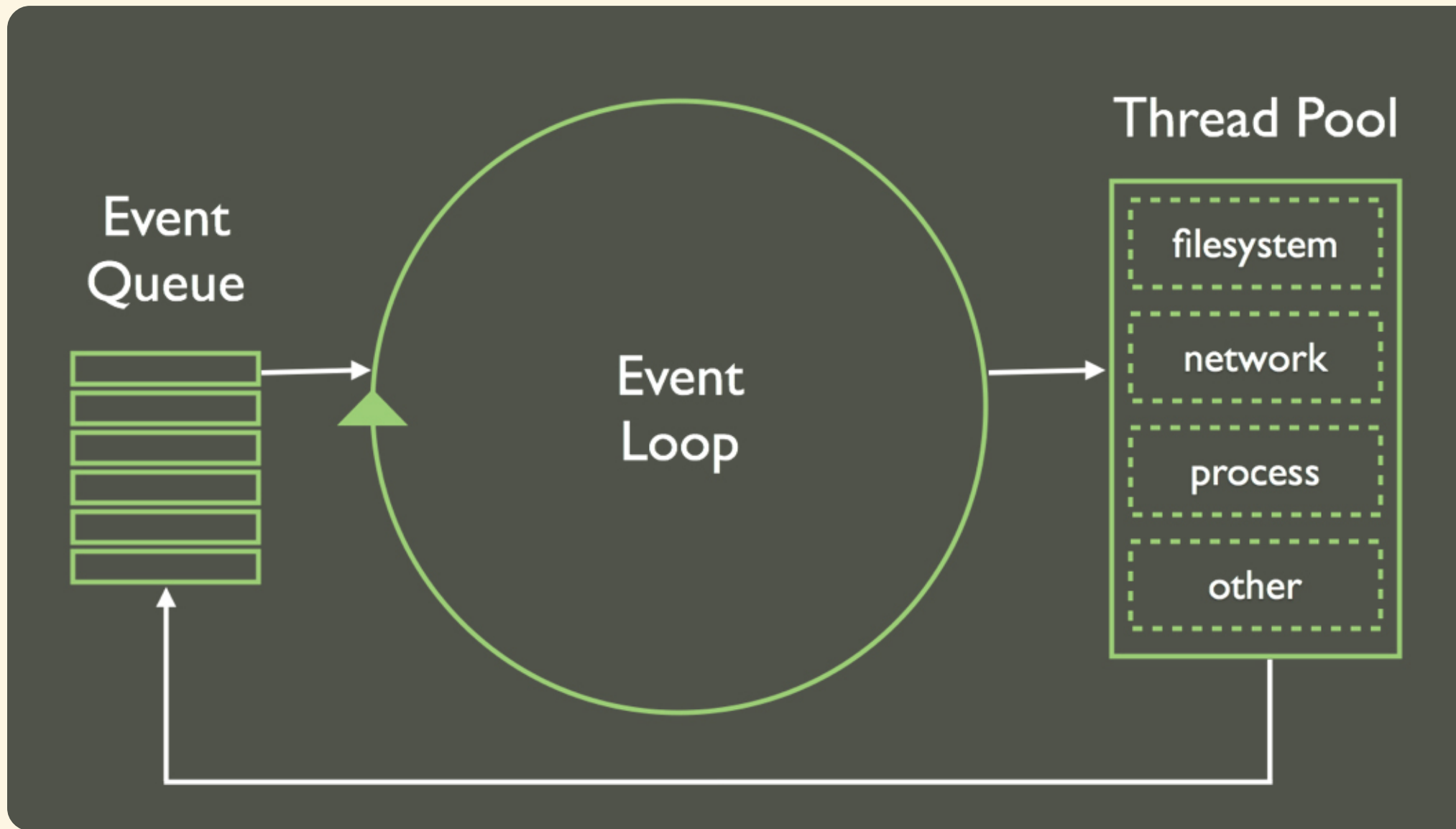
console.log("Hello");

setTimeout(function() {
  console.log("world!");
}, 5000);
```

# THE EVENT LOOP

```
while (true) {  
  // process all scheduled events and callbacks  
  eventQueue.forEach(process)  
  
  ...  
  // other code and logic  
  ...  
  
  // advance time one tick  
  tick++;  
}
```

Understanding the Node.js Event Loop



# RESULT: VERY CONCURRENT PROCESSES

Like your operating system multiplexes limited resources:

- Hard Disk I/O
- Network I/O
- CPU

**WHEN RESOURCES BECOME AVAILABLE, NODE USES THEM.**

**WHEN UNAVAILABLE, NODE PATIENTLY SCHEDULES FOR THEM  
(EVENT-LOOP)**

# CONCURRENT !== PARALLEL

```
// Fibonacci Sequence Generator

function f(n) {
  var start = 0;

  if(n == 0) return(s);

  if(n == 1) {
    s += 1;
    return(s);
  }
  else {
    return(f(n - 1) + f(n - 2));
  }
}
```



# WHY ASYNCHRONY :: WHY JAVASCRIPT

- Because that's how I/O should be done! OS has done it for decades, why can't web servers do it too.
- V8 has become one of the most advanced VMs out there
  - Google supports it, and the Browser Wars mean it will only get better
- Javascript is ubiquitous, most know and if not it is "easy" to learn.
- First-class functions and closures fit the callback model.

# CALLBACKS

```
var delayed = function(callback) {  
  setTimeout(function() {  
    callback(null, { ok: true })  
  }, 1000)  
}
```

```
delayed(function(err, result) {  
  if (err) throw err  
  
  console.log(result.ok)  
})
```

# **NPM INSTALL GIPHY**

**PLAY AROUND WITH GIPHY'S API IN NODE**

```
var token = 'dc6zaTOxFJmzC'  
var giphy = require('giphy')(token)  
  
giphy.translate({}, function(err, data) {  
  if (err) throw err  
})
```

# ASSIGNMENT #1

## GIPHY TRANSLATE TOOL

Use Giphy's "translate" API to create a script that outputs GIF based on user input. If you have extra time to spare, try jumping ahead and making a web server that serves these images to a browser based on the specified URL.

# **ASYNCHRONOUS PATTERNS**

# EVENTEMITTERS

```
var emitter = new EventEmitter

setInterval(function() {
  emitter.emit('data', Math.random())
}, 1000)
```

```
emitter.on('data', function(result) {
  console.log(result) // Random numbers
})
```

# STREAMS

*We should have some ways of connecting programs like garden hose – screw in another segment when it becomes necessary to massage data in another way. This is the way of IO also.*

Doug McIlroy. October 11, 1964



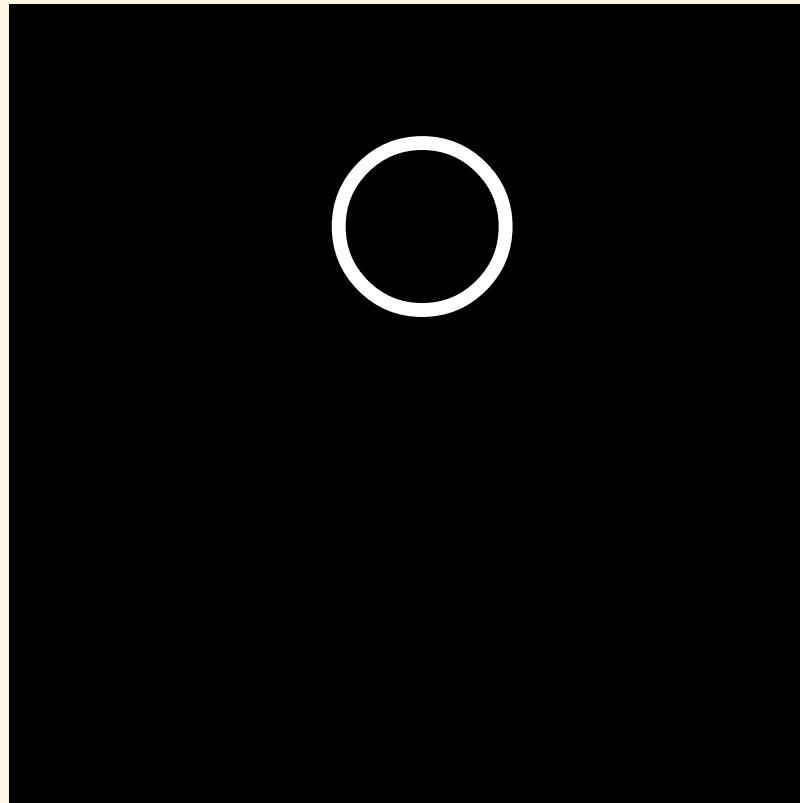
```
# Get unique values from column 1 of a .csv file
# Write output to index.txt
$ cat index.csv | cut -d, -f1 | uniq > index.txt
```

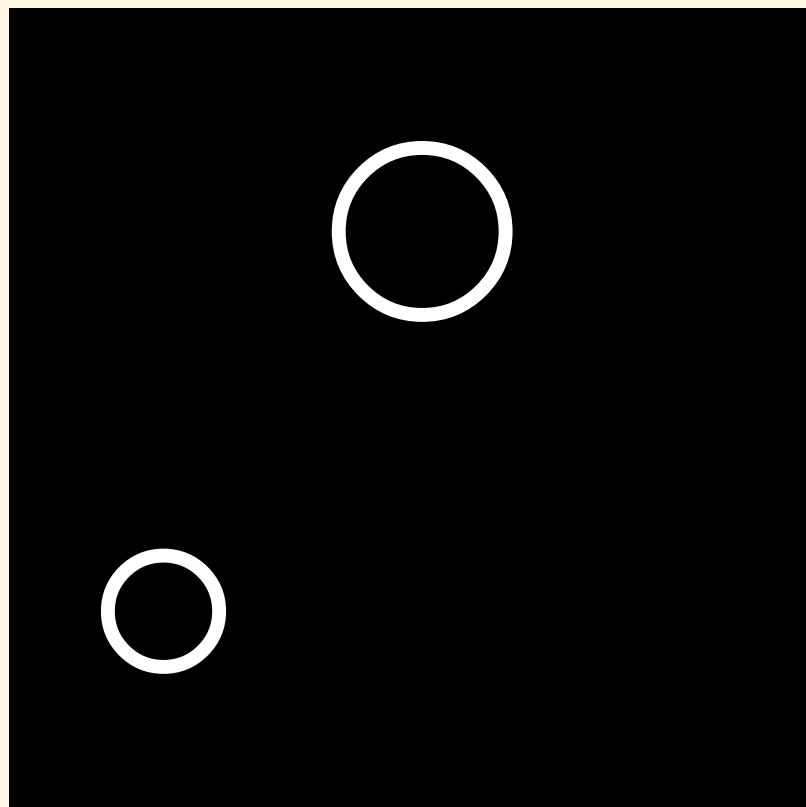
```
fs.createReadStream('index.csv')
  .pipe(split('\n')) // Split on newlines
  .pipe(csv2(','))   // Split each line into values (-d,)
  .pipe(pick(0))      // Pick the first value (-f1)
  .pipe(unique())     // Only output unique values (uniq)
  .pipe(join('\n'))  // Add the newlines back into your output
  .pipe(fs.createWriteStream('index.txt'))
```

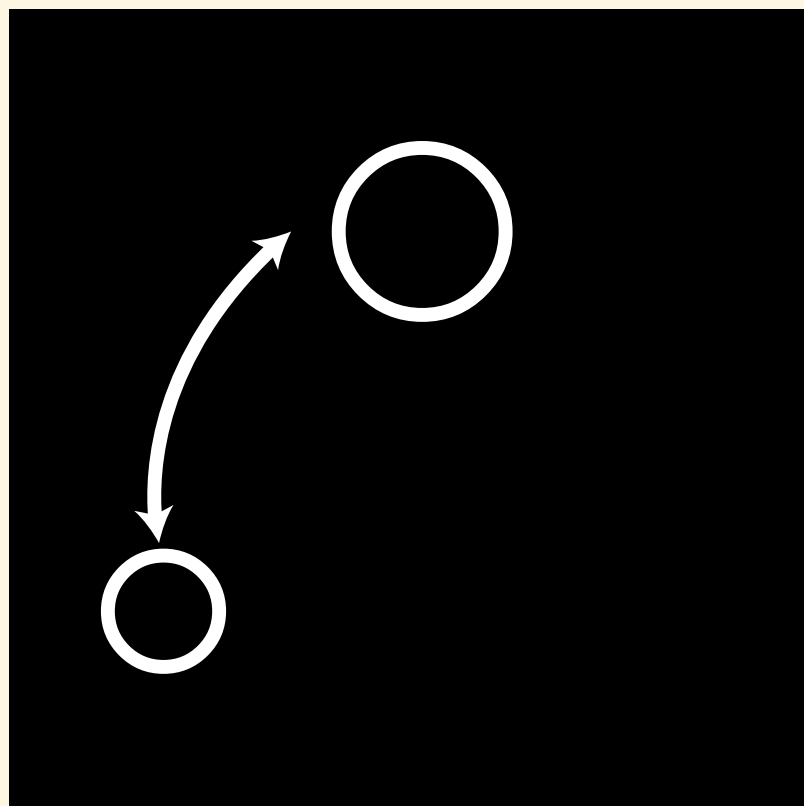
**STREAMS ARE HARD TO WRITE**

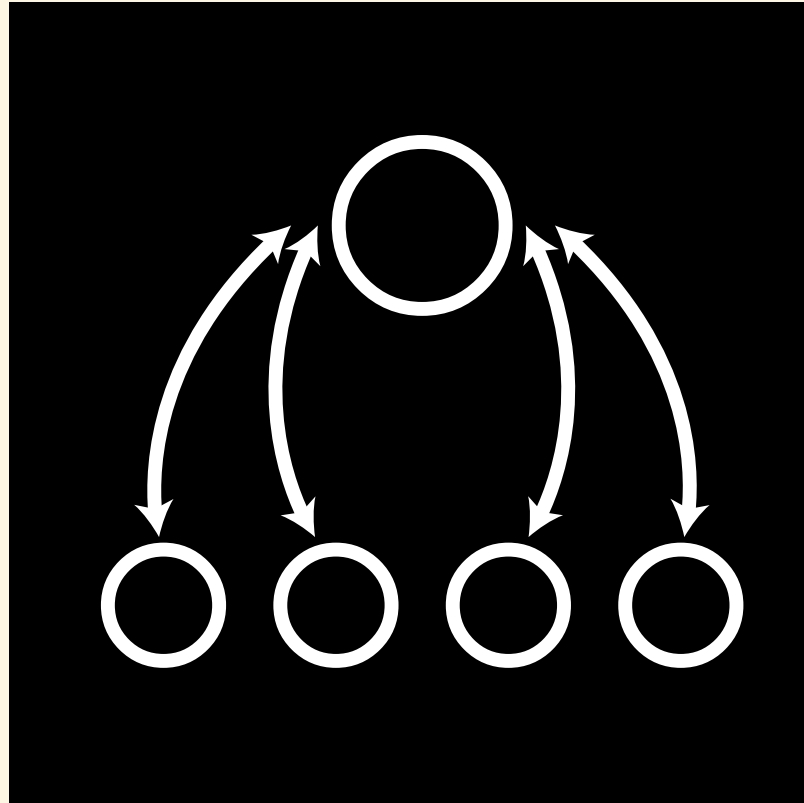
**BUT EASY TO USE**

# SERVERS AND CLIENTS









# TCP

**A "BASELINE" FOR RAW DATA**



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

var server = net.createServer(function(connection) {
  connection.write('hello\n')
  connection.write('world\n')
  connection.end()
})

server.listen(9999, function(err) {
  if (err) throw err
  console.log('telnet localhost 9999')
})
```



A terminal window titled "bash" with four tabs: "bash", "node", "bash", and "node". The active tab is "bash". The terminal output shows a telnet connection to localhost on port 9999. The connection is successful, and the remote host sends the message "hello world". The connection is then closed by the foreign host.

```
[13:42] ~ > telnet localhost 9999
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
hello
world
Connection closed by foreign host.
[13:42] ~ > |
```

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

var client = net.connect({
  host: 'localhost',
  port: 9999
})

client.pipe(process.stdout)
```

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

var client = net.connect({
  host: 'localhost',
  port: 9999
})

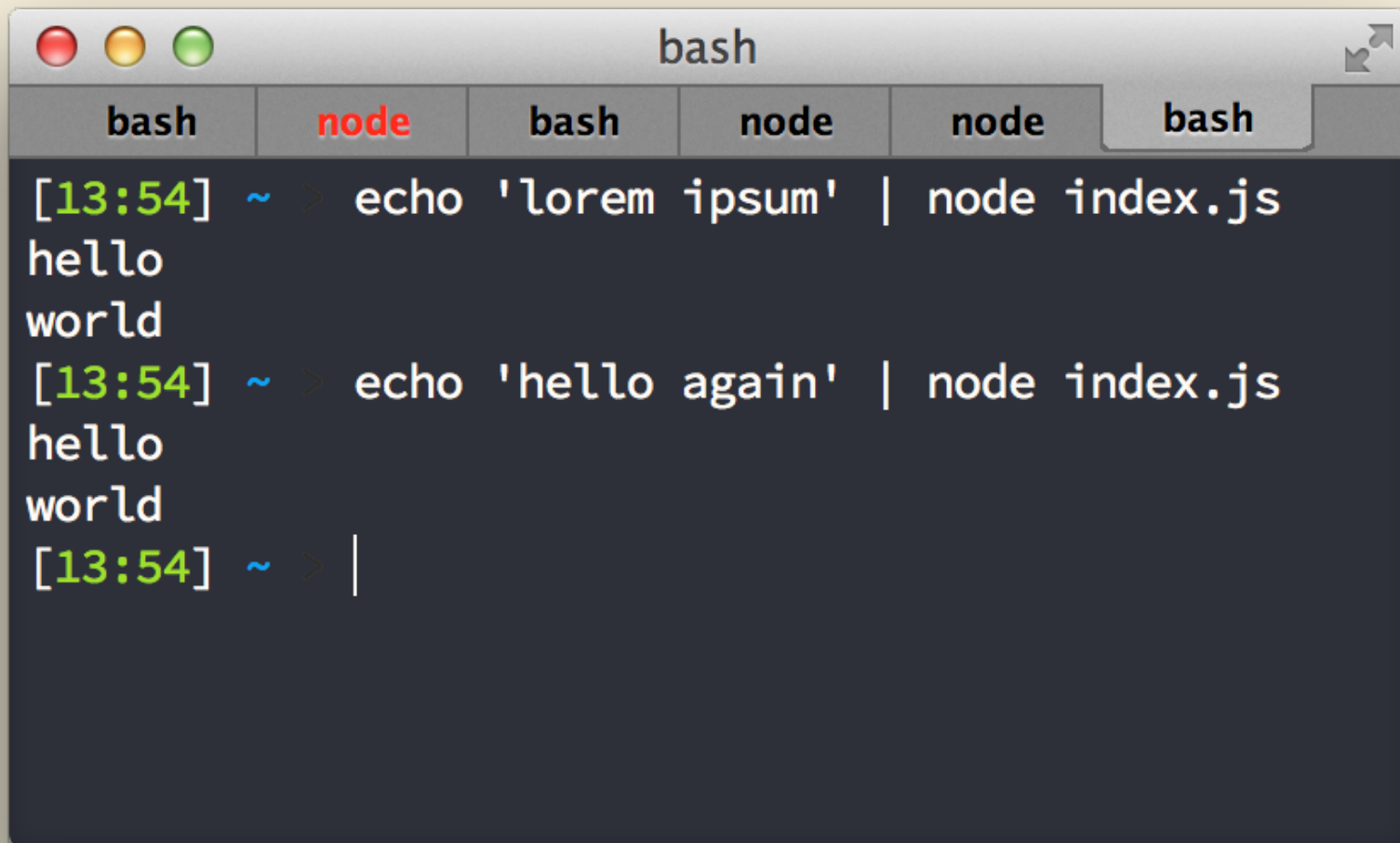
process.stdin
  .pipe(client)
  .pipe(process.stdout)
```

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

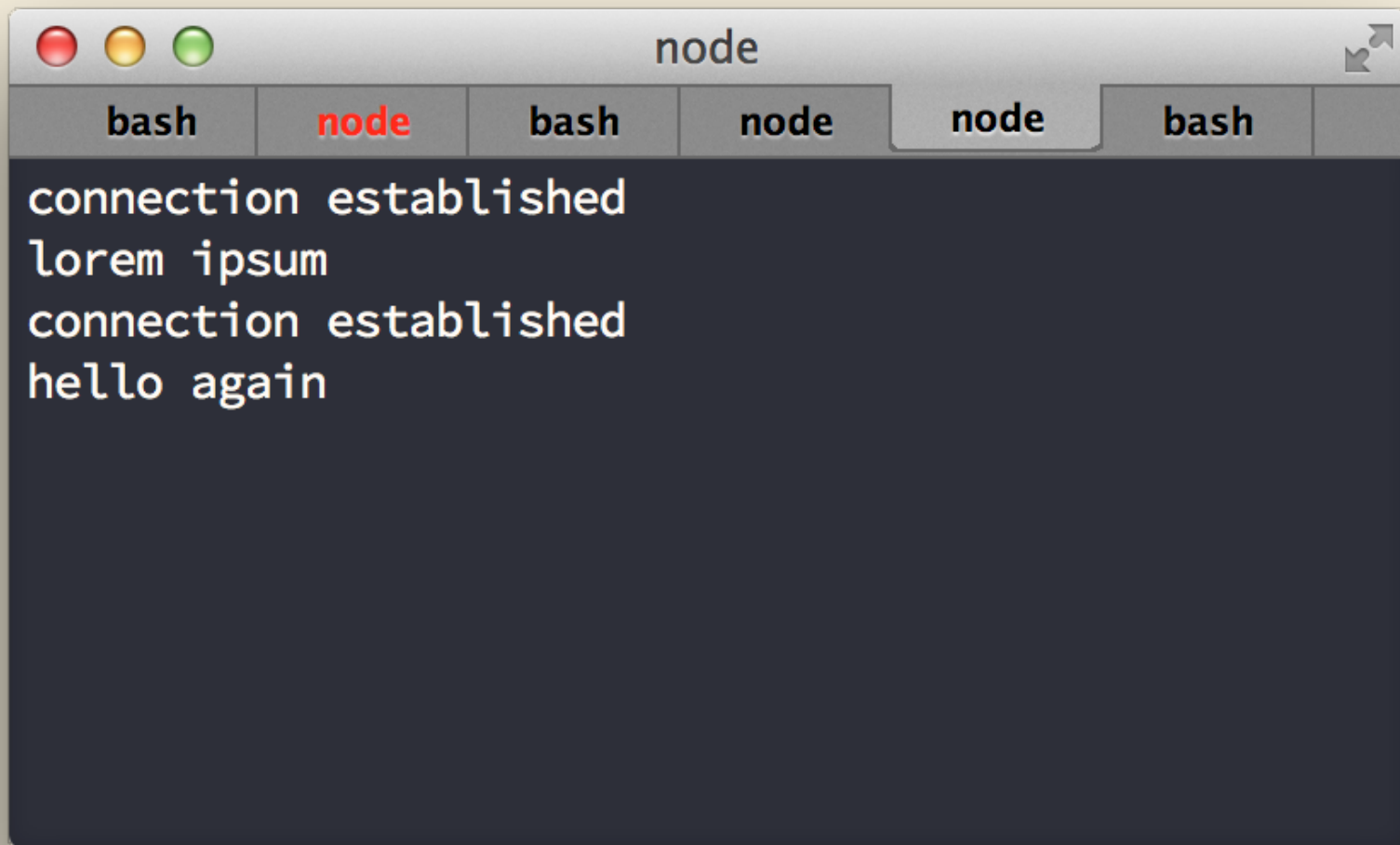
var server = net.createServer(function(connection) {
  console.log('connection established')
  connection.write('hello\n')
  connection.write('world\n')

  connection.pipe(process.stdout)
})

server.listen(9999, function(err) {
  if (err) throw err
  console.log('telnet localhost 9999')
})
```

A terminal window with a title bar containing three colored window control buttons (red, yellow, green) and the text "bash". Below the title bar are several tabs labeled "bash", "node" (in red), "bash", "node", "node", and "bash". The main area of the terminal is dark blue and contains the following text:

```
[13:54] ~ > echo 'lorem ipsum' | node index.js  
hello  
world  
[13:54] ~ > echo 'hello again' | node index.js  
hello  
world  
[13:54] ~ > |
```



# ASSIGNMENT #2

## CREATE A TCP CHAT SERVER/CLIENT

- You should be able to start the server using *node server.js*, and have it listen for incoming connections in the background.
- The client may send messages using *node client.js <server-address> <message>*.
- Try adding some functionality to the server or client: colored text, emoji support, ASCII art, are some fun examples.
- Try making the chat client persistent: that is to say, you can start it up once and continue typing messages without having to restart the process.

# THE ECOSYSTEM

Framework/Language	Strengths	Weaknesses
Server Side MVC (Rails/Django)	Full stack frameworks (all encompassing). Rapidly prototype complex apps or build from the ground up.	Monolithic and very heavy. Hard to swap out components to use a different library (walled garden)
Node.js (Express/Connect)	The "pipes". Great for connecting components and apps with multiple interacting services	Often too low level for complex workflows and heavy CRUD applications.
Client Side MVC (Backbone, Angular, Ember)	Very responsive and interactive. Manages complex transition and state on the client. Often coupled with a server framework.	Only lives on the client so unless you persist state with Ajax or browser storage, changes and updates are all forgotten on page navigation.



# PUTTING IT ALL TOGETHER

- Node is the "pipes" - really shines when you have to connect multiple components => Service oriented architecture
- The asynchronous nature makes it very efficient at handling multiple independent processes.
- But the single thread of execution means it is bad at computing in parallel in its context. Offshore it!