An Analysis of the Dynamic Behavior of JavaScript Programs

~ GREGOR RICHARDS, SYLVAIN LEBRESNE, BRIAN BURG
AND JAN VITEK

Who, where and when?

* Richards is currently an Assistant Professor at the University of Waterloo

  Research interests: Dynamically and gradually-typed languages, VM design

* Lebresne is still a Post-doc at Purdue University

  Ph. D in 2008 from Paris University.

  Research interest: Type Exception from a theoretical view point.

* Brian Burg is at Applie’s WebKit team.

  Ph. D from University of Washington.

  Research interests: Debugging tools

* Jan Vitek is a Professor at Northeastern. Co-founder of the Secure Software Systems (S3) Lab.

What is this paper about?

.... But why?

- Because researchers have not properly analyzed large JavaScript programs in the wild.
- Because JavaScript is too flexible and too many dynamic features to be completely studied.
- Paper aims to prove or disprove the many common assumptions of JavaScript programs.
- Inspired by Dufour et al.'s work on run-time metrics for Java.

JavaScript: Craziness and then some!

- Uses a prototype based inheritance system rather than a simple Object-oriented inheritance systems.
- An objects is just simply a collection of properties.
- Even a method is just a ‘property’!!!
- Any function can be a constructor
- The most infamous of the JavaScript quirks: *eval* and variadicity

eval == evil???

- *eval* is a string representation of JavaScript expression, statement or a sequence of statements.
- It is a property of JavaScript's global object.

```
eval(new String("2 + 2")); // returns a String object containing "2 + 2"
eval("2 + 2");             // returns 4
```

- *eval* is commonly used to construct JSON objects from strings

SOURCE: Mozilla Developer Network

Variadicity … yes that's a thing!

- Functions need not be called with the same number of arguments or type as it signature.
- Different from Object-oriented polymorphism.
- Functions may be variadic without being declared so
- They can have any degree of variadicity
- Many built-in functions are variadic.
- A function can even be called from any context using the *call* method.

JavaScript: Assumptions

The common assumptions about JavaScript programs in research and in implementations:

- Object protocol dynamism
- Properties are rarely deleted
- *eval* is infrequently used
- Low Call-site dynamism

More assumptions

- Invariance of prototype hierarchy
- Declared function signatures are indicative of types
- Program sizes are modest
- Running time is dominated by ‘hot spots’

Quis custodiet ipsos custodes?

* What if the JavaScript benchmarks themselves are not representative of its usage in the real-world?

* What if the kind of operations they perform on JavaScript programs is not really the usual kind of computations that is run on JavaScript?

* What if all that we know about JavaScript is one big, fat lie???

Dealing with the Devil

* Evaluation done using an instrumented version of theWebKit engine

* Records a trace of all the operations by the interpreter

* Even *eval* is traceable.

* These traces are collected and stored in a database from where it is later analyzed/mined.

* The offline ‘replays’ the state to replicate the heap state

* Static analyses are performed on the recovered files.

Measuring the sizes

Site	Source size	Unique source size	Trace size	Func. count	Hot	Live
280S	116 KB	81KB	11,931 K	4,293	6.8%	44%
BING	815 KB	186KB	1,199 K	2,457	6.4%	46%
BLOG	1,347 KB	775KB	91 K	5,087	11.5%	16%
DIGG	1,106 KB	759KB	1,734 K	2,957	8.7%	39%
EBAY	3,156 KB	1,034KB	2,239 K	10,791	11.7%	31%
FBOK	14,904 KB	1,604KB	5,309 K	43,469	5.8%	19%
FLKR	8,862 KB	246KB	490 K	19,149	14.0%	13%
GMAP	1,736 KB	833KB	13,125 K	5,146	7.8%	61%
GMIL	2,084 KB	1,719KB	6,047 K	10,761	7.6%	38%
GOGL	2,376 KB	839KB	1,815 K	10,250	15.0%	28%
ISHK	915 KB	420KB	5,376 K	2,862	0.6%	35%
LIVE	1,081 KB	938KB	48,324 K	2,936	7.4%	49%
MECM	4,615 KB	646KB	14,084 K	14,401	6.6%	24%
TWIT	837 KB	160KB	2,252 K	2,967	9.2%	45%
WIKI	1,009 KB	115KB	53 K	1,226	14.6%	24%
WORD	1,386 KB	235KB	6,403 K	3,118	1.0%	42%
YTUB	2,897 KB	562KB	541 K	11,321	13.0%	22%
ALL	2,544 KB	790KB	4,151 K	10,625	2.2%	26%

Figure 2. Program sizes. “Source size” is the total amount of source seen by the interpreter, including source loaded more than once and `eval`s. “Unique source size” excludes multiple loads of the same source, but still includes `eval`.

Function sizes

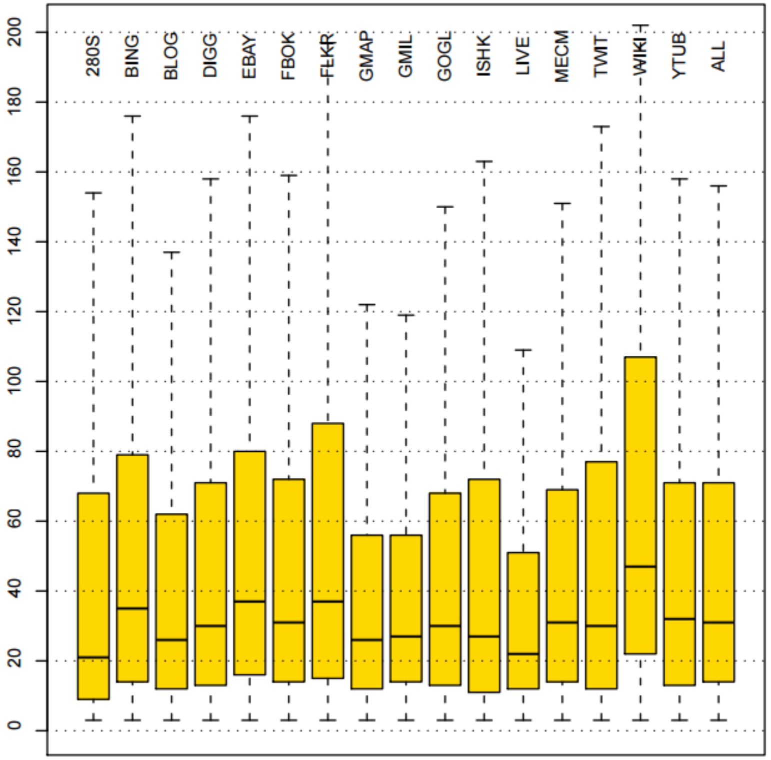

Figure 3. Static function size. The per-site quartiles and median static function size, measured by the number of AST nodes generated from parsing the function.

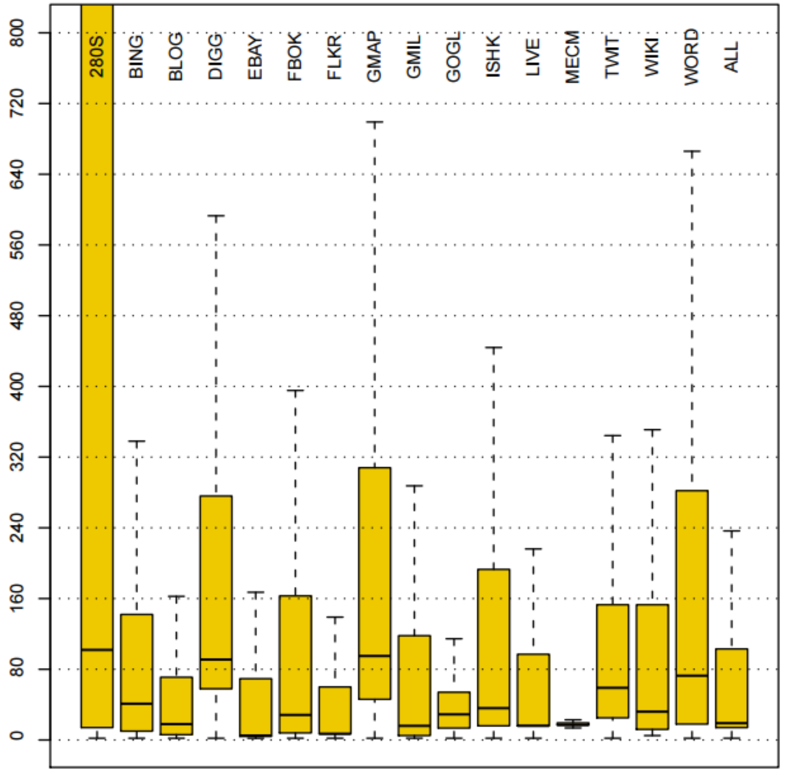

Figure 4. Dynamic function size. The per-site quartiles and median function size, measured in the number of trace events.

Instruction Mix

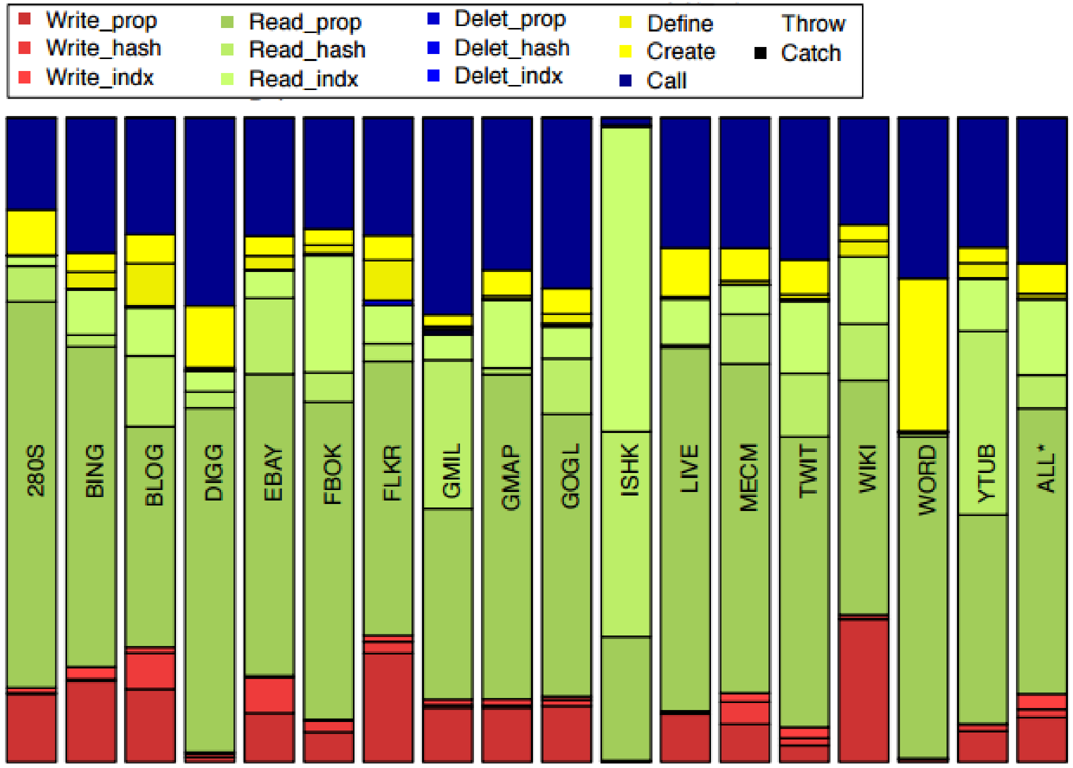

Figure 5. Instruction mix. The per-site proportion of read, write, delete, call instructions (averaged over multiple traces).

Object Kinds

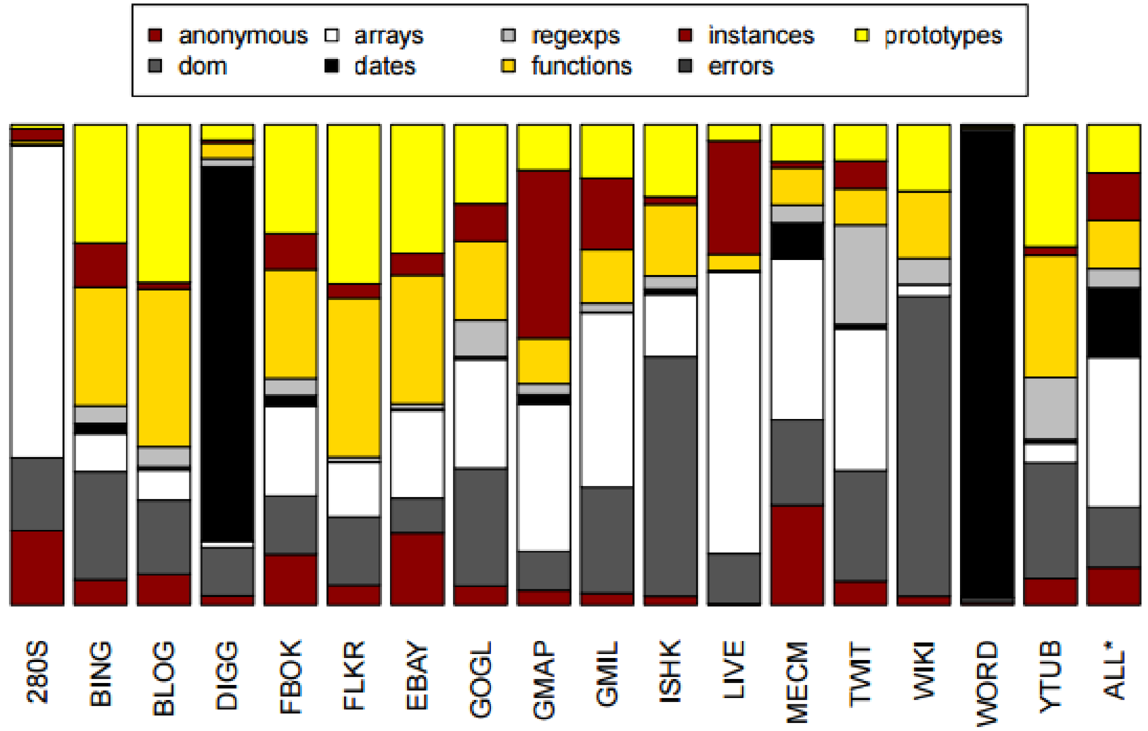

Figure 7. Kinds of allocated objects. The per-site proportion of runtime object kinds (averaged over multiple traces).

Call-site polymorphism

	Callsites with N function bodies					
Site	**1**	**2**	**3**	**4**	**>5**	**Max**
280S	99.9%	0.0%	0.0%	0.0%	0.0%	1,437
BING	93.6%	4.8%	1.0%	0.3%	0.3%	274
BLOG	95.4%	3.4%	0.5%	0.2%	0.5%	95
DIGG	95.4%	3.2%	0.4%	0.3%	0.7%	44
EBAY	91.5%	7.1%	0.5%	0.5%	0.5%	143
FBOK	76.3%	14.8%	3.7%	1.7%	3.5%	982
FLKR	81.9%	13.2%	3.6%	0.5%	0.8%	244
GMAP	98.2%	0.8%	0.4%	0.2%	0.4%	345
GMIL	98.4%	1.2%	0.2%	0.1%	0.2%	800
GOGL	93.1%	5.5%	0.6%	0.3%	0.6%	1,042
ISHK	90.2%	8.1%	1.0%	0.0%	0.8%	42
LIVE	97.0%	1.7%	0.5%	0.3%	0.5%	115
MECM	94.2%	4.1%	1.2%	0.2%	0.4%	106
TWIT	89.5%	7.2%	1.7%	0.3%	1.3%	60
WIKI	87.9%	6.7%	1.9%	0.2%	3.2%	32
WORD	86.8%	7.9%	2.7%	1.9%	0.6%	106
YTUB	83.6%	10.6%	5.4%	0.1%	0.4%	183
All	81.2%	12.1%	3.0%	1.2%	2.5%	1,437

Figure 9. Call site polymorphism. Number of different function bodies invoked from a particular callsite (averaged over multiple traces).

Variadicity

	Functions with N distinct arities					
Site	1	2	3	4	>5	Max
280S	99.3%	0.6%	0.0%	0.1%	0.1%	9
BING	94.2%	4.9%	0.7%	0.2%	0.0%	4
BLOG	97.1%	2.3%	0.4%	0.2%	0.0%	4
DIGG	92.5%	6.3%	0.9%	0.3%	0.1%	5
EBAY	95.9%	3.6%	0.3%	0.0%	0.3%	9
FBOK	93.9%	4.8%	0.6%	0.6%	0.1%	6
FLKR	94.2%	4.6%	0.9%	0.3%	0.0%	4
GMAP	93.4%	5.5%	0.6%	0.3%	0.2%	6
GMIL	95.3%	3.8%	0.6%	0.2%	0.2%	30
GOGL	94.6%	4.3%	0.7%	0.2%	0.2%	9
ISHK	97.6%	2.3%	0.1%	0.0%	0.0%	3
LIVE	92.7%	6.1%	0.8%	0.3%	0.1%	7
MECM	91.9%	6.5%	0.6%	0.5%	0.5%	7
TWIT	90.9%	7.4%	1.3%	0.5%	0.0%	4
WIKI	96.7%	3.3%	0.0%	0.0%	0.0%	2
WORD	92.6%	6.6%	0.6%	0.2%	0.0%	4
YTUB	98.5%	1.4%	0.1%	0.0%	0.0%	4
All	93.5%	4.8%	0.7%	0.4%	0.6%	30

Figure 10. Function variadicity. Proportion of functions used variadically.

The nature of evil

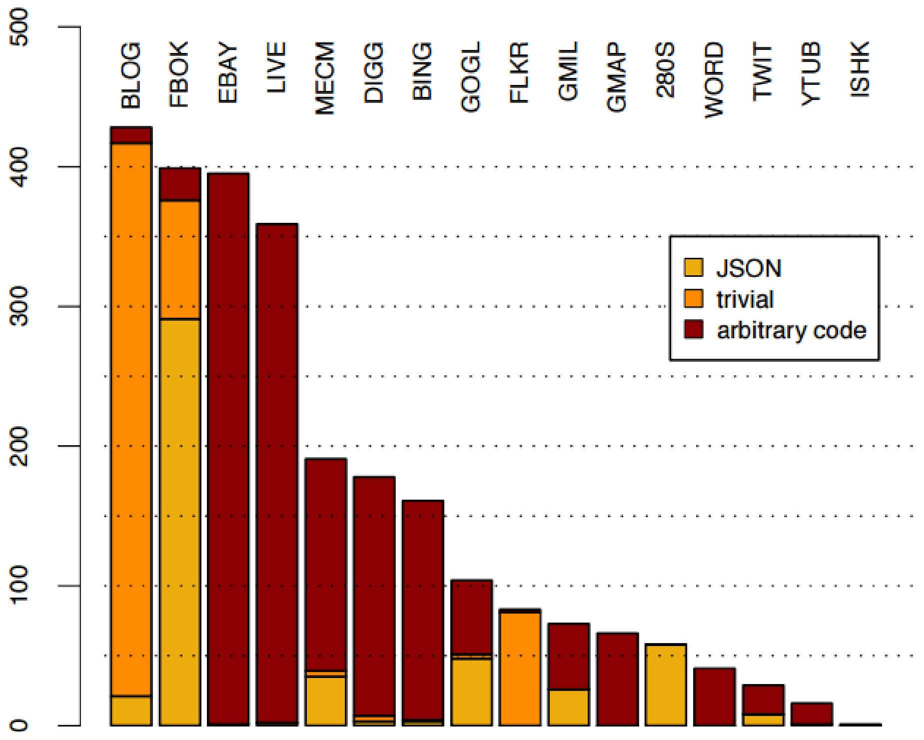

Figure 11. Uses of `eval`. Count of the invocations of `eval` (averaged over multiple traces). Sites sorted by total number of invocations, descending.

Object Protocol Dynamism

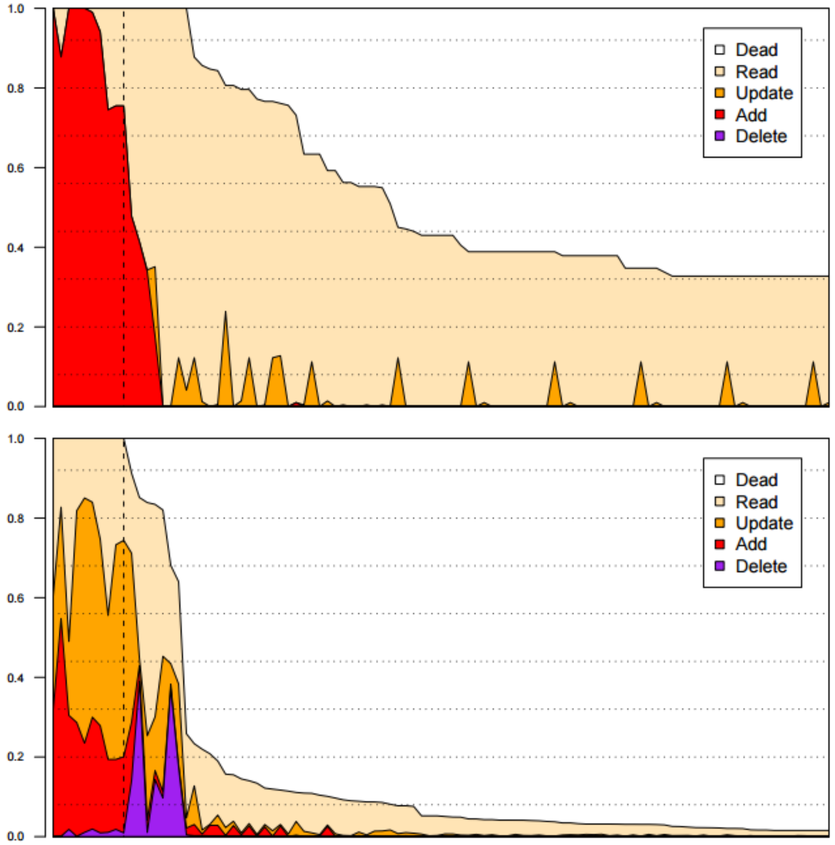

Figure 13. Object timelines. Above, TWIT. Below, GOGL. The dashed line indicates the end of object construction.

Constructor Polymorphism

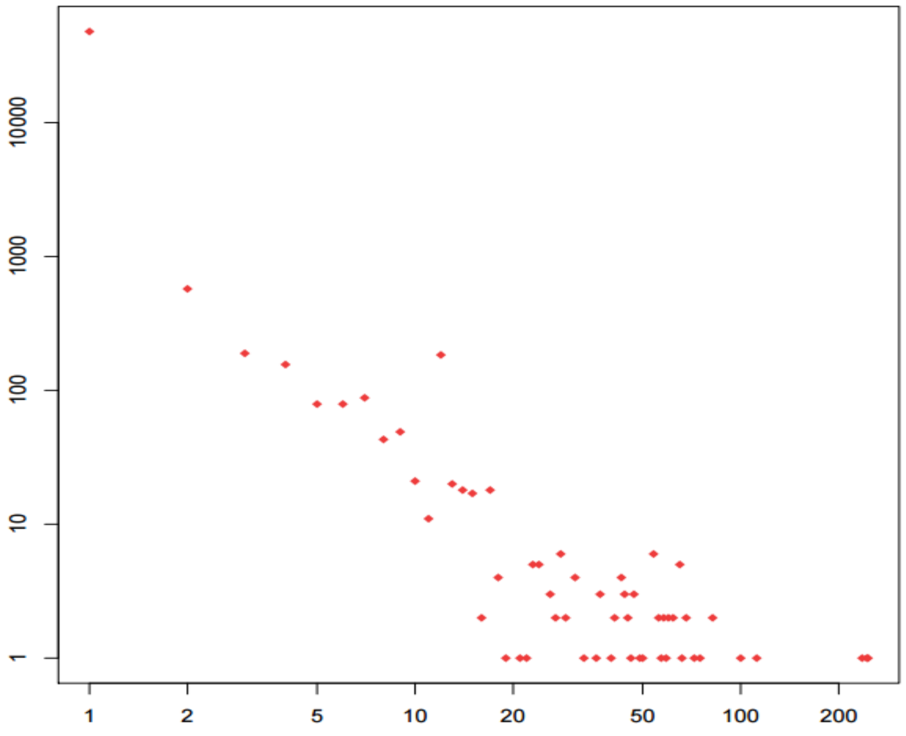

Figure 15. Constructor polymorphism. Plots the number of distinct sets of properties (x-axis) against the number of constructor functions observed to create objects with that many distinct sets of properties (y-axis). (Log scale)

Benchmarks: Allocated Objects

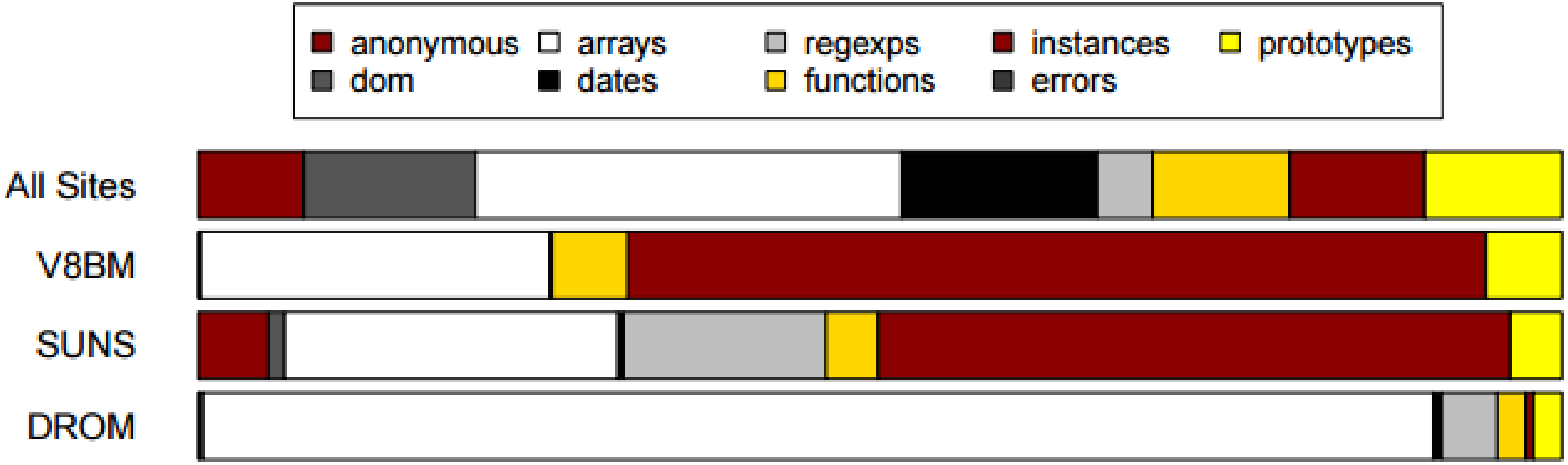

Figure 17. Kinds of allocated objects.

Benchmarks: Object timelines

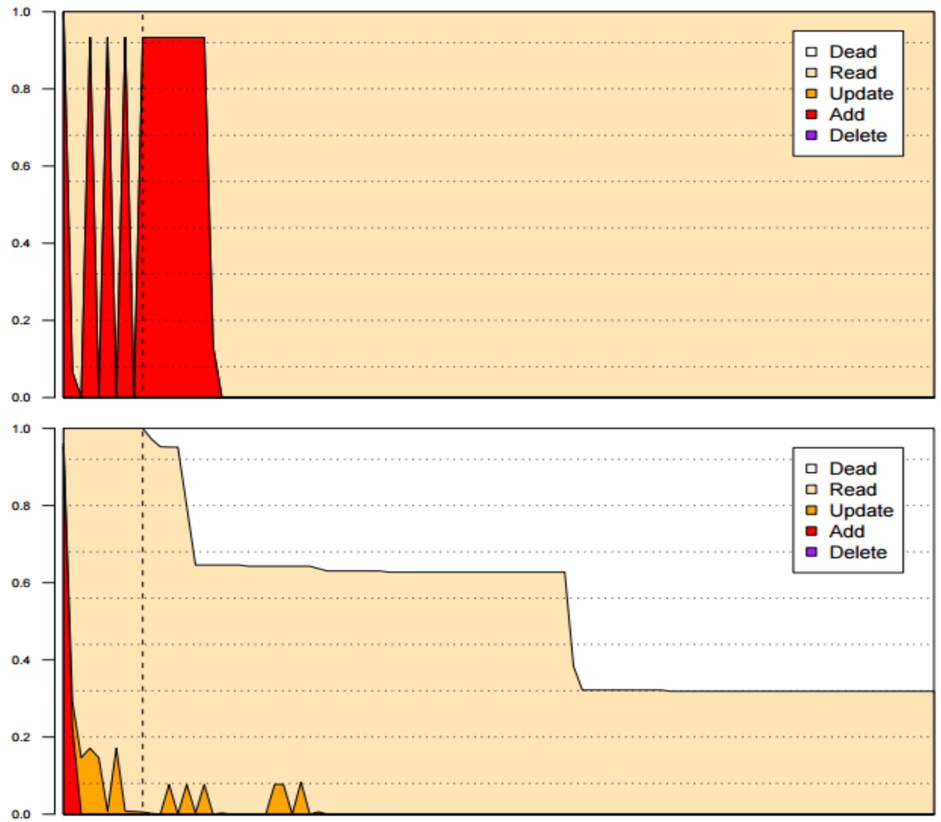

Figure 18. Object timelines. SUNS (above) and V8BM (below). The dashed line indicates the end of object construction.

… and the results are:

The authors conclude that most of the assumptions about production JavaScript programs are:

FALSE

In particular:

- Properties are added only at object initialization: BUSTED
- Properties are rarely deleted: BUSTED (but I think is PLAUSIBLE)
- Use of *eval* is infrequent: BUSTED
- Program sizes are modest: BUSTED

Results (continued)

- Prototype hierarchy is invariant: BUSTED (but I think is PLAUSIBLE)
- Call-site dynamism is low: BUSTED (but I think is PLAUSIBLE)
- Declared function signatures are indicative of types: BUSTED
- Execution time is dominated by hot loops: **CONFIRMED**

The Violators

	1	2	3	4[9]	5	6	7	8[10]
280S	X							
BING				X	X			
BLOG		X				X		X
DIGG	X	X		X	X	X		
EBAY				X		X		X
FBOK	X	X	X		X	X	X	
FLKR					X		X	X
GMAP	X			X	X	X		
GMIL	X	X	X	X		X		
GOGL	X	X	X	X	X	X		X
ISHK	X	X					X	
LIVE	X	X		X	X	X		
MECM	X	X		X	X	X		
TWIT	X				X		X	
WIKI							X	X
WORD	X			X	X		X	
YTUB		X	X			X	X	X

Figure 19. Violations. For each assumption (above), a subjective opinion of which sites (left) violate that assumption.


Questions?