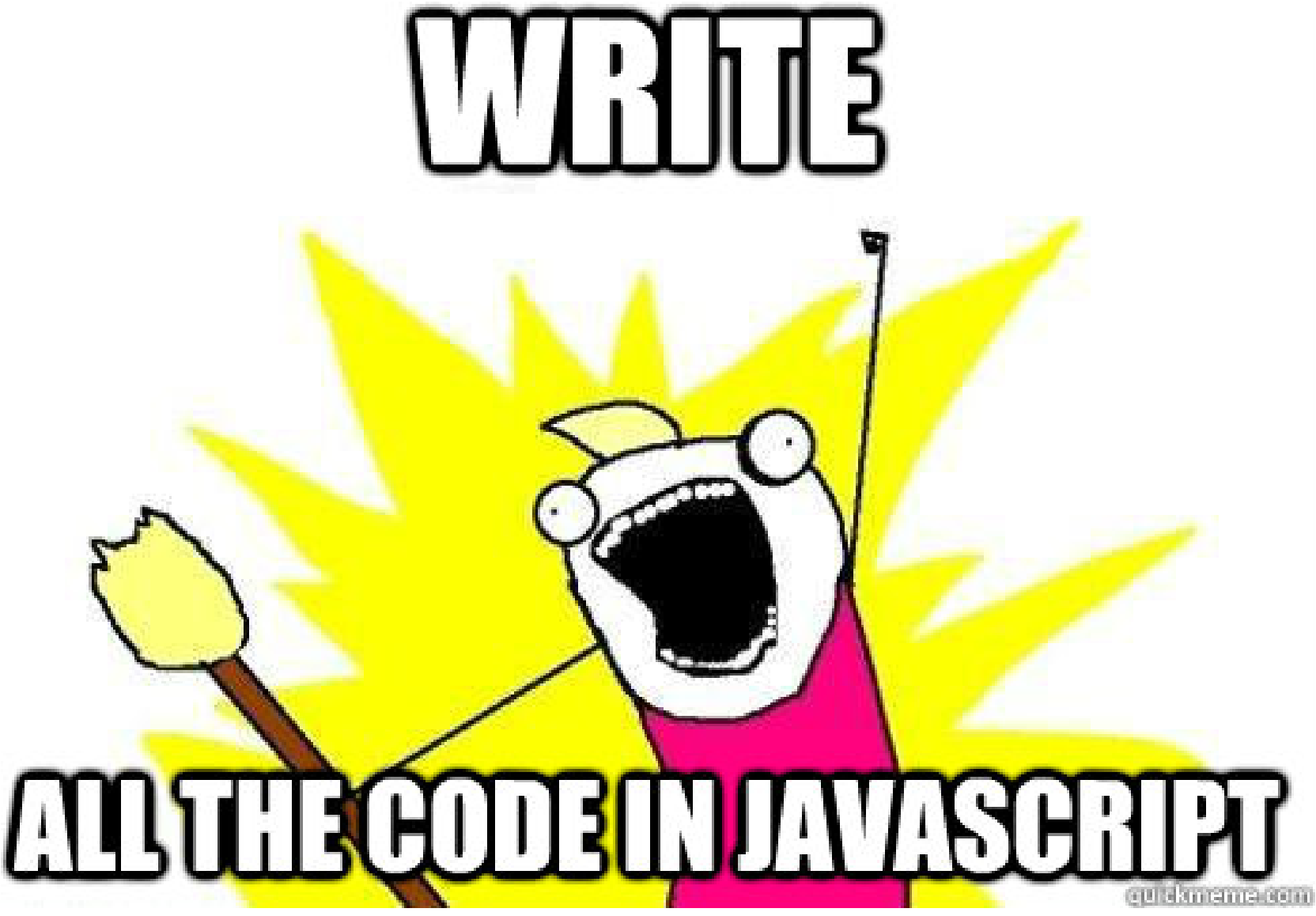
Fullstack Javascript Web Applications

Chris Gradwohl
University of California, Santa Cruz

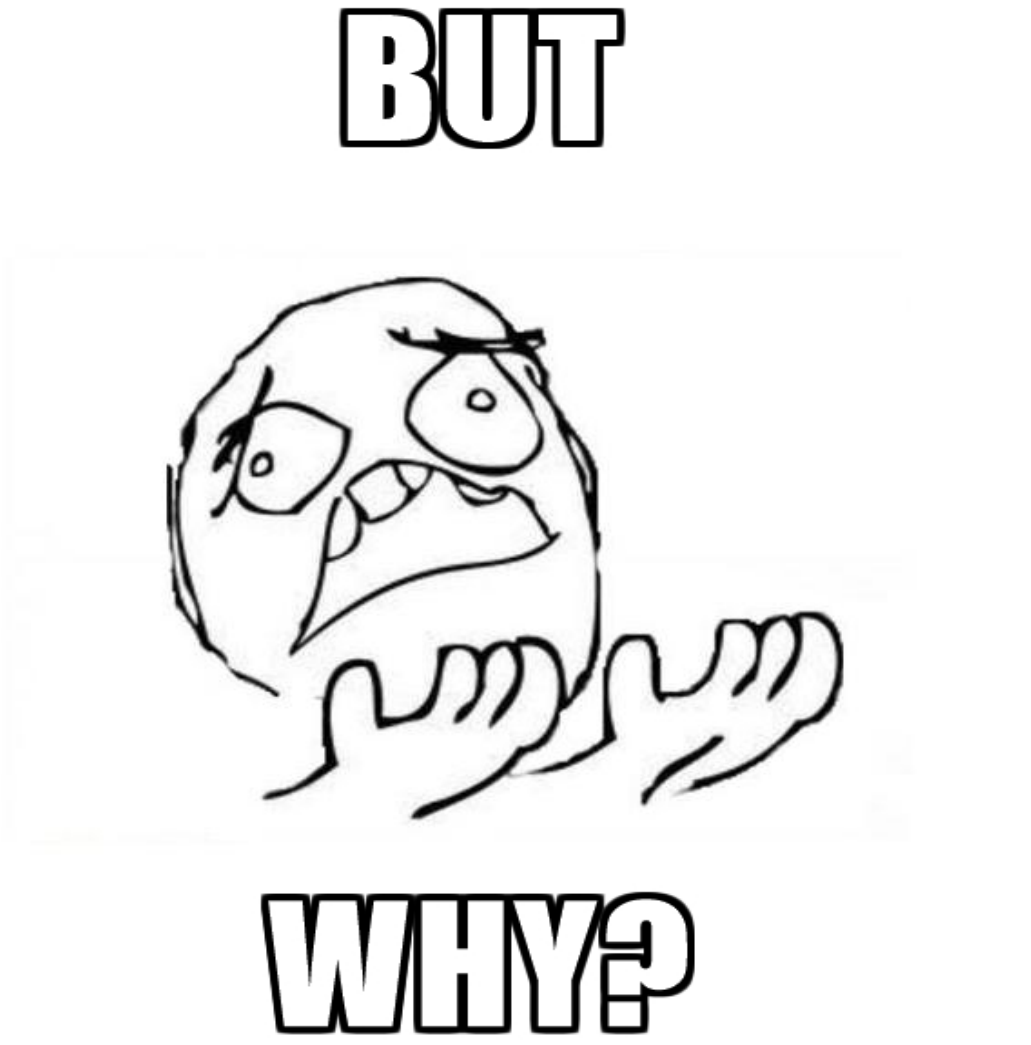
Introduction to Javascript

- Stack Overflow has ranked Javascript as the worlds most popular programming language(four years in a row).
- The language for client side web development.
- Node.js has made fullstack(client side AND server side) applications possible.

Introduction to Javascript



Introduction to Javascript



Why Javascript?

- It's convenient for developers
- But more importantly...javascript is a single threaded, non-blocking, asynchronous, concurrent language.

Why Javascript?

- It's convenient for developers
- But more importantly...javascript is a single threaded, non-blocking, asynchronous, concurrent language.

Javascript is single threaded

- one thread == one call stack == one thing at a time.
- the call stack records where we are in the program.
- calling a function means we .push() it onto the stack.
- returning from a function means we .pop() it off the stack.

Javascript is single threaded

- one thread == one call stack == one thing at a time.
- the call stack records where we are in the program.
- calling a function means we .push() it onto the stack.
- returning from a function means we .pop() it off the stack.

Javascript is single threaded

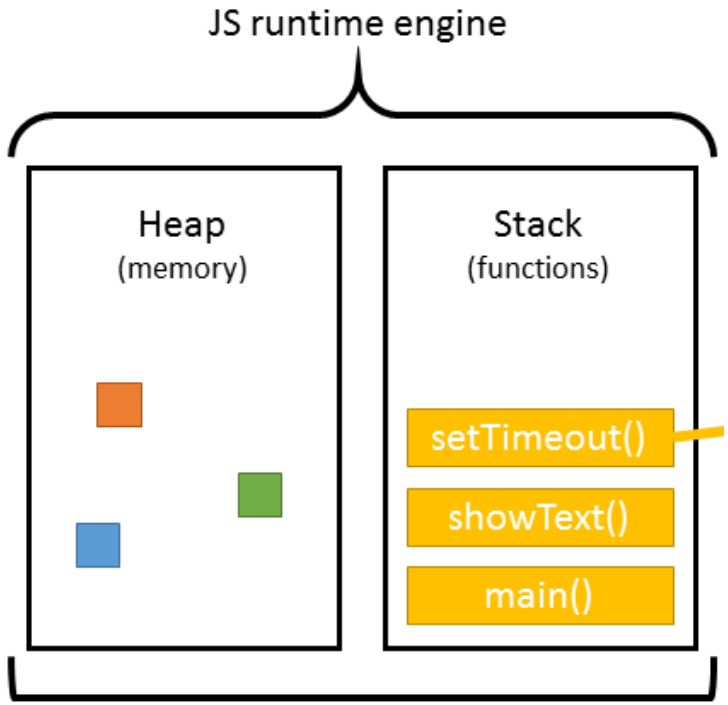
- one thread == one call stack == one thing at a time.
- the call stack records where we are in the program.
- calling a function means we .push() it onto the stack.
- returning from a function means we .pop() it off the stack.

Javascript is single threaded

- one thread == one call stack == one thing at a time.
- the call stack records where we are in the program.
- calling a function means we .push() it onto the stack.
- returning from a function means we .pop() it off the stack.

Blocking and the call stack

- Blocking refers to HOW we .push() functions onto the call stack.
- Synchronous function calls are the problem.



Blocking and the call stack

Javascript event loop solves blocking

- the javascript event loop makes javascript asynchronous and concurrent!

- .push() a function onto the stack

- .pop() the function off the call stack
- process functions in the event loop
- the stack is now free to execute remaining function calls.

- callback/return the function when its ready to execute

Blocking and the call stack

Javascript event loop solves blocking

- the javascript event loop makes javascript asynchronous and concurrent!

- .push() a function onto the stack

- .pop() the function off the call stack
- process functions in the event loop
- the stack is now free to execute remaining function calls.

- callback/return the function when its ready to execute

Blocking and the call stack

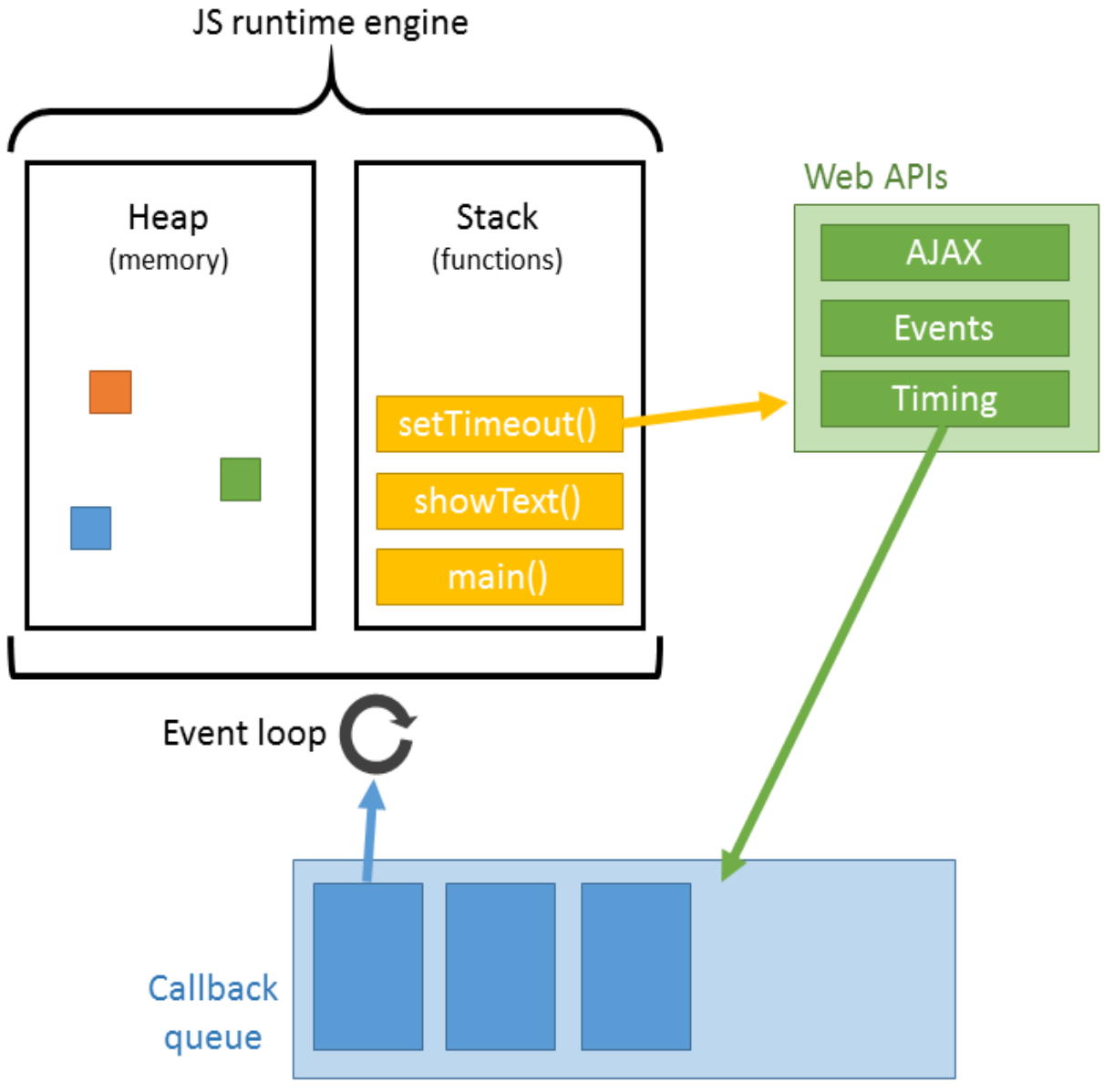
Javascript event loop solves blocking

- the javascript event loop makes javascript asynchronous and concurrent!

- .push() a function onto the stack

- .pop() the function off the call stack
- process functions in the event loop
- the stack is now free to execute remaining function calls.

- callback/return the function when its ready to execute


JS runtime engine
Heap
(memory)
Stack
(functions)
setTimeout()
showText()
main()
Web APIs
AJAX
Events
Timing
Event loop
Callback
queue

Javascript Recap

- Single threaded AND Non-Blocking
  - one stack == one thing at a time(kind of)
  - there is no downtime, works starts right away
  - made possible by the asynchronous and concurrent event loop.
- Benefits
  - fast, no downtime between function calls
  - easier to program versus multithreaded programming
  - less resources: memory, time

Javascript Recap

- Single threaded AND Non-Blocking
  - one stack == one thing at a time(kind of)
  - there is no downtime, works starts right away
  - made possible by the asynchronous and concurrent event loop.
- Benefits
  - fast, no downtime between function calls
  - easier to program versus multithreaded programming
  - less resources: memory, time

Javascript Recap

- Single threaded AND Non-Blocking
  - one stack == one thing at a time(kind of)
  - there is no downtime, works starts right away
  - made possible by the asynchronous and concurrent event loop.
- Benefits
  - fast, no downtime between function calls
  - easier to program versus multithreaded programming
  - less resources: memory, time

Javascript Recap

- Single threaded AND Non-Blocking
  - one stack == one thing at a time(kind of)
  - there is no downtime, works starts right away
  - made possible by the asynchronous and concurrent event loop.
- Benefits
  - fast, no downtime between function calls
  - easier to program versus multithreaded programming
  - less resources: memory, time

Javascript Recap

- Single threaded AND Non-Blocking
  - one stack == one thing at a time(kind of)
  - there is no downtime, works starts right away
  - made possible by the asynchronous and concurrent event loop.
- Benefits
  - fast, no downtime between function calls
  - easier to program versus multithreaded programming
  - less resources: memory, time

Javascript Recap

- Single threaded AND Non-Blocking
  - one stack == one thing at a time(kind of)
  - there is no downtime, works starts right away
  - made possible by the asynchronous and concurrent event loop.
- Benefits
  - fast, no downtime between function calls
  - easier to program versus multithreaded programming
  - less resources: memory, time

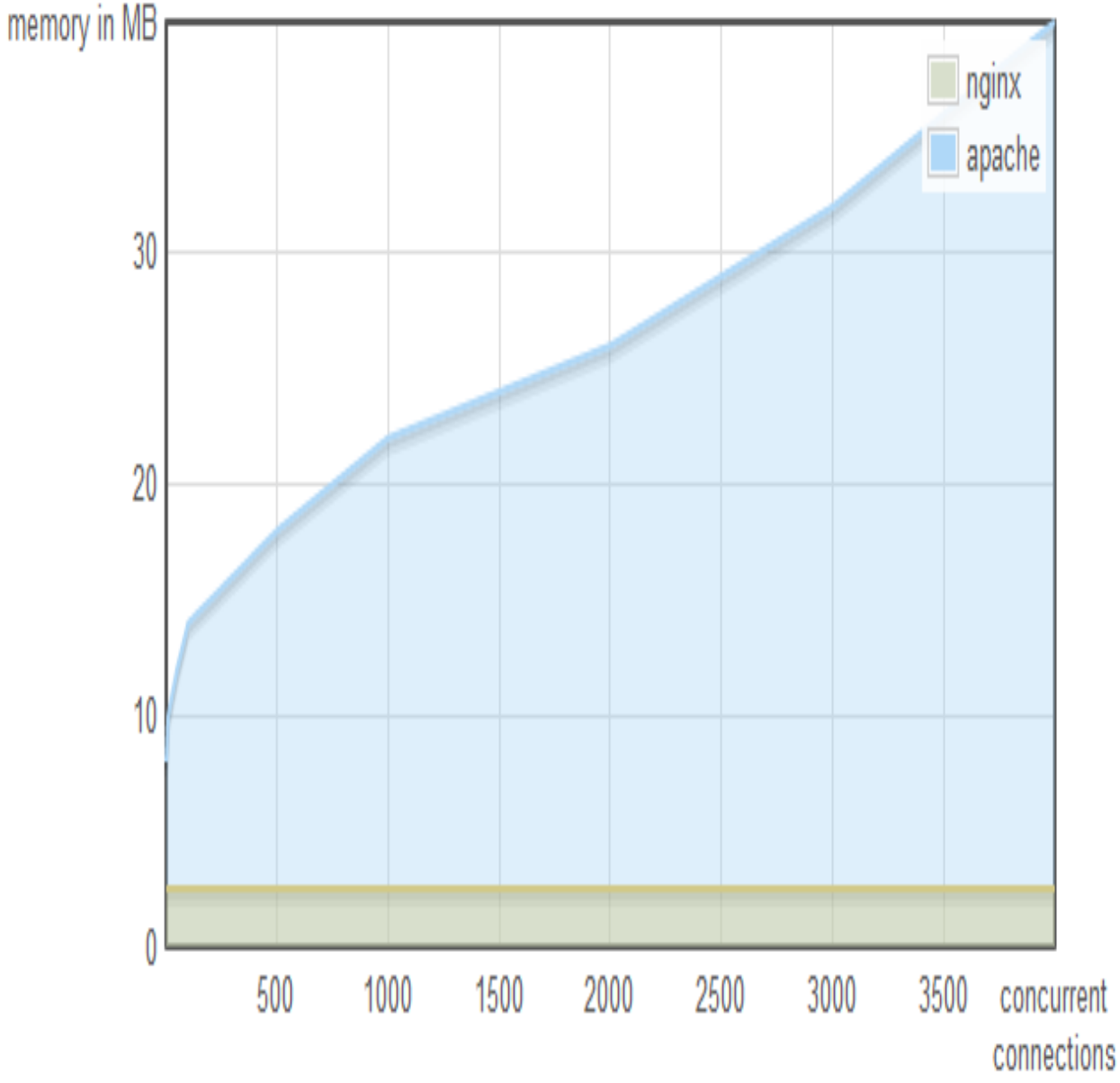
Javascript Recap

- Single threaded AND Non-Blocking
  - one stack == one thing at a time(kind of)
  - there is no downtime, works starts right away
  - made possible by the asynchronous and concurrent event loop.
- Benefits
  - fast, no downtime between function calls
  - easier to program versus multithreaded programming
  - less resources: memory, time

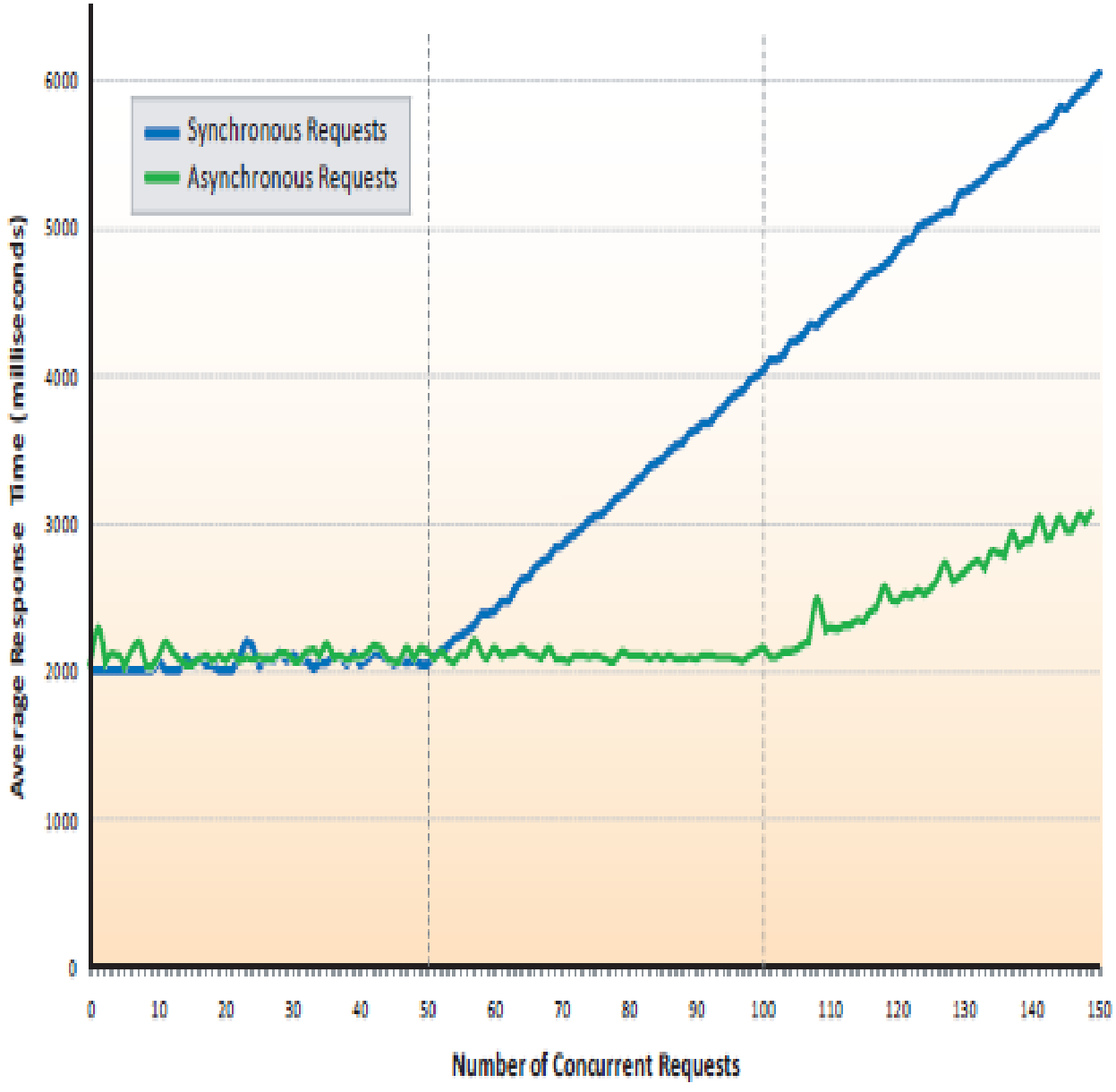
Javascript Recap

- Single threaded AND Non-Blocking
  - one stack == one thing at a time(kind of)
  - there is no downtime, works starts right away
  - made possible by the asynchronous and concurrent event loop.
- Benefits
  - fast, no downtime between function calls
  - easier to program versus multithreaded programming
  - less resources: memory, time

Memory Usage



Speed


Synchronous Requests
Asynchronous Requests
Average Response Time (milliseconds)
0
1000
2000
3000
4000
5000
6000
0
10
20
30
40
50
60
70
80
90
100
110
120
130
140
150
Number of Concurrent Requests

Conclusions

- Javascript is FAST: single threaded, non blocking, asynchronous, and concurrent
- Technologies like Node.js allows developers to use Javascript outside the the browser, and inside the server.
- Using javascript on the server, is fast and scalable.
- Allows developers to program both the front-end and backend.

References


Saba Alimadadi, Ali Mesbah, Karthik Pattabiraman.
Understanding Asynchronous Interactions in Full-Stack JavaScript.
ICSE '16, May 14 - 22, 2016, Austin, TX, USA


Prashant Bansal.
http://prashantb.me/javascript-call-stack-event-loop-and-callbacks/


Steven Sanderson.
http://blog.stevensanderson.com/2010/01/25/measuring-the-performance-of-asynchronous-controllers/


Stack Overflow.
https://stackoverflow.com/research/developer-survey-2016