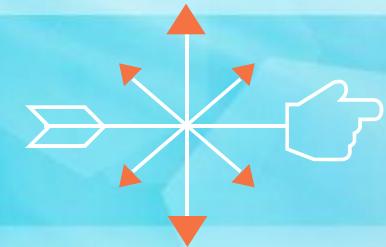


Qualcomm Research

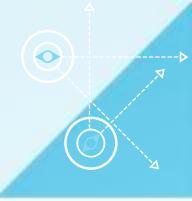
Silicon Valley



Are scripting languages ready for mobile computing?

Calin Cascaval, cascaval@qti.qualcomm.com

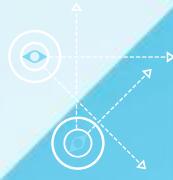
Feb. 18, 2014



Outline

- Mobile computing -- what is the fuss?
- Developers and their tools -- who is writing code?
- Browsers and JavaScript -- the push toward platform independence

Mobile Market

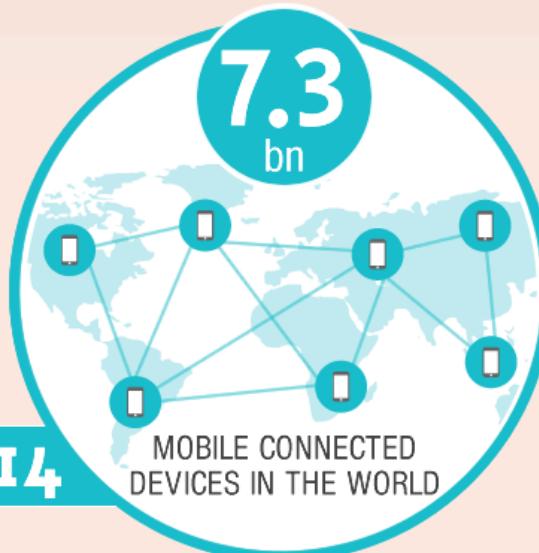


BROUGHT TO YOU BY: **Illuminas**
Global market research

MOBILE DEVICE USAGE IS EXPLODING

New age of mobile World

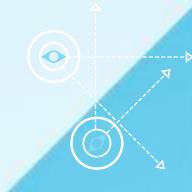
By **2014** there will be more mobile-connected devices on Earth than people¹



MORE
THAN

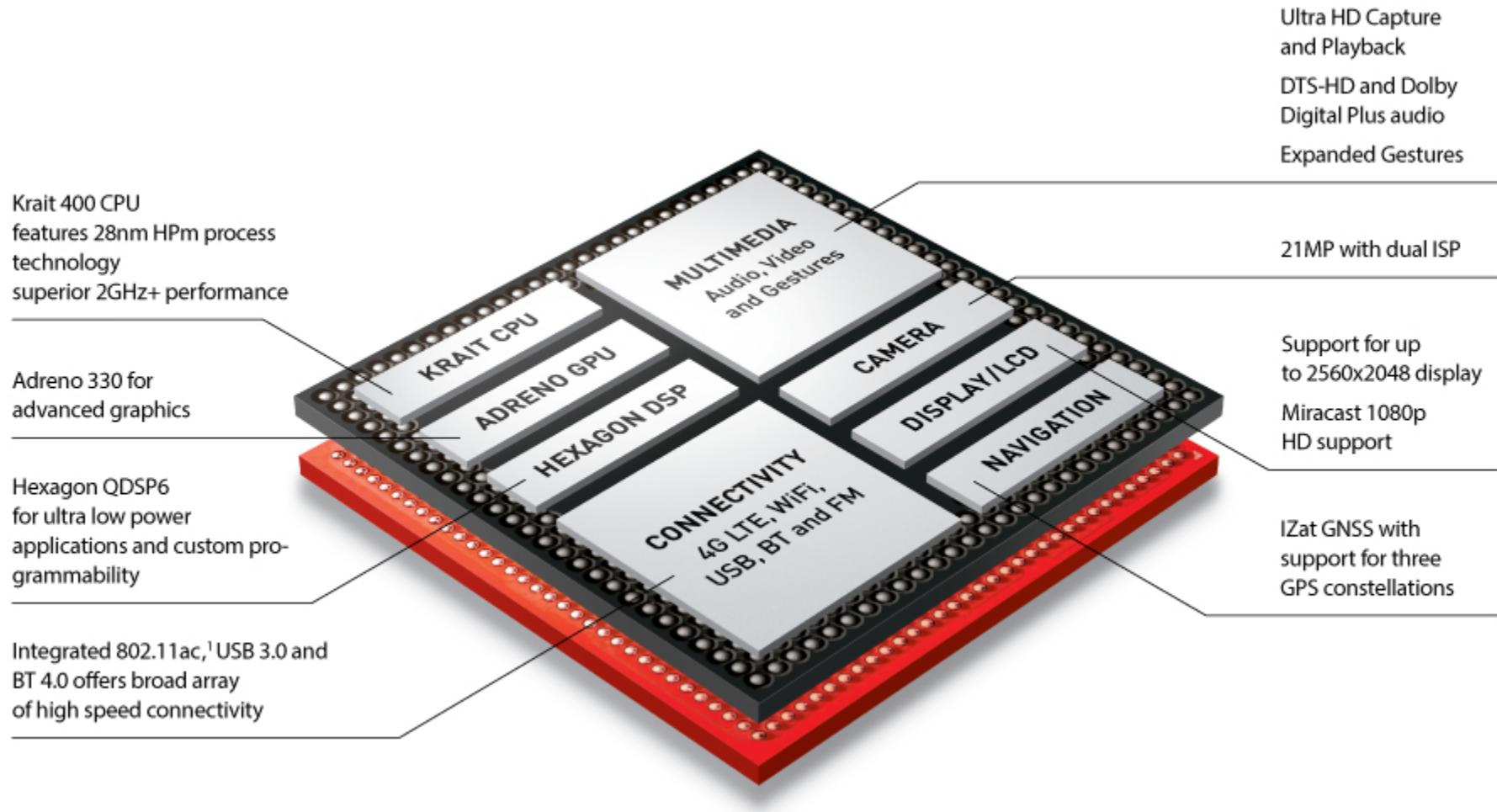


Mobile Computing Landscape



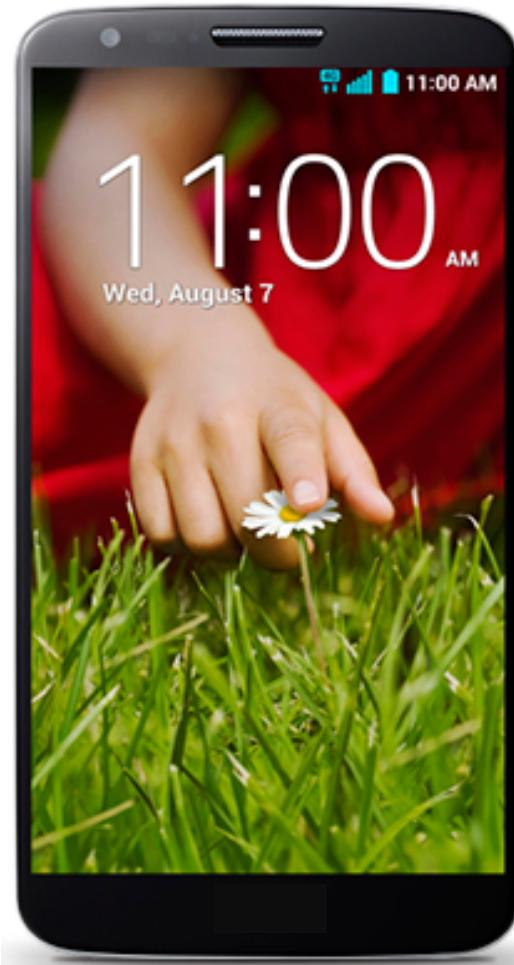
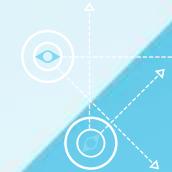
What's in a smartphone?

Anatomy of a Mobile SoC: Qualcomm Snapdragon 800



Source: <http://www.qualcomm.com/snapdragon/processors/800>

Mobile Constraints

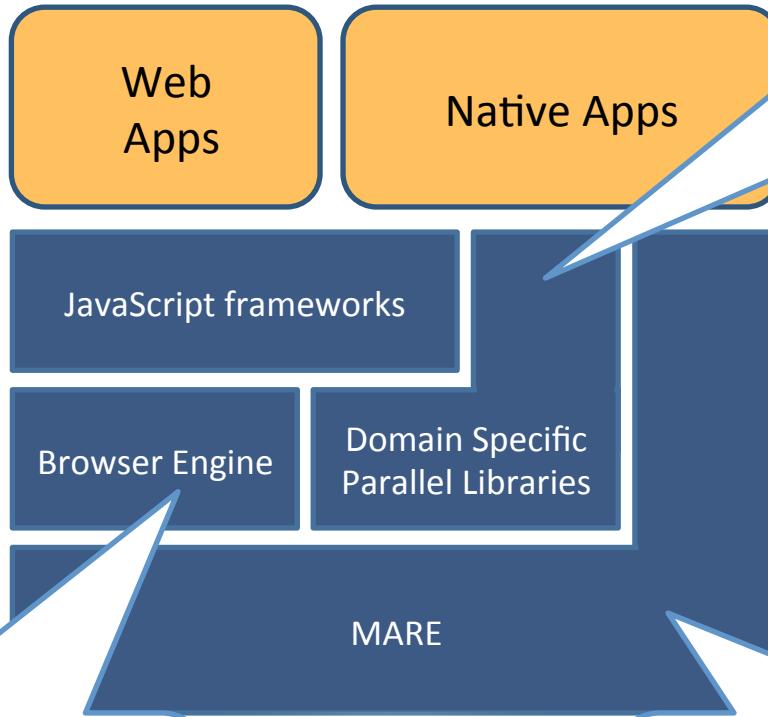


Energy



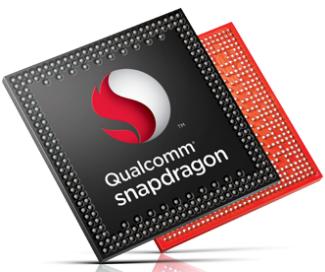
Thermal

Mobile Computing Software Stack



Portability: Parallel Browser

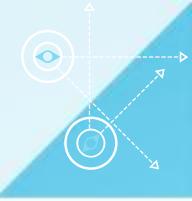
- Use of concurrency to optimize execution of Web Apps
- Muscaliet JavaScript Engine
<http://github.com/mcjs/mcjs.git>



Programmability: MARE

- Parallel, heterogeneous programming
- Power and performance optimizations

<http://developer.qualcomm.com/mare>



Scripting languages?

JavaScript!

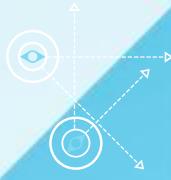
JavaScript: the language of the Web



JavaScript: Language features

- Dynamic types and operator overloads
 - Type of variables can change
 - Operators change behavior based on operand types
 - E.g. “+” changes to string concatenation either operand is string
 - E.g. “++” applies a type conversion if operand is not number
 - => “++” is not the same as “+=1”
 - E.g. “<, >, >= ...” apply type conversion to right operand based on the type of left operand
 - => “a < b” is not equivalent to “b > a”

```
var x = 0;
for (var i = 0; i < 5; ++i) {
    if (x > 5)
        x += ".";
    x += i;
}
print(x + " is " + typeof(x));.....→ 6.4 is string
print(x + 1);.....→ 6.41
print(6.4 <= x);.....→ true
print(x >= "6.4 ");.....→ false
print(++x + " is " + typeof(x));.....→ 7.4 is number
print(x + 1);.....→ 8.4
```



JavaScript: Objects

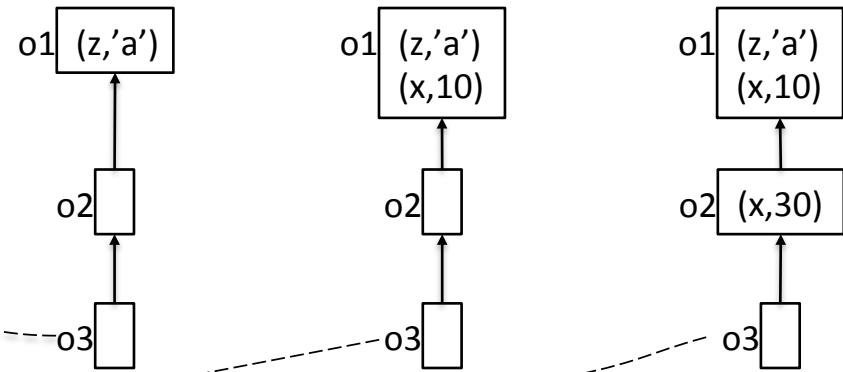
- JavaScript objects are nested hash tables (a.k.a prototype chain)
 - Semantically $o.x \equiv o["x"]$

```
var o1={ };
function F1(){ this.z = 'a'; }
F1.prototype = o1;

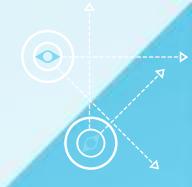
var o2 = new F1();
function F2(){}
F2.prototype = o2;

var o3 = new F2();

print(o3.x); .....> undefined
o1.x = 10;
print(o3.x); .....> 10 <-
o2.x += 20;
print(o3.x); .....> 30 <-
```



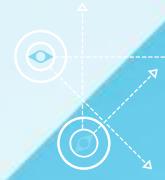
Status after property assignments



JavaScript: Language features (cont.)

- The “arguments” keyword allows alternative access to arguments of a function and its callers
- Function scope: is the only scope in JavaScript. All declarations (even within condition blocks) are moved to the top
- Arrays are special hash tables with different rules for property attributes and prototype chain
- The undefined and null values are the default in various seeming similar situations. The values behave differently in some particular situations

JavaScript Standardization



ECMAScript v1

- Brendan Eich: Mocha, LiveScript
- Initial version, supported in Netscape
- IE (JScript)

ECMAScript v5

- Strict mode
- Object reflection and properties
- Getters and setters
- JSON support

ECMAScript v7

- Promises/concurrency
- Math and numbers
- Guards and trademarks
- Value types
- Operator overloading and traits

ECMAScript v3

- Regular expressions
- try/catch
- strings and number formating

ECMAScript v6

- Classes and modules
- Iterators
- Generators and generator expressions

1995-1997

1999

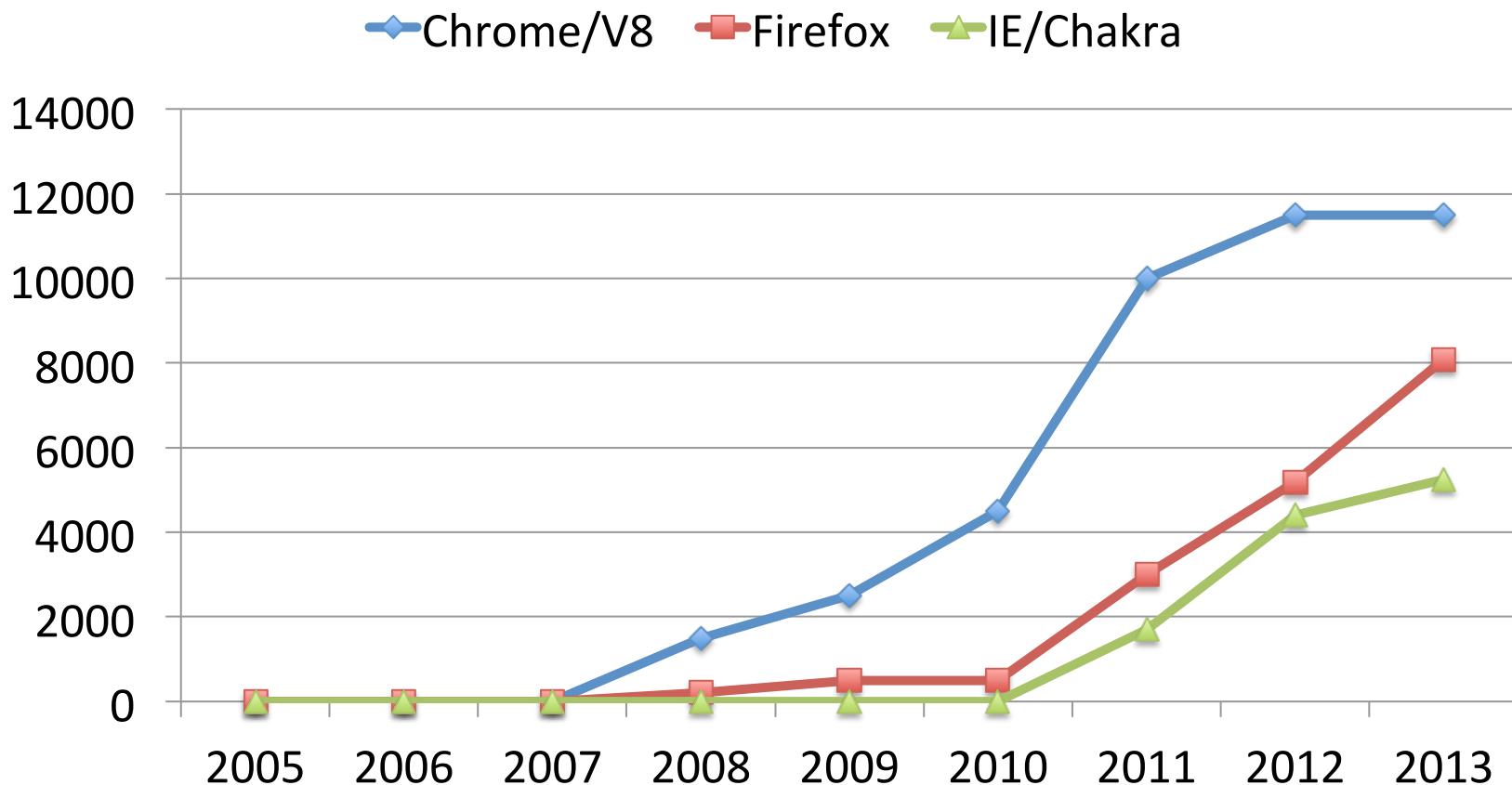
2009

2013+

Historical JavaScript Performance

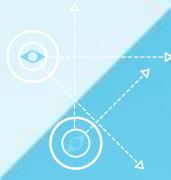


V8 Benchmark Score -- Higher is better



Sources: <https://confluence.ontotext.com/display/ResearchSpace/JavaScript+Frameworks>
<http://www.zdnet.com/the-big-browser-benchmark-january-2013-edition-7000009776/>

JavaScript Usage



jQuery

- What the DOM API should have been
- DOM navigation, traversal, animations
- Facilitates plugins on top of JavaScript

GWT

- Java to JavaScript
- In-browser debugging and optimizations

asm.js

- Restricted JavaScript that can be used as a target by compilers

Node.js

- Server-side JavaScript for networking apps
- Event-driven, non-blocking infrastructure to script highly concurrent programs.
- Built-in support network protocols: TCP, DNS, HTTP

Emscripten

- C++ to JavaScript
- “Big apps” running in the browser
- Generates asm.js

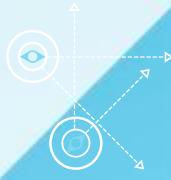
TypeScript

- Superset of JavaScript with type annotations and objects

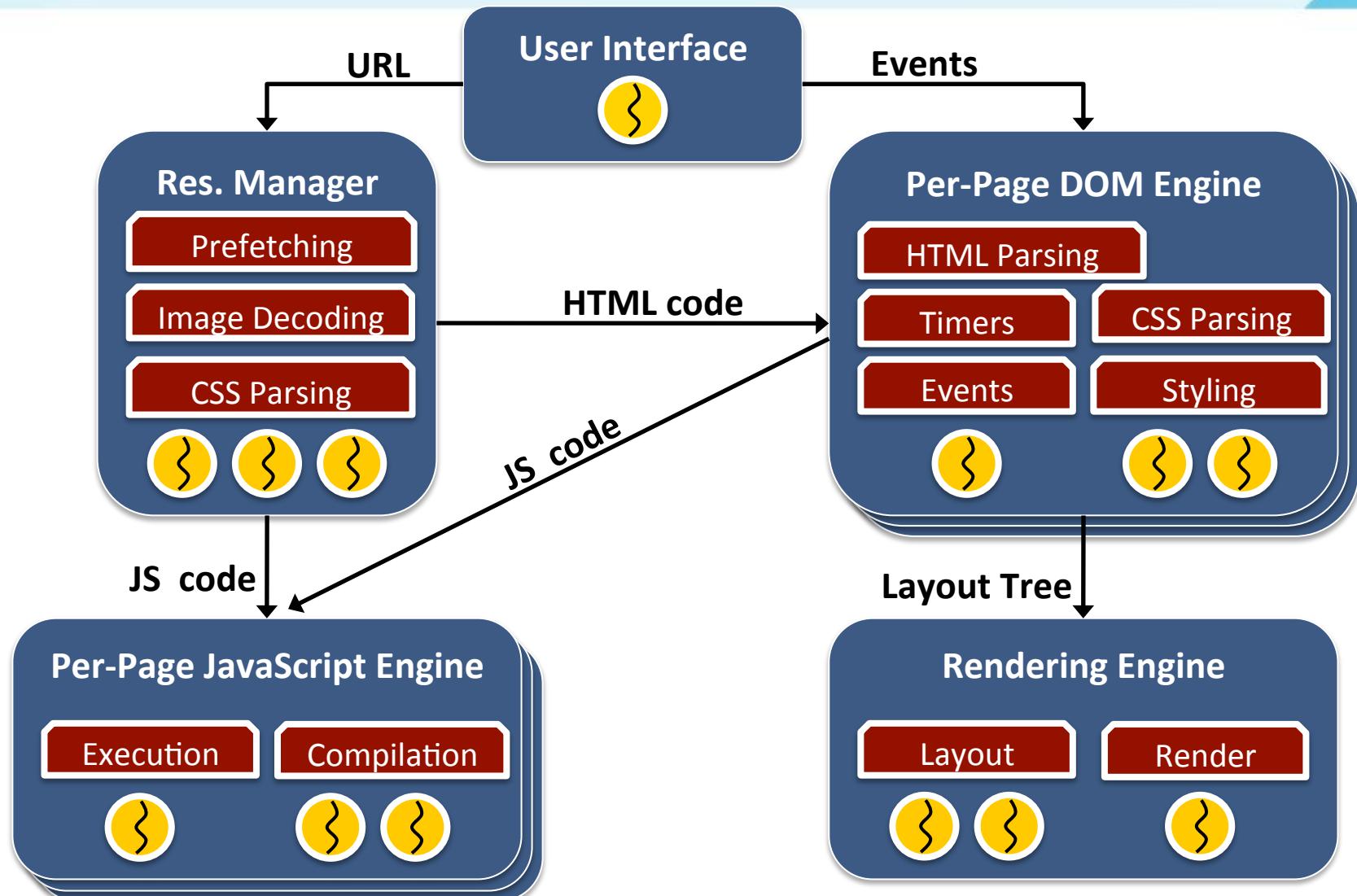
CoffeeScript

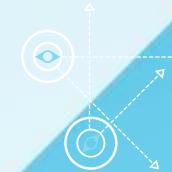
- Restricted JavaScript
- Extended with some nice features inspired by Haskell and Python

Muscaliet JavaScript Engine



Qualcomm Zoomm Browser: Pervasive Concurrency





1st

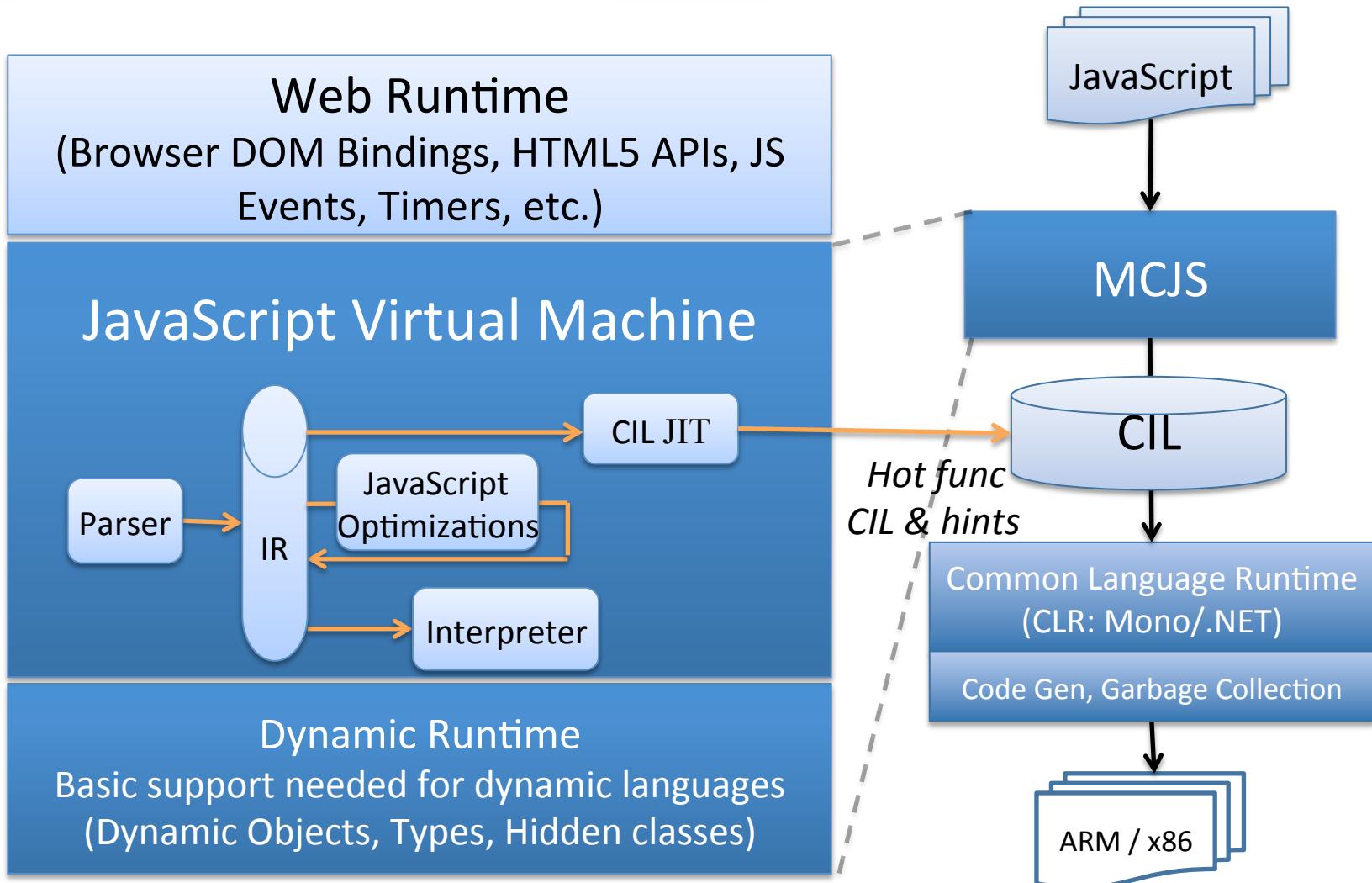
Execution

Full JIT
Compiler

Type
Inference

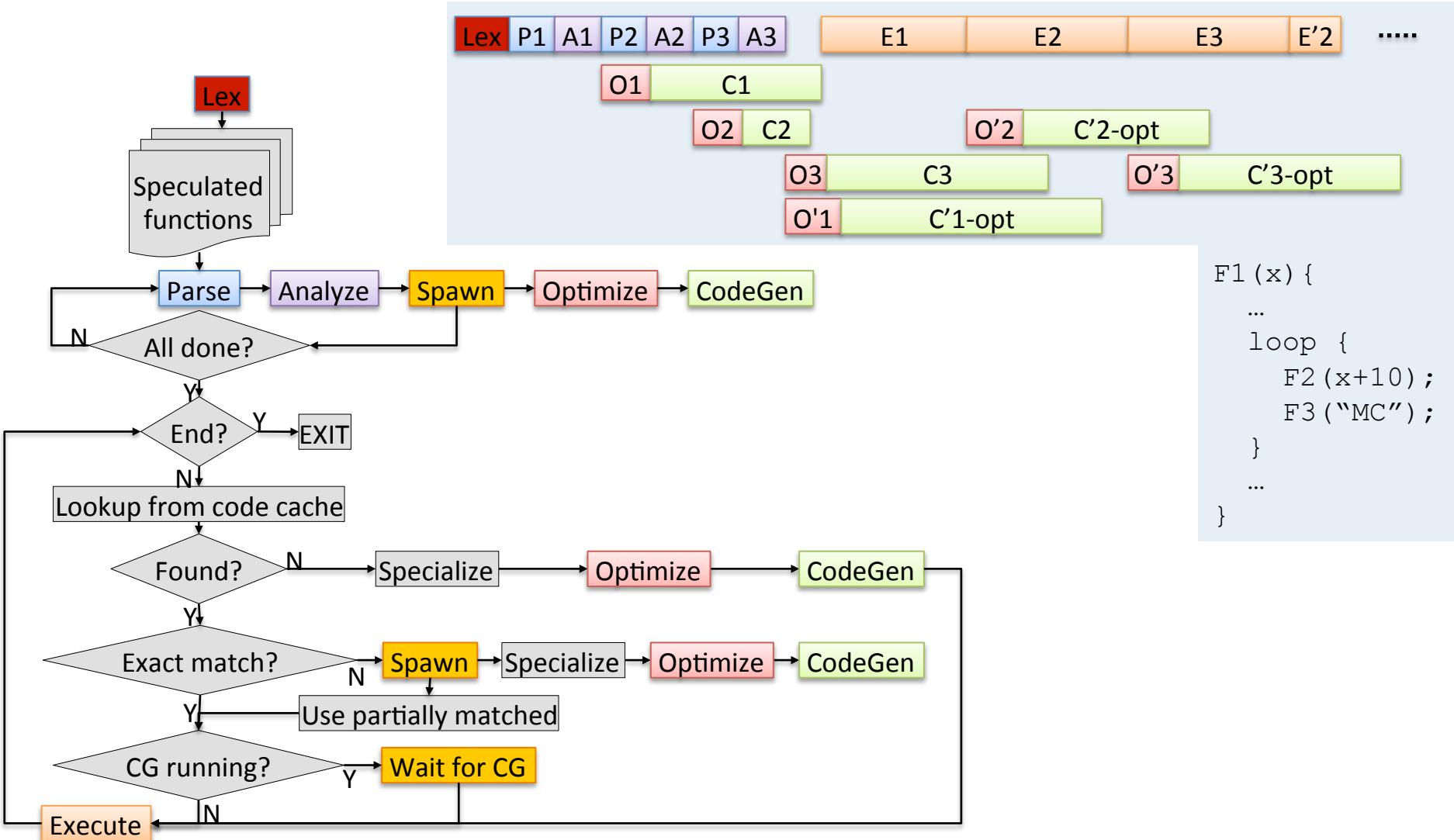
JITted code

MCJS Engine Architecture

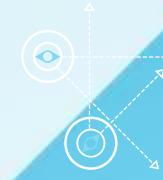




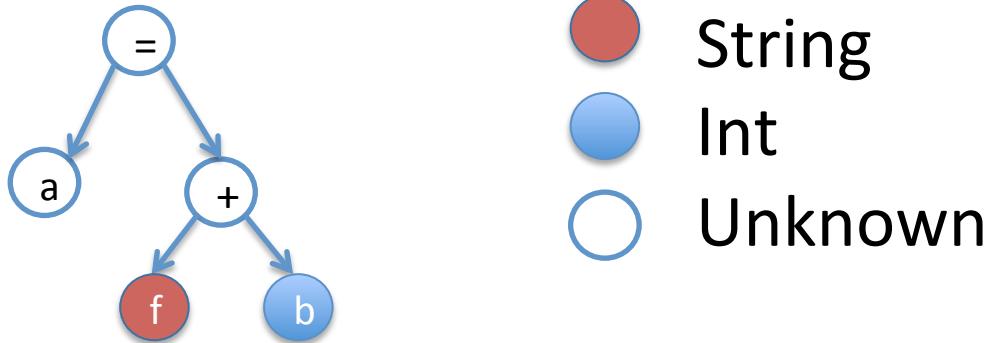
MCJS: Parallel JIT



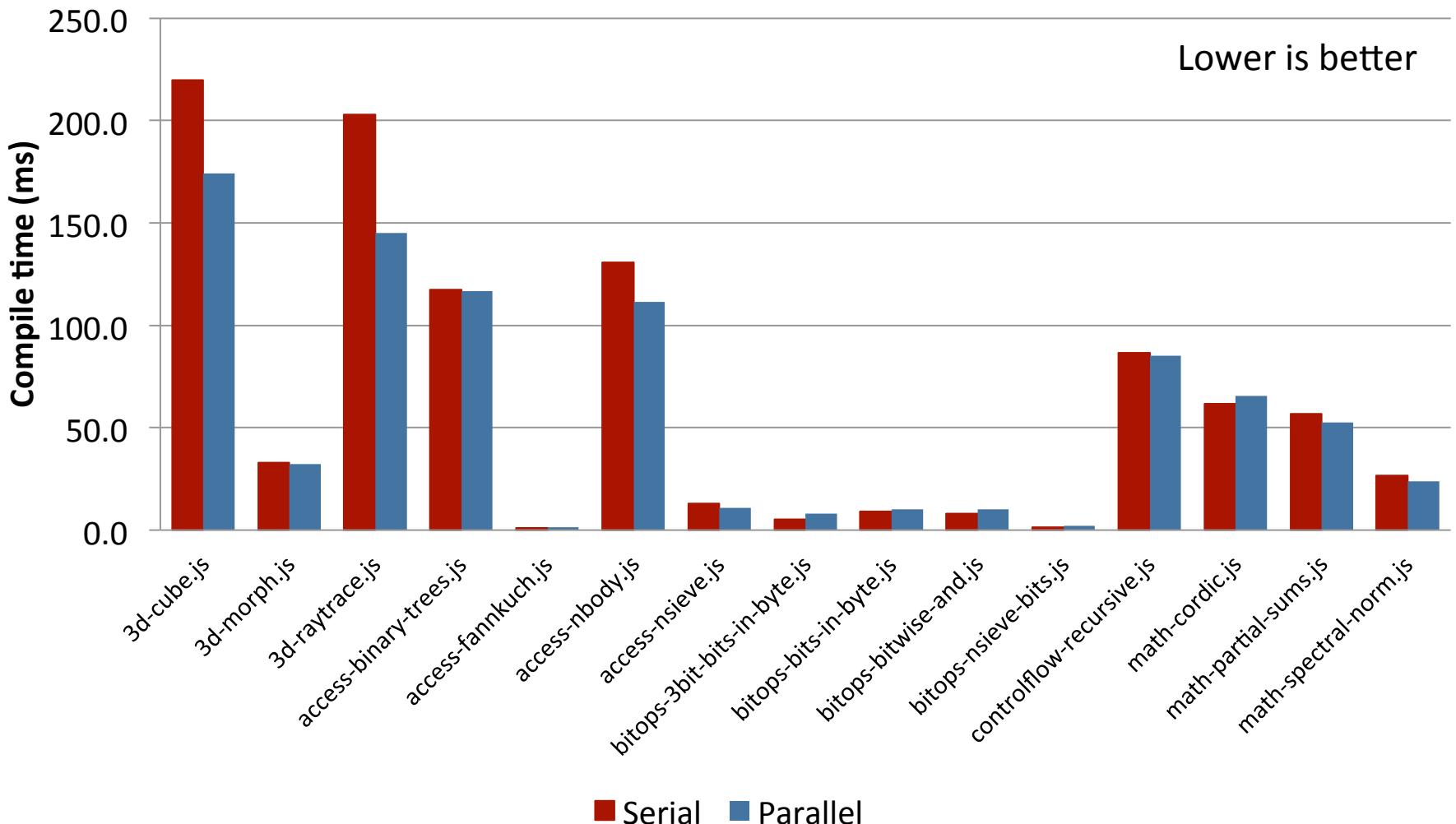
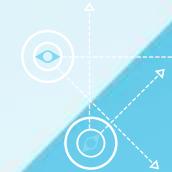
MCJS Type Inference Engine

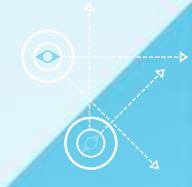


- Simple worklist based algorithm, using a type lattice
- Infers the type of local symbols
- Makes most of the internal operations run faster
 - e.g. $a = f + b$



MCJS: Parallel Compilation





Lessons

What worked

A layered architecture that allows for quick development and design iteration

Type inference and language level optimizations really pay off

Parallelism pays off for long compilation times

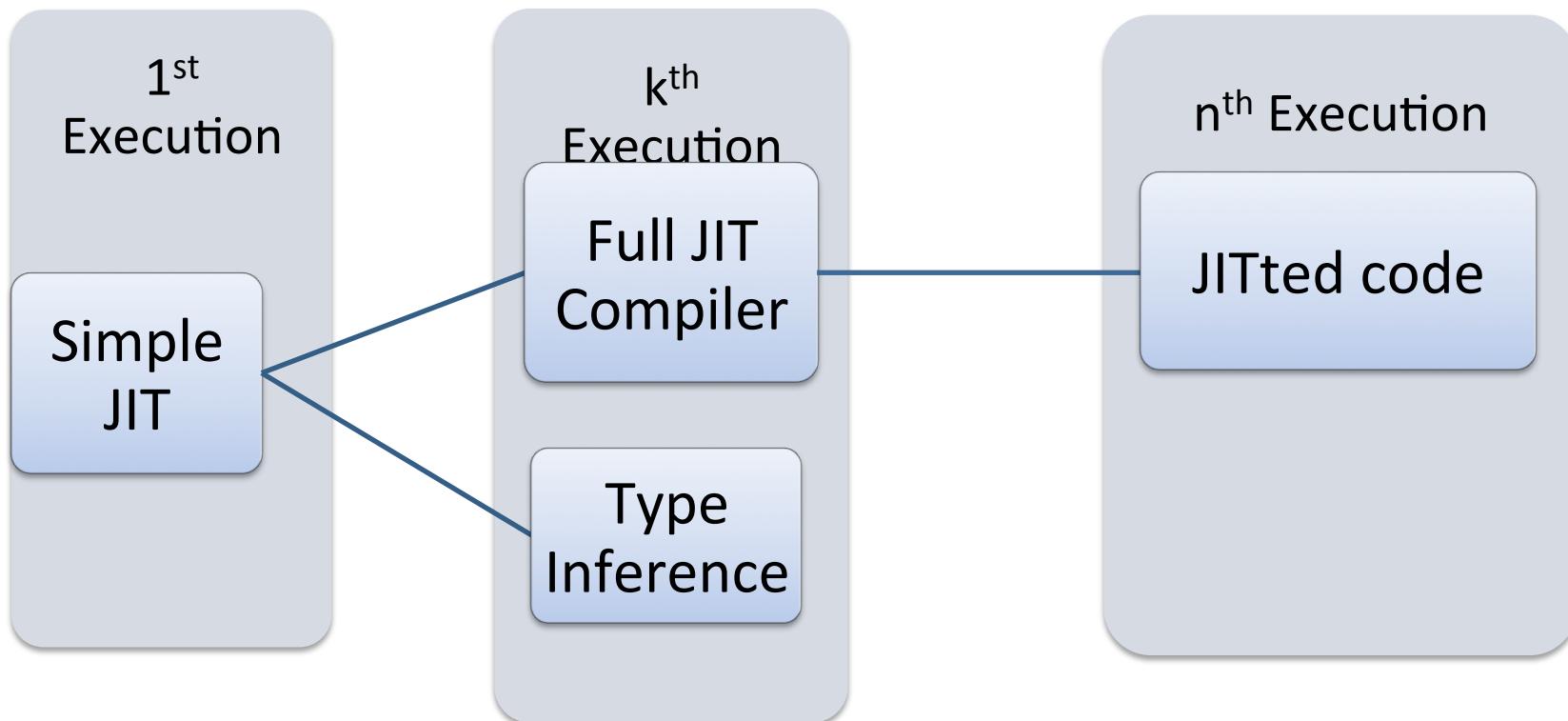
Proper browser integration is essential to reduce overhead

What did not

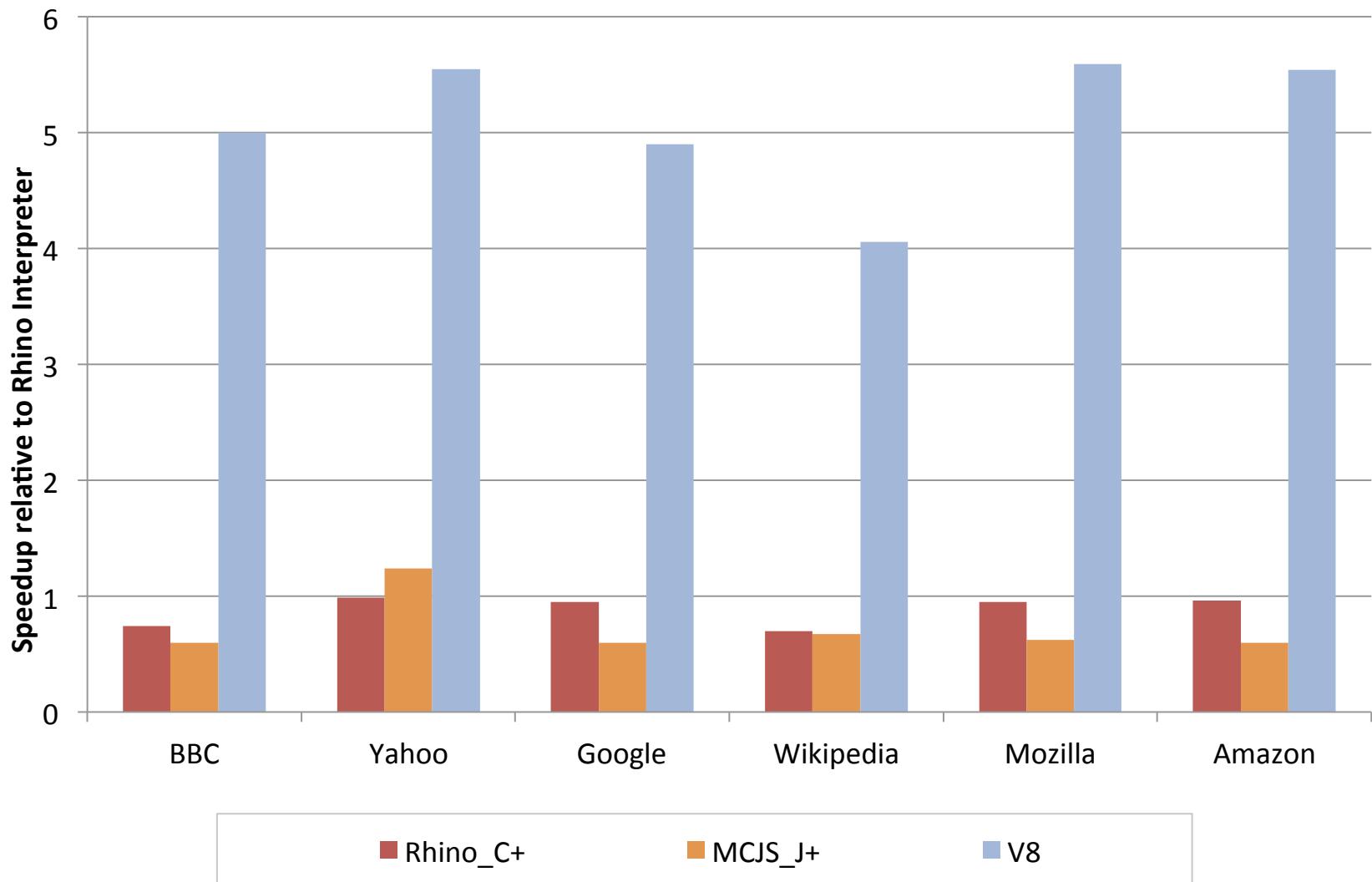
Full compilation works well for heavy benchmarks, but it is difficult to amortize

Concurrency opportunities and speculation require highly tuned heuristics, different across workloads

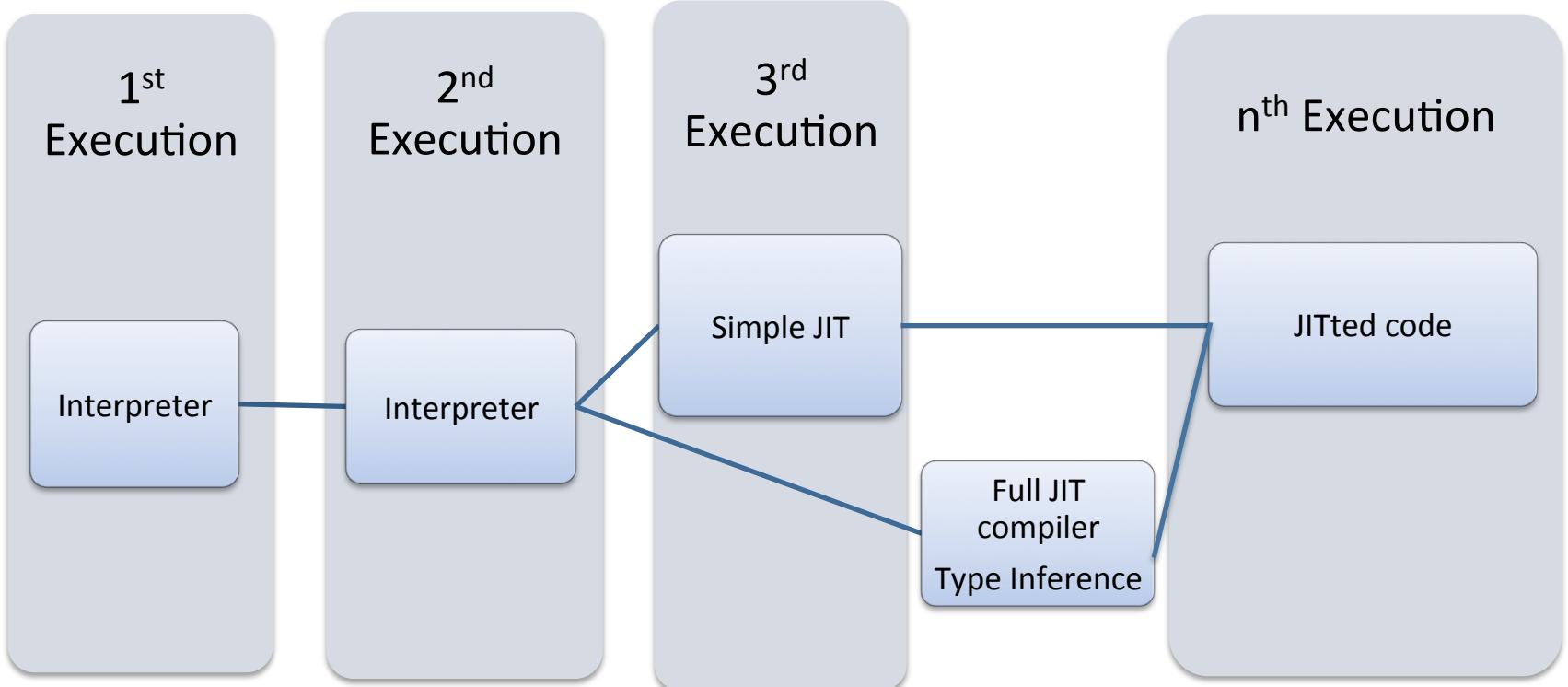
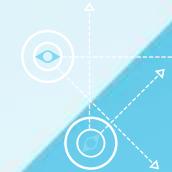
MCJS: Reducing compilation overhead



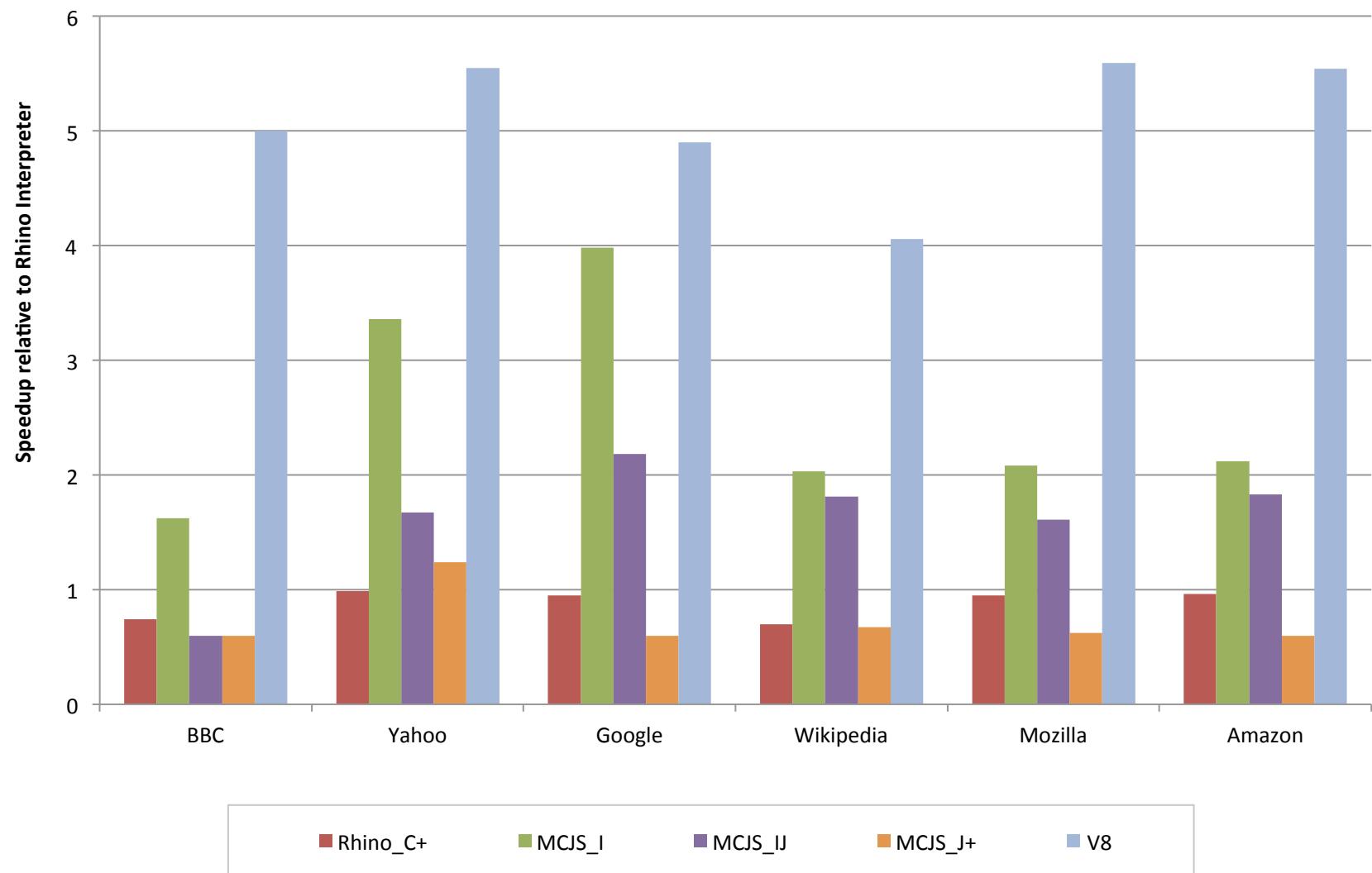
Website performance



MCJS: How about an interpreter?



Website performance



Lessons



What worked

A layered architecture that allows for quick development and design iteration

Fast compilation/interpretation is essential since there is almost no reuse for code in web pages

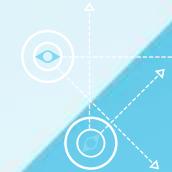
Benchmarks and web apps require a full compiler

What did not

Interaction with the native JIT has additional overheads

Language “features” prevent many optimizations

Challenges in type inferring JS code



```
var bar = 0

function foo()
{
    var a = 10;
    var b = a + 30;
    var c = a * 15 + b;
    a = bar + 2;
}
foo()
bar = "str"
foo()
```

Local symbols are blue
Global symbols are red

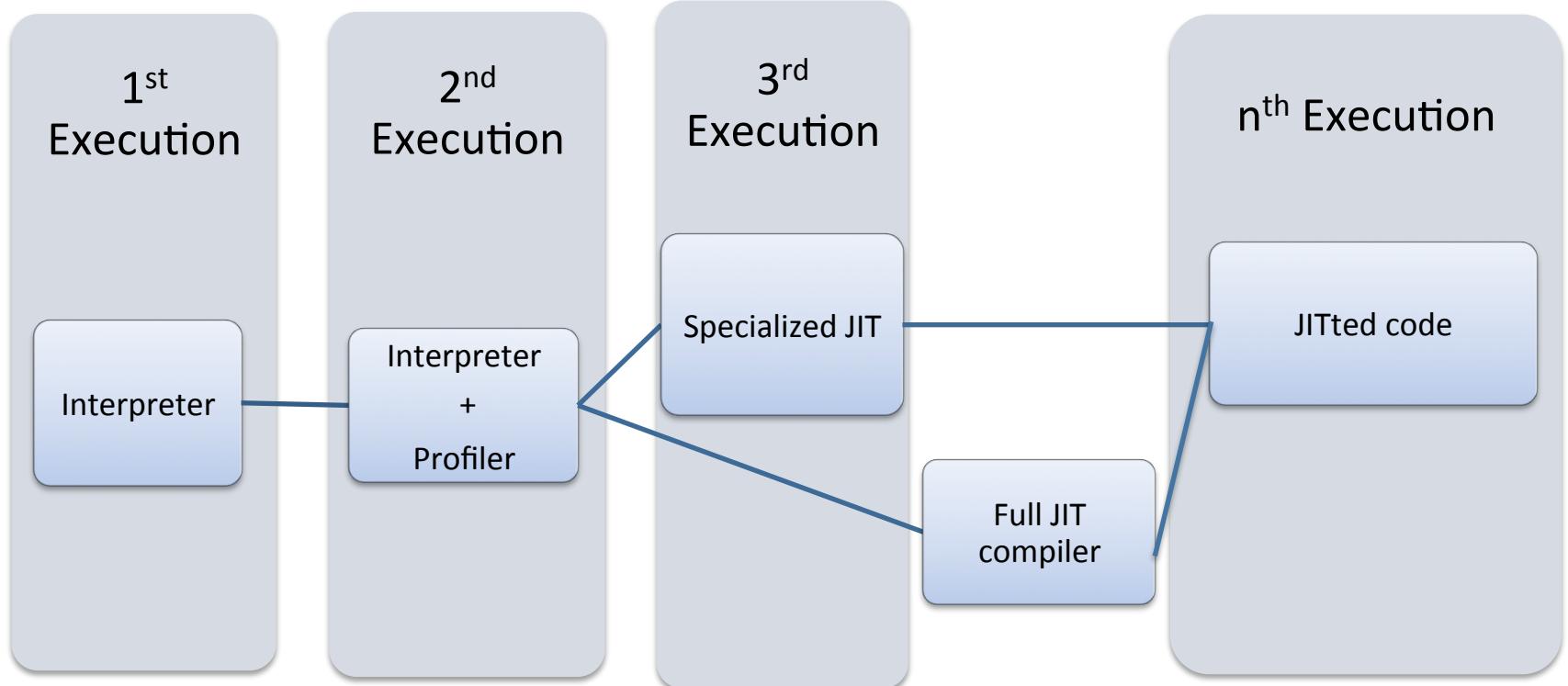
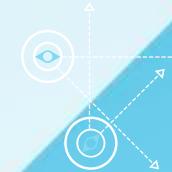
TI Result

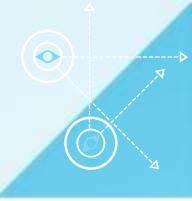
=====

a -> DValue
b -> DValue
c -> DValue
bar -> DValue

- Since **bar** is global symbol, there is no way we can infer its type.
 - Without **bar** everything would have been type inferred

MCJS: How about a profiler?

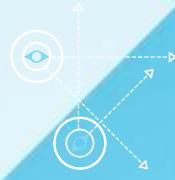




JavaScript on web pages

- 1.5% of the profiled nodes showed dynamic nature
 - Differs from previous published results because we profiled the nodes that would help the TI.
- 16% of the executed functions have nodes with dynamic types
- Only 2.7% of the callsites were polymorphic (had multiple call sites)

Type feedback driven type-inference



Reduce number of guards

Leverage type inference analysis to focus only on the dynamic variables

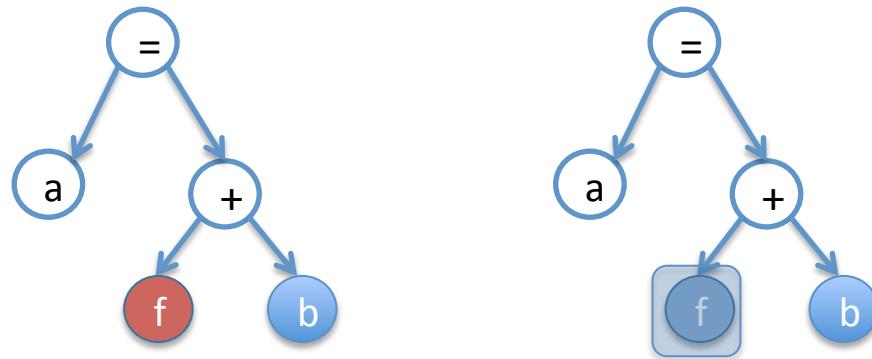
Arrays and strings are objects, inferring the index allows direct access

`a = f + b;`

Assume `b` is `int`
and `f` is a global symbol (`DValue`)

Augmented type info

Profiler information on inferred types is used to enhance compiler type information: more optimized static code





Example for profile driven TI

```
var bar = 0
function baz() { return 1; }
var obj = { i : 1 }
var arr = [1, "str"]

function foo()
{
    var a = 10;
    var b = 30;
    var c = a / 15 + b;
    var d = obj.i;
    a = bar + 2 + baz();
    c = d + a;
    d = b * c;
    arr[0] = arr[1];
}
```

 Represents 1 guard
Total 6!

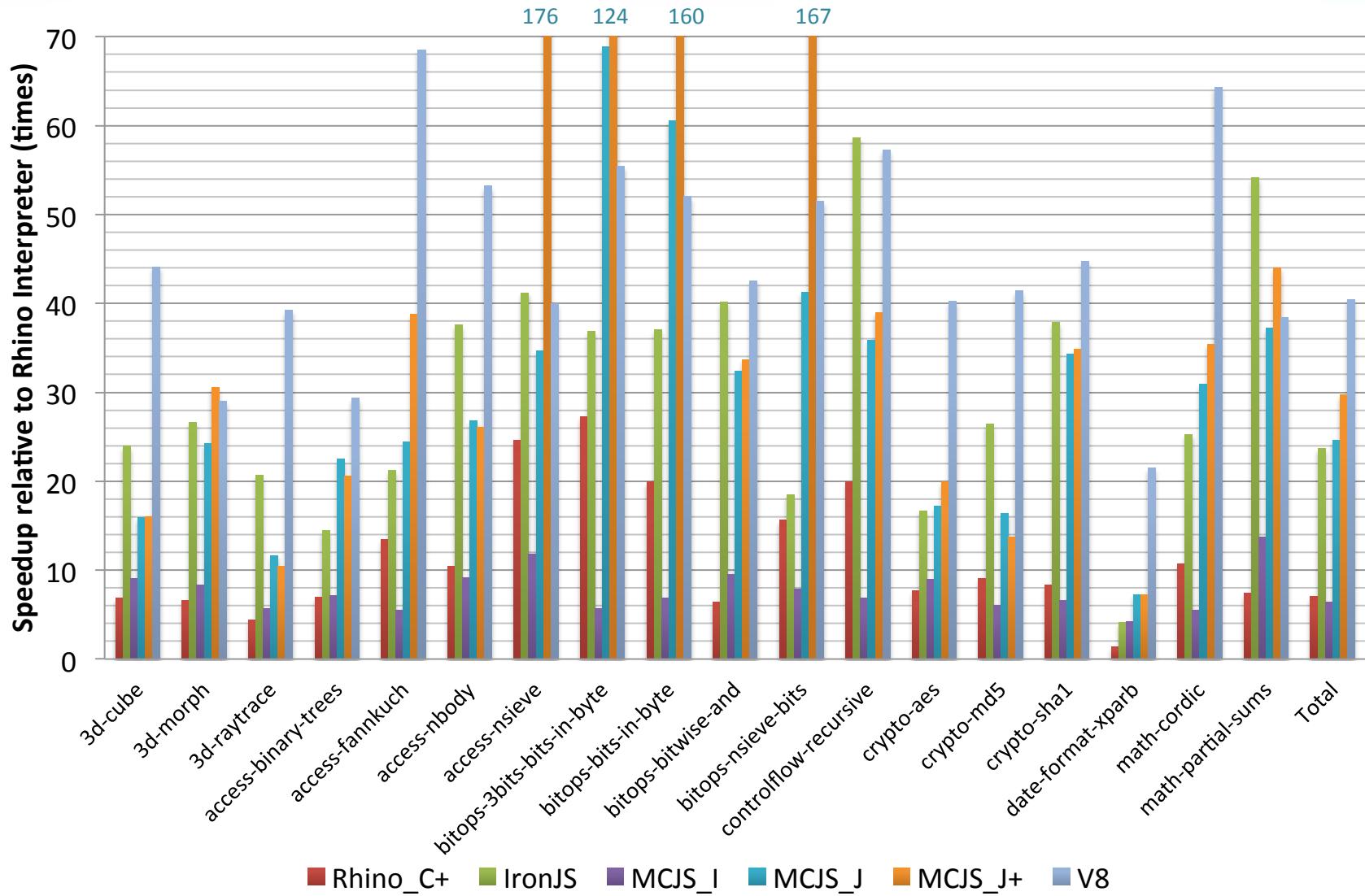
Old TI Result
=====

a	→ DValue
b	→ Int32
c	→ DValue
d	→ DValue
bar	→ DValue
baz	→ DValue
obj	→ DValue
arr	→ DValue

New TI Result
=====

a	→ Int32
b	→ Int32
c	→ Int32
d	→ Int32
bar	→ DValue
baz	→ DValue
obj	→ DValue
arr	→ DValue

SunSpider performance



Lessons



What worked

A layered architecture that allows for quick development and design iteration

Exploitation of program structure

There are many more compiler tricks that we've implemented.

What did not

However, compiler support can take it only up to a point

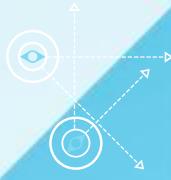
For more we need a system level approach, combining programmer knowledge, software architecture, and language design

And the right language for the task

See our upcoming VEE 2014 papers.

The code is available at: <http://github.com/mcjs/mcjs.git>

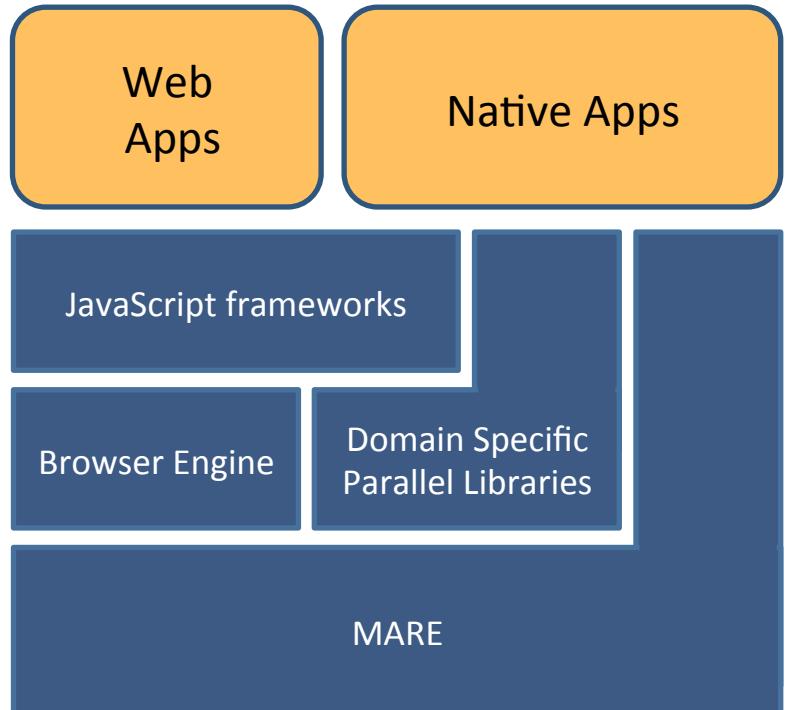
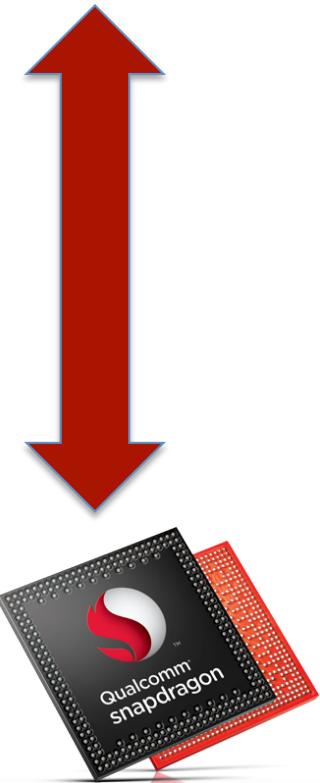
Are scripting languages ready for mobile computing?



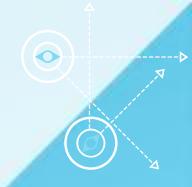
The Mobile Software Stack



Gap!



The BIG Challenge



Portability and ease of development

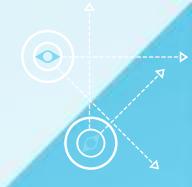
VS.

Power and performance programming



Acknowledgments

- Manticore team
 - Behnam Robatmili, Pablo Montesinos Ortego, Michael Weber, Dario Suarez-Gracia, Jimi Xenidis, Han Zhao, Weiwei Chen, Kishore Puskuri, Freark Van Der Berg, Ravi Hastantram
- Interns
 - Madhukar Kedlaya, Christian Delozier, Christoph Kershbaumer, Adrian Sampson, Andrey Ermolinski
- Former members
 - Mehrdad Reshadi, Seth Fowler, Alex Shye, Wayne Piekarski, Vrajesh Bhavsar, Babak Salamat
- Qualcomm Research Silicon Valley



Legal Disclaimers

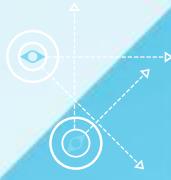
All opinions expressed in this presentation are mine and may not necessarily reflect those of my employer.

Qualcomm and Snapdragon are trademarks of Qualcomm Incorporated, registered in the United States and other countries. All Qualcomm Incorporated trademarks are used with permission.

Other products and brand names may be trademarks or registered trademarks of their respective owners.

A large, faint watermark of the Qualcomm logo is visible across the background of the slide.

cascaval@qti.qualcomm.com



Qualcomm Multicore Asynchronous Runtime Environment

MARE is a **programming model** and a **runtime system** for heterogeneous mobile programming

Simple

Tasks are a natural way to express parallelism.
Familiar C++ programming.

Productive

Focus on application logic, not on thread management

Efficient

Task dependences allow the MARE runtime to perform more intelligent scheduling decisions

- User level, native C++ library, supported on Android, Linux, Mac OS X, and Windows
- Optimized for the Qualcomm Snapdragon platform
- Available at: <http://developer.qualcomm.com/mare>