Bootstrapping a Self-Hosted Research Virtual Machine for JavaScript

An Experience Report

Maxime Chevalier-Boisvert

{chevalma,lavoeric,feeley,dufour}@iro.umontreal.ca

Université de Montréal

DLS, October 24, 2011

Motivation

- Dynamic languages rapidly increasing in popularity
  - Dramatic rise in the last two decades
  - JavaScript, pushed as the language of the web
- Currently available JS VMs highly complex
  - Large (V8 375 KLOC, SpiderMonkey 550 KLOC)
  - Complex, legacy constraints
  - Difficult to modify, maintain
- Need for a flexible research VM
  - Allows exploring implementation alternatives easily
  - Customizable frontend, IR, backend, runtime system
- Tachyon: self-hosted VM with JIT compiler for JS
  - Currently 75 KLOC, highly commented

Self-hosting

- Tachyon is a JS compiler, itself written in JS
- Tachyon can already compile itself
- Many advantages from self-hosting
  - Higher-level implementation language than C/C++
  - Less code duplication. Same runtime for VM, hosted programs
  - No need for compatibility layer between VM, hosted programs
  - Possibility for VM to optimize itself
- Some issues
  - JS needs to be extended for JIT compiler writing
  - Possible conflictual self-interactions

Why JavaScript?

- Dynamic languages are an interesting research topic
  - Difficult to analyze
    - Dynamic typing, eval, etc.
  - Difficult to compile efficiently
    - Performance gap vs static languages
- JavaScript is:
  - Very popular
    - The language of the web
  - Of manageable complexity
    - ECMAScript 5 (ES5) spec is fairly small
  - Representative of dynamic languages
    - ...And their associated complexities

What is JavaScript?

- Dynamic language
  - Dynamic typing, no type annotations
  - `eval` function
- Basic types include:
  - Doubles (no int!), strings, booleans, objects, arrays, first-class functions, null, undefined
- Objects as hash maps
  - Can add/remove properties at any time
  - Prototype-based, no classes
- Functional component

JavaScript Example

```
function Num(x)
{
    this.val = x;

    if (x !== 0)
        this.div = function() { return this.val / x; };
}

Num.prototype.toString = new Function("return 'NUM';");

var a = new Num(0);
var b = new Num(2);

b.val = 6;

print( a + b.div() ); // prints NUM3
```

JavaScript Example

```
function Num(x)    <- constructor function
{
    this.val = x;

    if (x !== 0)
        this.div = function() { return this.val / x; };
}

Num.prototype.toString = new Function("return 'NUM';");

var a = new Num(0);    <- objects created using "new"
var b = new Num(2);    <-

b.val = 6;

print( a + b.div() ); // prints NUM3
```

JavaScript Example

```
function Num(x)
{
    this.val = x;

    if (x !== 0)
        this.div = function() { return this.val / x; };
}

Num.prototype.toString = new Function("return 'NUM';");

var a = new Num(0);
var b = new Num(2);

b.val = 6;

print( a + b.div() ); // prints NUM3
```

the object will have the "div" method only if x is not 0

only b has the "div" method

JavaScript Example

```
function Num(x)
{
    this.val = x;

    if (x !== 0)
        this.div = function() { return this.val / x; };
}
```

Num objects inherit "toString" from their prototype

```
Num.prototype.toString = new Function("return 'NUM';");

var a = new Num(0);
var b = new Num(2);

b.val = 6;
```

a.toString is called here

```
print( a + b.div() ); // prints NUM3
```

JavaScript Example

```
function Num(x)
{
    this.val = x;

    if (x !== 0)
        this.div = function() { return this.val / x; };
}

Num.prototype.toString = new Function("return 'NUM';");

var a = new Num(0);
var b = new Num(2);

b.val = 6;

print( a + b.div() ); // prints NUM3
```

code generated dynamically from a string

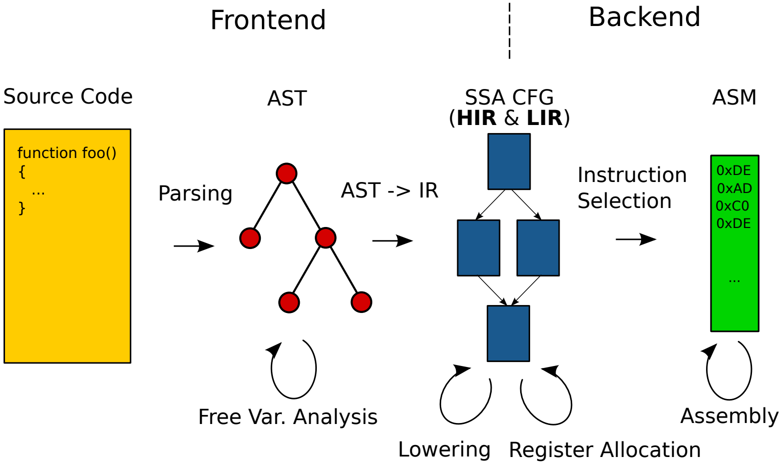
Related work

- Meta-circular VMs
  - Squeak: a Smalltalk VM (OOPSLA, 1997)
  - Jalapeño a.k.a. JikesRVM: Java in Java (OOPSLA, 1999)
  - Klein: SELF in SELF (OOPSLA, 2005)
  - PyPy: the meta-VM (ICOOOLPS, 2009)
  - Cog: extends the Smalltalk VM with a JIT (VMIL, 2011)
- Modern JS implementations
  - Firefox: SpiderMonkey (PLDI, 2009)
  - WebKit: JavaScriptCore (since 2002)
  - Chrome: V8 (since 2008)
  - Internet Explorer: Chakra (since 2009)

Contributions

- Presentation of the design of our compiler
- Design of low-level extensions to JS for JIT compiler writing, compatible with the existing syntax
- An execution model for the VM
- Description of the bootstrap process required to compile & initialize the Tachyon VM
- Experience in writing a large system in JS

Design Overview



Simple Example

```
function add1(n)
{
    return n + 1;
}
```

Multiple Semantics

```
function add1(n)
{
    return n + 1;
}
```

```
add1(2)          ⟼ 3

add1('hello')    ⟼ 'hello1'

add1(true)       ⟼ 2
add1(null)       ⟼ 1
add1(undefined)  ⟼ NaN

add1({ toString: function() { return '3'; } }) ⟼ '31'

add1({ toString: function() { return 3; } })   ⟼ 4
```

High-Level IR

```
entry:
box n = arg 2;
box $t_4 = call <fn "add">, undef, undef, n, box:1;
ret $t_4;
```

High-Level IR

Control-Flow Graph (CFG)

one basic block (function entry point)

```
entry:
box n = arg 2;
box $t_4 = call <fn "add">, undef, undef, n, box:1;
ret $t_4;
```

High-Level IR

Static-Single Assignment (SSA)

```
entry:
box n = arg 2;
box $t_4 = call <fn "add">, undef, undef, n, box:1;
ret $t_4;
```

all temps have dynamic "box" type

High-Level IR

Call to "add" primitive, implements "+" operator

```
entry:
box n = arg 2;
box $t_4 = call <fn "add">, undef, undef, n, box:1;
ret $t_4;
```

High-Level IR

Calls have hidden arguments

```
entry:
box n = arg 2;
box $t_4 = call <fn "add">, undef, undef, n, box:1;
ret $t_4;
```

closure pointer (first undef) — this pointer (second undef)

IR Lowering

- Transformation of HIR into LIR
- Multiple passes
  - Inlining of primitive functions
  - Sparse Conditional Constant Propagation (SCCP)
    - Constant propagation
    - Dead code elimination
    - Algebraic simplifications
  - Global Value Numbering (GVN)
  - Optimization patterns
    - Control-flow graph simplifications
    - Strength reduction
    - Redundant phi elimination
    - Dead code elimination
  - Simplistic purity/side-effect analysis

Low-Level IR

```
entry:
box n = arg 2;
pint $t_4 = and_box_pint n, pint:3;
if $t_4 === pint:0 then cmp_true else if_false;

cmp_true:
box $t_14 = add_ovf n, box:1 normal call_res overflow ovf;

if_false:
ref $t_17 = get_ctx;
box global_3 = load_box $t_17, pint:36;
box $t_19 = call <fn "addGeneral">, undef, global_3, n, box:1;
jump call_res;

ovf:
ref $t_9 = get_ctx;
box global_2 = load_box $t_9, pint:36;
box $t_11 = call <fn "addOverflow">, undef, global_2, n, box:1;
jump call_res;

call_res:
box phires = phi [$t_14 cmp_true], [$t_19 if_false], [$t_11 ovf];
ret phires;
```

Low-Level IR

```
entry:
box n = arg 2;
pint $t_4 = and_box_pint n, pint:3;
if $t_4 === pint:0 then cmp_true else if_false;

cmp_true:
box $t_14 = add_ovf n, box:1 normal call_res overflow ovf;

if_false:
ref $t_17 = get_ctx;
box global_3 = load_box $t_17, pint:36;
box $t_19 = call <fn "addGeneral">, undef, global_3, n, box:1;
jump call_res;

ovf:
ref $t_9 = get_ctx;
box global_2 = load_box $t_9, pint:36;
box $t_11 = call <fn "addOverflow">, undef, global_2, n, box:1;
jump call_res;

call_res:
box phires = phi [$t_14 cmp_true], [$t_19 if_false], [$t_11 ovf];
ret phires;
```

Low-Level IR

```
entry:
box n = arg 2;
pint $t_4 = and_box_pint n, pint:3;
if $t_4 === pint:0 then cmp_true else if_false;

cmp_true:
box $t_14 = add_ovf n, box:1 normal call_res overflow ovf;

if_false:
ref $t_17 = get_ctx;
box global_3 = load_box $t_17, pint:36;
box $t_19 = call <fn "addGeneral">, undef, global_3, n, box:1;
jump call_res;

ovf:
ref $t_9 = get_ctx;
box global_2 = load_box $t_9, pint:36;
box $t_11 = call <fn "addOverflow">, undef, global_2, n, box:1;
jump call_res;

call_res:
box phires = phi [$t_14 cmp_true], [$t_19 if_false], [$t_11 ovf];
ret phires;
```

Low-Level IR

```
entry:
box n = arg 2;
pint $t_4 = and_box_pint n, pint:3;
if $t_4 === pint:0 then cmp_true else if_false;

cmp_true:
box $t_14 = add_ovf n, box:1 normal call_res overflow ovf;

if_false:
ref $t_17 = get_ctx;
box global_3 = load_box $t_17, pint:36;
box $t_19 = call <fn "addGeneral">, undef, global_3, n, box:1;
jump call_res;

ovf:
ref $t_9 = get_ctx;
box global_2 = load_box $t_9, pint:36;
box $t_11 = call <fn "addOverflow">, undef, global_2, n, box:1;
jump call_res;

call_res:
box phires = phi [$t_14 cmp_true], [$t_19 if_false], [$t_11 ovf];
ret phires;
```

Low-Level IR

```
entry:
box n = arg 2;
pint $t_4 = and_box_pint n, pint:3;
if $t_4 === pint:0 then cmp_true else if_false;

cmp_true:
box $t_14 = add_ovf n, box:1 normal call_res overflow ovf;

if_false:
ref $t_17 = get_ctx;
box global_3 = load_box $t_17, pint:36;
box $t_19 = call <fn "addGeneral">, undef, global_3, n, box:1;
jump call_res;

ovf:
ref $t_9 = get_ctx;
box global_2 = load_box $t_9, pint:36;
box $t_11 = call <fn "addOverflow">, undef, global_2, n, box:1;
jump call_res;

call_res:
box phires = phi [$t_14 cmp_true], [$t_19 if_false], [$t_11 ovf];
ret phires;
```

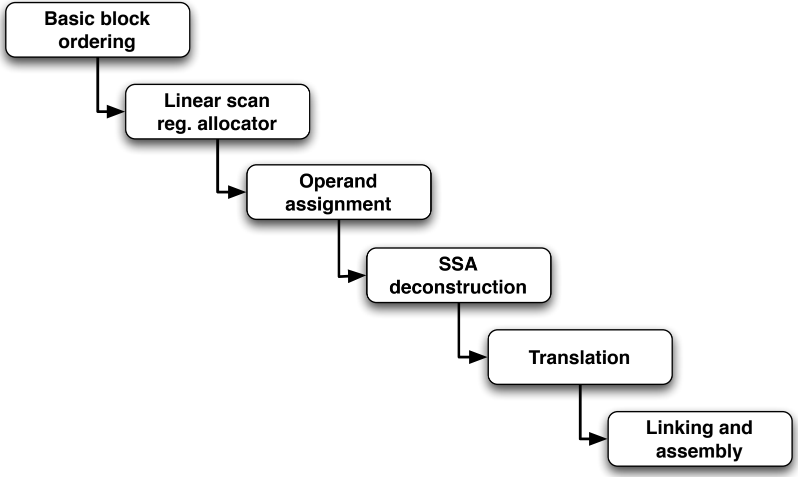
Low-Level IR

```
entry:
box n = arg 2;
pint $t_4 = and_box_pint n, pint:3;
if $t_4 === pint:0 then cmp_true else if_false;

cmp_true:
box $t_14 = add_ovf n, box:1 normal call_res overflow ovf;

if_false:
ref $t_17 = get_ctx;
box global_3 = load_box $t_17, pint:36;
box $t_19 = call <fn "addGeneral">, undef, global_3, n, box:1;
jump call_res;

ovf:
ref $t_9 = get_ctx;
box global_2 = load_box $t_9, pint:36;
box $t_11 = call <fn "addOverflow">, undef, global_2, n, box:1;
jump call_res;

call_res:
box phires = phi [$t_14 cmp_true], [$t_19 if_false], [$t_11 ovf];
ret phires;
```

Low-Level IR

```
entry:
box n = arg 2;
pint $t_4 = and_box_pint n, pint:3;
if $t_4 === pint:0 then cmp_true else if_false;

cmp_true:
box $t_14 = add_ovf n, box:1 normal call_res overflow ovf;

if_false:
ref $t_17 = get_ctx;
box global_3 = load_box $t_17, pint:36;
box $t_19 = call <fn "addGeneral">, undef, global_3, n, box:1;
jump call_res;

ovf:
ref $t_9 = get_ctx;
box global_2 = load_box $t_9, pint:36;
box $t_11 = call <fn "addOverflow">, undef, global_2, n, box:1;
jump call_res;

call_res:
box phires = phi [$t_14 cmp_true], [$t_19 if_false], [$t_11 ovf];
ret phires;
```

Code Generation



x86 Machine Code

```
<fn:add1>
movl 4(%ecx),%edi
subl $3,%edi
testl %edi,%edi
je L7828
cmpl $0,%edi
jg L7829
movl $25,%ebp
movl 4(%ecx),%edi
cmpl $0,%edi
cmovlel %ebp,%edx
cmpl $1,%edi
cmovlel %ebp,%ebx
cmpl $2,%edi
cmovlel %ebp,%eax
jmp L7828
L7829:
movl %eax,12(%ecx)
movl %esp,%ebp
subl $1,%edi
cmpl $0,%edi
jle L7828
L7831:
cmpl %esp,%ebp
jl L7830
movl (%ebp),%eax
movl %eax,(%ebp,%edi,4)
subl $4,%ebp
jmp L7831
L7830:
movl 12(%ecx),%eax
sall $2,%edi
addl %edi,%esp
L7828:

entry:
movl %eax,%ebx
andl $3,%ebx
testl %ebx,%ebx
movl $0,%ebx
cmovzl %esp,%ebx
testl %ebx,%ebx
je if_false
jmp log_and_sec

if_false:
movl %ecx,%ebx
movl 36(%ebx),%ebx
movl <addGeneral_fast>,%edi
movl $25,%edx
movl $4,%esi
movl $4,4(%ecx)
call *%edi
jmp call_res

log_and_sec:
movl %eax,%ebx
addl $4,%ebx
jno ssa_dec
jmp iir_false

ssa_dec:
movl %ebx,%eax
jmp call_res

iir_false:
movl %ecx,%ebx
movl 36(%ebx),%ebx
movl <addOverflow_fast>,%edi
movl $25,%edx
movl $4,%esi
movl $4,4(%ecx)
call *%edi
jmp call_res

call_res:
ret $0
```

x86 Machine Code

```
<fn:add1>
movl 4(%ecx),%edi
subl $3,%edi
testl %edi,%edi
je L7828
cmpl $0,%edi
jg L7829
movl $25,%ebp
movl 4(%ecx),%edi
cmpl $0,%edi
cmovlel %ebp,%edx
cmpl $1,%edi
cmovlel %ebp,%ebx
cmpl $2,%edi
cmovlel %ebp,%eax
jmp L7828
L7829:
movl %eax,12(%ecx)
movl %esp,%ebp
subl $1,%edi
cmpl $0,%edi
jle L7828
L7831:
cmpl %esp,%ebp
jl L7830
movl (%ebp),%eax
movl %eax,(%ebp,%edi,4)
subl $4,%ebp
jmp L7831
L7830:
movl 12(%ecx),%eax
sall $2,%edi
addl %edi,%esp
L7828:

entry:
movl %eax,%ebx
andl $3,%ebx
testl %ebx,%ebx
movl $0,%ebx
cmovzl %esp,%ebx
testl %ebx,%ebx
je if_false
jmp log_and_sec

if_false:
movl %ecx,%ebx
movl 36(%ebx),%ebx
movl <addGeneral_fast>,%edi
movl $25,%edx
movl $4,%esi
movl $4,4(%ecx)
call *%edi
jmp call_res

log_and_sec:
movl %eax,%ebx
addl $4,%ebx
jno ssa_dec
jmp iir_false

ssa_dec:
movl %ebx,%eax
jmp call_res

iir_false:
movl %ecx,%ebx
movl 36(%ebx),%ebx
movl <addOverflow_fast>,%edi
movl $25,%edx
movl $4,%esi
movl $4,4(%ecx)
call *%edi
jmp call_res

call_res:
ret $0
```

x86 Machine Code

```
<fn:add1>
movl 4(%ecx),%edi
subl $3,%edi
testl %edi,%edi
je L7828
cmpl $0,%edi
jg L7829
movl $25,%ebp
movl 4(%ecx),%edi
cmpl $0,%edi
cmovlel %ebp,%edx
cmpl $1,%edi
cmovlel %ebp,%ebx
cmpl $2,%edi
cmovlel %ebp,%eax
jmp L7828
L7829:
movl %eax,12(%ecx)
movl %esp,%ebp
subl $1,%edi
cmpl $0,%edi
jle L7828
L7831:
cmpl %esp,%ebp
jl L7830
```

```
movl (%ebp),%eax
movl %eax,(%ebp,%edi,4)
subl $4,%ebp
jmp L7831
L7830:
movl 12(%ecx),%eax
sall $2,%edi
addl %edi,%esp
L7828:

entry:
movl %eax,%ebx
andl $3,%ebx
testl %ebx,%ebx
movl $0,%ebx
cmovzl %esp,%ebx
testl %ebx,%ebx
je if_false
jmp log_and_sec

if_false:
movl %ecx,%ebx
movl 36(%ebx),%ebx
movl <addGeneral_fast>,%edi
movl $25,%edx
movl $4,%esi
```

```
movl $4,4(%ecx)
call *%edi
jmp call_res

log_and_sec:
movl %eax,%ebx
addl $4,%ebx
jno ssa_dec
jmp iir_false

ssa_dec:
movl %ebx,%eax
jmp call_res

iir_false:
movl %ecx,%ebx
movl 36(%ebx),%ebx
movl <addOverflow_fast>,%edi
movl $25,%edx
movl $4,%esi
movl $4,4(%ecx)
call *%edi
jmp call_res

call_res:
ret $0
```

x86 Machine Code

```
<fn:add1>
movl 4(%ecx),%edi
subl $3,%edi
testl %edi,%edi
je L7828
cmpl $0,%edi
jg L7829
movl $25,%ebp
movl 4(%ecx),%edi
cmpl $0,%edi
cmovlel %ebp,%edx
cmpl $1,%edi
cmovlel %ebp,%ebx
cmpl $2,%edi
cmovlel %ebp,%eax
jmp L7828
L7829:
movl %eax,12(%ecx)
movl %esp,%ebp
subl $1,%edi
cmpl $0,%edi
jle L7828
L7831:
cmpl %esp,%ebp
jl L7830
movl (%ebp),%eax
movl %eax,(%ebp,%edi,4)
subl $4,%ebp
jmp L7831
L7830:
movl 12(%ecx),%eax
sall $2,%edi
addl %edi,%esp
L7828:

entry:
movl %eax,%ebx
andl $3,%ebx
testl %ebx,%ebx
movl $0,%ebx
cmovzl %esp,%ebx
testl %ebx,%ebx
je if_false
jmp log_and_sec

if_false:
movl %ecx,%ebx
movl 36(%ebx),%ebx
movl <addGeneral_fast>,%edi
movl $25,%edx
movl $4,%esi
movl $4,4(%ecx)
call *%edi
jmp call_res

log_and_sec:
movl %eax,%ebx
addl $4,%ebx
jno ssa_dec
jmp iir_false

ssa_dec:
movl %ebx,%eax
jmp call_res

iir_false:
movl %ecx,%ebx
movl 36(%ebx),%ebx
movl <addOverflow_fast>,%edi
movl $25,%edx
movl $4,%esi
movl $4,4(%ecx)
call *%edi
jmp call_res

call_res:
ret $0
```

Extended JavaScript

- Primitive functions implement the JS semantics
  - e.g.: `add`, `sub`, `newObject`, `getProp`, `putProp`
  - These need direct memory access, machine integer types
  - JS by itself isn't quite expressive enough
- Extended JavaScript
  - Foreign Function Interface (FFI) system to call into C code
  - Function prologue annotations
  - Inline IR (like inline assembly)
  - Object layouts (like C structs)
  - Named symbolic constants

Function Annotations

`"static"`	Statically linked function
`"inline"`	Always inline function
`"noglobal"`	No access to global object
`"cproxy"`	Function callable from C
`"arg <name> <type>"`	Low-level argument types
`"ret <type>"`	Low-level return type

Inline IR

- Inline IR system
    - Exposes low-level VM instructions
    - Direct pointer and memory manipulation
    - Machine integer and FP types (e.g.: int32, float64)
    - Like inline assembly, but machine-independent, portable
- IIR instructions include:
    - `load`, `store`, `add`, `add_ovf`, `sub`, `sub_ovf`, etc.
    - Appear like function calls in JS code
- Manipulating objects using `load`, `store` is cumbersome
    - Layout system to describe memory layouts (C struct-like)
    - Auto-generate method to allocate, get/set layout fields

Example Primitive (1/2)

```
function newObject(proto) {
    "tachyon:static";
    "tachyon:noglobal";

    assert (
        proto === null || boxIsObjExt(proto),
        'invalid object prototype'
    );

    var obj = alloc_obj();

    set_obj_proto(obj, proto);

    set_obj_numprops(obj, u32(0));

    var hashtbl = alloc_hashtbl(HASH_MAP_INIT_SIZE);
    set_obj_tbl(obj, hashtbl);

    return obj;
}
```

Example Primitive (1/2)

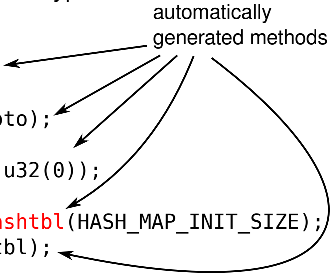
```
function newObject(proto) {
    "tachyon:static";
    "tachyon:noglobal";

    assert (
        proto === null || boxIsObjExt(proto),
        'invalid object prototype'
    );

    var obj = alloc_obj();

    set_obj_proto(obj, proto);

    set_obj_numprops(obj, u32(0));

    var hashtbl = alloc_hashtbl(HASH_MAP_INIT_SIZE);
    set_obj_tbl(obj, hashtbl);

    return obj;
}
```

statically linked function

no access to global object

Example Primitive (1/2)

```
function newObject(proto) {
    "tachyon:static";
    "tachyon:noglobal";

    assert (
        proto === null || boxIsObjExt(proto),
        'invalid object prototype'
    );

    var obj = alloc_obj();

    set_obj_proto(obj, proto);

    set_obj_numprops(obj, u32(0));

    var hashtbl = alloc_hashtbl(HASH_MAP_INIT_SIZE);
    set_obj_tbl(obj, hashtbl);

    return obj;
}
```



Example Primitive (1/2)

```
function newObject(proto) {
    "tachyon:static";
    "tachyon:noglobal";

    assert (
        proto === null || boxIsObjExt(proto),
        'invalid object prototype'
    );

    var obj = alloc_obj();

    set_obj_proto(obj, proto);

    set_obj_numprops(obj, u32(0));

    var hashtbl = alloc_hashtbl(HASH_MAP_INIT_SIZE);
    set_obj_tbl(obj, hashtbl);

    return obj;
}
```

named symbolic constant

Example Primitive (2/2)

```
function cStringToBox(strPtr) {
    "tachyon:static";
    "tachyon:noglobal";
    "tachyon:arg strPtr rptr";

    if (strPtr === NULL_PTR) return null;

    for (var strLen = pint(0); ; strLen++) {
        var ch = iir.load(IRType.i8, strPtr, strLen);
        if (ch === i8(0)) break;
    }

    var strObj = alloc_str(strLen);

    for (var i = pint(0); i < strLen; i++) {
        var cCh = iir.load(IRType.i8, strPtr, i);
        var ch = iir.icast(IRType.u16, cCh);
        set_str_data(strObj, i, ch);
    }

    compStrHash(strObj);
    return getTableStr(strObj);
}
```

Example Primitive (2/2)

```
function cStringToBox(strPtr) {
    "tachyon:static";
    "tachyon:noglobal";
    "tachyon:arg strPtr rptr";

    if (strPtr === NULL_PTR) return null;

    for (var strLen = pint(0); ; strLen++) {
        var ch = iir.load(IRType.i8, strPtr, strLen);
        if (ch === i8(0)) break;
    }

    var strObj = alloc_str(strLen);

    for (var i = pint(0); i < strLen; i++) {
        var cCh = iir.load(IRType.i8, strPtr, i);
        var ch = iir.icast(IRType.u16, cCh);
        set_str_data(strObj, i, ch);
    }

    compStrHash(strObj);
    return getTableStr(strObj);
}
```

statically linked function

no access to global object

strPtr is a raw pointer (char*)

Example Primitive (2/2)

```
function cStringToBox(strPtr) {
    "tachyon:static";
    "tachyon:noglobal";
    "tachyon:arg strPtr rptr";

    if (strPtr === NULL_PTR) return null;

    for (var strLen = pint(0); ; strLen++) {
        var ch = iir.load(IRType.i8, strPtr, strLen);
        if (ch === i8(0)) break;
    }

    var strObj = alloc_str(strLen);

    for (var i = pint(0); i < strLen; i++) {
        var cCh = iir.load(IRType.i8, strPtr, i);
        var ch = iir.icast(IRType.u16, cCh);
        set_str_data(strObj, i, ch);
    }

    compStrHash(strObj);
    return getTableStr(strObj);
}
```

low-level integer types

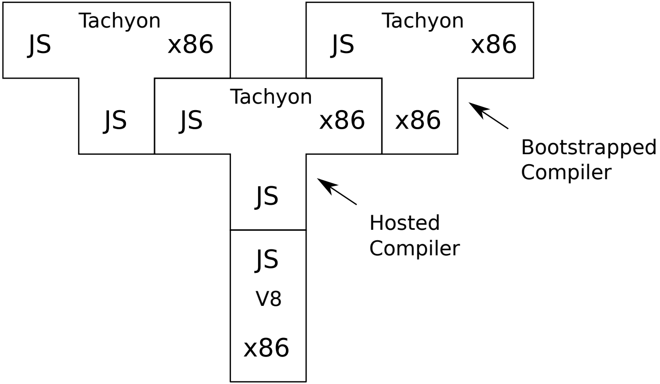
Example Primitive (2/2)

```
function cStringToBox(strPtr) {
    "tachyon:static";
    "tachyon:noglobal";
    "tachyon:arg strPtr rptr";

    if (strPtr === NULL_PTR) return null;

    for (var strLen = pint(0); ; strLen++) {
        var ch = iir.load(IRType.i8, strPtr, strLen);
        if (ch === i8(0)) break;
    }

    var strObj = alloc_str(strLen);

    for (var i = pint(0); i < strLen; i++) {
        var cCh = iir.load(IRType.i8, strPtr, i);
        var ch = iir.icast(IRType.u16, cCh);
        set_str_data(strObj, i, ch);
    }

    compStrHash(strObj);
    return getTableStr(strObj);
}
```

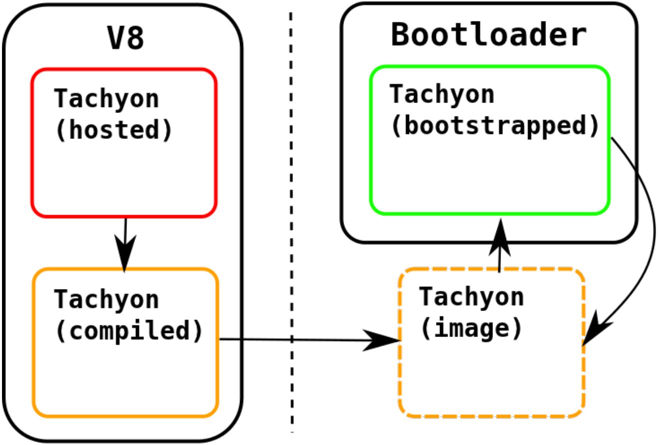
memory load from pointer

low-level integer cast

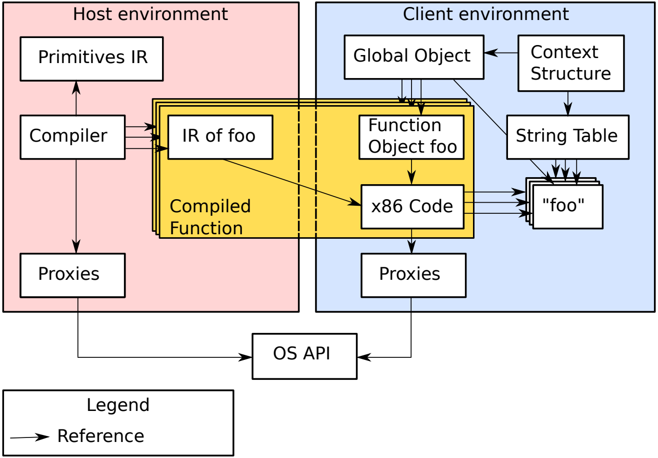
The Bootstrap



Tachyon's Independence



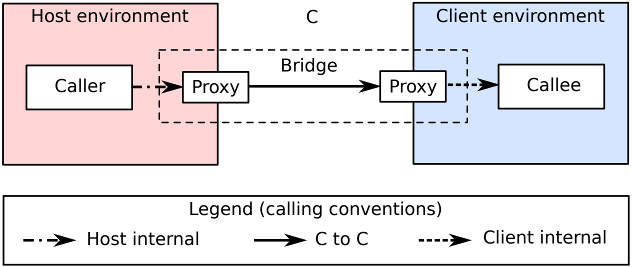
VM Execution Model



VM Initialization

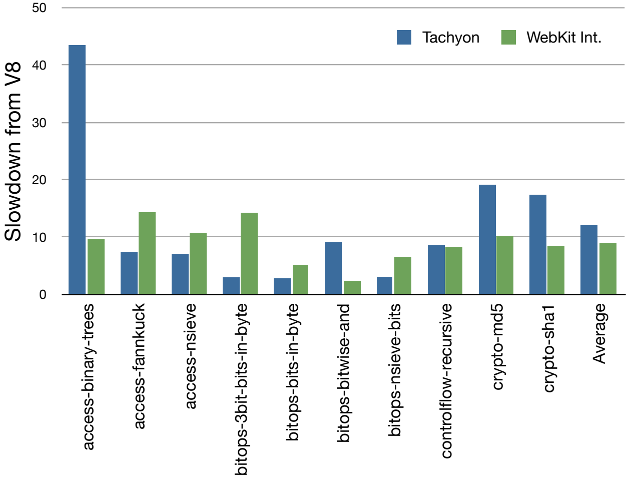
- Self-initialization
  - Host VM does not manipulate Tachyon objects directly
  - Can call Tachyon functions through bridges
- Initialization in multiple steps
  - Compilation & initial linking of primitives
  - Memory block allocated for heap
  - Call to `initHeap(heapPtr, heapSize)`
    - Allocates context structure, global object
  - Re-linking of primitives
    - Strings allocated w/ `getStrObj(rawStr, strLen)`
  - Compilation, linking of stdlib
  - Compilation, linking of the rest of Tachyon

Bridges



Host environment
C
Client environment
Bridge
Caller
Proxy
Proxy
Callee
Legend (calling conventions)
Host internal
C to C
Client internal

Early Performance Numbers



JavaScript for Compiler Writing

- JS lends itself nicely to data manipulation
  - Makes implementing analyses, optimizations easier
- The ES5 standard library is rather incomplete
  - No data structures (e.g.: hash map/set), few string functions
- Lack of static checking can make refactorings harder
  - Unit tests, assertions are critical
- Lack of module system is annoying (will be fixed soon!)
- Low-level code successfully limited to a few areas (backend, primitives)

Current Project Status

- What we have
  - All ES5 language constructs
  - Objects, closures, arrays
  - Almost complete ES5 standard library
    - Array, String, RegExp, Date, etc.
  - Fairly comprehensive unit test suite
  - Many useful tools
    - JS parser, pretty-printer, profiler
- To be completed
  - Object property attributes (e.g.: read-only)
  - Garbage collector (!)
  - Exceptions
  - Full floating-point support

Recap & Conclusion

- Tachyon is a self-hosted JS compiler
  - Pure JIT compiler
  - Extended JS dialect
- Bootstrap using "self-initialization" mechanism
- Supports most of ES5
  - Working on adding missing features
- Plan to use Tachyon to optimization ideas
  - Type inference
  - Self-optimization
- Open source (BSD license)

Thanks for listening!

We welcome your questions/comments

Feel free to contact the Dynamic Language Team (DLT):

{chevalma,lavoeric,feeley,dufour}@iro.umontreal.ca

Dynamic Language Team at UdeM

- Tachyon
  - Dynamic type analysis
  - Optimistic optimization
- Photon
  - Highly-dynamic system
  - Live programming
- Program analysis
  - Type profiling
- All our code is open source