In Praise of Metacircular Virtual Machine Layering

Erick Lavoie, Bruno Dufour, Marc Feeley
Université de Montréal
ericklavoie.com

Motivation

Context

- Program comprehension
- Benchmark generation
- Security invariant monitoring
- Hybrid analysis design

Run-Time Instrumentation

- Object model operations
- Function calls
- Scope chain
- Control-flow
- ...

Direct Instrumentation of Virtual Machine

+ Guaranteed compliance
+ Integrated in a browser
+ Straightforward on a simple interpreter
- Needs maintenance
- Tied to a single browser
- Becoming more complex

Research Problem

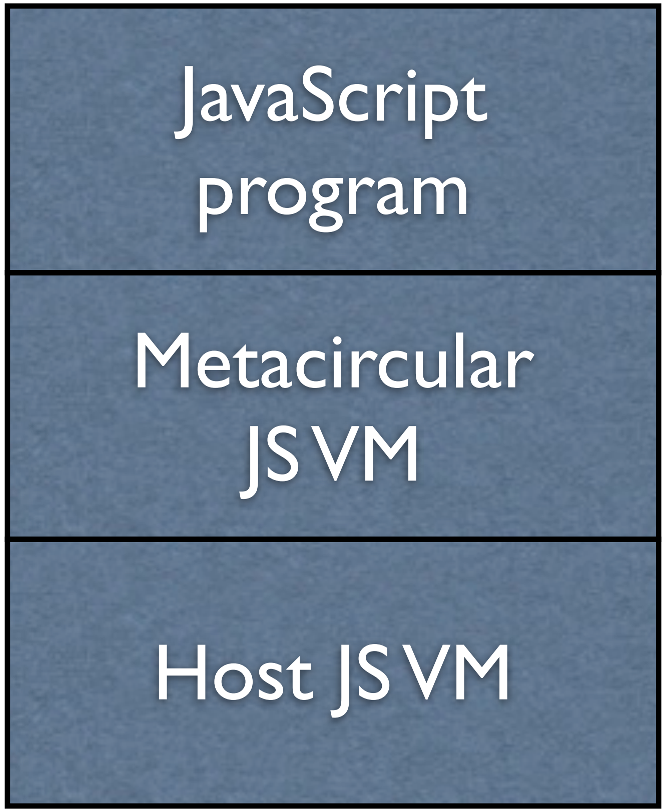
Research Problem

How can we reify opaque aspects of JavaScript for high-level run-time instrumentation at a minimal cost in performance?

Approach

Metacircular Virtual Machine Layering

- \+ Differential implementation
- \+ Independent from Host VM implementation
- \- Run-time cost



Application

Program Comprehension

- What is the contribution of application data structures to its overall memory footprint?
- Where are these objects created with regard to the source code?

Run-Time Instrumentation

- Object model operations
- Function calls
- Scope chain
- Control-flow
- ...

Reifying Object Model Operations

`var o = {};`	`var o = send(root.object,“__new__”);`
`o.p = 42;`	`send(o, “__set__”, “p”, 42);`
`o.p;`	`send(o, “__get__”, “p”);`
`delete o.p;`	`send(o, “__delete__”, “p”);`
`new F();`	`send(F, “__ctor__”);`

Counting the Number of Property Accesses

```
var getCounter = 0;

send(root.object, “__set__”, “__get__”,
    function (name) {
      getCounter++;
      return this.get(name);
    }
);
```

Reifying Function Calls

```
function (g) {g();};
```

```
function (g) {
  send(g, “call”);
};
```

```
h.call();
```

```
send(h, “call”);
```

```
f();
```

```
send(global, “f”);
```

```
o.p();
```

```
send(o, “p”);
```

Semantics of Message Sending

```
function send(obj, msg, ..args) {
    var method = obj.get(msg);
    return method.call(obj, ..args);
}
```

Semantics of Message Sending

```
function send(obj, msg, ..args) {
    var method = obj.get(msg);
    var callFn = method.get("call");
    return callFn.call(method, obj, ..args);
}
```

Intercepting Function Calls

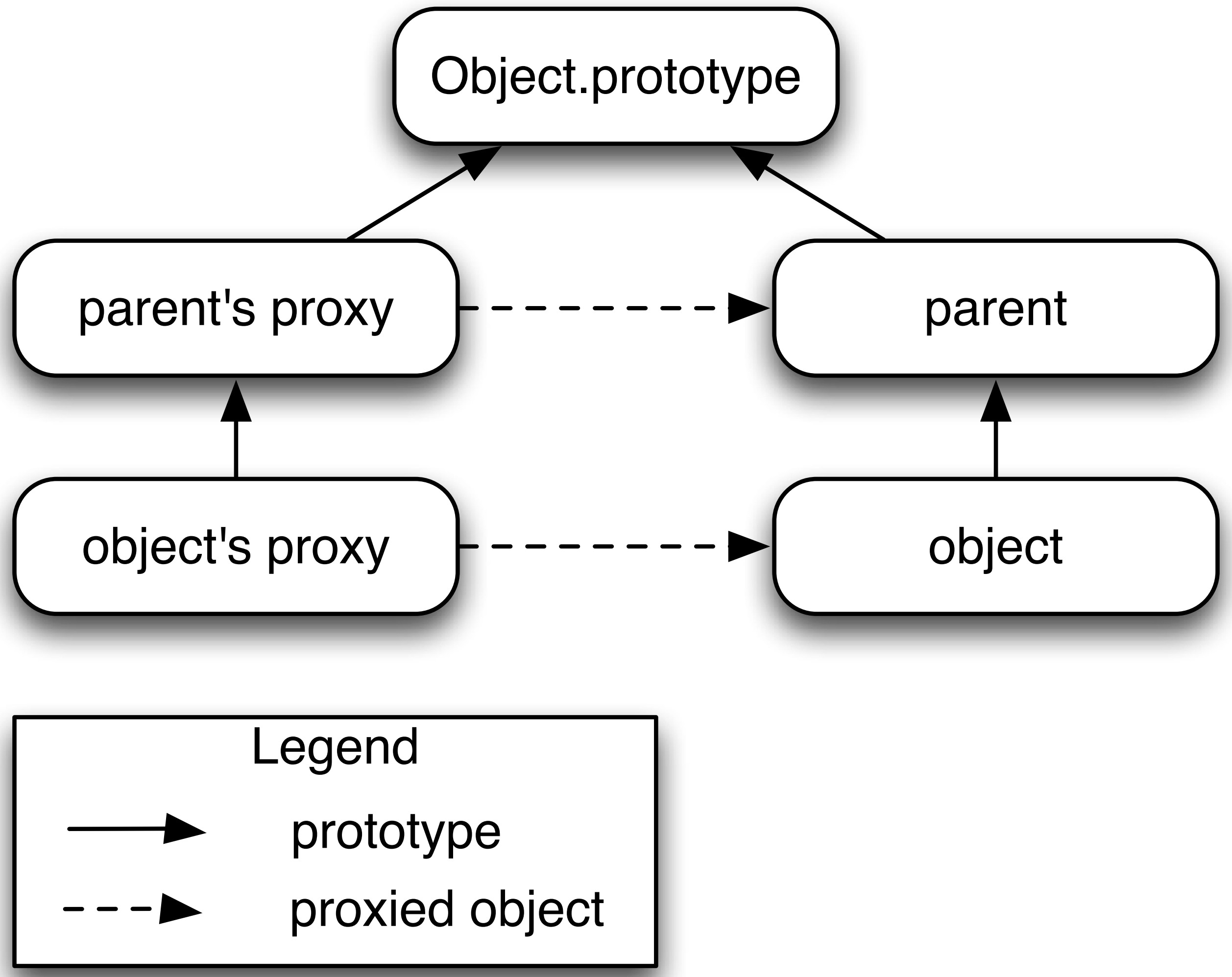
```
function beforeCall() {...}

send(root.function, “__set__”, “call”,
    function (obj, ..args) {
      beforeCall(this, obj, ..args);
      return this.call(obj, ..args);
    }
 );
```

Optimization

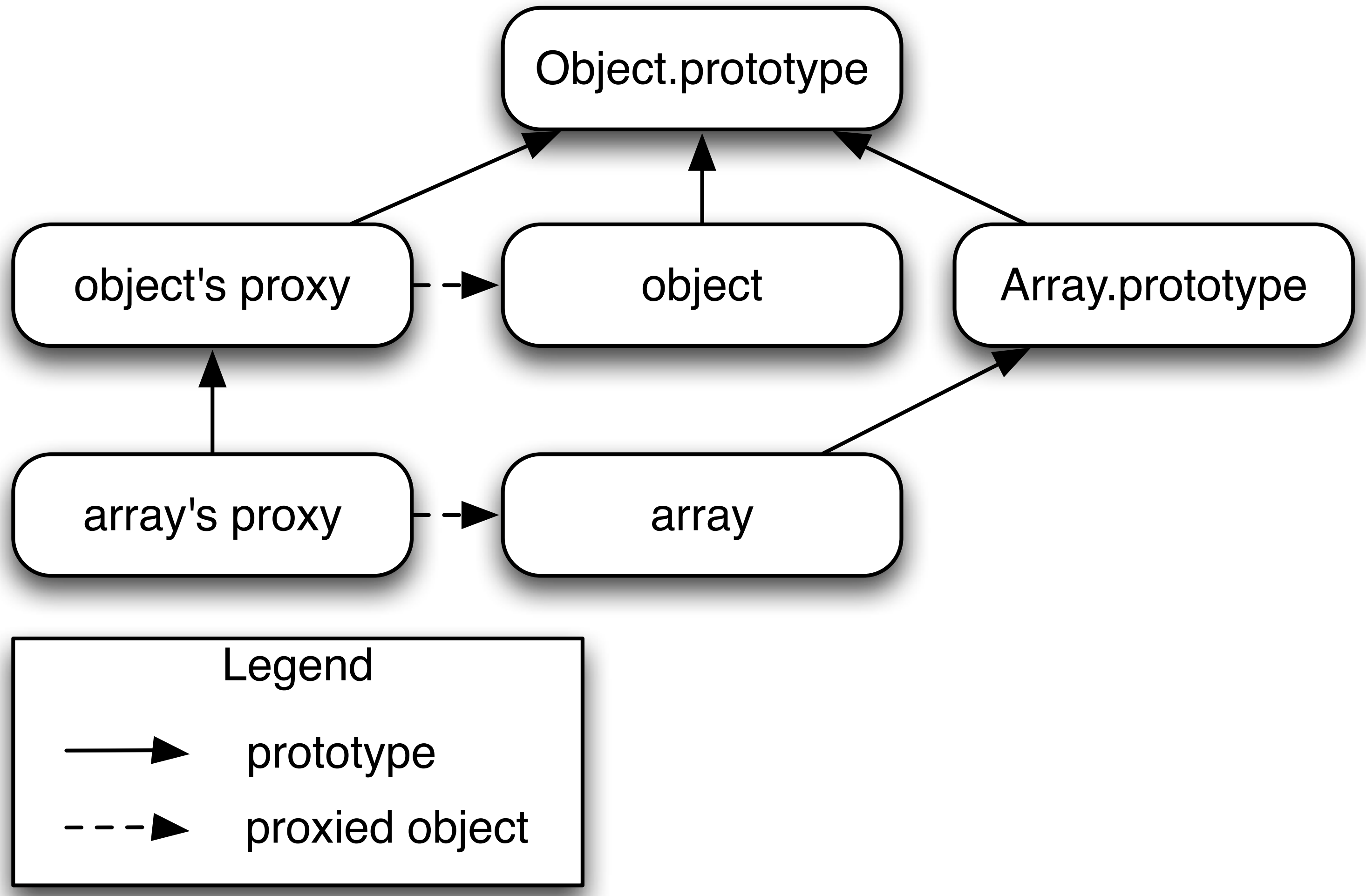
Object Representation

- Encapsulate invariants of the implementation
- Provide fast object creation, accesses and updates
- Allow transparent per-object information collection

Basic Object Representation



Special Object Representation



Object Representation Operations

```
var o = {};          root.object.create();

o.p = 42;            o.set(“p”, 42);

o.p;                 o.get(“p”);

delete o.p;          o.delete(“p”);
```

Call-Site Specialization of Arguments Nb

```
function f(g) {
  return g();
}
```

```
new FunctionProxy(function f(g) {
  return (g instanceof FunctionProxy) ?
    g.proxiedObject.call($global) :
    error(g);
});
```

Call-Site Specialization of Arguments Nb

```
function f(g) {
  return g();
}
```

```
new FunctionProxy(function f($this,g) {
  return (g instanceof FunctionProxy) ?
    g.proxiedObject($global) :
    error(g);
});
```

Call-Site Specialization of Arguments Nb

```
function f(g) {
  return g();
}
```

```
new FunctionProxy(function f($this,g) {
  return g.call($global);
});
```

```
FunctionProxy.prototype.call =
  function (obj) {
    return this.proxiedObject.apply(obj,
      Array.prototype.slice.call(arguments, 1));
  };
```

Call-Site Specialization of Arguments Nb

```
function f(g) {
  return g();
}
```

```
new FunctionProxy(function f($this,g) {
  return g.callWith0Arg($global);
});
```

```
FunctionProxy.prototype.callWith0Arg = function (obj) {
  return this.proxiedObject(obj);
};
```

```
function invFn() { throw Error(“Invalid Function”); };
```

```
ObjectProxy.prototype.callWith0Arg = invFn;
```

```
String.prototype.callWith0Arg = invFn;
```

```
...
```

Counting the Number of Property Accesses (revisited)

```
var getCounter = 0;

send(root.object, "__set__", "__get__",
    new FunctionProxy(
      function ($this, name) {
        getCounter++;
        return $this.get(name);
      }
    )
);
```

Empirical Evaluation

Baseline Performance

Benchmark	Pn	SM	V8	V8/Pn	SM/Pn
Crypto	529.0	348.0	17025.0	32.2	0.7
DeltaBlue	82.8	249.0	19306.0	233.2	3.0
EarleyBoyer	738.0	808.0	34170.0	46.3	1.1
NavierStokes	908.0	564.0	20947.0	23.1	0.6
RayTrace	156.0	560.0	19442.0	124.6	3.6
RegExp	441.0	781.0	3902.0	8.8	1.8
Richards	120.0	219.0	14149.0	117.9	1.8
Splay	118.0	1508.0	5850.0	49.6	12.8
V8 Score	270.0	524.0	14002.0	51.9	1.9

Baseline Memory Usage

Benchmark	Pn	V8	Pn/V8
Crypto	56.0	20.0	2.8
DeltaBlue	33.0	20.0	1.6
EarleyBoyer	128.0	20.0	6.4
NavierStokes	29.0	19.0	1.5
RayTrace	35.0	20.0	1.8
RegExp	54.0	22.0	2.5
Richards	28.0	18.0	1.6
Splay	84.0	97.0	0.9

Instrumented Performance

Benchmark	Pn	Pn-spl	Pn-fast	Pn/Pn-spl	Pn/Pn-fast
Crypto	529.0	41.4	566.0	12.8	0.9
DeltaBlue	82.8	36.2	103.0	2.3	0.8
EarleyBoyer	738.0	162.0	767.0	4.6	1.0
NavierStokes	908.0	51.4	871.0	17.7	1.0
RayTrace	156.0	85.1	158.0	1.8	1.0
RegExp	441.0	324.0	476.0	1.4	0.9
Richards	120.0	30.5	113.0	3.9	1.1
Splay	118.0	453.0	117.0	0.3	1.0
V8 Score	270.0	91.2	281.0	3.0	1.0

Conclusion

Conclusion

- Metacircular VM layering can be used to reify object model operations and function calls at a performance level similar to a state-of-the-art interpreter
- Simple implementation (~1700 LOC runtime library excluding JS-to-JS translator)

Future Work

- Support DOM to integrate with a browser
- Apply approach to reify scope chain and control-flow for other applications
- Improve performance by exploiting dynamic recompilation to remove redundant checks
- Develop metacompilers to generate custom VMs for specific instrumentation tasks