

Fun in the Web: Web based programming environments for functional programs



Danny Yoo
WPI

Functional

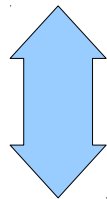
event-driven programming

on the Web!

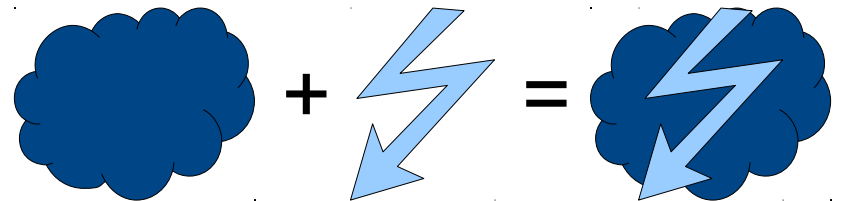
Functional...

This matters for education: algebra and functional programming have a relationship that can reinforce each other

$$h(a, b) = \sqrt{a^2 + b^2}$$



```
(define (h a b)
  (sqrt (+ (sqr a) (sqr b))))
```




... event driven...

- Works well for writing games (driven by events like user interaction, clock ticks, etc...)
- Behaves appropriately for multiple platforms, including both the desktop browser and the cell phone

... on the Web

The Web is ubiquitous on both desktops and smartphones

```
> (+ 1/2 1/3)
5/6
> 22/7
3.142857
> (list (circle 20 "solid" "green")
        (rectangle 30 20
                  "outline" "maroon"))
```

```
(list  )
> (add1 (/ 1 0))
```

/: division by zero

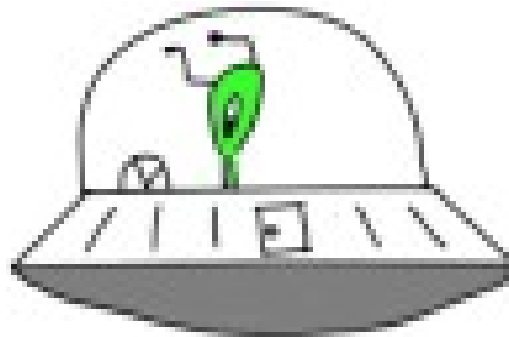
[at line 1, column 6, in <interactions3>](#)

[at line 1, column 0, in <interactions3>](#)

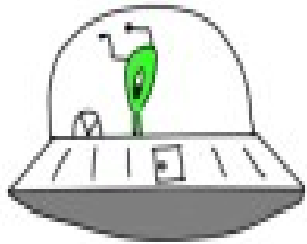
```
> |
```



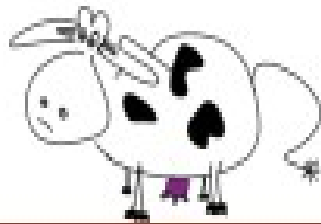
How do we make this program?



The World Programming Model



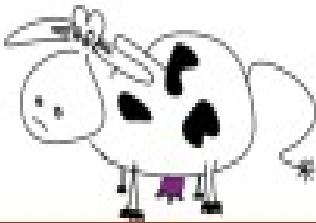
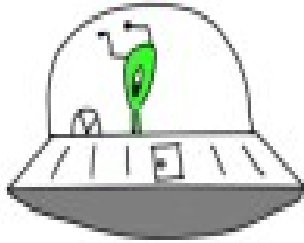
The “WORLD” represents the state of our program, like the distance from the sky



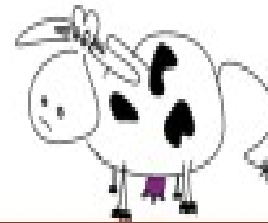
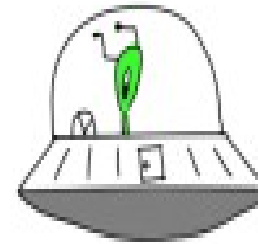
We can write functions that consume the world and produce an output

`draw(world)` = a picture of the UFO at the given distance

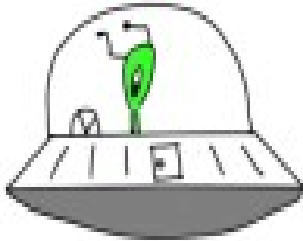
draw(0) =



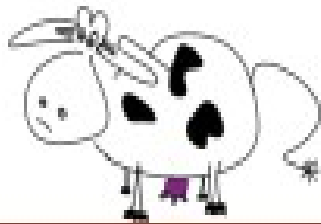
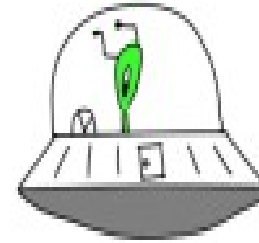
draw(20) =



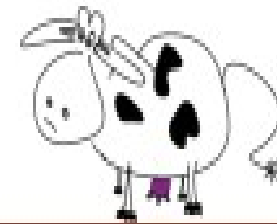
The World Programming Model



We can write a function that computes a new world from the previous one.



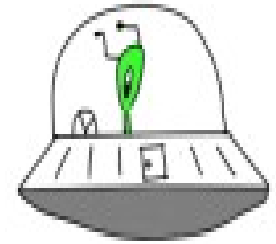
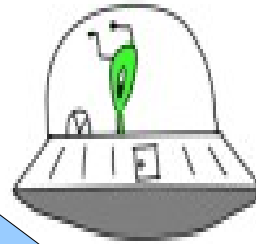
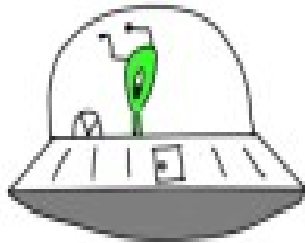
```
descend(world) =  
world + 20
```



`world_0 = 0`

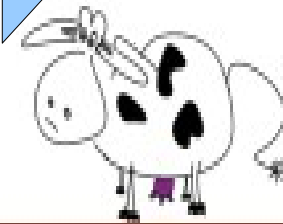
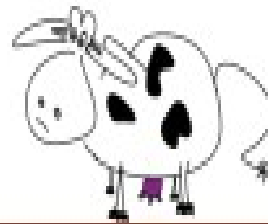
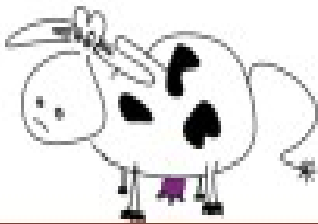
`world_1 = 20`
`= descend(world_0)`

The World Programming Model



Clock tick: descend

Key Press: ...



A World Program

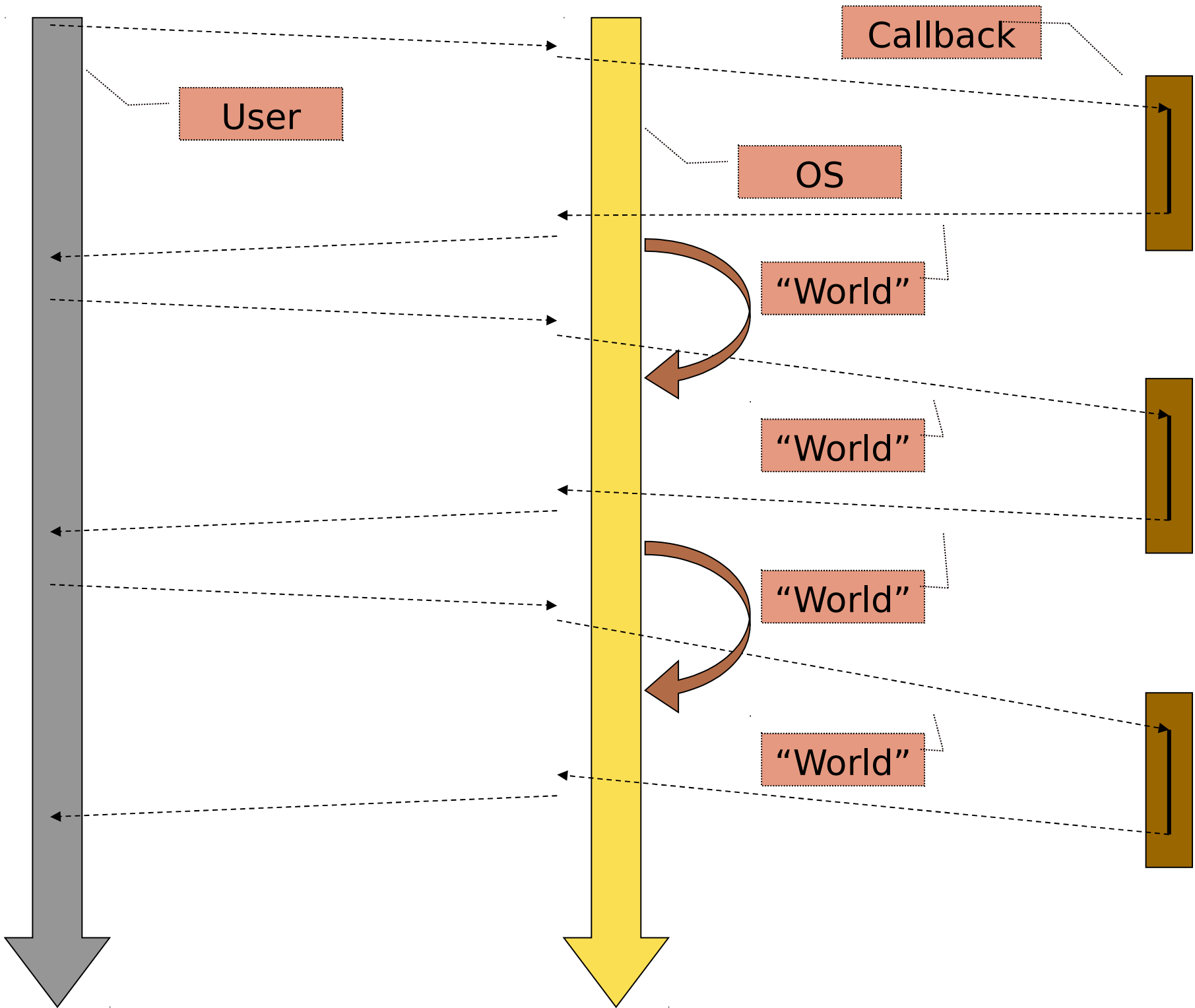
```
(define UFO  
  (bitmap/url "http://world.cs.brown.edu/1/clipart/ufo.png"))
```

```
(define (descend height) (+ height 5))
```

```
(define (draw height)  
  (place-image UFO 150 height  
    (empty-scene 300 300)))
```

;; The use of big-bang starts the world program.

```
(big-bang 0 ;; initially, let the height be zero.  
  (on-tick descend 1/15) ;; tick every 15 frames a second  
  (to-draw draw)) ;; use draw to draw the scene
```



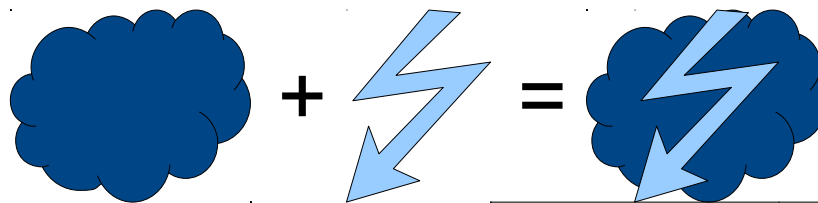
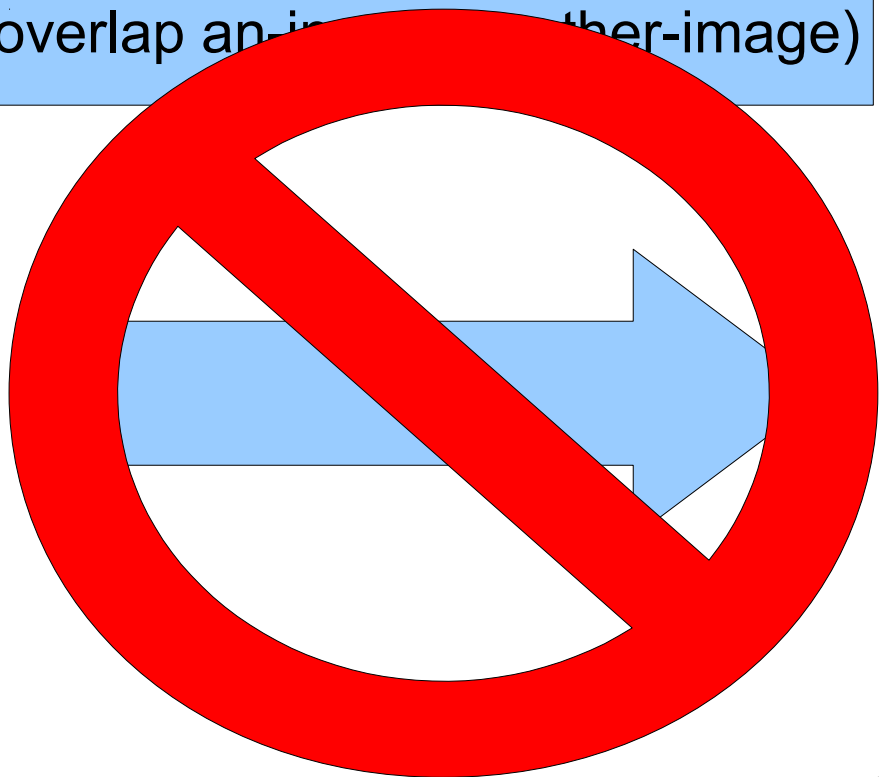
Problems

Challenges to World Programming on the Web

- JavaScript is the language of the web browser
- ***We should be able to run World in JS***
- There are problems inherent to JavaScript:
 - Lack of user-friendly error messages
 - Important for beginner programmers, since no one is perfect
- But there are more serious issues...
asynchrony!

Asynchrony example 1: Image Initialization

```
(define an-image  
  (bitmap/url "http://..."))  
  
(define another-image  
  (bitmap/url "http://..."))  
  
(overlap an-image another-image)
```



“block until loaded”

```
var anImage = new Image();  
anImage.src = "http://...";  
  
var anotherImage =  
  new Image();  
anotherImage.src =  
  "http://...";  
  
overlap(anImage, anotherImage);
```

What happens when we don't wait?

- In the best case...

uncaught exception: [Exception... "Component returned failure code: 0x80040111 (NS_ERROR_NOT_AVAILABLE) [nsIDOMCanvasRenderingContext2D.drawImage]" nsresult: "0x80040111 (NS_ERROR_NOT_AVAILABLE)" location: "JS frame :: file:///gpfs/main/home/dyoo/foo.html :: <TOP_LEVEL> :: line 13" data: no]

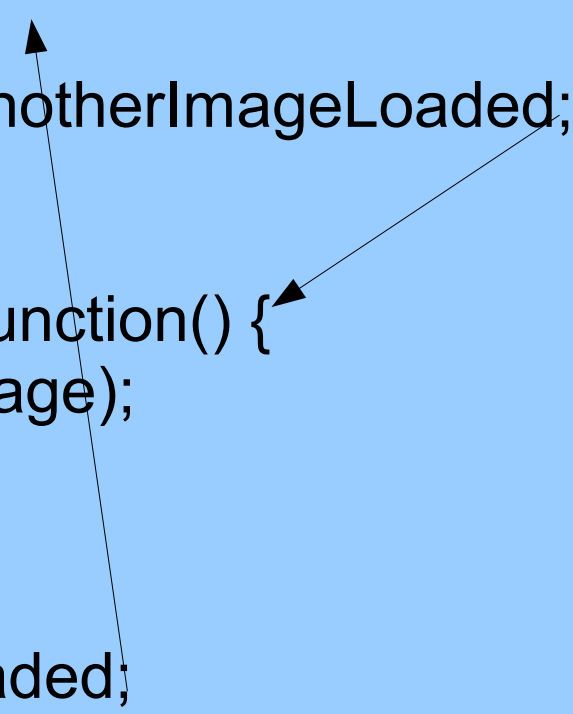
Functional programs on top of
JavaScript need to be able to block.

```
var anImage, anotherImage;

var afterImageLoaded = function() {
  anotherImage = new Image();
  anotherImage.onload = afterAnotherImageLoaded;
  anotherImage.src = "http://...";
};

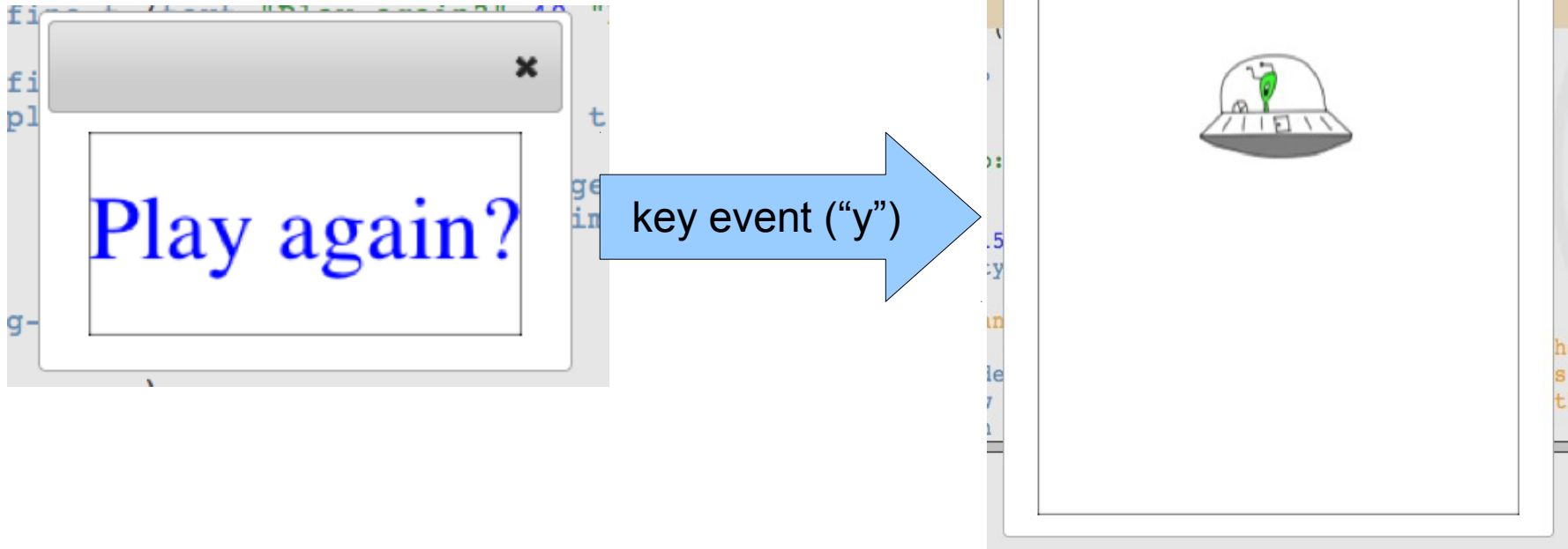
var afterAnotherImageLoaded = function() {
  overlap(anImage, anotherImage);
};

anImage = new Image();
anImage.onload = afterImageLoaded;
anImage.src = "http://...";
```

A diagram with two arrows. One arrow starts from the line 'anotherImage.onload = afterAnotherImageLoaded;' in the first function and points to the opening curly brace of the second function. The other arrow starts from the line 'anotherImage.src = "http://...";' in the second function and points to the opening curly brace of the first function.

This should be handled by the language,
not the programmer.

Asynchrony example 2: Modal dialogs



Big-bang should be a function!

```
(define (user-chooses-yes?)  
  (big-bang 'not-yet  
            (to-draw ...)   
            (stop-when choice-made?)  
            (on-key key-pressed)))
```

This big-bang should wait until choice-made? is true

```
(define (choice-made? w key)  
  (not (eq? w 'not-yet)))
```

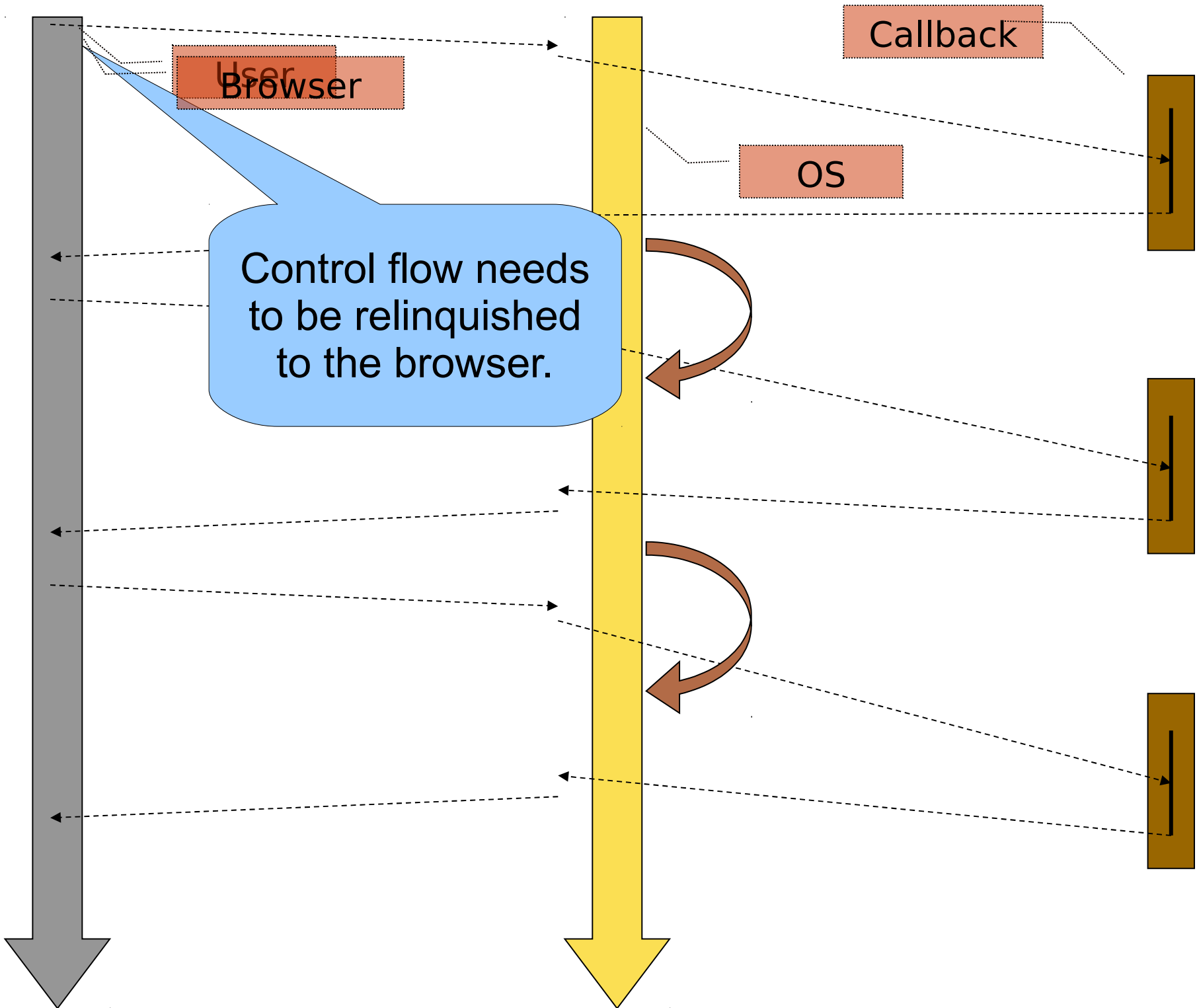
```
(define (key-pressed w a-key)  
  (cond [(key=? a-key "y") true]  
        [(key=? a-key "n") false]  
        [else w]))
```

```
(if (user-chooses-yes?)  
    (start-again)  
    (quit-game))
```

After big-bang returns its value (the world), we should be able to branch on its value

Using big-bang for dialogs

- Big-bang should be a function that can return a value
- Mechanically, big-bang needs to give control to the browser to process events, to handle events like clock ticks or key events
 - JavaScript is a single-threaded, event-driven language
- Again, we need to abandon control back to the browser



Can big-bang be an function?

```
(define (user-chooses-yes?)  
  (big-bang 'not-yet  
    (to-draw ...)   
    (stop-when choice-made?)  
    (on-key key-pressed)))
```

```
(define (choice-made? w key)  
  (not (eq? w 'not-yet)))
```

```
(define (key-pressed w a-key)  
  (cond [(key=? a-key "y") true]  
        [(key=? a-key "n") false]  
        [else w]))
```

```
(if (user-chooses-yes?)  
    (start-again)  
    (quit-game))
```

This big-bang must save the rest of the computation... and then release control!

The rest of the computation needs to be reinstated at some future point

Functional programs on JavaScript
need to be able to abandon and
restore their control context.

Solving these problems

Applying delimited control to asynchronicity

- Javascript alone doesn't give us the necessary control primitives to implement images as values and functional big-bang.
- Control operators can be applied toward this problem. (i.e. continuations!)

Delimited control operators

- Save
 - Saves the current state of the computation
- Restore
 - Restarts a saved computation
- Abort
 - Abandons the current computation
- Prompt
 - Delimits the extent to which the continuation is captured and restored

Applying delimited control

- For image initialization:

```
var image = new Image();
saveContinuation();
image.onload = function() {
    restoreContinuation(image);
}
img.src = ...;
```

Applying delimited control

- For big-bang:

```
function bigBang(initialWorld, ...) {  
  theWorld = initialWorld;  
  initializeEventHandlers();  
  saveContinuation();  
  abortContinuation();  
}
```

- On stopWhen():

```
restoreContinuation(theWorld);
```

Summary of the approach

- Delimited control operations allow programs to wait
 - Addresses the image loading race condition
- They allow computation to be suspended and restored
 - Allows browser events to be processed in the middle of a long-running computation
 - big-bang can be functional

Implementing the Evaluator

Prototypes

- Explored simplest, direct solutions.
- Issues and problems informed the final design of the evaluator.

Design 1: BSL $\xrightarrow{\text{compile}}$ JavaScript $\xrightarrow{\text{eval()}}$ value

Design 2: Racket $\xrightarrow{\text{racket compilation}}$ bytecode $\xrightarrow{\text{managed_eval}}$ value

Final Design Racket $\xrightarrow{\text{racket compilation}}$ bytecode $\xrightarrow{\text{compile}}$ JavaScript $\xrightarrow{\text{managed_eval}}$ value

The elements in boldface are those that are implemented in this dissertation.

Prototype 1: naïve compilation

Design 1: BSL $\xrightarrow{\text{compile}}$ JavaScript $\xrightarrow{\text{eval()}}$ value

- Image initialization bug exposed need to control for asynchronicity
- Similarly, big-bang could only be used at toplevel due to asynchronicity constraint
- Student programs couldn't use advanced language features (macros, modules).
- Advanced recursive programs could hit the JavaScript stack ceiling.

Prototype 2: interpretation

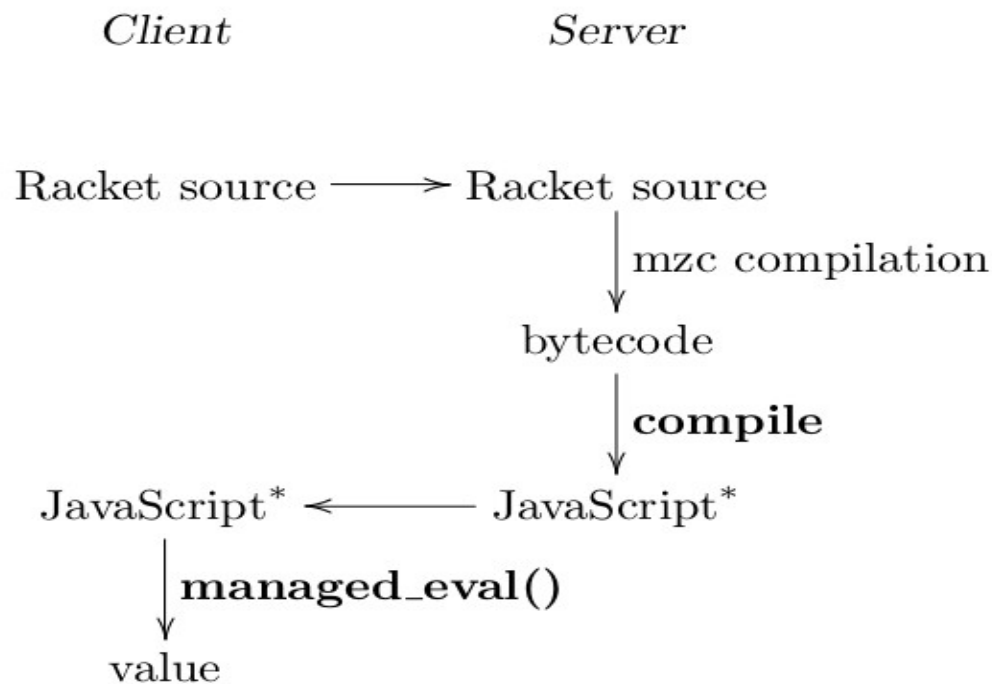
Design 2: `Racket` $\xrightarrow{\text{racket_compilation}}$ `bytecode` $\xrightarrow{\text{managed_eval}}$ `value`

Racket bytecode interpreter with explicit representation for the control stack

- Reuses Racket's production level compiler
- Solves asynchronicity with delimited control
- But: interpretive overhead proved costly enough to affect sophisticated programs (1000-5000X slower than Racket)

Final design: 1 + 2

JS compiler, like Design 1, but avoid the JavaScript stack entirely and maintain explicit control stack representation, like Design 2



Example run

Can reuse Racket compiler's optimizations (constant folding, loop unrolling, etc)

```
(let* ([x 3] [y 4])  
  (+ x y))  
...
```

Racket compile

```
(Top  
 (Prefix (list (ModuleVariable '+ '#%kernel))))  
 (Let1  
  (Constant 3)  
  (Let1 (Constant 4)  
    (App (ToplevelRef 4 0)  
      (list (LocalRef 3)  
            (LocalRef 2))))))  
...)
```

JS translate

```
env.push([Primitives["+"]]);  
env.push(undefined);  
env[env.length - 1 - 0] = 3;  
env.push(undefined);  
env[env.length - 1 - 0] = 4;  
val = (env[env.length - 1 - 1] + env[env.length - 1]);  
env.length = env.length - 2;  
...  
GOTO returnAddress;
```

JS as simple intermediate language, including assignments, stack operations, GOTOs..

How do we get to GOTO?

- JavaScript doesn't have GOTO statements
 - GOTO for primitive flow control, and to allow for delimited control operations like continuation capture

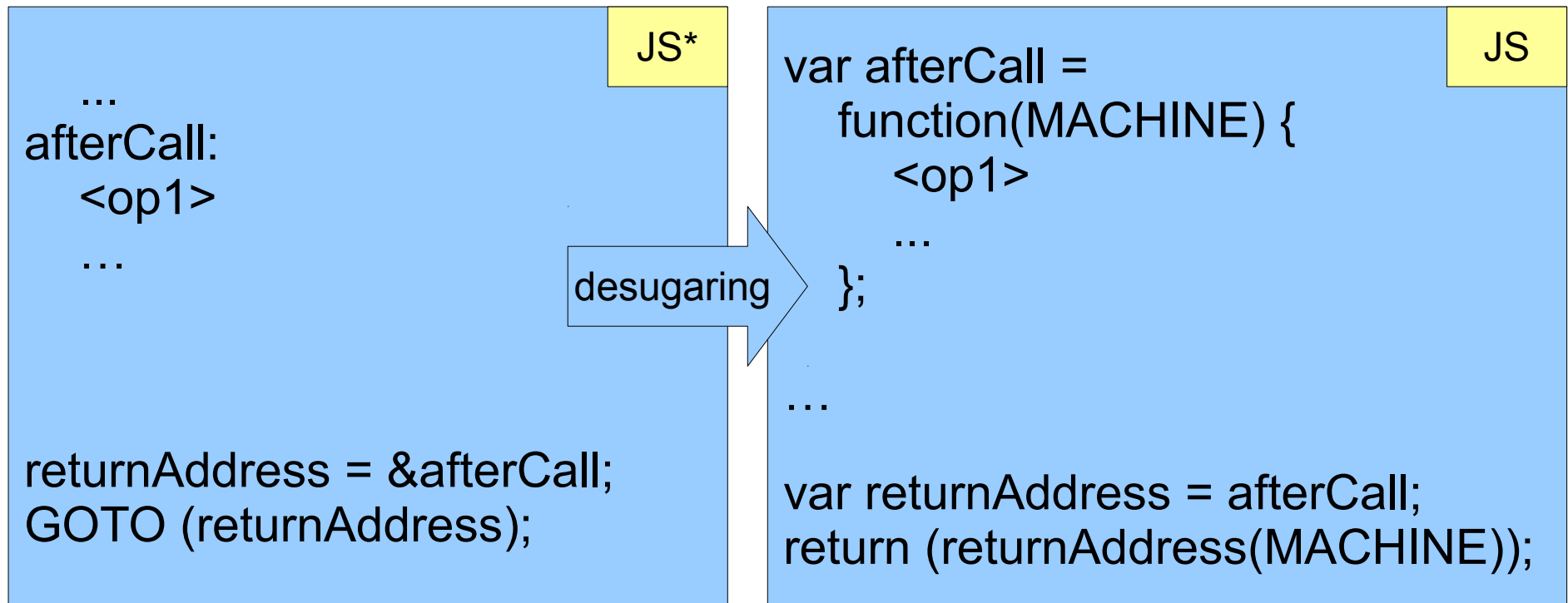
```
...  
afterCallSingle:  
...  
returnAddress = ...  
GOTO (returnAddress)
```



??

Lambda, the Ultimate GOTO

- However, JS supports functions
- Each basic block transforms to a function



Example run 2

```
(let-values  
  [(x rest-stack)  
   (stack-pop a-stack)])  
...
```

Racket compile →

```
(Top  
 (Prefix (list (ModuleVariable  
 '+ '#%kernel))))  
 (Let-Values  
  (list 'x 'rest-stack)  
  (App stack-pop a-stack))  
 ...)
```



??

Multiple value returns

- Multiple value returns are pervasive in functional languages
 - Allow return of several values, without boxing or mutation
- Multiple value returns affect all procedure application positions
 - Return points need to report reliable errors when they receive the wrong number of values

Multiple values with explicit check

- A standard approach uses a special value or flag to mark a return as multi-value; all return contexts are responsible for checking that value

```
afterCallSingle:  
  if multipleValuesReceived:  
    raise error "Expected single value ..."  
  ...  
...  
  
multipleValuesReceived = false  
GOTO returnAddress
```

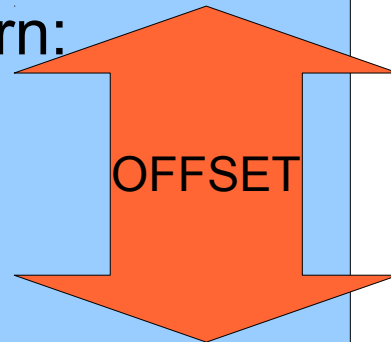
Multiple values with pointer arithmetic

onMultipleValueReturn:

...

afterCallSingle:

...



returnAddress = ...

GOTO (returnAddress - OFFSET)

Trick: implicit branch via address arithmetic [Ashley & Dybvig, 1994]

When returning multiple values, returners will jump to offset of regular return address.

onMultipleValueReturn:

...

afterCallSingle:

...

returnAddress = ...

GOTO (returnAddress -
OFFSET)

JS doesn't have pointer arithmetic...

... but functions are values...

```
var f = function() { ... };  
var g = function() { ... };
```

... and we can assign attributes into them!

```
f.successor = g;  
g.predecessor = f;
```

```
onMultipleValueReturn:
  ...
afterCallSingle:
  ...

returnAddress = ...
GOTO (returnAddress -
      OFFSET)
```

translates to:



```
var onMultipleValueReturn =
  function(MACHINE) {
    ...
  };

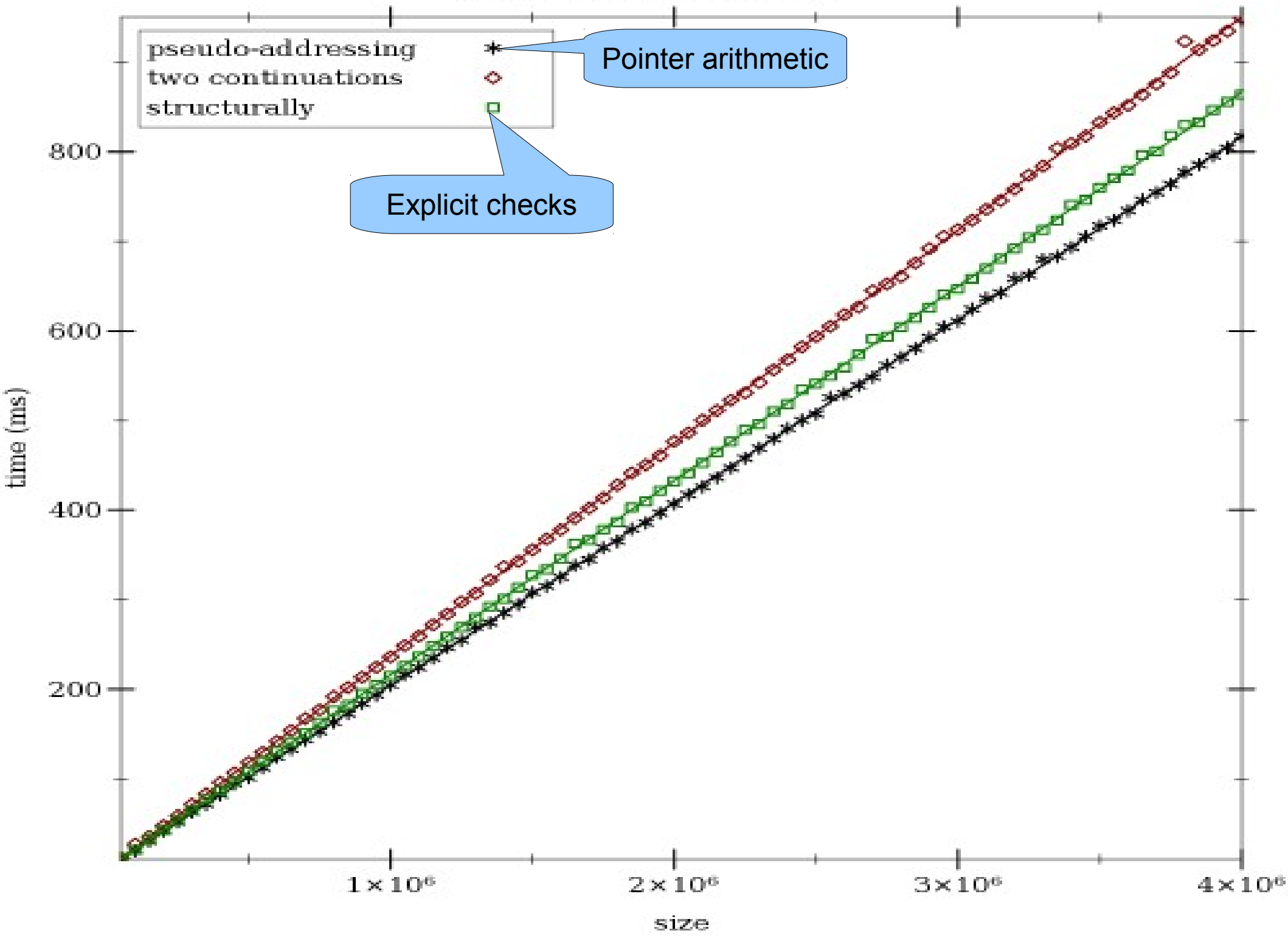
var afterCallSingle =
  function(MACHINE) {
    ...
  };
afterCallSingle.pred =
  onMultipleValueReturn;

...

var returnAddress = ...

return(returnAddress.pred(MACHINE));
```

mine-google-12-large.data



More details about the evaluator

Periodic, preemptive yielding of control to the browser after n function calls

- Allows for the implementation of interrupts
- Keeps the web browser responsive to input
- Avoids the stack ceiling for long-running computations
- Application of continuation marks on the control frames to produce good error messages with stack traces

Evaluation

www.wescheme.org/openEditor

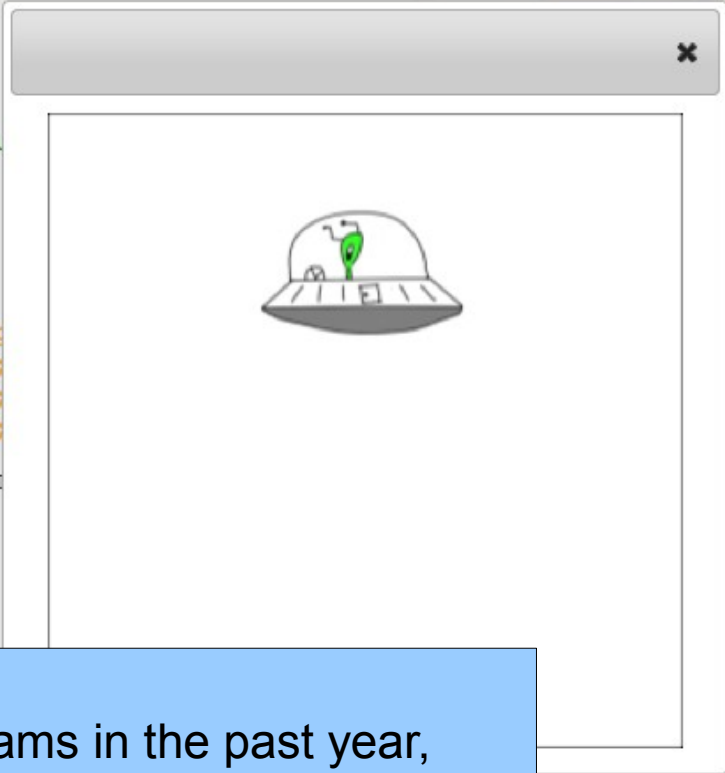
WeScheme

Sometimes YouTube. Perhaps iPhone. Together, WeScheme!

Run Stop Save Share Programs Logout API

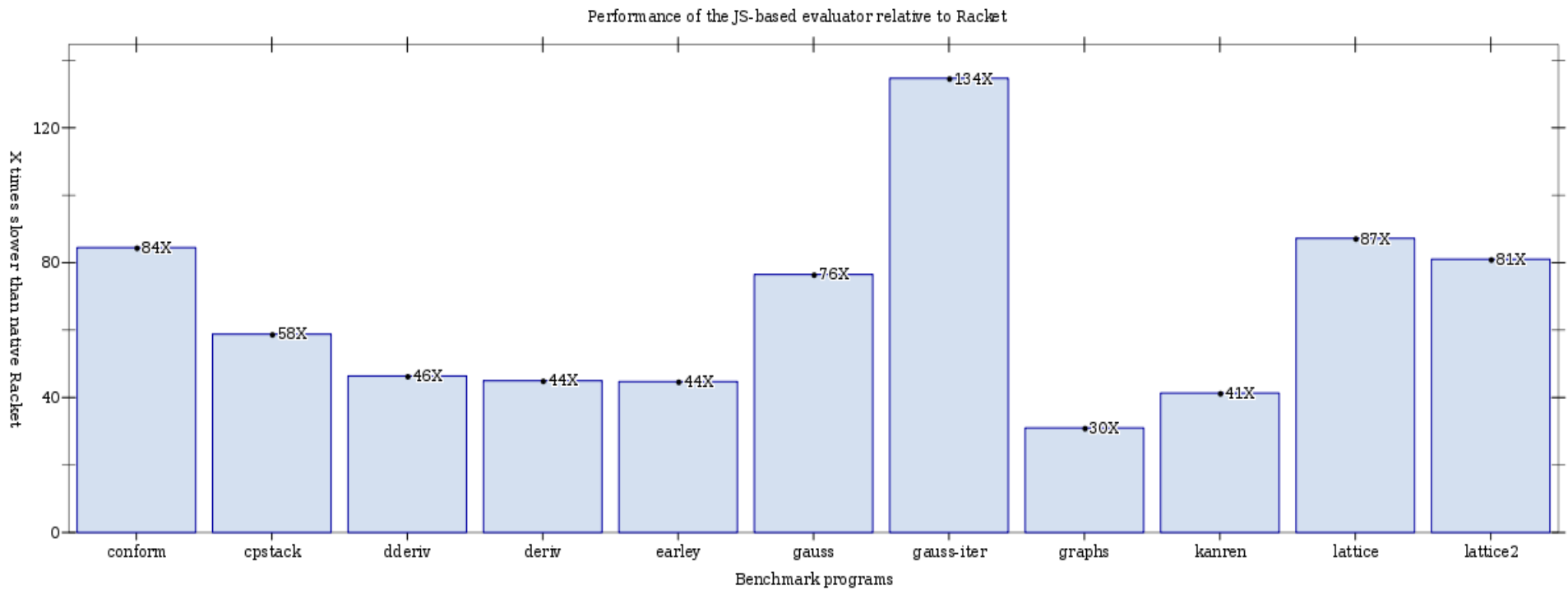
Project name: falling ufo

```
1 (define (descend w) (+ w 5))
2
3 (define (hits-floor? w) (> w 300))
4
5 (define UFO
6   (image-url "http://world.cs.brown.edu
7
8 (define (draw w)
9   (place-image UFO 150 w
10    (empty-scene 300 300)))
11
12 ;; The use of big-bang starts the world
13 (big-bang 0
14   (on-tick descend 1)
15   (on-redraw draw)
16   (stop-when hits-floor?))
```



~2000 programs in the past year,
from hundreds of users

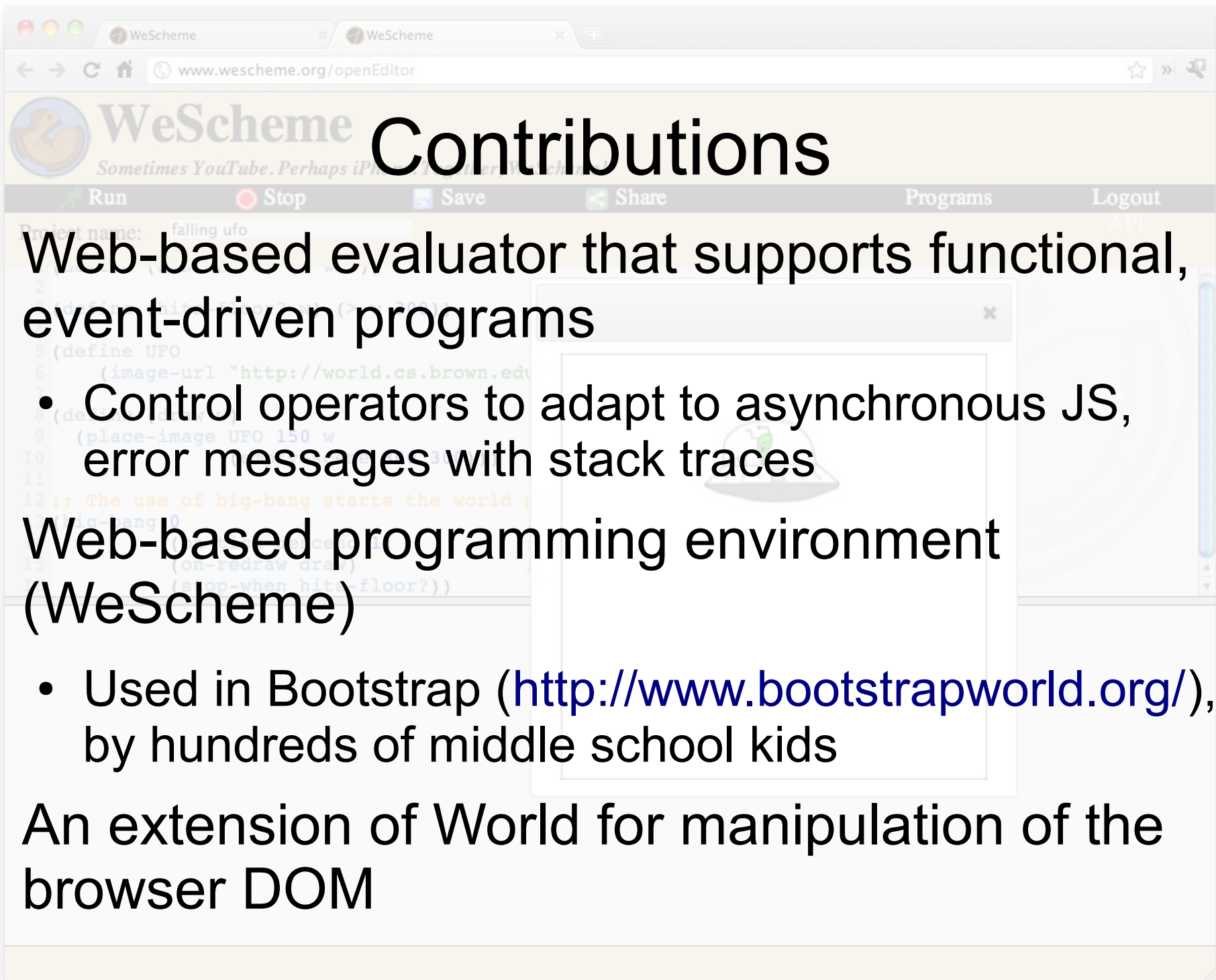
Performance vs. Racket JIT



Related work

Web evaluators and programming environments

- Server-side evaluators (Try Ruby, Try Haskell, ideone)
 - Textual output
 - Non-real-time interaction with the user
- Client-side evaluators (wscheme, HotRuby, Obrowser)
 - No direct support for control operators



WeScheme Contributions

- Web-based evaluator that supports functional, event-driven programs
 - Control operators to adapt to asynchronous JS, error messages with stack traces
- Web-based programming environment (WeScheme)
 - Used in Bootstrap (<http://www.bootstrapworld.org/>), by hundreds of middle school kids
- An extension of World for manipulation of the browser DOM

Future work

- Optimize the compiler and evaluator
 - Improve run-time, reduce code size
- Improve WeScheme to become a premiere programming environment for education
 - Enable embedding, develop tools for to support curricula
- Phone development
 - Automatically generate Android & iOS packages
 - Expose APIs for phone I/O

Thanks to...

- Very sincere thanks to my advisors: Kathi Fisler and Shriram Krishnamurthi
- WPI ALAS and Brown PLT
- Emmanuel Schanzer, Guillaume Marceau, Tim Nelson, Sam Tobin-Hochstadt, Jay McCarthy, Jens Axel Sogaard, Matthew Flatt, Robby Findler, John Clements, Scott Newman, Ethan Cecchetti, Zhe Zhang, Will Zimrin, ... too many to list!