Comet with node.js and V8

by amix

About me

- Cofounder and lead developer of Plurk.com
  Coded most of the frontend and backend
- Mostly code in Python and JavaScript
  But I am proficient in other languages as well (C, Java, Lisp, Lua etc.)
- I have coded since I was 12
  24 years old now and still love to program :-)
- Not a node.js expert...
  But I find it very interesting

Overview of the talk

- Why JavaScript matters
- What makes V8 VM special
- node.js in some details
- Characteristics of comet communication
- Implementing comet using node.js and WebSockets
- Perspective: JavaScript as the future platform

Why JavaScript matters

- The pool of JavaScript programmers is huge and it's getting bigger
- JavaScript's distribution is among the largest think of all the browsers that support it and all the mobile platforms that support it or will support it... Think of Flash
- Big companies like Adobe, Google, Apple and Microsoft are spending tons of $ in improving JavaScript
- JavaScript is likely to become one of the most popular languages
- JavaScript is gaining ground on the backend

What makes V8 special

- V8 JavaScript VM is used in Google Chrome and is developed by a small Google team in Denmark. V8 is open-source
- V8 team is led by Lars Bak, one of the leading VM engineers in the world with 20 years of experience in building VMs
- Lars Bak was the technical lead behind HotSpot (Sun's Java VM). HotSpot improved Java's performance 20x times
- Before HotSpot Lars Bak worked on a SmallItalk VM

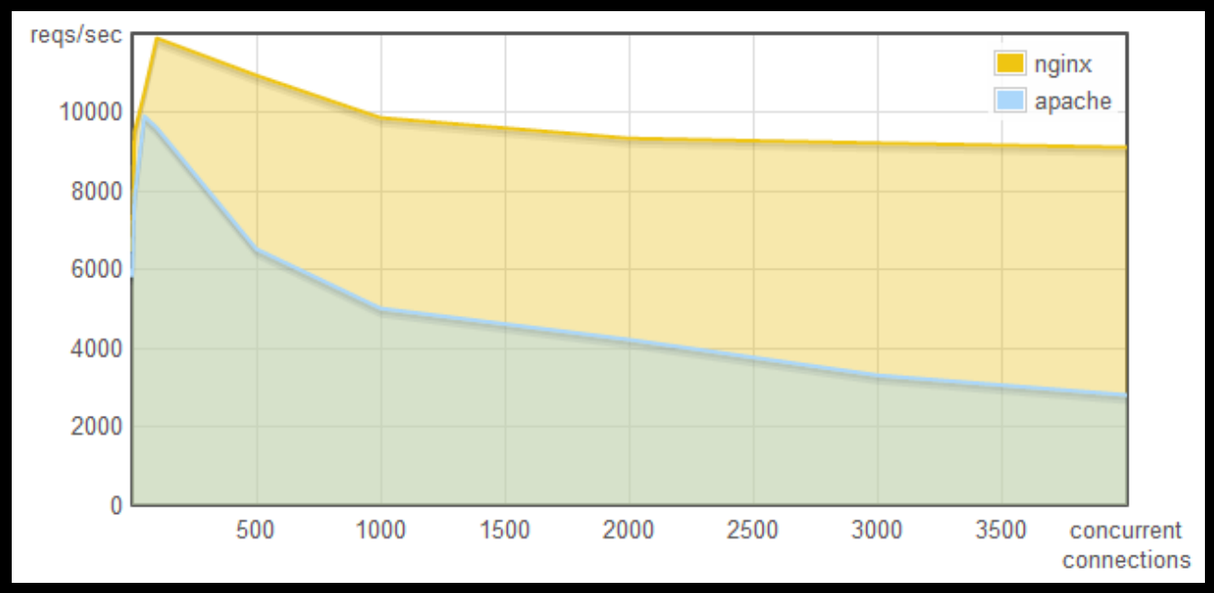
What makes V8 special

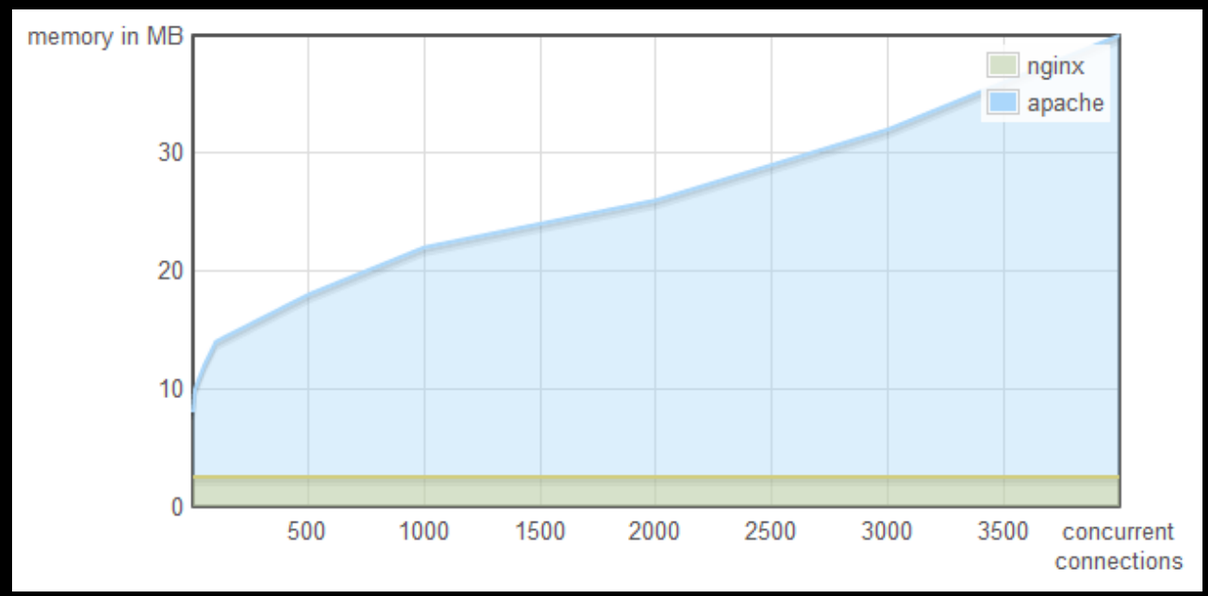
- No JIT, all JavaScript is compiled to assembler
- Hidden classes optimization from Self
  V8 does not dynamically lookup access properties, instead it uses hidden classes that are created behind the scene
- Improved garbage collector
  stop-the-world, generational, accurate, garbage collector
- V8 is independent of Google Chrome
- Remarks / “limitations”:
  No bytecode language, no threads, no processes

What is node.js?

- A system built on top of V8
- Introduces:
  - non-blocking IO
  - ability to do system calls
  - HTTP libraries
  - module system (+ other things)
- The non-blocking nature makes node.js a good fit for comet and next generation realtime web-applications
- 8000 lines of C/C++, 2000 lines of Javascript, 14 contributors

Advantages of non-blocking





- nginx: non-blocking apache: threaded
- non-blocking can handle more req. pr. sec and uses a lot less memory
- comet does not scale at all for threaded servers...

graph source: http://blog.webfaction.com/a-little-holiday-present

Major point

JavaScript programming is already geared towards event based programming:

- Events in browsers....
- Closures (anonymous functions) are natural part of the language

```
document.addEventListener("click", function(event) {
  alert(event)
}, false)
```

Hello world using node.js

```
var sys = require('sys'),
    http = require('http')

http.createServer(function (req, res) {
  setTimeout(function () {
      res.sendHeader(200, {'Content-Type': 'text/plain'})
      res.sendBody('Hello World')
      res.finish()
  }, 2000)
}).listen(8000)

sys.puts('Server running at http://127.0.0.1:8000/')
```

hello_world.js

Blocking vs. non-blocking

The way to do it in most other languages:

```
puts("Enter your name: ")
var name = gets()
puts("Name: " + name)
```

The way to do it in node.js!

```
puts("Enter your name: ")
gets(function (name) {
    puts("Name: " + name)
})
```

node.js design philosophy:
To receive data from disk, network or another process there must be a callback.

Events in node.js

- All objects which emit events are are instances of process.EventEmitter
- A promise is a EventEmitter which emits either success or error, but not both

```
var tcp = require("tcp")

var s = tcp.createServer()
s.addListener("connection",
   function (c) {
      c.send("hello nasty!")
      c.close()
})
s.listen(8000)
```

```
var stat = require("posix").stat,
    puts = require("sys").puts

var promise = stat("/etc/passwd")
promise.addCallback(function (s) {
   puts("modified: " + s.mtime)
})
promise.addErrback(function(orgi_promise) {
   puts("Could not fetch stats on /etc/passwd")
})
```

Comet vs. Ajax

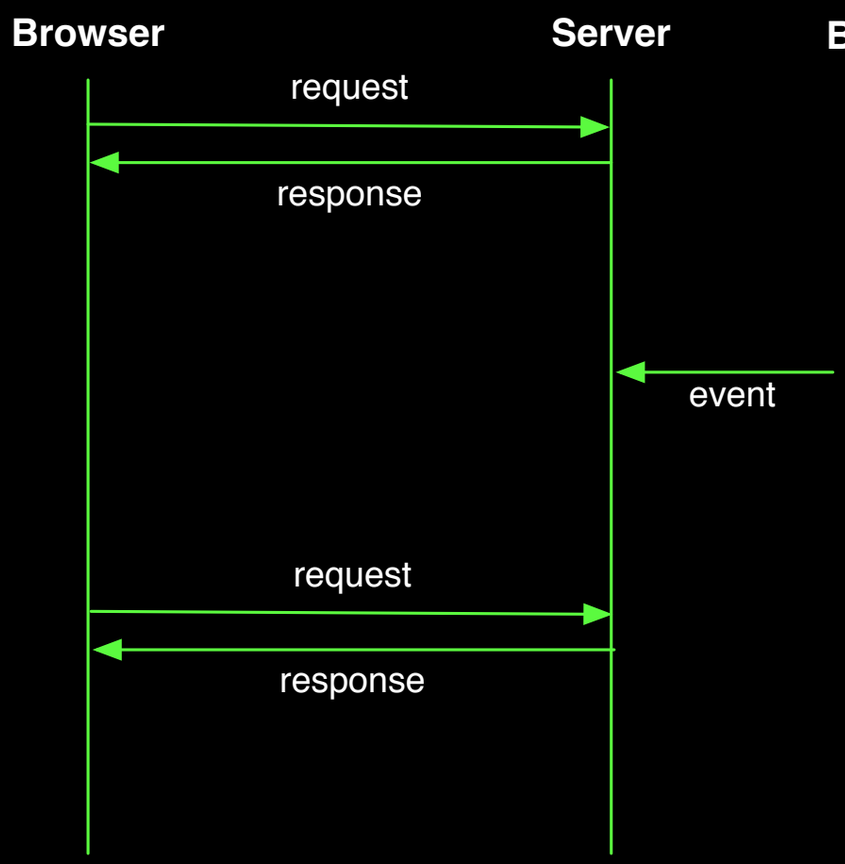
Ajax is so yesterday...
Comet is the new superhero on the block

Comet can be used to create real time web-applications

Examples include Plurk and Google Wave. Demo of Plurk real time messaging

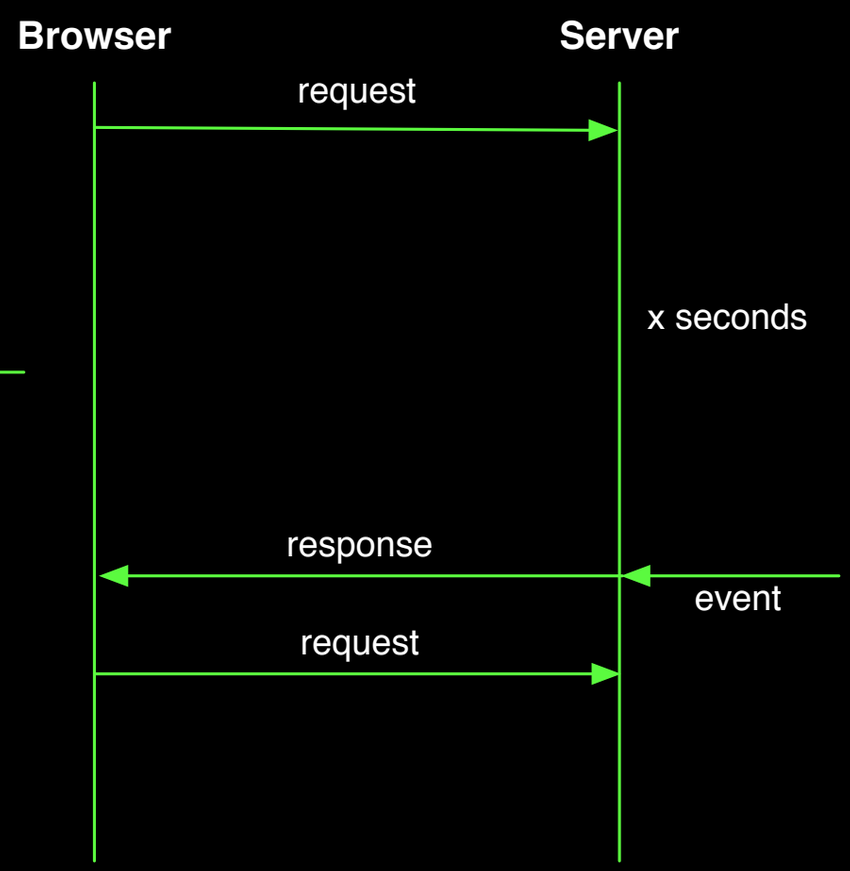
Ajax vs. Comet

Ajax



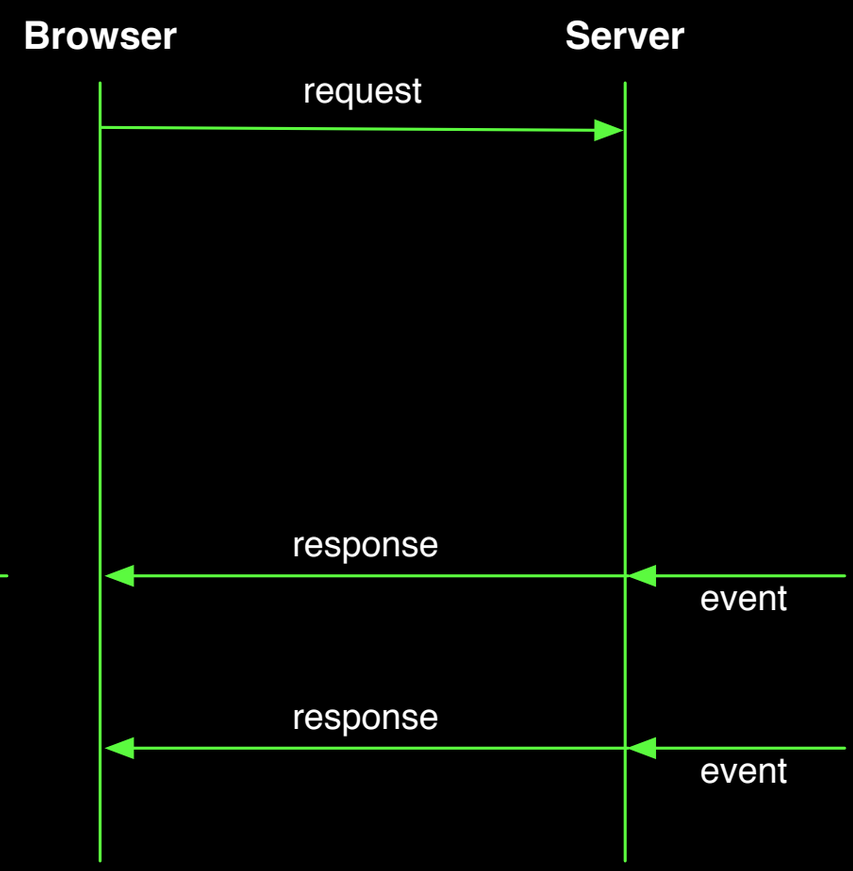
How most do it today

Comet (long poll)



How some do it today

Comet (streaming)



How we will do it soon

Major point

- Comet servers need to have a lot of open connections
- One thread pr. connection does not scale
- The solution is to use event based servers
- It's only possible to create event based servers in node.js!

Implementing comet

- Long polling
  - works for most parts. Used in Plurk
  - is more expensive and problematic than streaming
- Streaming without WebSockets:
  - Very problematic due to proxies and firewalls
- Streaming with WebSockets (HTML5):
  - Makes comet trivial on both client and server side
- In this presentation we will focus on the future: Building a chat application with WebSockets and node.js

WebSockets

- Aims to expose TCP/IP sockets to browsers, while respecting the constraints of the web (security, proxies and firewalls)
- A thin layer on top of TCP/IP that adds:
  - origin based security model
  - Addressing and protocol naming mechanism for supporting multiple services on one port and multiple host names on one IP address
  - framing mechanism for data transporation
- More information: http://dev.w3.org/html5/websockets/
- Currently implemented in Google Chrome

node.websocket.js implementation
demo

```
var sys = require('sys')
    members = []

var Module = this.Module = function() { }

Module.prototype.onData = function(data, connection) {
   for(var i in members)
         members[i].send(data)
};

Module.prototype.onConnect = function(connection) {
   members.push(connection)
}

Module.prototype.onDisconnect = function(connection) {
   for(var i in members) {
         if(members[i] == connection) {
              members.splice(i, 1)
              break;
         }
   }
}
```

```
Room = {
   init: function() {
         Room.ws = new WebSocket('ws://127.0.0.1:8080/chat')
         Room.ws.onopen = Room._onopen
         Room.ws.onmessage = Room._onmessage
         Room.ws.onclose = Room._onclose
   },
    //...
    _send: function(user, message){
         Room.ws.send(serializeJSON({
              'from': user,
              'message': message
         }))
},
 _onmessage: function(m) {
         if (m.data) {
              var data = evalTxt(m.data)

              from = data.from
              message = data.message
 //...
```

Note: I have patched
node.websocket.js to include onConnect...

Other comet servers

- JBoss Netty: Java library for doing non-blocking networking, based on java.nio
  used by Plurk to handle 200.000+ open connections
- erlycomet: Erlang comet server based on MochiWeb
- Tornado: FriendFeed's Python non-blocking server
- I have tried most of the popular approaches and none of them feel as natural as node.js!

How does node.js perfrom?

- Hello World benchmark node.js vs. Tornado
  non scientific - take it with a grain of salt!
- Tornado is one of the fastest Python based servers

```
$ ab -c 100 -n 1000 http://127.0.0.1:8000/
Concurrency Level:      100
Time taken for tests:   0.230 seconds
Complete requests:      1000
Failed requests:        0
Write errors:           0
Total transferred:      75075 bytes
HTML transferred:       11011 bytes
Requests per second:    4340.26 [#/sec] (mean)
Time per request:       23.040 [ms] (mean)
Time per request:       0.230 [ms]
Transfer rate:          318.21 [Kbytes/sec]
```

node.js

```
$ ab -c 100 -n 1000 http://127.0.0.1:8000/
Concurrency Level:      100
Time taken for tests:   0.427 seconds
Complete requests:      1000
Failed requests:        0
Write errors:           0
Total transferred:      171864 bytes
HTML transferred:       12276 bytes
Requests per second:    2344.36 [#/sec] (mean)
Time per request:       42.656 [ms] (mean)
Time per request:       0.427 [ms]
Transfer rate:          393.47 [Kbytes/sec]
```

Tornado

CoffeScript

- A Ruby inspired language that compiles to JavaScript

CoffeScript

```
square: x => x * x.
cube:   x => square(x) * x.
```

```
sys = require('sys')
http = require('http')

http.createServer( req, res =>
    setTimeout( =>
        res.sendHeader(200, {'Content-Type': 'text/plain'})
        res.sendBody('Hello World')
        res.finish()., 2000).
).listen(8000)
```

JavaScript

```
var cube, square;
square = function(x) {
  return x * x
}
cube = function(x) {
  return square(x) * x
}
```

```
var sys = require('sys'),
http = require('http')

http.createServer(function (req, res) {
   setTimeout(function () {
         res.sendHeader(200, {'Content-Type': 'text/plain'})
         res.sendBody('Hello World')
         res.finish()
   }, 2000)
}).listen(8000)
```

JavaScript: The platform of the future?

References

- Official page of node.js: http://nodejs.org/
- Official page of V8: http://code.google.com/p/v8/
- CoffeScript: http://jashkenas.github.com/coffee-script/
- Websockets spec: http://dev.w3.org/html5/websockets/
- node.websocket.js: http://github.com/guille/node.websocket.js/
- Tornado: http://www.tornadoweb.org/
- JBoss Netty: http://jboss.org/netty
- erlycomet: http://code.google.com/p/erlycomet/

PS: Plurk API

- Plurk API is now available: http://www.plurk.com/API
- Python, Ruby, PHP and Java implementations are under development
- Use it to build cool applications :-)

Questions?

- These slides will be posted to:
  www.amix.dk (my blog)

- You can private plurk me on Plurk:
  http://www.plurk.com/amix