node.js
&
Asynchronous Programming

astro@spaceboyz.net http://spaceboyz.net/~astro/

2010-09-24 C3D2 Themenabend

Concurrency

Nebenläufigkeit ≠ Parallelität

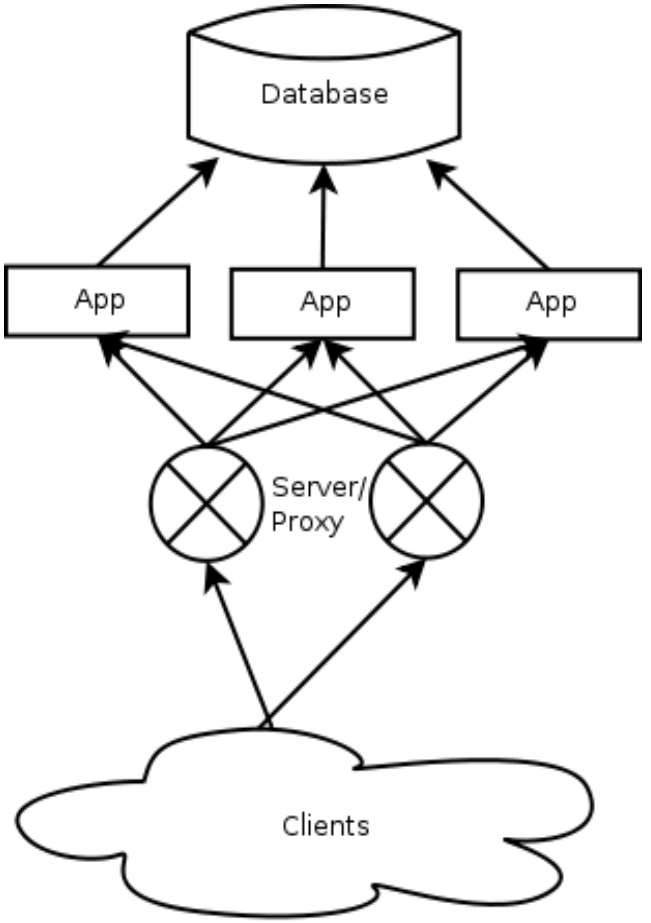
http://www.kegel.com/c10k.html

Definitions

- *Parallelism* means running a *[...]* program on multiple processors, with the goal of improving performance. Ideally, this should be done invisibly, and with no semantic changes.
- *Concurrency* means implementing a program by using multiple I/O-performing threads. While a concurrent *[...]* program can run on a parallel machine, the primary goal of using concurrency is not to gain performance, but rather because that is the simplest and most direct way to write the program. Since the threads perform I/O, the semantics of the program is necessarily non-deterministic.

http://haskell.org/ghc/docs/6.12.2/html/users_guide/lang-parallel.html

Common Architecture



Threads

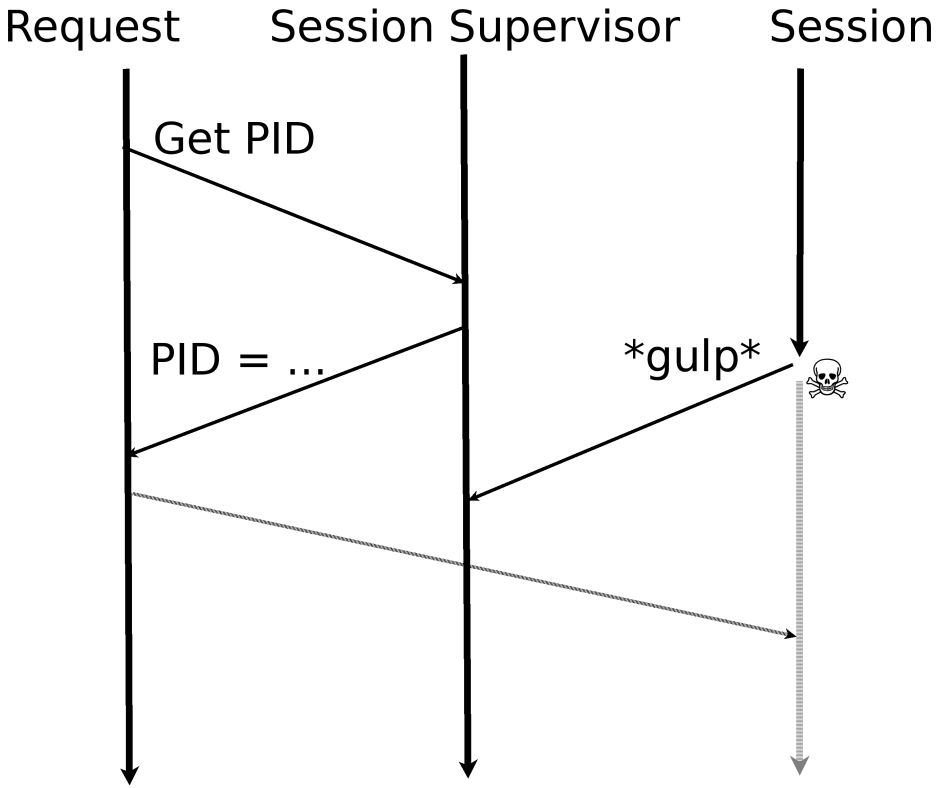
- Hard to get right for IO
- Synchronization increases complexity
- Native threads: expensive, scale across cores
- Green threads: light-weight or poor

Locking

- Semaphores: **V**erhogen & **P**robeer te verlagen
- Mutexes: Lock & Unlock
- In Ruby:
  - ```
    mutex.synchronize {
       # Critical section
    }
    ```
- Deadlocks, Locking Contention, Granularity left to programmer
    ```

Message Passing/ Actor Concurrency

- Works for Erlang
- Shared-nothing concurrency
- Race conditions



Software Transactional Memory

- Less locking by optimistic concurrency control
- But: don't do IO in transactions!
- Gets unwieldy

Asynchronous Programming/ Event-based Programming/ Reactor Pattern

- No threads
- 1 event loop (“*Reactor*”)
- Register callbacks for:
  - IO events for FDs (can read, can write, error)
  - Timers
- Do. Not. Block. Ever.
- OS backends: select(), poll(), /dev/poll, kqueue(), epoll

Asynchronous Hello World

```
puts "Enter your name:"

name = gets.strip

puts "Hello, #{name}"
```

```
process.stdout.write("Enter your
name:\n");

var stdin = process.openStdin();

stdin.on('data', function(name) {

    process.stdout.write(

        "Hello, " + name);

    process.exit(0);

});

process.stdout.write("Look, I'm
waiting for input but still can do
stuff.\n");
```

Use with...

- C: libevent < libev
  http://libev.schmorp.de/bench.html
- C++: Boost.Asio
- Python: Twisted
- Ruby: EventMachine
- GHC: natively soon
- JavaScript: in browsers & node.js

Still CPU-bound?

- Even Supercomputers are limited to 1000s of cores & Terabytes of RAM
- Use message-passing
- Dealing with the network prepares scaling to clusters/Cloud

node.js

www.nodejs.org

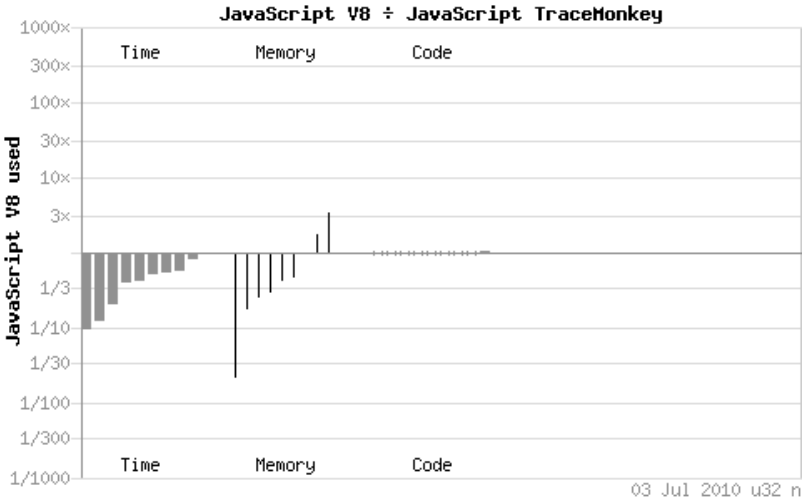
JavaScript?

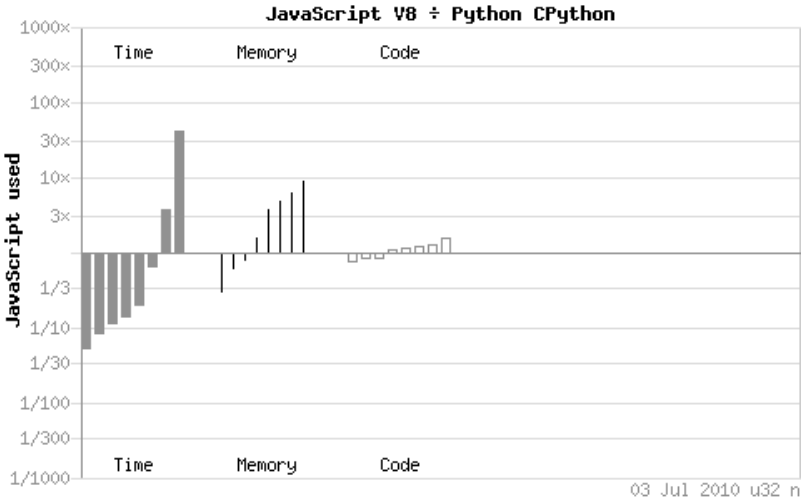
- Bad reputation in the 90s
- Asynchronous patterns widespread
- Easy syntax
- Many programmers
- Required for the web

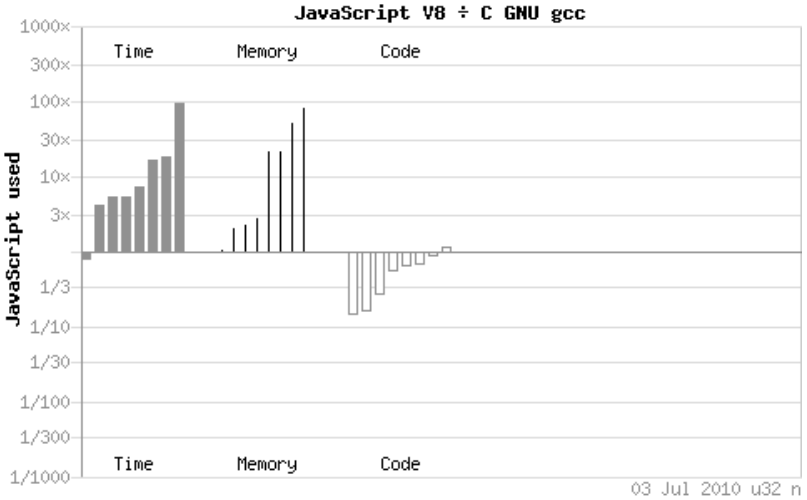
v8

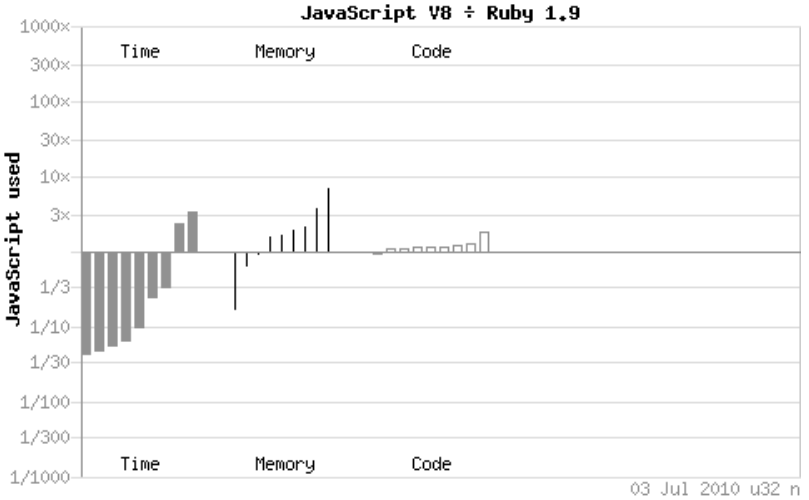
- JS VM of Google Chrome
- C++
- Only for x86, amd64, arm
- Performance goals:
  - Fast Property Access
  - Dynamic Machine Code Generation
  - Efficient Garbage Collection

v8 Performance









http://shootout.alioth.debian.org/u32/benchmark.php?test=all&lang=v8

node.js

- Interfaces to:
  - v8
  - libev
  - libeio
  - c-ares
  - http_parser
- Convenient interfaces

Convenient Interfaces I

C

- Resolve host (asynchronously)
- Create socket
- Set non-blocking
- connect()
- Pass FD to poll()
- Can write?

node.js

```
var conn = new net.Stream();
conn.connect(port,host);
conn.addListener('connect', ...);
```

Convenient Interfaces II

C

- Enqueue FD in poll()
- Can read?
- read() non-blockingly

node.js

```
conn.addListener('data',
  function(data) {
    /* handle data */
  });
```

Write/Send Buffering

- Receivers indicate window size
- Sender throttled by Nagle's Algorithm:
  - Blocking send() takes time
  - Reactor doesn't get “can write” all the time
- node.js takes care, but be aware

```
Event: 'drain'

function () { }

Emitted when the write buffer becomes empty.
Can be used to throttle uploads.
```

Events & Callbacks

- process.EventEmitter:
  - emit(type, args...)
  - addListener(type, function(args...) { });
  - **New:** on ≡ addListener
- *type* references on nodejs.org

sys.pump()

```
function pump (readable, writable, cb) {
  readable.addListener('data', function (data) {
    if (!writable.write(data)) readable.pause();
  });
  readable.addListener('end', function () {
    writable.close();
  });
  writable.addListener('drain', function () {
    readable.resume();
  });
  writable.addListener('close', function () {
    if (cb) cb();
  });
}
```

HTTP

- node.js killer feature
- Hand-woven http-parser:
  - http://github.com/ry/http-parser
- For servers & clients
- Frameworks emerge...

Buffers

- Strings are Unicode
- Buffers are binary
- Raw data buffers
- Cannot be resized

```
stream.setEncoding('binary');
stream.on('data', function(buffer) { });
```

Modules

Exporting from rockets.js:

```
exports.launchRockets = function(...) { ... };
```

Using:

```
var rockets = require('./rockets');
rockets.launchRockets(...);

var sys = require('sys');
sys.puts("Hello, World");
```

npmjs.org

- http://github.com/isaacs/npm
- `make install` or `node cli.js`
- Always fetches http://registry.npmjs.org/
- `npm list`
- `npm install` *`$pkg`*
- `npm publish` *`$dir_with_package.json`*

npm & modules

- Load by packet name:
  - `var xmpp = require('node-xmpp');`
- Load by packet name & version:
  - `var xmppOld = require('node-xmpp-0.1.1');`
  - `var xmppNew = require('node-xmpp-0.2.0');`
- `require()` re-evaluates modules
  - Load a module with global state only once!

package.json

- CommonJS Modules/1.0

```
{ "name": "node-expat"
 ,"version": "1.1.0"
 ,"main": "./build/default/node-expat"
 ,"description": "NodeJS binding for fast XML parsing."
 ,"scripts" : { "install" : "./install.sh" }
 ,"dependencies": []
 ,"licenses": [{ "type": "MIT" }]
 ,"engine": "node"
}
```

Connect

- Webserver middleware
- Asynchronous: `req`, `res`, `next`
- Static file serving, Cookies, Sessions, Gzip, Error handler, Vhost dispatching, Routing, ...
- Base for expressjs

Garbage Collection

- Observe VmRSS!
- v8 invokes GC when memory usage gets high:
- Make calculations preemptible
  - `setTimeout(callback, 1);`
  - `process.nextTick(callback);`
- Invoke GC manually: http://github.com/billywhizz/node-gc
- Beware of premature optimization!

Resources

- API documentation: nodejs.org
- http://groups.google.com/group/nodejs/
- GitHub:
  - Many projects
  - http://wiki.github.com/ry/node/

Douglas Crockford doesn't sleep, he waits to be triggered.

Under Douglas Crockford's beard, there is not a chin. There is only another namespace.

JavaScript

If your javascript does not lint in your first try, then you are obviously not Douglas Crockford

Object inherits from Crockford

State of JavaScript

- Standardized as ECMAScript
- Different DOM APIs in browsers resulted in a bad reputation during the 90s
- Microsoft JScript = JavaScript
- DOM standardized by W3C
- Programming for browsers: pretty libraries
- Programming for servers: less portability issues

JSON

- **J**ava**S**cript **O**bject **N**otation
- application/json
- language-independent
- Specified by Douglas Crockford
- `{ "sublab": "Leipzig", "Mate": 0, "Astro": null }`
- Not binary-safe
- Types: Number, String, Boolean, Array, Object, null

```
eval("(" + json + ")");
JSON.parse(string);
JSON.stringify(object);
```

Semicolons

- Best practise: end each statement with **;**
- Google: *JavaScript Automatic Semicolon Insertion*
- Newlines replace **;**
  - Except if followed by **(**
  - http://mislav.uniqpath.com/2010/05/semicolons/

Type Coercion

```
> 0 == ''
true
> if (" \r\n\t ") console.log("WTF");
WTF
> 0 === ''
false
```

- Falsy Values:
  - `0`
  - `NaN`
  - `''`
  - `false`
  - `null`
  - `undefined`

Shortcuts

```
var isAwesome = dude && dude.awesomeness > 10;

var body, text = msg &&
    ((body = msg.getChild('body'))) &&
    body.getText();
```

Creating Objects

```
function Rocket(speed) {
    this.speed = speed;
    this.launch = function() { /* ... */ };
}

var my_rocket = new Rocket(343);
my_rocket.launch();
```

Numbers

- Always double precision floating-point:

```
> 9999999999999999
10000000000000000

> NaN == NaN
false
> isNaN(NaN)
true
```

Arrays

- Actually Objects

```
> Object.keys([23, 42, 5])
[ '0', '1', '2' ]
```

Strings

- Always 16-bit Unicode

```
> String.fromCharCode(0x1d11e)
'\ud11e'
```

- Use “Buffer” for binary data

Function Scope

```
function foo() {
    for(var i = 0; i < n; i++) {
        // ...
    }
    for(var i = 0; i < n; i++) {
        // ...
    }
}
```

Closures

```
function makeAdder() {
    var sum = 0;
    return function(n) {
        sum += n;
        return sum;
    };
}

var a1 = makeAdder();
var a2 = makeAdder();

a1(1);
a2(5);
a1(1);
```

Explicit this

```
function Stuff() {
    this.contents = "stuff";
    this.printContents = function() {
        console.log(this.contents);
    };
}

var s = new Stuff();
s.printContents();
```

stuff

```
var f1 = s.printContents;
f1();
```

undefined

```
var f2 = function() { s.printContents();
};
f2();
```

stuff

Prototype-based Inheritance

```
function Client() {
    Connection.apply(this, arguments);
    /* ... */
}

/* Client.prototype = new Connection();
 * Client.prototype.constructor = Client;
 * Much better: */
sys.inherits(Client, Connection);
```

Objects as Dictionaries & Prototypes

```
> ('toString' in {})
true
> ({}.hasOwnProperty('toString'))
false
```

Type Checking

```
> typeof null
'object'
> typeof []
'object'
> 5..constructor === Number
true
```

JSONP

JavaScript Object Notation + Padding