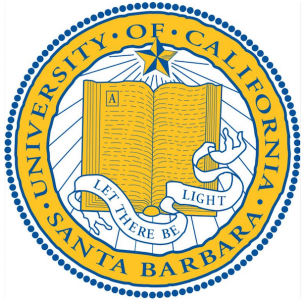
# CS 290B

## Scalable Internet Services

Andrew Mutz

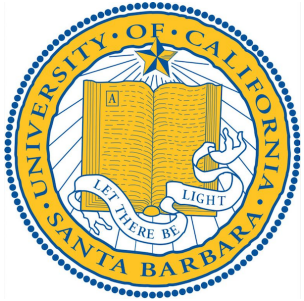
December 2, 2014



# For Today

Client-side renaissance continued

- Asm.js
- Emscripten
- MRuby

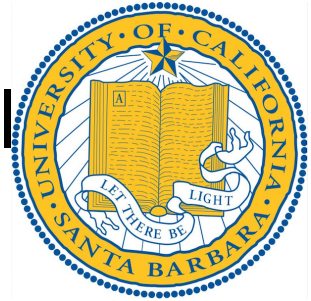


# Introduction

http://coolwanglu.github.io/vim.js/experimental/vim.html

http://danielsadventure.info/html5fractal

http://crypt-webgl.unigine.com/game.html

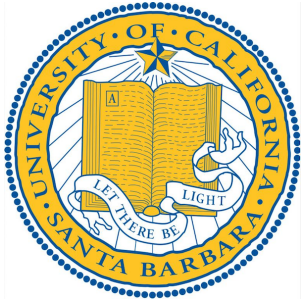


# Introduction

Web-based applications have taken over many domains

- Email, calendaring, word processing, social, etc.
- HTML5 has Geo-location, WebRTC, etc.

What can’t web browsers do?

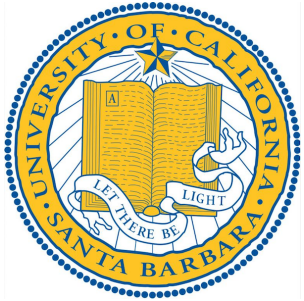


# Introduction

Web browsers struggle with compute heavy tasks...

- Games
- Speech recognition
- Image recognition
- Image processing, effects

Lots of work has been put into making JS fast, but its still a scripting language.

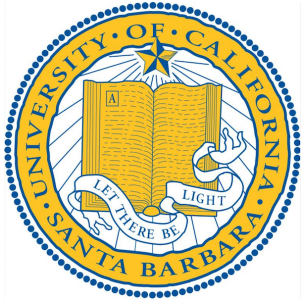


# Introduction

Competitive pressure on browser vendors.
More and more is desired from browsers.

Javascript engine performance has been improving

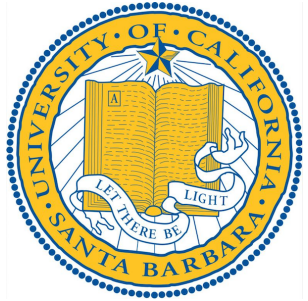
- Dynamic compilation
- Advanced garbage collection
- Other VM optimzations



# Introduction

There is no other language on the client.

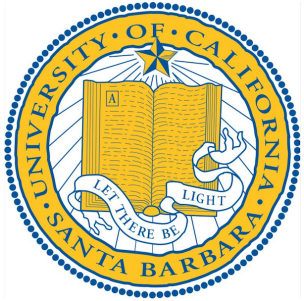
- `<script type=”text/c”>` doesn’t exist
  - Why doesn’t this exist?
- This means we are limited in the compute performance on the client.
- This also means we don’t have language heterogeneity on the client
  - Why is this bad?



# Introduction

Key observation:

- Javascript engines are limited in the performance they can deliver because of the rich feature set of Javascript.
  - Garbage collection, Dynamic dispatch, etc.
- If Javascript didn't have some of these features, we could build really high performance VMs
- **More usefully, if restrict ourselves to the fast subset of features, VM designers can make that go very fast.**

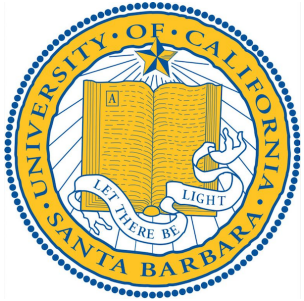


# Asm.js

A subset of Javascript that VMs can execute very fast.

Removes most memory management and dynamic typing.

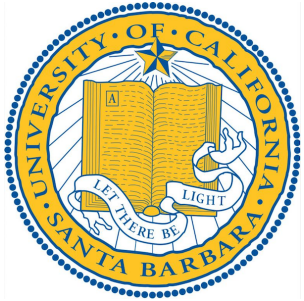
Developed at Mozilla in 2013

# Asm.js

Example:

```
f = function(a, b){
  return a + b;
}
f(5, 2)
=> 7

f(“foo”, “bar”)
=> “foobar”
```



# Asm.js

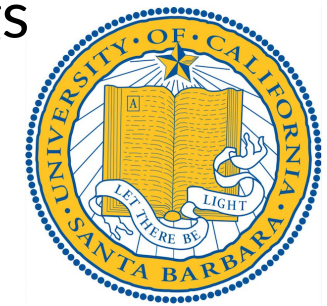
## Example:

```
f = function(a, b){
  return a + b;
}
f(5, 2)
=> 7

f(“foo”, “bar”)
=> “foobar”
```

How is `f()` compiled to machine code?

Which “+” should be used?

Depends on the args passed to `f()` at runtime



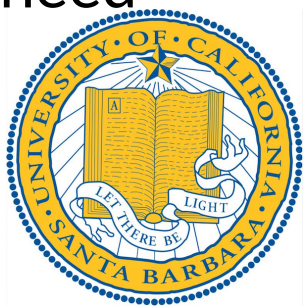
# Asm.js

Example:

```
f = function(a, b){
  a = a|0;
  b = b|0;
  return a + b;
}
f(5, 2)
=> 7
```

The bitwise `|0` doesn't affect the value, but guarantees the result is an integer

We've removed the need to check types to implement "+"

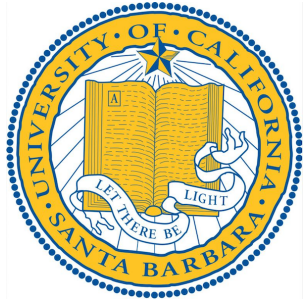


# Asm.js

## Example:

```
f = function(a, b){
  a = a|0;
  b = b|0;
  return a + b;
}
f(5, 2)
=> 7
```

In addition to guaranteeing an integer, `|0` serves as a hint to VM implementors that are Asm.js aware.



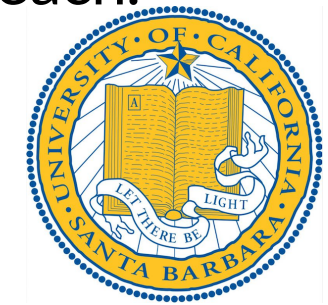
# Asm.js

## Another example:

```
result = ""
for(i=10; i>0; i--){
  result += "a"
}
```

In this code, the variable “result” is getting repeatedly reassigned.

The garbage collector here is cleaning up after each.

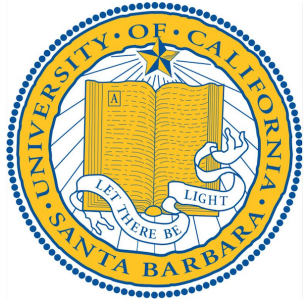


# Asm.js

Another example:

```
result = ""
for(i=10; i>0; i--){
  result += "a"
}
```

**If strings are immutable in Javascript, how do we avoid this?**



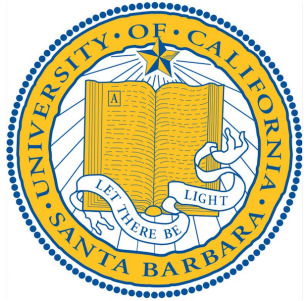
# Asm.js

## Another example:

```
buff = Int32Array(10);
for(i=10; i>0; i--){
  buff[i] = "a".charCodeAt(0)
}
result = String.fromCharCode.
apply(
  String, buff);
```

Avoid the garbage collector completely.

Allocate a reusable array of integers and manage your memory by hand.



# Asm.js

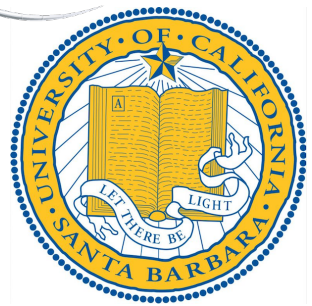
So, by restricting our use of Javascript to static typing and manual memory management, we can generate JS that can be executed very quickly.

**Who would ever want to write code like this?**

# LLVM

## LLVM: Low Level Virtual Machine project

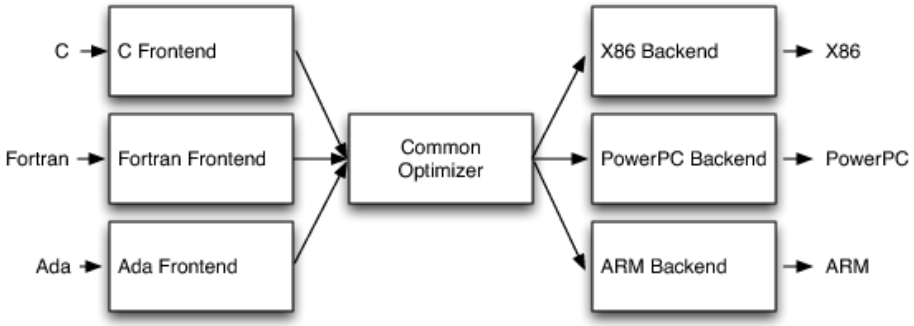
- Language-agnostic low level bytecode that can be translated to machine code.
- Many languages support generating LLVM: C, C++, Python, Ruby, Rust, Scala, etc.

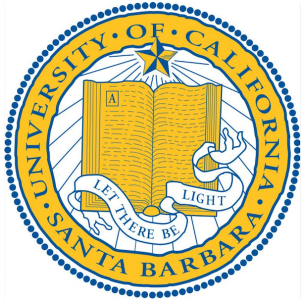


# LLVM

Basic idea:

- Have many front-end libraries able to generate LLVM bytecode
- LLVM optimizations are language/architecture agnostic
- Many back-ends to generate bytecode for each architecture.

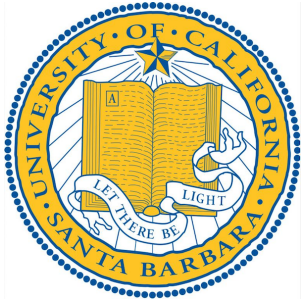


# LLVM

## Sample:

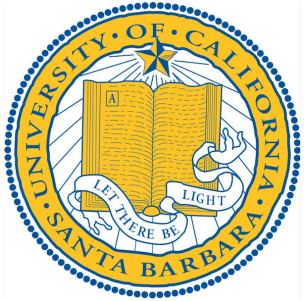
```
#include<stdio.h>

int main() {
  printf("hello, world!\n");
  return 0;
}
```



# LLVM

## Sample:

```
@.str = private unnamed_addr constant [15 x i8] c"hello, world!\0A\00", align 1
; Function Attrs: nounwind ssp uwtable
define i32 @main() #0 {
  %1 = alloca i32, align 4
  store i32 0, i32* %1
  %2 = call i32 (i8*, ...)* @printf(
    i8* getelementptr inbounds ([15 x i8]* @.str, i32 0, i32 0))
  ret i32 0
}
```



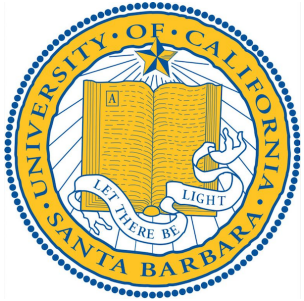
# Asm.js

So we have many languages compiling to LLVM, and many backends that translate that to machine instructions.

What do we get if we write a backend that translates to Javascript?

# Emscripten

Emscripten translates LLVM bytecode into Asm.js

# Emscripten

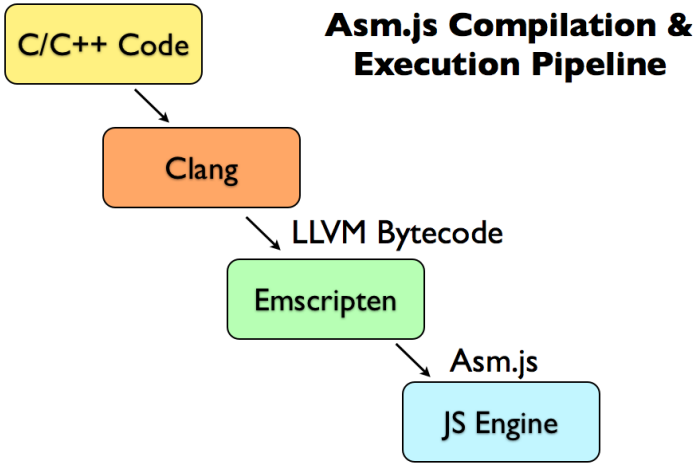
Example:

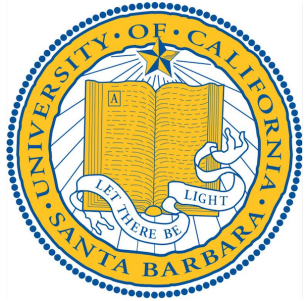
```
//LLVMIR
define i32 @func(i32* %p){
  %r= load i32* %p
  %s= shl i32 %r, 16
  %t= call i32 @calc(i32 %r, i32 %s)
  ret i32 %t
}
```

```
// JS
function func(p){
  var r = HEAP[p];
  return calc(r, r << 16);
}
```

# Emscripten

Because we have many languages that compile to LLVM, we can now use Emscripten to execute these languages in browsers

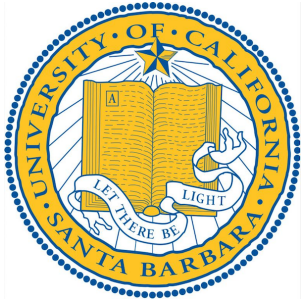


# Emscripten

What happens to application performance?

- What % performance decrease do you think?

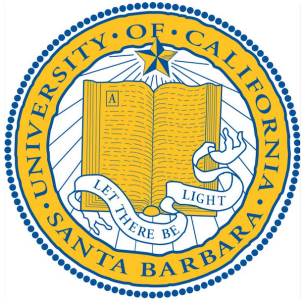


# Emscripten

What happens to application performance?

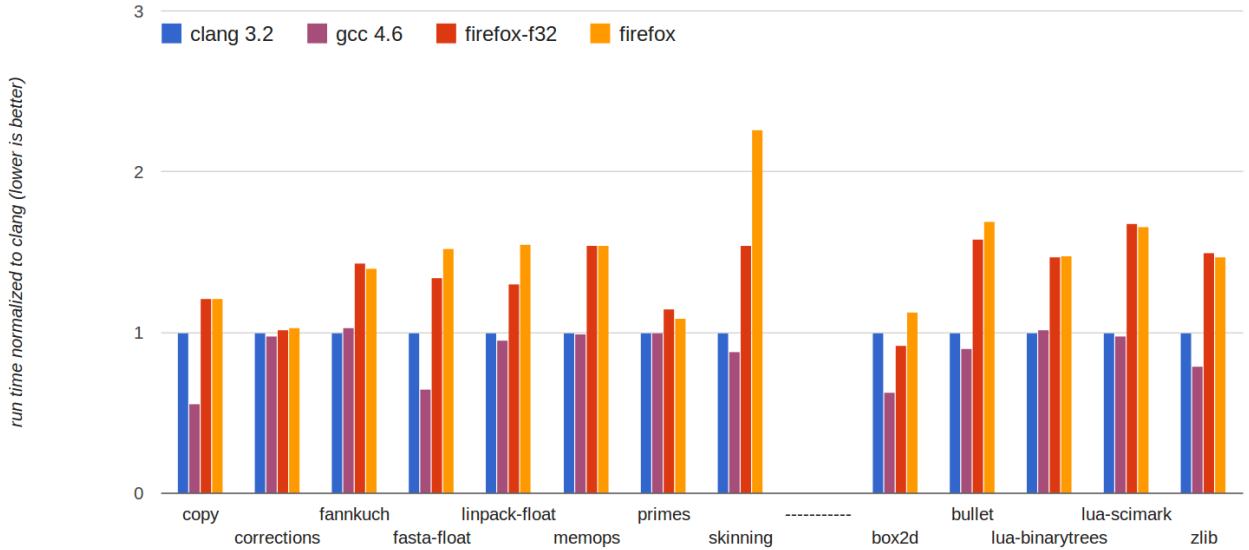
- What % performance decrease do you think?

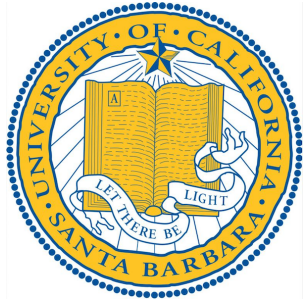
**Project authors suggest 2X slowdown over native code!**



# Emscripten

## What happens to application performance?



# MRuby

**Demo!**

**http://joshnuss.github.io/mruby-web-irb/**