# Lambda Calculus
# Functional Programing
# and


# Lambda Calculus
# Functional Programing
# and

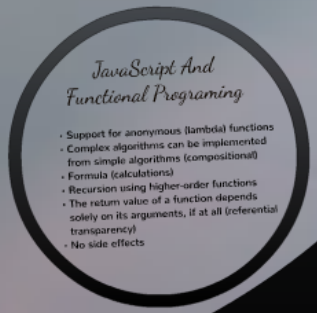


What is a Closure?
• A closure is the local variables for a function - kept alive after the function has returned.
• A function saves its context (referencing environment)
JavaScript and Closures
Cons
Group E -xcellent
Jesper Rage
Pritpaul Padda
Wesley Chow
Jason Curt
Jason Yu
Aahan Suneja
Dennis Honey
Yifang La
Andrew Li
John Cox
Prezi

# In the Beginning

18 years ago
1995
Brendan Eich
Lambda Calculus
Functional Programing





# Who is Brendan Eich?

- CTO of Mozilla Corporation
- Rust programming language
- Creator of JavaScript
- Helped found mozilla.org

# What is JavaScript

- Created to formalize the concept of computability
- Interpreted programming language
- Supports functions as arguments
- Prototyped-based scripting language
- Uses dynamic and duck typing
- Functional language that uses call-by-value
- Imperative
- Mostly Non Lazy
- But there are libraries that are implemented with lazy evaluation

# JavaScript And Functional Programing

- Support for anonymous (lambda) functions
- Complex algorithms can be implemented from simple algorithms (compositional)
- Formula (calculations)
- Recursion using higher-order functions
- The return value of a function depends solely on its arguments, if at all (referential transparency)
- No side effects

# JavaScript And Lazy Evaluation

- JavaScript is imperative,procedural, functional and call-by-value
- JavaScript does not have lazy Evaluation
- But now there are libraries that are implemented with lazy evaluation

## What is Lazy Evaluation?

- A policy that delays evaluation of expressions and only does them on demand
- In other words, expressions that are not necessary to compute in a function will not be computed
- Also known as normal order, delayed/deferred execution, call-by-need
- Found in languages like Haskell and Scheme
- Was introduced by Christopher Wadsworth in 1971

## Lazy implementation

## Pros / Cons of Being Lazy

Pros

Cons

# What is Lazy Evaluation?

- A policy that delays evaluation of expressions and only does them on demand
- In other words, expressions that are not necessary to compute in a function will not be computed
- Also known as normal order, delayed/deferred execution, call-by-need
- Found in languages like Haskell and Scheme
- Was introduced by Christopher Wadsworth in 1971

# Pros / Cons of Being Lazy

## Pros

- Can be used to reduce loading times on web pages
- Loading only necessary images and other images as needed
- The ability to construct potentially infinite data structures

## Cons

- Lots of overhead with lazy evaluation
- Improper use causes undefined behavior
- Must track object was previously referenced
- Difficult with imperative programming
- Exception handling and i/o - Exception handling is difficult to integrate with lazy evaluation

# Pros

- Can be used to reduce loading times on web pages
- Loading only necessary images and other images as needed
- The ability to construct potentially infinite data structures

# Cons

- Lots of overhead with lazy evaluation
- Improper use causes undefined behavior
- Must track object was previously referenced
- Difficult with imperative programming
- Exception handling and i/o - Exception handling is difficult to integrate with lazy evaluation

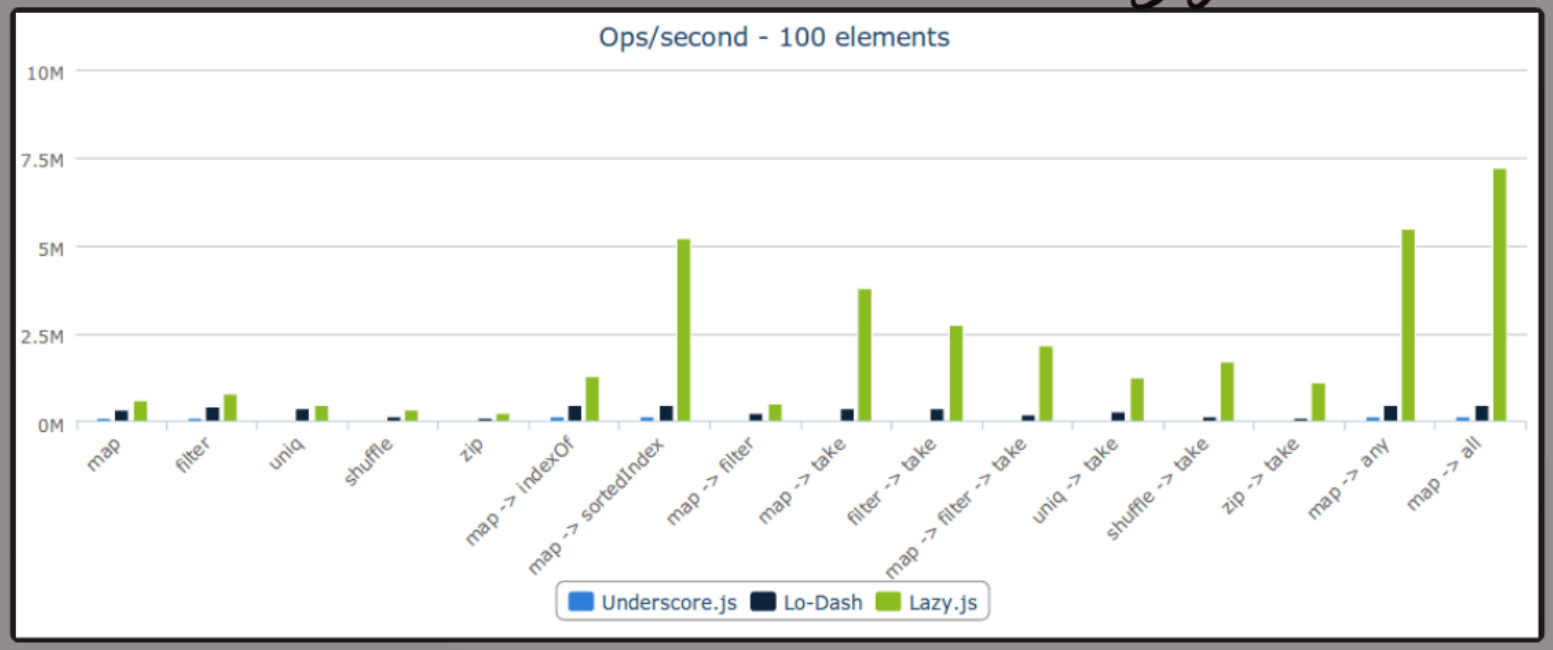
# Lazy implementation

stream.js
-Is a module that contains the data structure ‘Streams’ that is essentially an infinite list

lazy.js
-Is a module that is similar to Underscore, which is a framework for developing web applications.

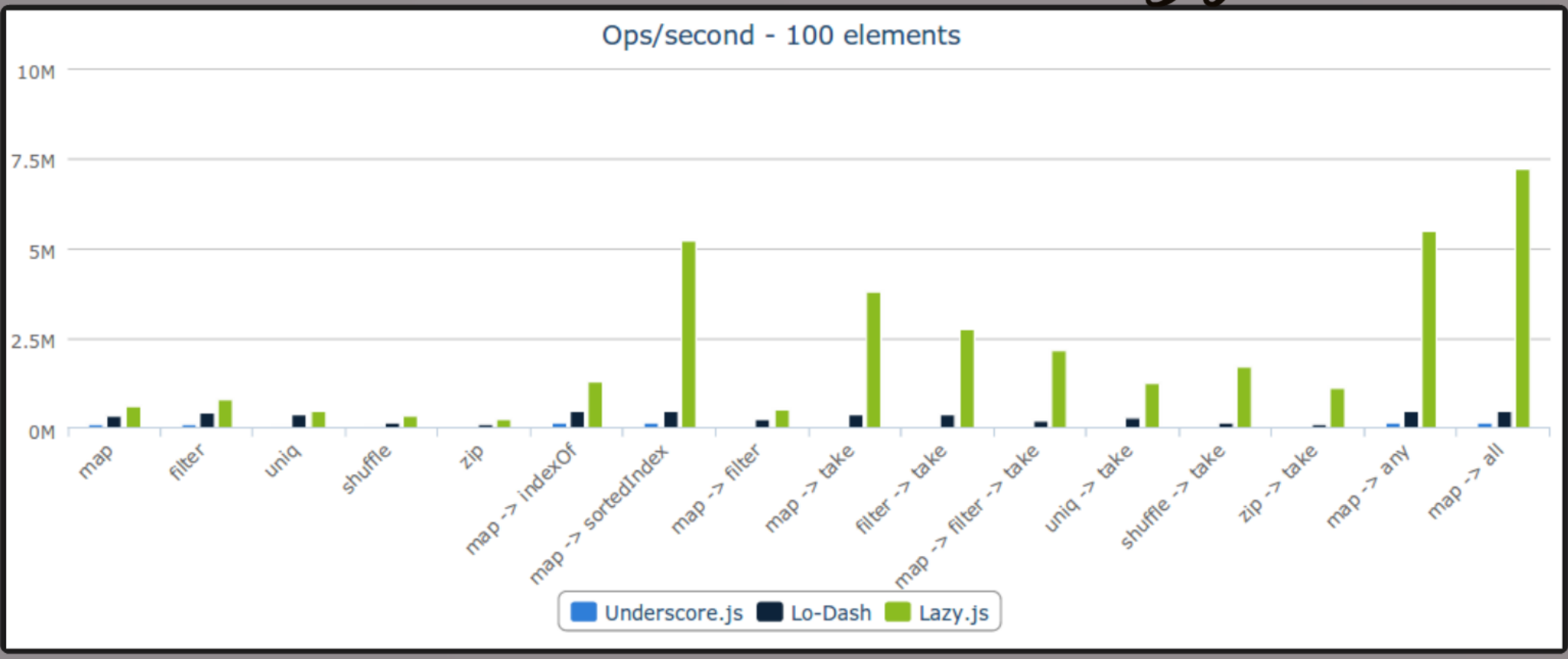
linq.js
-Implements all NET 4.0 methods and supports extensions

# Chart about Lazy




- Underscore, Lo-Dash, and Lazy.js implement very similar functions
- The number of operations per second do not differ between stand-alone functions
- However, the number of operations per second exponentially increases for Lazy.js when composition of functions are used in comparison to Underscore and Lo-Dash

# Chart about Lazy




- Underscore, Lo-Dash, and Lazy.js implement very similar functions
- The number of operations per second do not differ between stand-alone functions
- However, the number of operations per second exponentially increases for

# Functional Programing And JavaScript

## Haskell List Comprehension

- Math set-builder notation
- Intuitive and flexible

$$S = \{2 * x \mid x \in \mathbb{N}, x^2 > 3\}$$
$$S = \{2^*x \mid x \leftarrow [0 \ldots], x^2 > 3\}$$

## What is a Closure?

- A closure is the local variables for a function - kept alive after the function has returned.
- A function saves its context (referencing environment)

## What is a higher order Function?

A higher-order function is a first-class function which takes a function as an argument and returns a function as a result.

## JavaScript and Higher Order

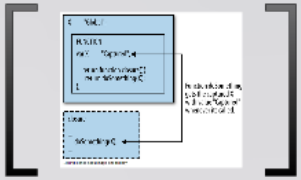
- JavaScript implemented map(), reduce() and filter() functions as a part of their Array class in JavaScript 1.6.
- Since there is no lazy evaluation in JavaScript, these operations are limited to finite arrays.

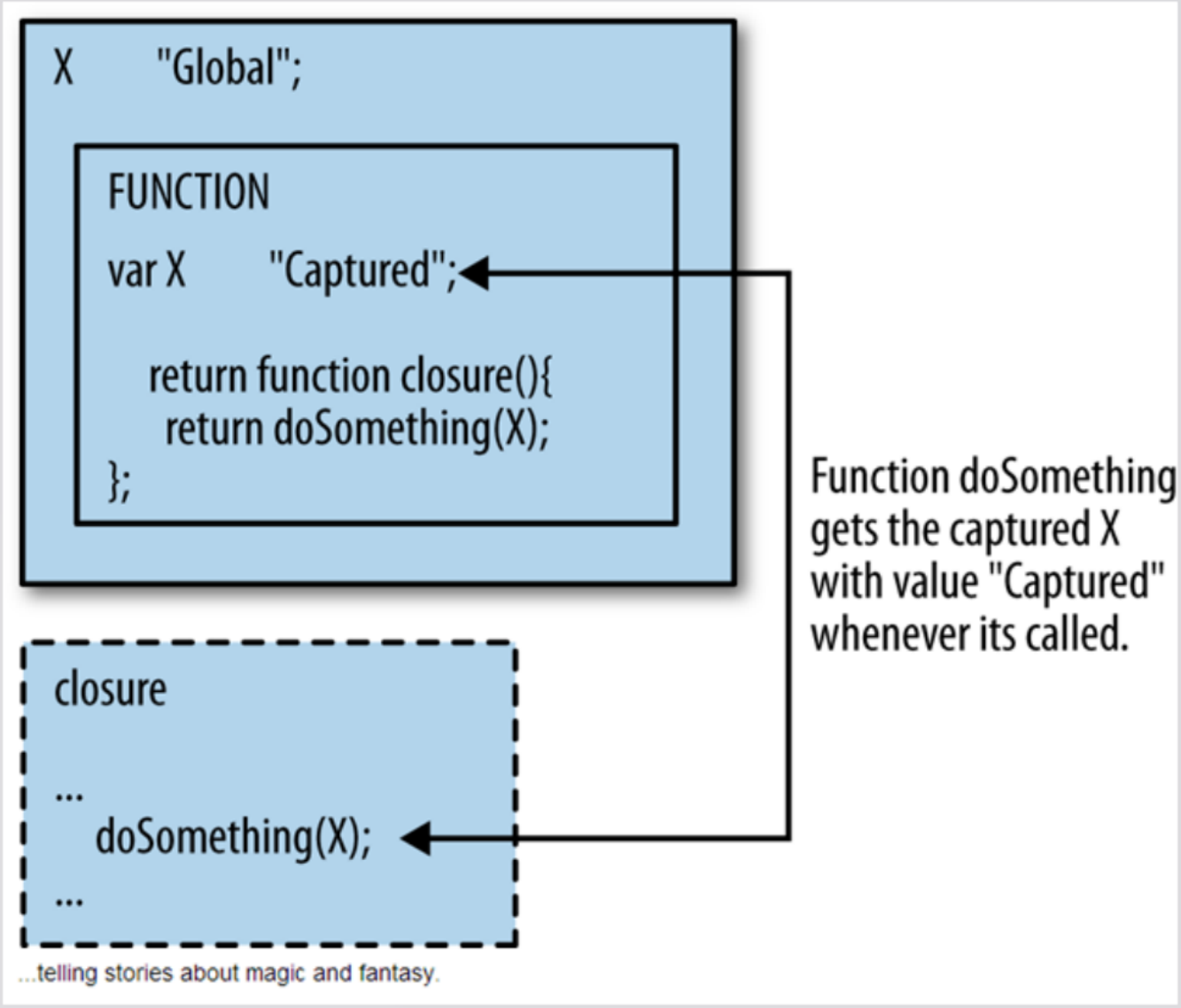
## JavaScript and Closures

- JavaScript creates an activation object for each scope
- Each object references all function arguments and variables within that scope
- The activation objects become part of a scope chain that JavaScript will traverse when trying to find a variable
- All a nested function has to do to create a closure is carry a reference to its activation object, which will keep all the variables and arguments in the scope chain alive
- Closures in JavaScript can also be used to implement the notion of private data. Let's exhibit this with a fairytale:

# What is a Closure?

- A closure is the local variables for a function - kept alive after the function has returned.
- A function saves its context (referencing environment)




X "Global";
FUNCTION
var X "Captured";
return function closure(){
return doSomething(X);
};
closure
...
doSomething(X);
...
Function doSomething gets the captured X with value "Captured" whenever its called.
...telling stories about magic and fantasy.

# JavaScript and Closures

- JavaScript creates an activation object for each scope
- Each object references all function arguments and variables within that scope
- The activation objects become part of a scope chain that JavaScript will traverse when trying to find a variable
- All a nested function has to do to create a closure is carry a reference to its activation object, which will keep all the variables and arguments in the scope chain alive
- Closures in JavaScript can also be used to implement the notion of private data. Let's exhibit this with a fairytale:

**Once upon a time:**

There was a princess...

```
function princess() {
```

She lived in a wonderful world full of adventures. She met her Prince Charming, rode around her world on a unicorn, battled dragons, encountered talking animals, and many other fantastical things.

```
    var adventures = [];

    function princeCharming() { /* ... */ }

    var unicorn = { /* ... */ },
        dragons = [ /* ... */ ],
        squirrel = "Hello!";
```

But she would always have to return back to her dull world of chores and grown-ups.

```
    return {
```

And she would often tell them of her latest amazing adventure as a princess.

```
        story: function() {
            return adventures[adventures.length - 1];
        }
    };
}
```

But all they would see is a little girl...

```
var littleGirl = princess();
```

...telling stories about magic and fantasy.

```
littleGirl.story();
```

And even though the grown-ups knew of real princesses, they would never believe in the unicorns or dragons because they could never see them. The grown-ups said that they only existed inside the little girl's imagination.

But we know the real truth; that the little girl with the princess inside...

...is really a princess with a little girl inside.

# What is a higher order Function?

A higher-order function is a first-class function which takes a function as an argument and returns a function as a result.

Java

- JavaScrip
  reduce() a
  part of th

# JavaScript and Higher Order

- JavaScript implemented map(), reduce() and filter() functions as a part of their Array class in JavaScript 1.6.
- Since there is no lazy evaluation in JavaScript, these operations are limited to finite arrays.

# Array and List Comprehension

## What is List / Array Comprehension?

- Creating a list from existing lists
- Syntactic Sugar
- Mathematical set-builder notation
- Well-understood and popular language feature
  - Found in Languages like Python, Haskell

## JavaScript Array Comprehension

- Pythonic generators along with iterators were first introduced in the 1.7 version of JavaScript
- A drawback of array comprehension is that they create an entire new array to be constructed in memory.

## Haskell List Comprehension

- Math set-builder notation
- Intuitive and flexible

S = {2 x| x  N, x2 > 3}
S = [2*x | x <- [0 . . ], x^2 > 3]

# What is List / Array Comprehension?

- Creating a list from existing lists
- Syntactic Sugar
- Mathematical set-builder notation
- Well-understood and popular language feature
    - Found in Languages like Python, Haskell

```
boomBangs xs = [ if x < 10 then "BOOM!" else "BANG!" | x <- xs, odd x]
```

The last part of the comprehension is the predicate. The function `odd` returns `True` on an odd number and `False` on an even one. The element is included in the list only if all the predicates evaluate to `True`.

```
ghci> boomBangs [7..13]
["BOOM!","BOOM!","BANG!","BANG!"]
```

# JavaScript Array Comprehension

- Pythonic generators along with iterators were first introduced in the 1.7 version of JavaScript
- A drawback of array comprehension is that they create an entire new array to be constructed in memory.

# Haskell List Comprehension

- Math set-builder notation
- Intuitive and flexible

S = {2  x| x   N, x2 > 3}
S = [2*x | x <- [0 . . ], x^2 > 3]

# Lambda Calculus
# Functional Programing
# and