COSC345
How JavaScript works
March 24 and 27 2015
What JavaScript is

• A **dynamically typed** language

• Compiled to **bytecodes** for a **virtual machine**

• With one or more **Just-In-Time Compilers** from byte codes to native
No standard VM!

- Java has the Java VM
- C# (F#, VB.Net, &c) has the CLR
- There is no standard VM for JavaScript
- SpiderMonkey (Mozilla) V8 (Chrome) are two of several. They are different!
One consequence

- You can compile language X to JVM .class files without going through Java.
- You can compile language X to CLR assemblies without going through C#.
- But to run with JavaScript, you have to compile language X to JavaScript source.
Statically typed

- Examples: C, C++, Fortran, Ada
- Yes, even Java and C#.
- More exotic examples like ML and F#.
- Compiler catches errors at compile time
- Not the original motivation!
What’s what (32)?

- A 32-bit word could be a float, unsigned int, int, unsigned short[2], short[2], unsigned char[4], signed char[4], bit[32], &c,

- Or it could be a pointer to something, like an array (char*) or record.
What’s what (64)?

• A 64-bit word could be a double, float[2], unsigned long long, long long, unsigned int[2], int[2], unsigned short[4], short[4], unsigned char[8], signed char[8], bit[64], &c,

• Or it could be a pointer to something, like an array (char*) or record.
Fortran types

- Fortran introduced types in the late 1950s
- So the compiler could generate fast code
- \( X = Y + Z \Rightarrow \text{ld.f r, Y; add.f r, Z; st.f r, X} \)
- OR \( \text{ld.w i, Y; add.w i, Z; st.w i, X} \)
- depending on the types of \( X, Y, Z \)
- *no* run-time overhead
Fortran types (2)

- The compiler knows what is in a word and can generate native code that does the right thing.
- Detecting uninitialised variables got harder; you couldn’t have a special ‘undefined’ value as JavaScript has.
And arrays

- Arrays in Fortran (C, Pascal, &c) use arithmetic on addresses to locate array elements in constant time

- float x[20][30]; ... y = x[i][j] ⇒

- ld.w r,i; mul.w r,=30; add.w r,j; mul.w r,=4; ld.f t, x(r); st.f t, y
Array price

- Checking subscript bounds requires extra code: `ld.w r,i; cmp.uw r,=20; trap.hs`

- Or it requires a smart compiler that can omit checks in known safe cases.

- C omits the checks for speed; some Fortran compilers make it optional; Ada and modern Java optimise them.
Records (structs)

- The compiler knows the layout of a record
- `typedef struct Date { short /*0*/year;
  char /*2*/month; char /*3*/ day; } Date;`
- Field access uses address arithmetic:
  - `pDate->y` ⇒ `ld.b r,3(pDate)`
- Access to nested structs: 0 runtime cost
Dynamic typing

- A type is part of a **value** at run time, not part of a compiler’s knowledge.
- The type has to be **stored** somehow.
- The type has to be **checked** somehow.
- All those nice fast things go away.
So who wants it?

- Dynamic typing simplifies simple-minded compilers.
- Less work for programmer
- May be safer at run time, always checking
- Revision doesn’t require recompiling
- Lisp, Smalltalk, Prolog, Python, JavaScript
Storing the type

- Make everything a pointer. Everything is the same size: simple.

- Blg Bag Of Pages: each page holds the same kind of thing, shift the pointer and index a table to get the type number. Use a reserved range of pages to encode small integers. Xerox Lisp: worked very well.
Storing the type (2)

- Alternative, make every kind of value a record. Put the type information in the first word. (Just like what C++ does with objects.)
- Type information is called a TAG.
- Takes an extra memory reference.
- Icon does this. Simple, portable.
Storing the type (3)

- Put the type in the pointer. Ensure that every object is 4-byte or 8-byte aligned and steal 2 or 3 tag bits out of every pointer.
- Allows 30-bit integers, booleans, characters, &c to be represented directly; immediate representation
- Other things are boxed, including floats
SpiderMonkey tagging

- Integer*2 + 1 (31-bit immediate ints)
- Object pointer + 0
- Double pointer + 2
- String pointer + 4
- Boolean*8 + 6 (immediate booleans)
null is object 0

*Everything* that is not a number or string or boolean or null is an object.

This includes arrays and functions.

Which is fine because in JavaScript those *are* objects.
Arithmetic

• x-y : only doubles exist in programmer’s model but implementation has ints as well
• See case JSOP_SUB: in jsinterp.c
• x+y : could also be string concatenation
• See case JSOP_ADD: in jsinterp.c