JS Responsibilities

Brendan Eich
<brendan@mozilla.org>
@BrendanEich

Quick Recap

- JavaScript: who is to blame?
  - Me first, Netscape second, Java third
- Who has the power?
  - Ecma TC39, made up of both
  - Browser vendors, who compete for
  - Developers: extensiblewebmanifesto.org
- Yet I still feel responsible
- There are a few things more to do

ES6, ES7 in parallel, rapid-ish release

- `class MyNodeList extends NodeList …`
- `const square = (x) => x * x;`
- `const aRot = (h, ...t) => t.push(h);`
- `let {top, left} = e.getClientRect();`
- `function conv(from, to = "EUR") …`
- `import {keys, entries} from "@iter";`
- `for (let [k, v] of entries(obj)) …`
- `function* gen() {yield 1; yield 2;}`
- Etc., see Kangax's compat table for progress

Let's talk about new stuff

- ES6 is kind of old news (ES5 is old news)
- ES7 `Object.observe` involves microtasks
  - Skip it! Maybe another time...
- Low-level JS looks important for
  - Emscripten, Mandreel, other compilers
  - Unreal Engine 3 => 4, + other game engines
  - Hand-coded WebGL-based modules (e.g. OTOY's ORBX codec)

Value Objects

- `symbol`, maybe
- `int64`, `uint64`
- `int32x4`, `int32x8` (SIMD)
- `float32` (to/from `Float32Array` today)
- `float32x4`, `float32x8` (SIMD)
- `bignum`
- `decimal`
- `rational`
- `complex`

Overloadable Operators

- `|` `^` `&`
- `==`
- `<` `<=`
- `<<` `>>` `>>>`
- `+` `-`
- `*` `/` `%`
- `~` *`boolean-test`* *`unary-`* *`unary+`*

Preserving Boolean Algebra

- `!=` and `!` are not overloadable, to preserve identities including
  - `X ? A : B  <=>  !X ? B : A`
  - `!(X && Y)  <=>  !X || !Y`
  - `!(X || Y)  <=>  !X && !Y`
  - `X != Y  <=>  !(X == Y)`

Preserving Relational Relations

- `>` and `>=` are derived from < and <= as follows:
  - `A > B  <=>  B < A`
  - `A >= B  <=>  B <= A`
- We provide <= in addition to < rather than derive `A <= B` from `!(B < A)` in order to allow the <= overloading to match the same value object's == semantics -- and for special cases, e.g., unordered values (NaNs)

Strict Equality Operators

- The strict equality operators, `===` and `!==`, cannot be overloaded
- They work on frozen-by-definition value objects via a structural recursive strict equality test (beware, `NaN !== NaN`)
- Same-object-reference remains a fast-path optimization

Why Not Double Dispatch?

- Left-first asymmetry (v value, n number):
  - `v + n  ==>  v.add(n)`
  - `n + v  ==>  v.radd(n)`
- Anti-modular: exhaustive other-operand type enumeration required in operator method bodies
- Consequent loss of compositionality: `complex` and `rational` cannot be composed to make `ratplex` without modifying source or wrapping in proxies

Cacheable Multimethods

- Proposed in 2009 by Christian Plesner Hansen (Google) in es-discuss
- Avoids double-dispatch drawbacks from last slide: binary operators implemented by 2-ary functions for each pair of types
- Supports Polymorphic Inline Cache (PIC) optimizations (Christian was on the V8 team)
- Background reading: [Chambers 1992]

Binary Operator Example

- For the expression `v + u`
  - Let `p = v.[[Get]](@@ADD)`
  - If `p` is not a `Set`, throw a `TypeError`
  - Let `q = u.[[Get]](@@ADD_R)`
  - If `q` is not a `Set`, throw a `TypeError`
  - Let `r = p` *`intersect`* `q`
  - If `r.length != 1` throw a `TypeError`
  - Let `f = r[0]`; if `f` is not a function, throw
  - Evaluate `f(v, u)` and return the result

API Idea from CPH 2009

```
function addPointAndNumber(a, b) {
  return Point(a.x + b, a.y + b);
}

Function.defineOperator('+', addPointAndNumber, Point, Number);

function addNumberAndPoint(a, b) {
  return Point(a + b.x, a + b.y);
}

Function.defineOperator('+', addNumberAndPoint, Number, Point);

function addPoints(a, b) {
  return Point(a.x + b.x, a.y + b.y);
}

Function.defineOperator('+', addPoints, Point, Point);
```

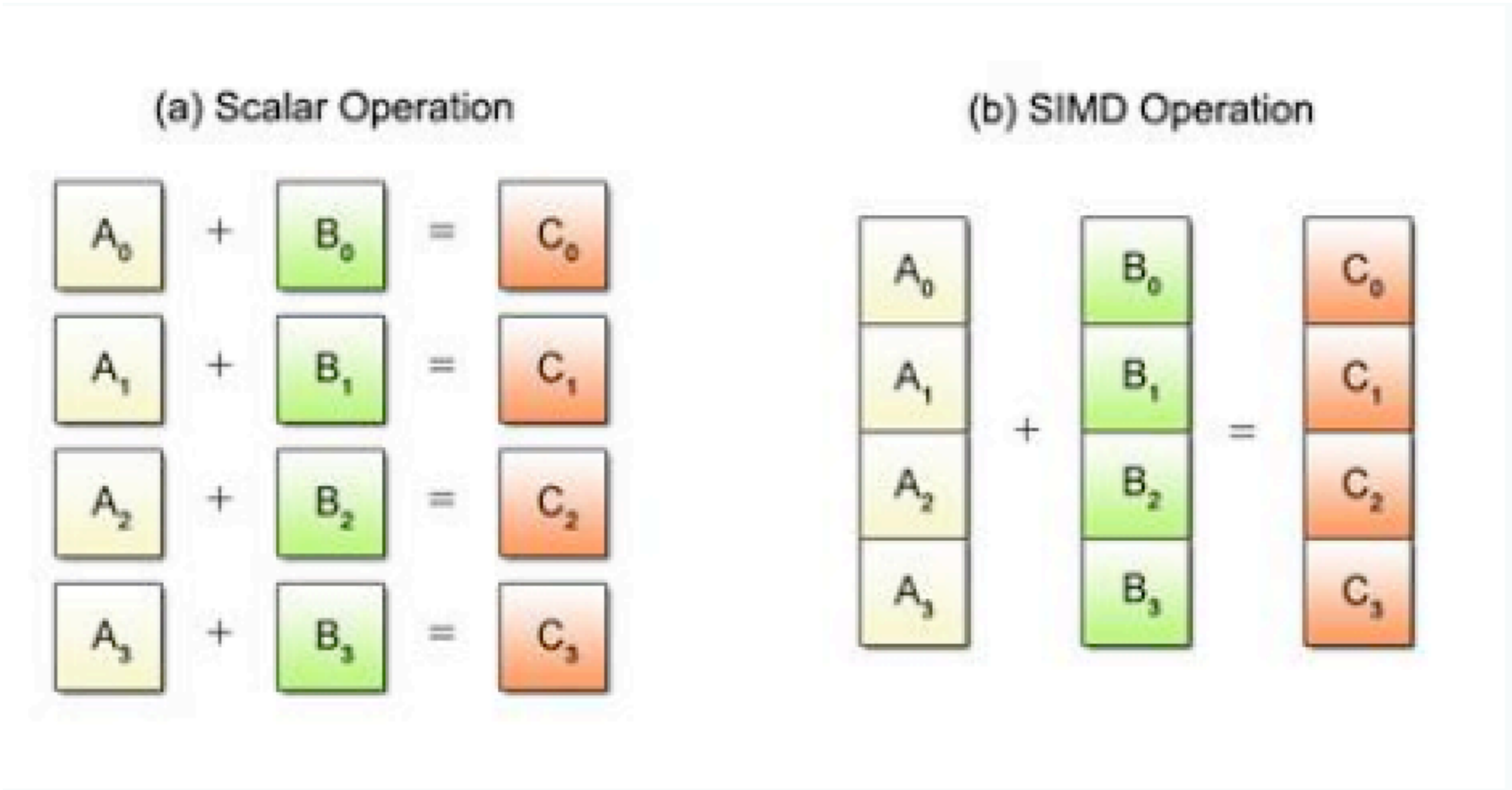
Literal Syntax

- `int64(0)   ==>  0L // as in C#`
- `uint64(0)  ==> 0UL // as in C#`
- `float32(0) ==>  0f // as in C#`
- `bignum(0)  ==>  0n // avoid i/I`
- `decimal(0) ==>  0m // or M, C/F#`
- We want a syntax extension mechanism, but declarative not runtime API
- This means new syntax for operator and suffix definition

Straw Value Object Declaration Syntax

```
value class point2d { // implies typeof “point2d”
  constructor point2d(x, y) {
    this.x = +x;
    this.y = +y;
    // implicit Object.freeze(this) on return
  }
  point2d + number (a, b) {
    return point2d(a.x + b, a.y + b);
  }
  number + point2d (a, b) {
    return point2d(a + b.x, a + b.y);
  }
  point2d + point2d (a, b) {
    return point2d(a.x + b.x, a.y + b.y);
  }
  // more operators, suffix declaration handler, etc.
}
```

SIMD



Single Instruction, Multiple Data (SSE, NEON, etc.)

SIMD intrinsics

- Game, DSP, other low-level hackers need them
- John McCutchan added them to DartVM
- Dart-to-the-heart? No! Dart2JS needs ‘em in JS
- A Google, Intel, Mozilla, Ecma TC39 joint

Possible ES7 Polyfillable SIMD API

https://github.com/johnmccutchan/ecmascript_simd

```
var a = float32x4(1.0, 2.0, 3.0, 4.0);
var b = float32x4(5.0, 6.0, 7.0, 8.0);
var c = SIMD.add(a, b);

// Also SIMD.{sub,mul,div,neg,abs} etc.
// See ES7 Value Objects for some sweet
// operator overloading sugar.
```

Why Operator Syntax Matters

From Cameron Purdy's blog:

"At a client gig, they were doing business/financial coding, so were using `BigDecimal`.

Of course, `.add()` and friends is too difficult, so they ended up with roughly:

```
BigDecimal subA = ...
BigDecimal subB = ...

BigDecimal total = new BigDecimal(
  subA.doubleValue() + subB.doubleValue() );
```

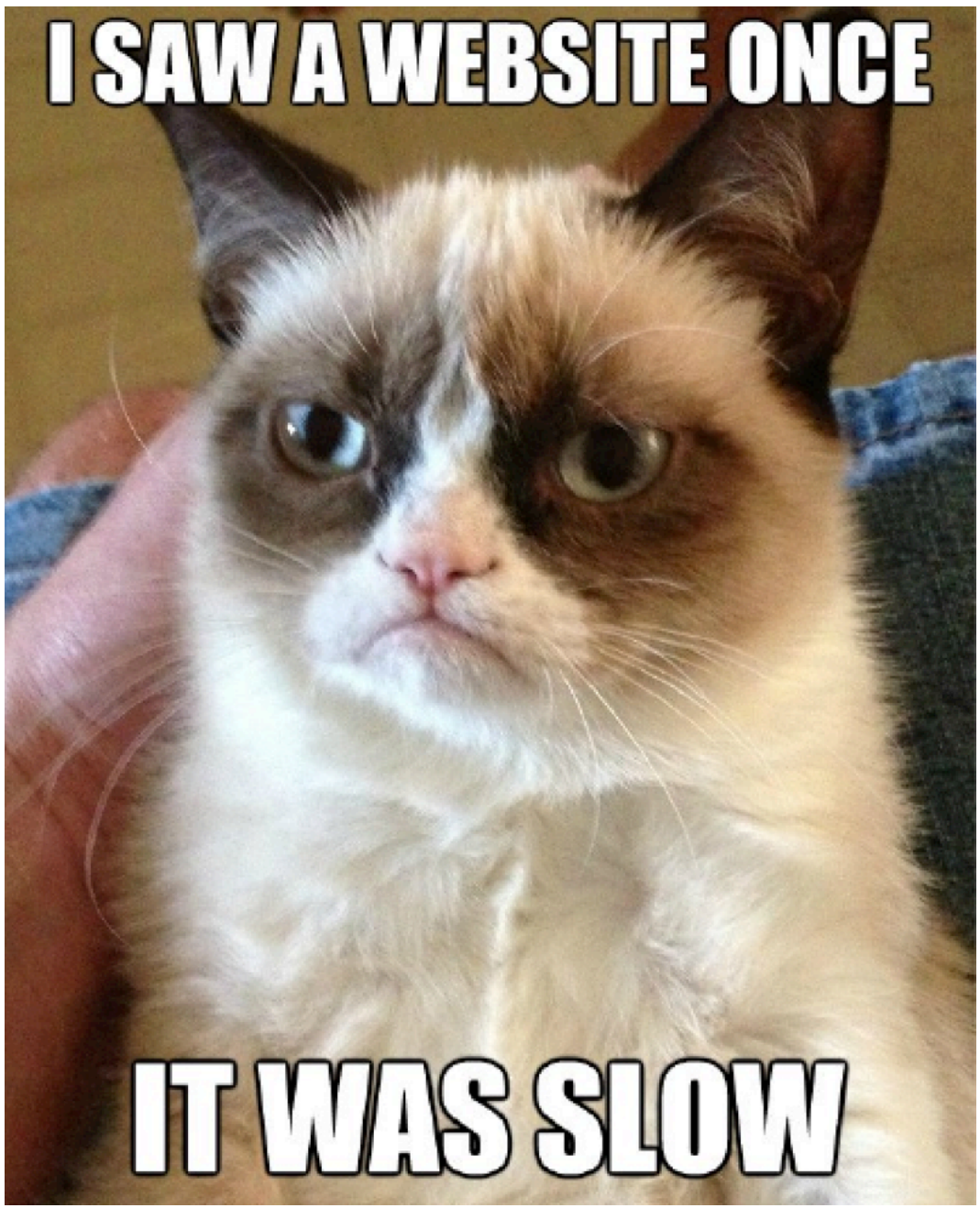
It was beautiful."

Posted by **Bob McWhirter** on October 31, 2005 at 08:17 AM EST

Threads?

- "Threads Suck" - me, 2007
- Emscripten and Mandreel want shared-memory threads for top-of-line C++ game engines
- Data races? C and C++ don't care
- Researching asm.js-confined shared buffers at Mozilla: github.com/sstangl/pthreads-gecko
- We will not expose data races to JS user code or JS VMs (all competitive runtimes are single-threaded for perf)

asm.js progress



Speed on the Web

- Speed is determined by many things
  - Network layer
  - DOM
  - Graphics
  - JS
- This is JSConf.eu, so let's talk about JS speed

How Fast is JS?

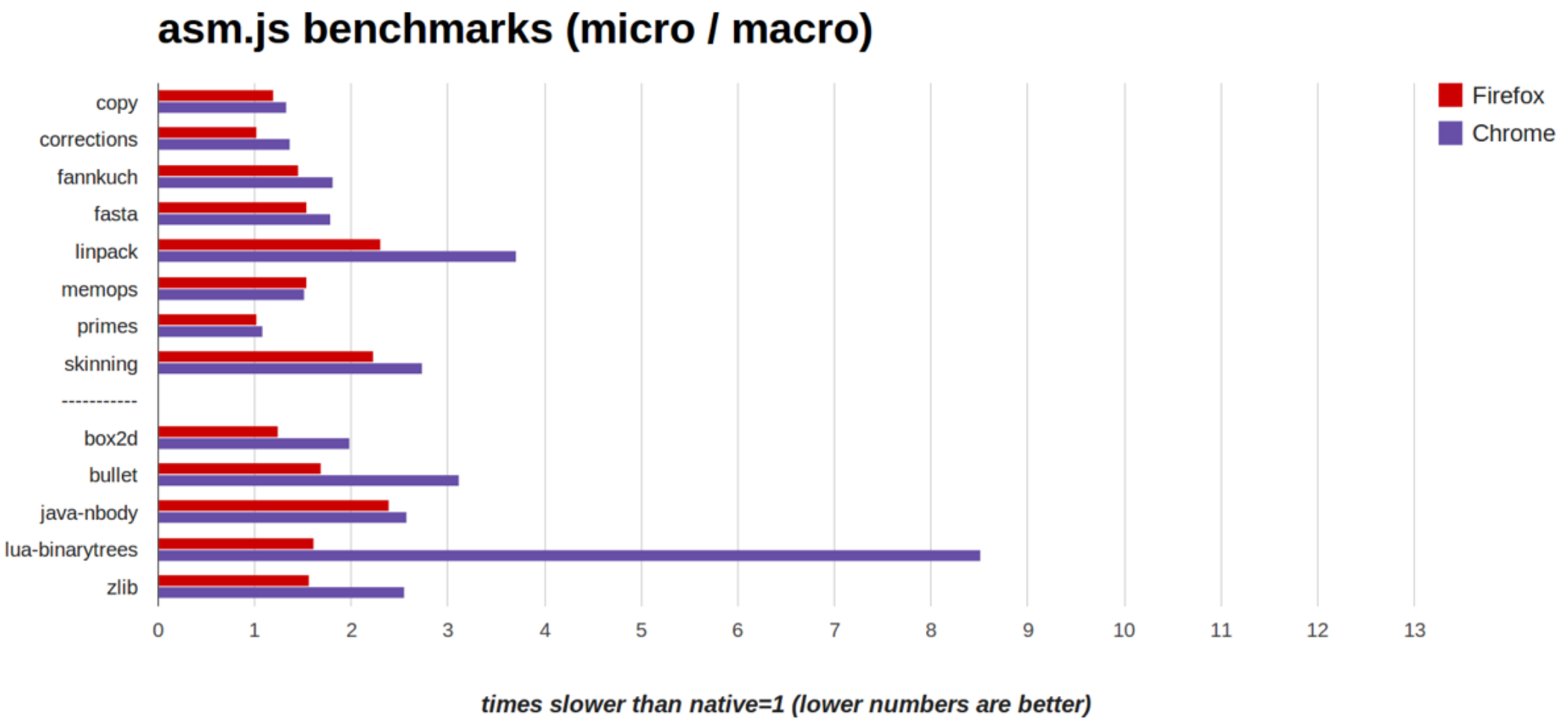
- The Epic Citadel Demo is one way to measure
- Over one million lines of C++ compiled to JS using Emscripten
- OpenGL renderer, compiled to use WebGL
- Uses only standardized web technologies

Epic Citadel Progress

Launched March, 2013
Over 6 months, browsers have improved

back then	today
only ran in Firefox	runs in Firefox & Chrome
20 second startup	10 second startup
40fps	60fps in Firefox & Chrome

Current Performance Results



Emscripten+asm.js Demos

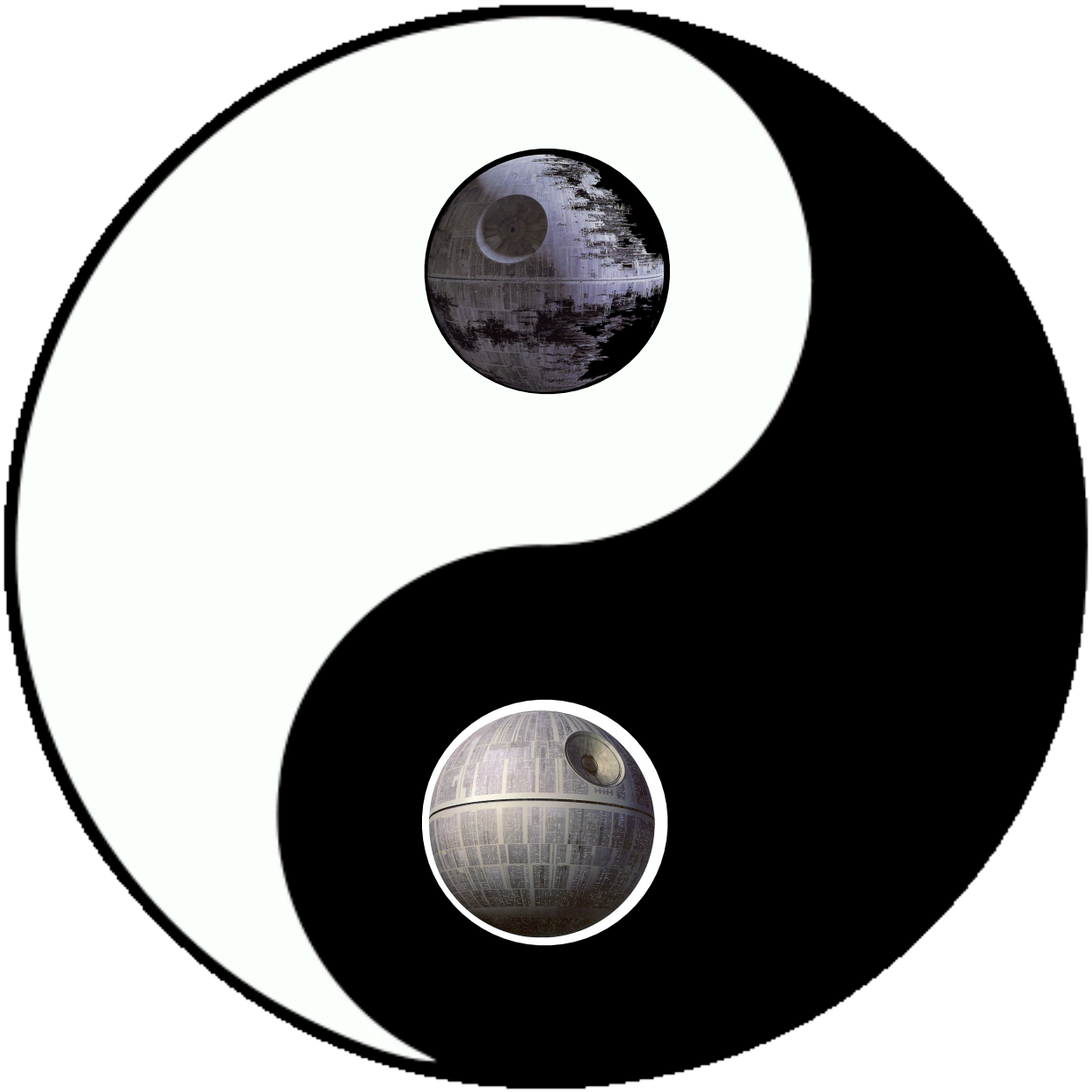
- Where's My Water, an alpha-stage port to Firefox OS (booted on a Nexus 4)
- Unreal Tournament, Sanctuary level
- FreeDoom (Boon) embedded in a worker inside BananaBread (only ~100 lines of JS to integrate the two)

End Demos
Back to Responsibilities

John Henry vs. the Steam Hammer

Responsibilities...

“Two of them b!tches” -Skinny Pete



Always Bet On...WTF Evolution?

Always bet on JS

- First they said JS couldn't be useful for building "rich Internet apps"
- Then they said it couldn't be fast
- Then they said it couldn't be fixed
- Then it couldn't do multicore/GPU
- Wrong every time!
- My advice: always bet on JS