NATIVE SPEED ON THE WEB

JAVASCRIPT & ASM.JS

ALON ZAKAI

@kripken

mozilla

THE WEB

Biggest **open** and **standards-based** platform

Great way to reach people

Would be great if it were **fast** as well ;)

HOW FAST IS IT?

SPEED ON THE WEB

Speed is determined by many things

DOM, **graphics**, **JavaScript**

This talk is about **JavaScript**

HOW FAST IS JAVASCRIPT?

The **Epic Citadel demo** is one way to measure

Large C++ codebase compiled to JS using **Emscripten**

Uses only **standard web technologies**, JS and WebGL

High-end native apps can run in JS!

EPIC CITADEL: PROGRESS

Launched March 2013, browsers improved since

back then	today
only ran in Firefox	runs in Firefox, Chrome
20 sec. startup	10 sec. startup
40fps	60fps in Firefox, Chrome

JS engines don't always run at full speed

Motivated **asm.js**, an **easy to optimize** subset of JS

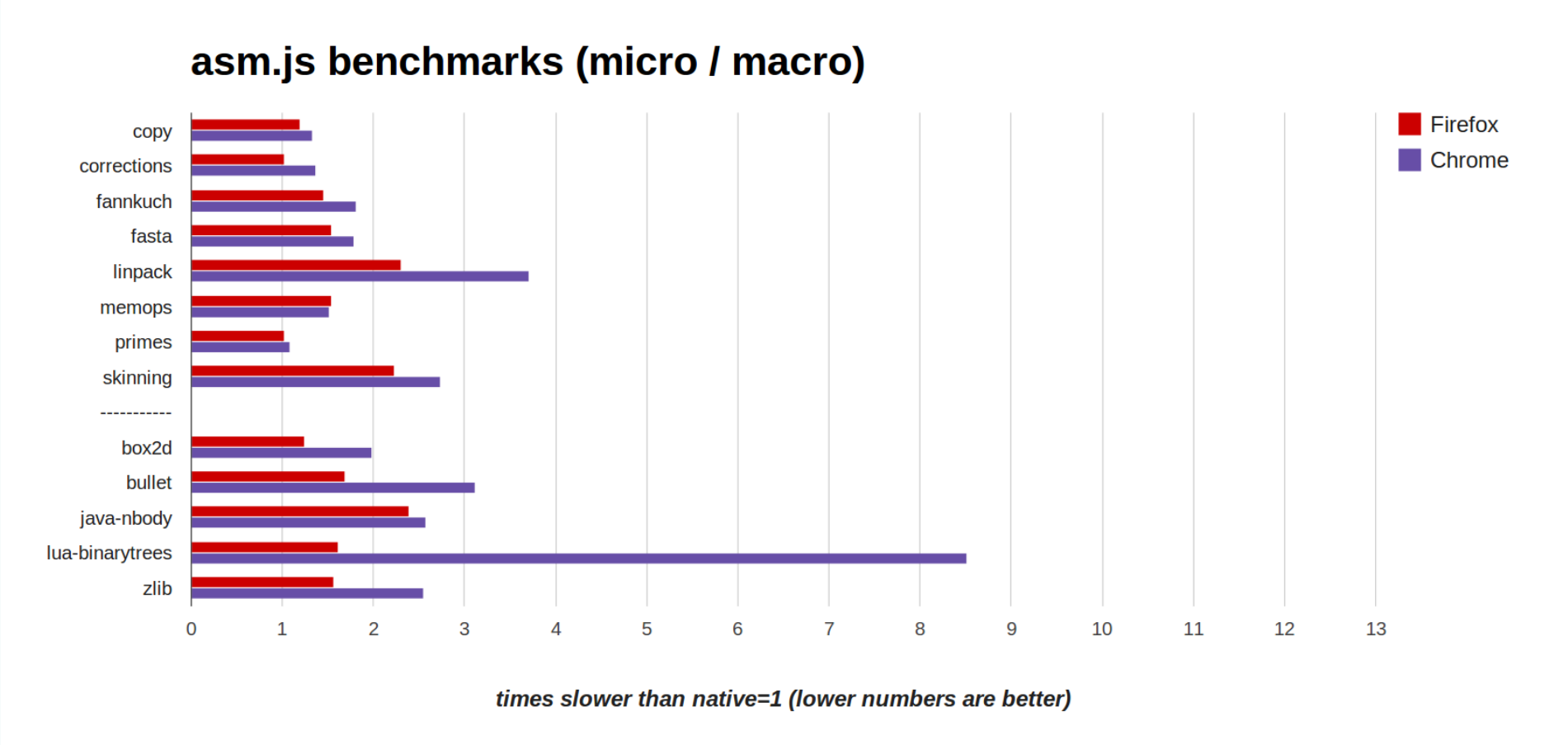
```
function asmCode(global, env, buffer) {
  'use asm';
  var HEAP = new global.Uint8Array(buffer);
  function fib_like(x) {
    x = x|0;
    if ((x >>> 0) < 2) return HEAP[x]|0;
    return ((fib_like((x-2)|0)|0) + (fib_like((x-1)|0)|0))|0;
  }
  return fib_like;
}
```

HEAP cannot be replaced

|0 trick ensures 32-bit ints

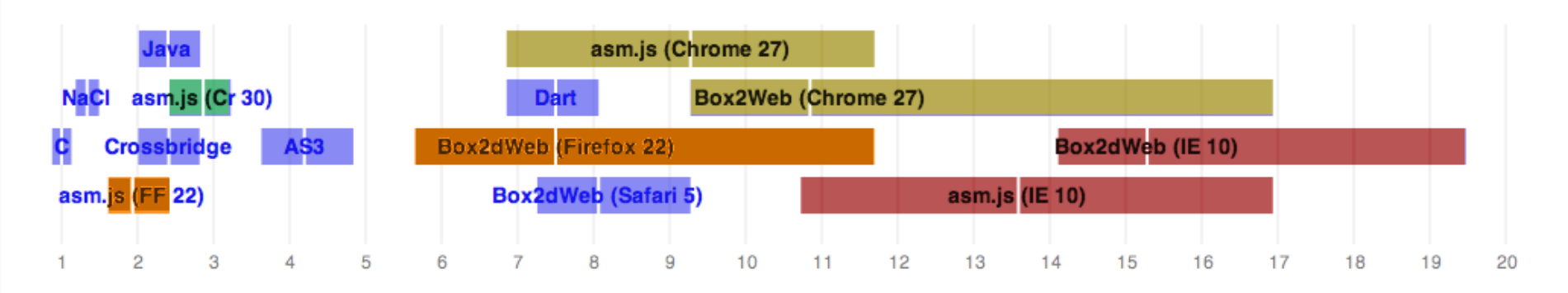
Typed arrays

ASM.JS PERFORMANCE



Emscripten benchmark suite (VMs and Emscripten from Sep 13 2013, run on 64-bit ubuntu 12.04)

BOX2D RESULTS



source: @jgw, Box2D Addendum; axis is time slower than native, lower is better

Box2dWeb - Port using "typical" JS

asm.js - Port using asm.js

asm.js is **faster** than "typical" JavaScript on Internet Explorer, Chrome and Firefox

ASM.JS - BACKGROUND

Began as a **research project** at Mozilla

Every compiler (CoffeeScript, Google Web Toolkit, etc.) generates a particular pattern of JS

Emscripten and Mandreel **converged** on a pattern for compiled **C++** in JS

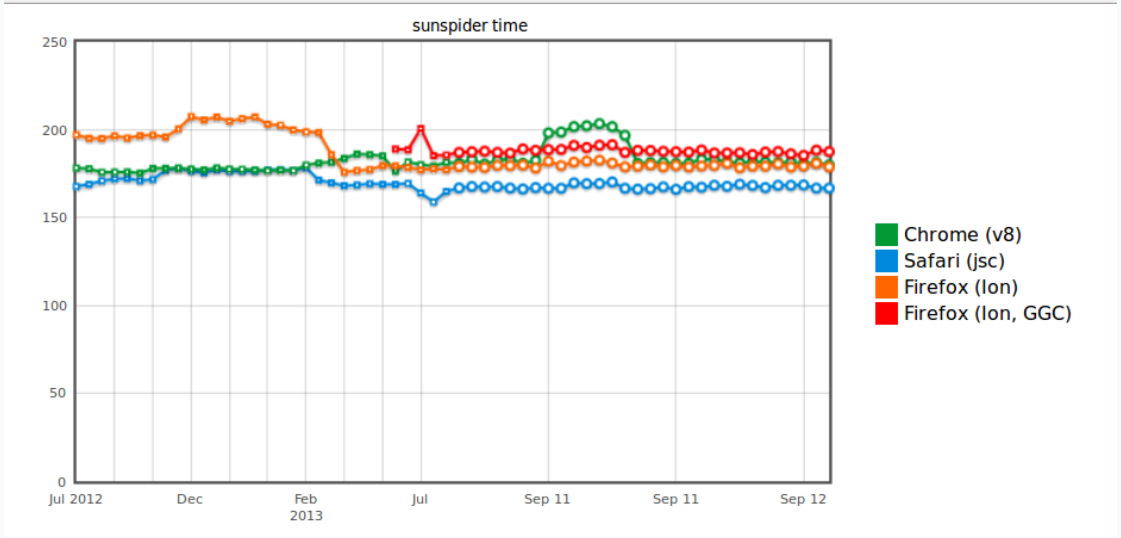
asm.js is a **formal definition** of that pattern, with some improvements

Ok, near-native speed on **Firefox** and **Chrome**...

...but some of us use **Safari** or **IE**?

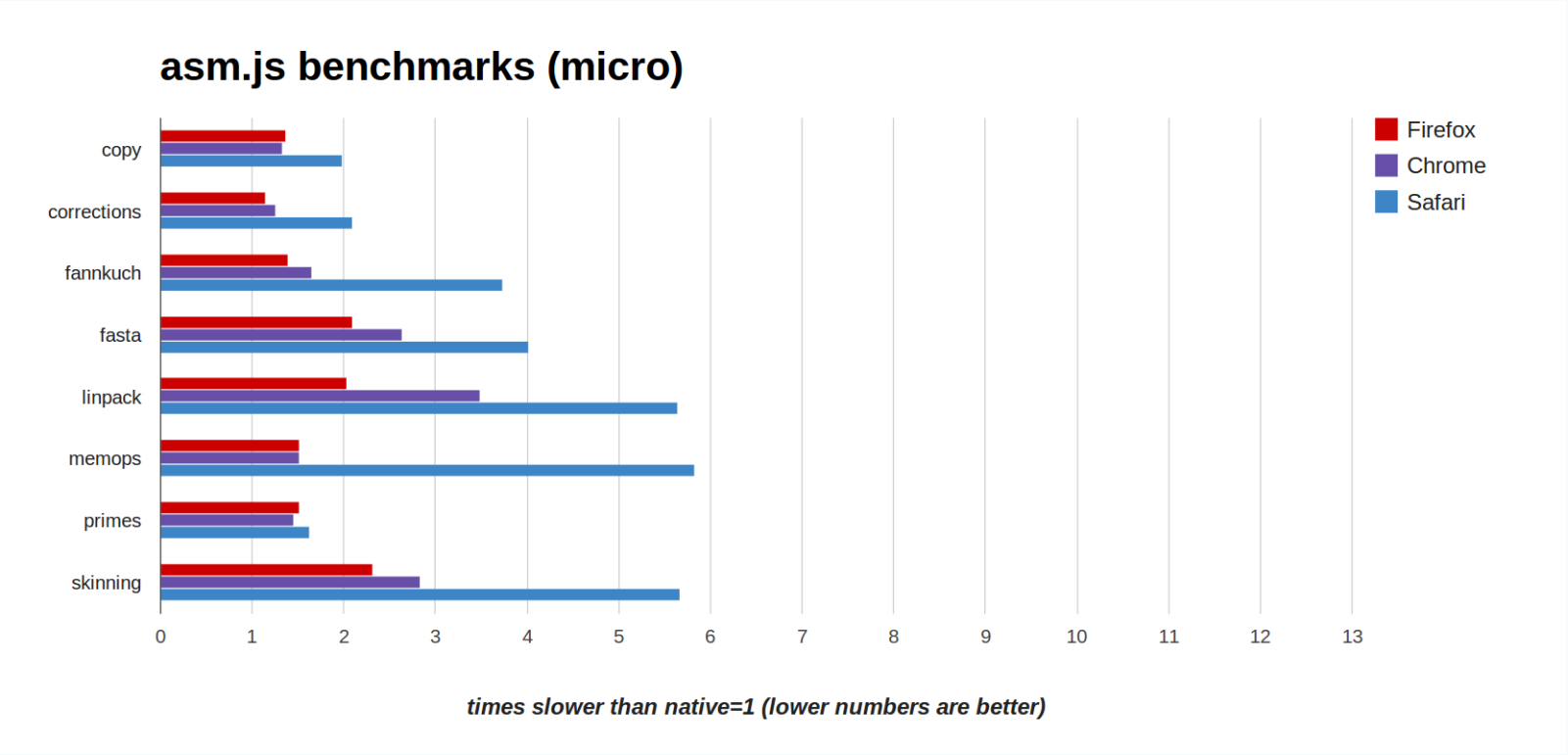
SAFARI

All browsers optimize JS, but decide which aspects to focus on at which times



source: **arewefastyet.com**; x axis is time, y axis is milliseconds (lower is better)

SAFARI



Very close on some, farther on others

Experimenting with using **LLVM as a JIT**

INTERNET EXPLORER

IE11, currently in pre-release, looks **promising** - I've seen it beat other browsers on some JS tests

Supports **WebGL!**

So safe to assume typed arrays will be fast, which means **asm.js** will be fast

BIG PICTURE

JS can run at about **half** the speed of native code

Why just half - what are the **remaining issues**?

1. 32-BIT FLOATS

JavaScript numbers are **64**-bit doubles

Often take more CPU cycles to compute

Require more memory bandwidth

Preliminary tests on Firefox show **10-20%** speed difference on relevant code

32-BIT FLOATS

Sometimes possible to do 32-bit math as an **optimization**

```
var floats = new Float32Array(calc());
floats[0] = floats[1] + floats[2];
floats[1] = floats[0] + floats[2];
floats[2] = floats[1] + floats[2];
```

Mathematically provable that those additions can be 32-bit

No reason JS engines cannot do this **right now** (and Firefox is **working on it**)

32-BIT FLOATS

A "fragile" optimization though

```
var floats = new Float32Array(calc());
floats[0] = floats[1] + floats[2] + 1; // can't
floats[1] = floats[0] + floats[2] + 1; // be
floats[2] = floats[1] + floats[2] + 1; // optimized
```

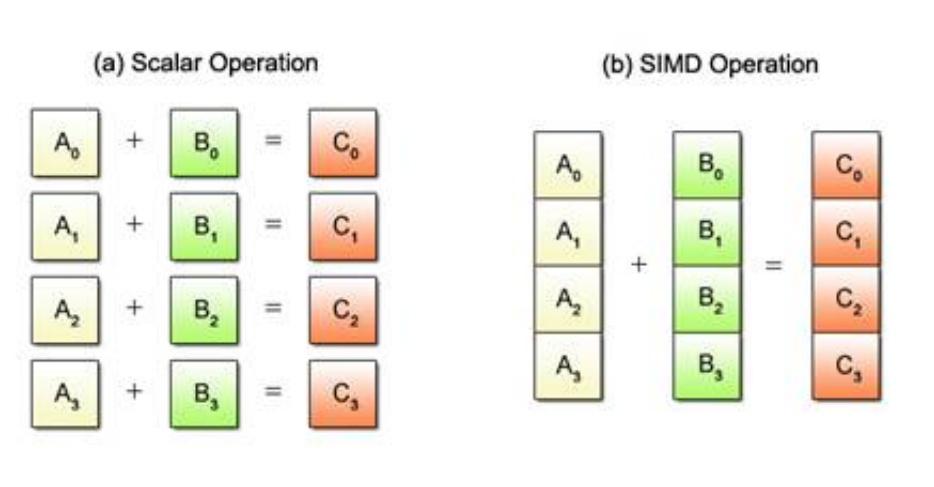
But ES6 (proposal for the next version of JS) includes **Math.fround**, which rounds to 32-bit precision

```
var floats = new Float32Array(calc());
floats[0] = Math.fround(floats[1] + floats[2]) + 1;
floats[1] = Math.fround(floats[0] + floats[2]) + 1;
floats[2] = Math.fround(floats[1] + floats[2]) + 1;
```

Now it's optimizable!

2. SIMD

Single Instruction, Multiple Data: **SSE**, **NEON**, etc.



source

Large parts of CPUs dedicated to SIMD

Usable from C++, Mono, Dart, etc.

SIMD

John McCutchan from Google wrote a **proposal for SIMD in JS**

```
var x = float32x4(1, 2, 3.14159, 22.5);
var y = float32x4(0, 0, 1, 0);
var z = SIMD.add(x, y);
```

Immutable 128-bit values, designed for what CPUs provide

Potentially big (e.g. 300% in some cases) speedups on certain types of code

Collaboration is ongoing with Google, Mozilla, Intel and TC39

3. THREADS

Web workers allow use of multiple CPU cores

Can **transfer** typed arrays, no copying

But threads can't read&write to the **same** data

Good for preventing data races, but **bad** for some types of projects (for example, modern game engines are heavily multithreaded)

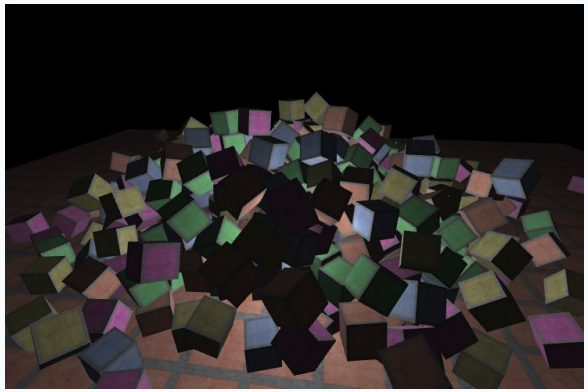
THREADS

Various proposals, some experiments, but no clear direction yet

Would require all vendors to agree and standardize something new and complex

Part of the challenge is figuring out how it **fits in** with the rest of the web, which does **not** currently have data races at all

Already many apps run +- native speed



BananaBread, a port of the **Sauerbraten** First Person Shooter (FPS) to JS+WebGL

Running **inside** it: **Boon**, **another** FPS

(= Doom code (**PrBoom**) + **Freedoom** content)

DEMO TECH DETAILS

Boon runs in a **web worker**

Input events are **proxied** to it, output of software renderer is sent back

We push those pixels to a **WebGL texture**

Less than **100 lines of code** to integrate the two

Both games run at once at **60fps** on a very average laptop

SUMMARY

JS is **close and getting closer** to native speed

Even on missing pieces (SIMD, float32, etc.), **progress is happening**

That's it! :) Questions?

(these slides were just tweeted **@kripken**)