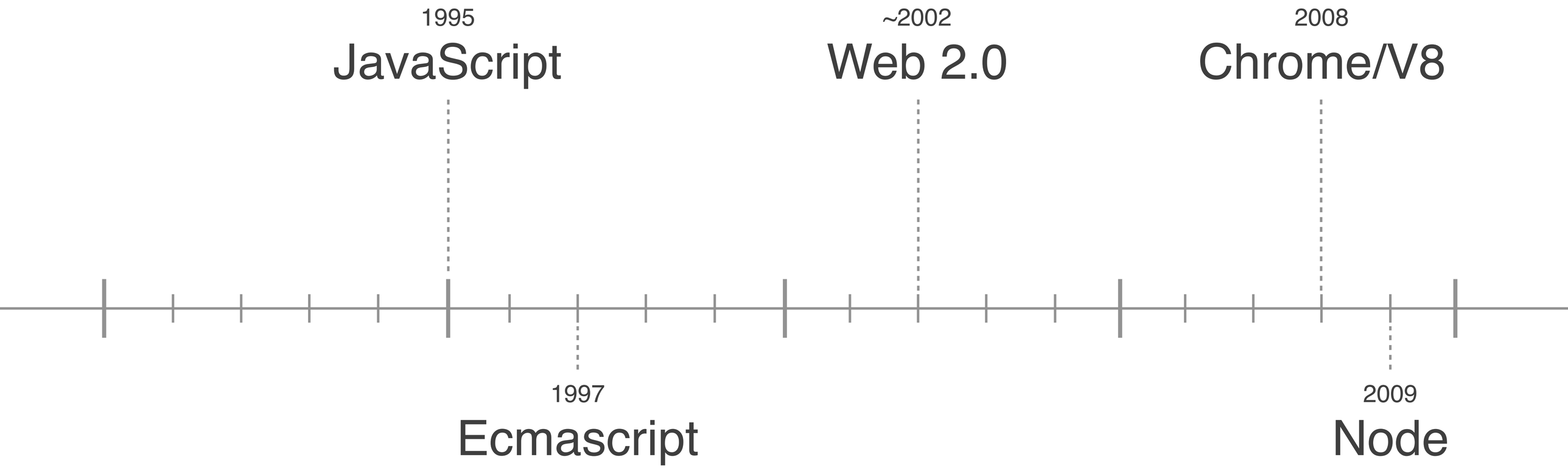
```
NODE.INTRO(function (err, ideas) {
    if (err) throw new Question(err)
    else understand(ideas)
})
```

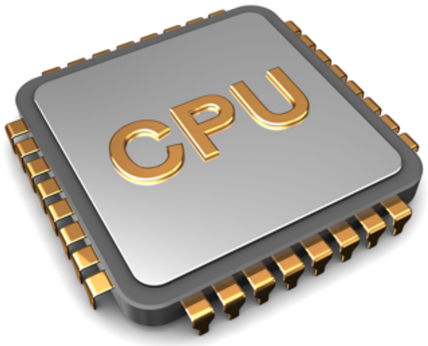
# BACKGROUND

# TIMELINE

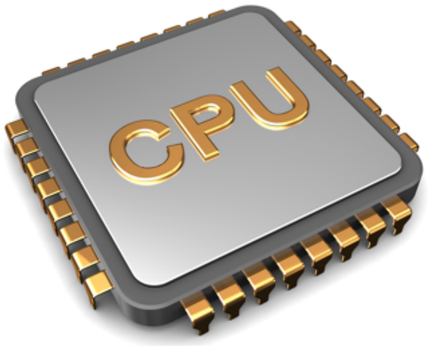

1995
JavaScript
~2002
Web 2.0
2008
Chrome/V8
1997
Ecmascript
2009
Node

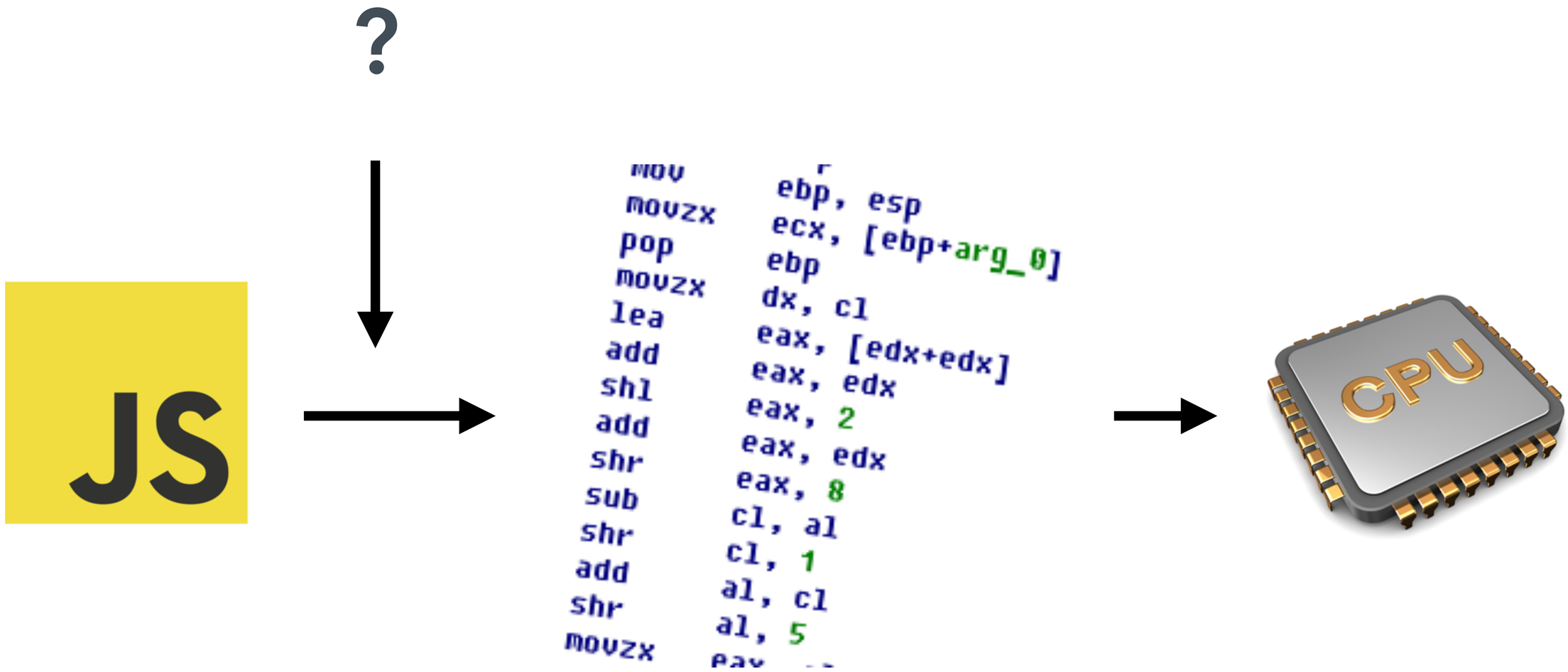

- JS runtime based library
- Built on Chrome's V8 JavaScript engine
- Event-driven
- Non-blocking I/O model
- Executes JavaScript on an operating system, instead of a browser

# HOW DO WE RUN ANY CODE IN ANY PROGRAMMING LANGUAGE?


CPU

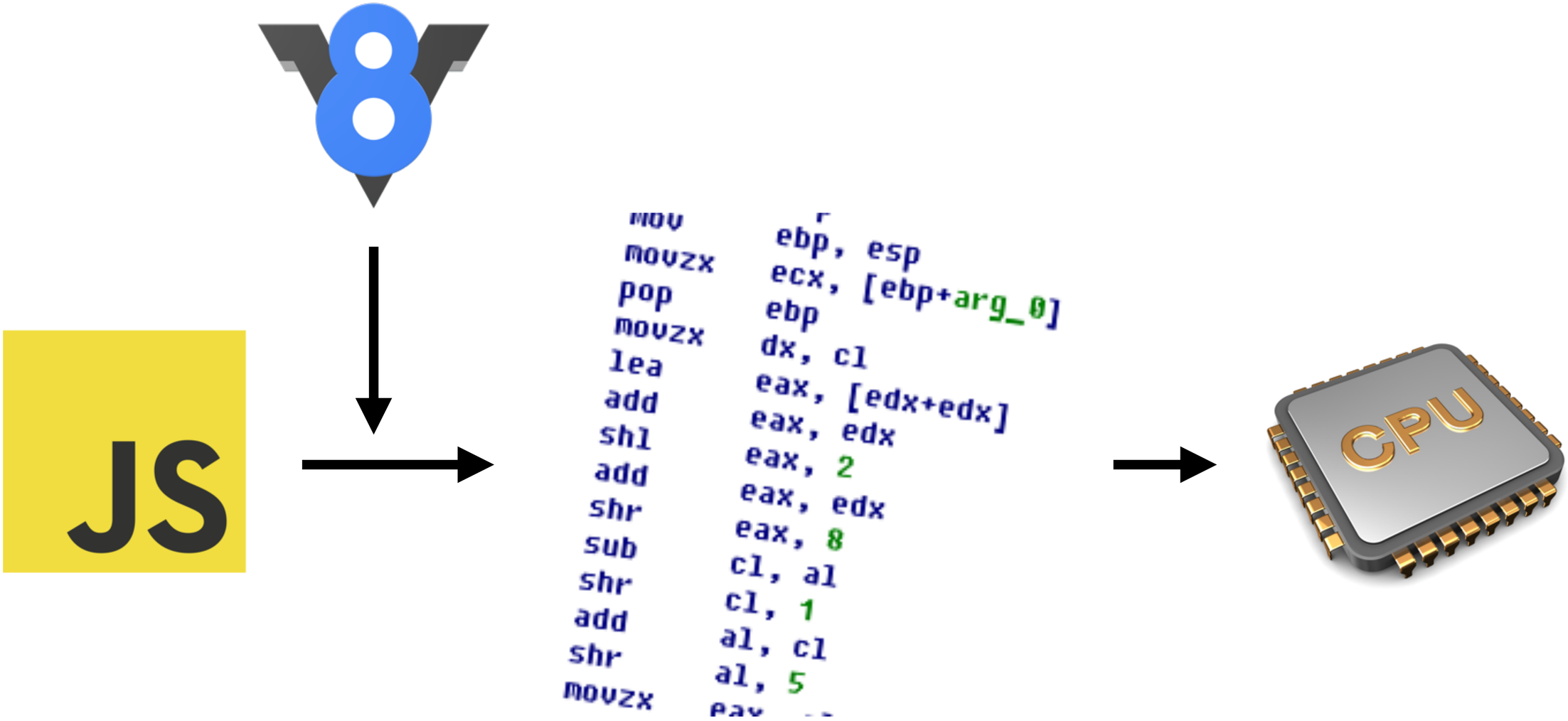
```
mov     ebp, esp
movzx   ecx, [ebp+arg_0]
pop     ebp
movzx   dx, cl
lea     eax, [edx+edx]
add     eax, edx
shl     eax, 2
add     eax, edx
shr     eax, 8
sub     cl, al
shr     cl, 1
add     al, cl
shr     al, 5
movzx   eax
```



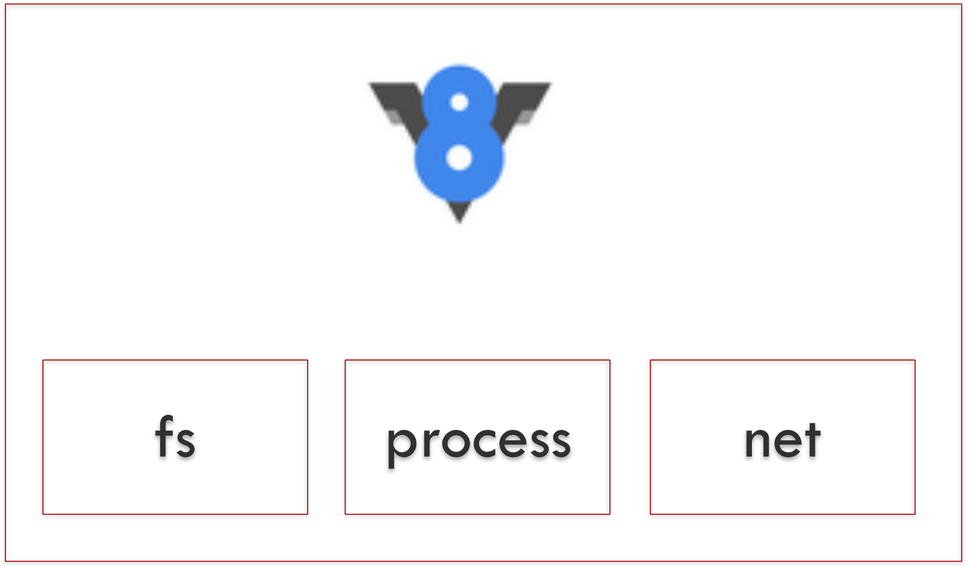

?
JS
mov ebp, esp
movzx ecx, [ebp+arg_0]
pop ebp
movzx dx, cl
lea eax, [edx+edx]
add eax, edx
shl eax, 2
add eax, edx
shr eax, 8
sub cl, al
shr cl, 1
add al, cl
shr al, 5
movzx
CPU

# GOOGLE V8 ENGINE

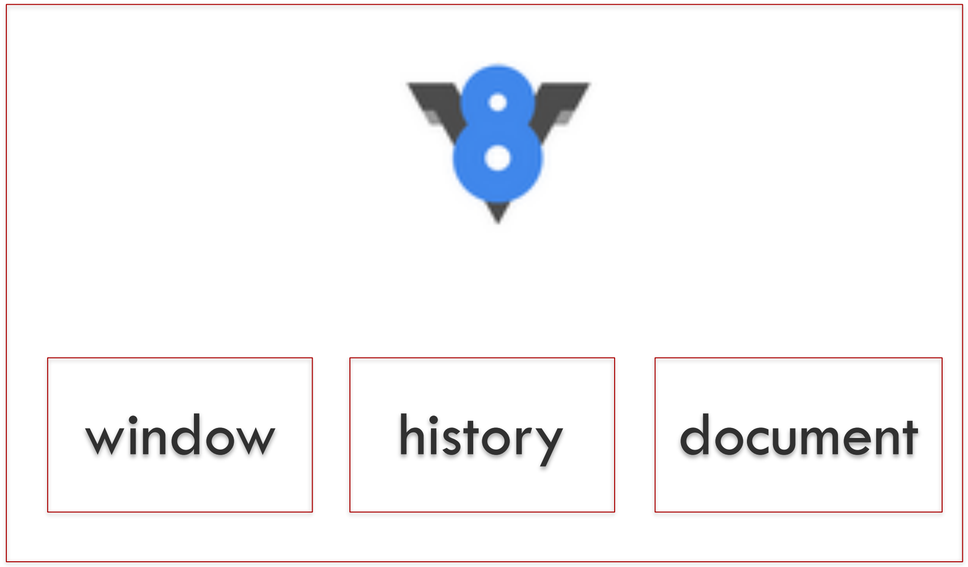
- Open Source
- Written in C++
- Used for client side (Chrome) AND server side (node.js)
- “Understands” (parses and compiles) and executes JS code
- Has to implement new specs (like ES7)


JS
mov ebp, esp
movzx ecx, [ebp+arg_0]
pop ebp
movzx dx, cl
lea eax, [edx+edx]
add eax, edx
shl eax, 2
add eax, edx
shr eax, 8
sub cl, al
shr cl, 1
add al, cl
shr al, 5
movzx
CPU

files (e.g. app.js)




<script></script>



# GOODBYE GLOBALS

- window
- document
- alert/prompt
- history
- etc, etc, etc

# HELLO GLOBALS

- global
- process
- __dirname, __filename
- Buffer
- require/module

# FAMILIAR GLOBALS

- console
- setTimeout / clearTimeout / setInterval / clearInterval

# WHY CARE?

*If you want to create a server and know JavaScript*

# WHY CREATE A SERVER?

*If you want to create a custom website or webapp*

# SERVER

- A program running on a computer connected to the internet
- Serves content requested by remote clients

# IF PROGRAMMING WERE COOKING...

# Program vs. Process

"recipe"

- Program is data
  - text file (can be interpreted)
- Inert — not doing anything
- Ready to be run as a process

- Process is execution "cooking"
  - memory allocated (STATE)
  - CPU performing steps
- "Live"
- Produces results
- Interactive
- Can be started/stopped
- Multiple processes from one program…

# COOKING METAPHOR

| | (term) | (metaphor) |
|---|---|---|
| `log('hi');` | program | recipe |
| JavaScript | programming language | recipe language |
| V8 | engine | chef |
| Node | runtime environment | kitchen |
| Sierra | operating system | building (restaurant?) |

# MODULES

# `module.exports`

- Assign it the data you want to expose
- A `require` of this file will return its `module.exports`

# require

- **Finds a file**
- Executes it
- Imports that file's exports

path → follows that path

no path → tries
`./node_modules`
`../node_modules`
`../../node_modules`
(…etc)

# npm

- **n**ode **p**ackage **m**anager
- Command line tool
- Can find libraries of code online
- Downloads them locally (into `node_modules` directory)
- Keeps list of project dependencies in `package.json`

# `package.json`

***Describes your project, e.g. its dependencies…***

- **Collaboration within your team**
- **Sharing within the node community**

# ASYNCHRONICITY

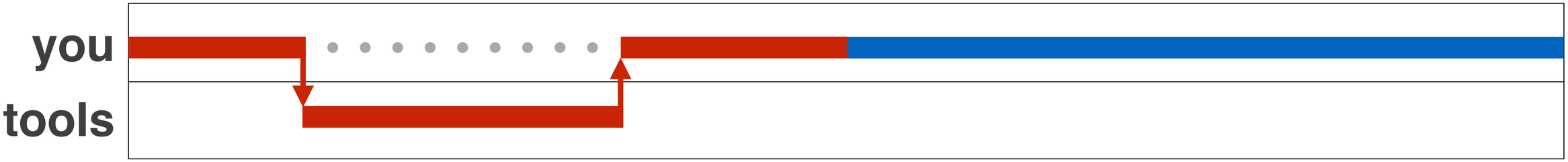
# CONCURRENCY

*"Let's bake a cake"*

1. You only make the icing after the cake comes out of the oven
2. You make the icing while the cake is in the oven
3. I only make the icing and you only make the cake

# CONCURRENCY

*Blocking…*



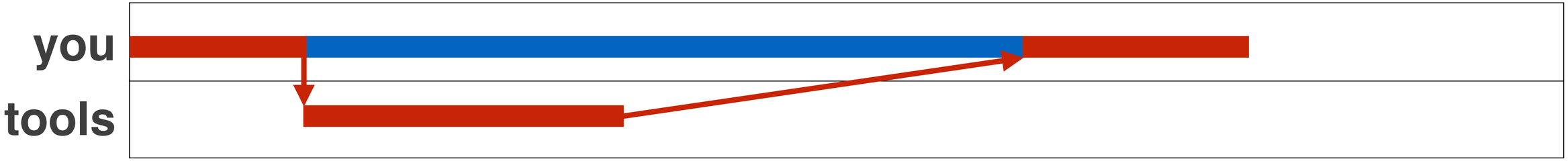

1. You only make the icing after the cake comes out of the oven


— cake

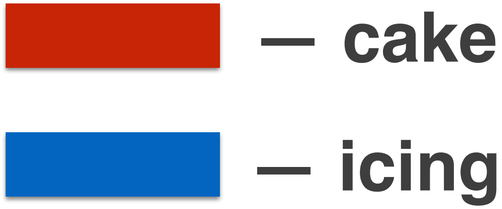

— icing

# CONCURRENCY

*Non-blocking…*



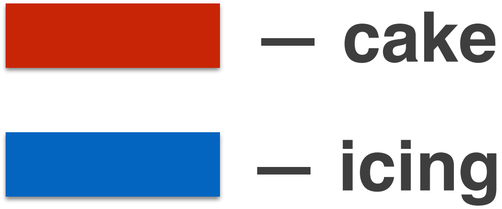

2. You make the icing while the cake is in the oven



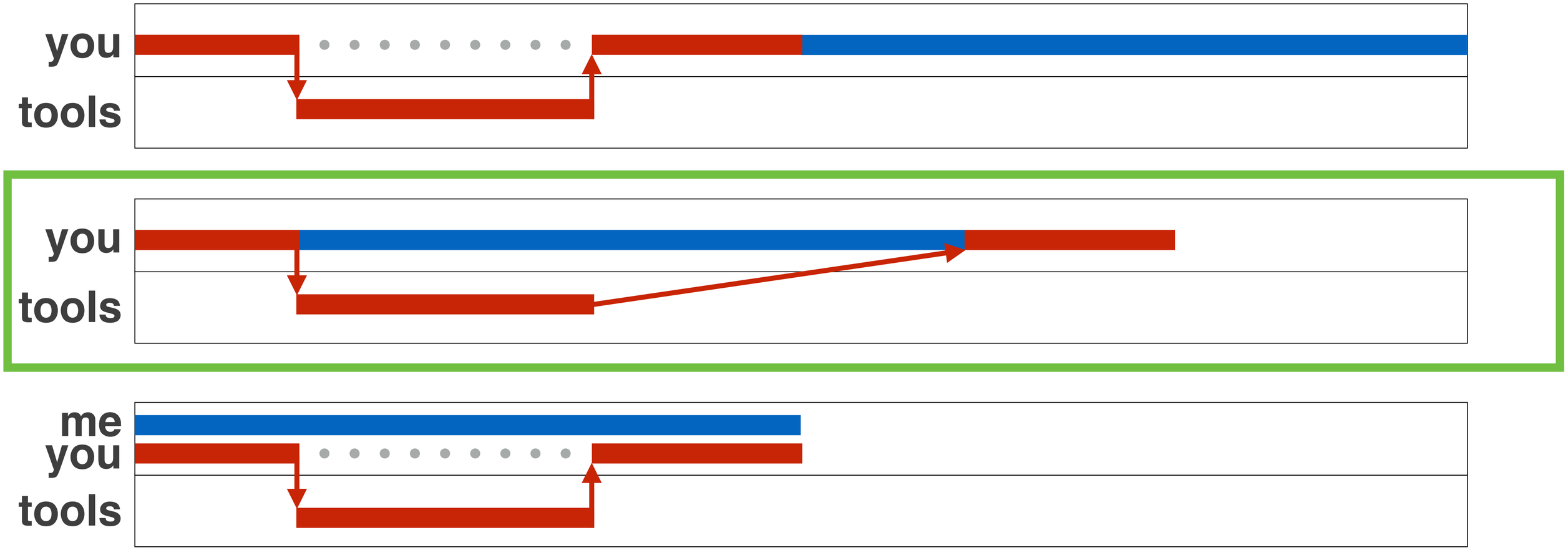
# CONCURRENCY

*Parallel…*


3. I only make the icing and you only make the cake



# WHICH DESCRIBES JAVASCRIPT?



Er, not exactly

*"Node.js is a ~~single-threaded~~, event-driven, non-blocking I/O platform"*

– SOME PEOPLE ON THE INTERNET

*"JavaScript is single-threaded"* …arguably yes

– OTHER PEOPLE ON THE INTERNET

# ASYNC

***(Code is asynchronous if) the execution order is not dependent upon the command order***

# WHAT HAPPENS?

```
console.log(‘Some callbacks');
setTimeout(function(){
  console.log('you');
}, 3000);
console.log('love');
```

```
Some callbacks
love
you
```

# EVENT BASED

*A function that executes asynchronously…*

1. Kicks off some external process
2. Registers an event handler for when that process finishes (callback)

# WHAT HAPPENS?

```
var start = new Date;
setTimeout(function(){
  var end = new Date;
  console.log('Time elapsed:', end - start, 'ms');
}, 500);

while (new Date - start < 1000) {};
```

```
=> Time elapsed: 1000 ms
```

# WHY?

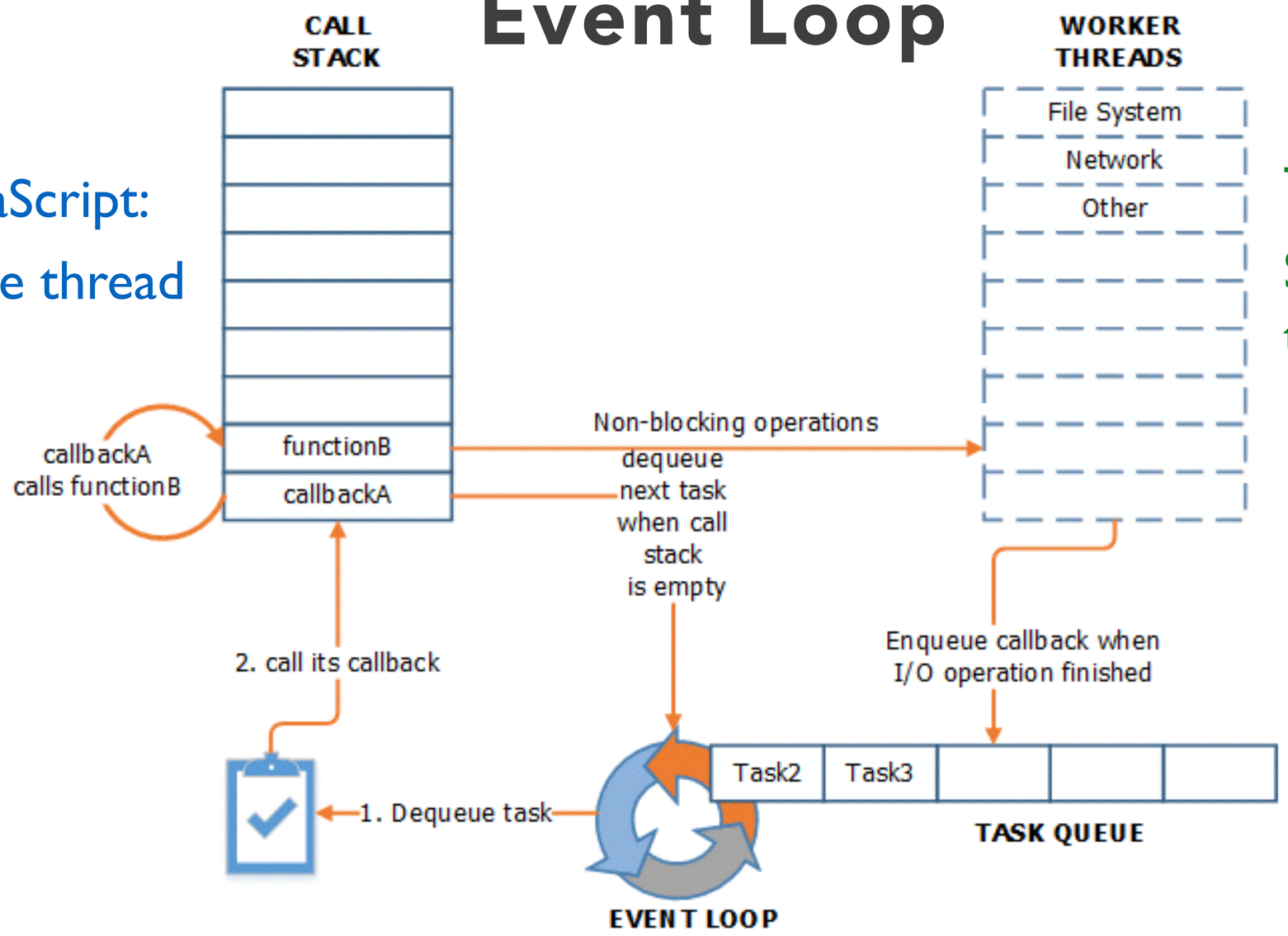
```
var start = new Date;
setTimeout(function(){ // starts up a timeout only
  var end = new Date;
  console.log('Time elapsed:', end - start, 'ms');
}, 500);

while (new Date - start < 1000) {}; // idles for 1000 ms
// meanwhile, halfway through, the timer finishes
// but while loops are blocking
// and js does not interrupt blocking commands
// after the while it has no other commands
// so it will execute the queued callback
```

# Event Loop

JavaScript:

One thread




Thread pool (libeio):

Slow stuff, multiple threads

Event loop (libev):

One thread

How do I know if a function is asynchronous?

If you want to be sure, you have to look it up

That doesn't help

…Wait really?

Well, async operations often have the following callback pattern:

```
asyncThing(function(err,data){…})
```

# SUMMARY

- **Node === platform for running server-side JavaScript**
  - Same language different available tools (globals, etc)
- `require` **pulls in what** `module.exports` **puts out**
- **JavaScript is single-threaded but its runtime environment is not**
- **A callback executes when its async event finishes**