ClojureScript Anatomy

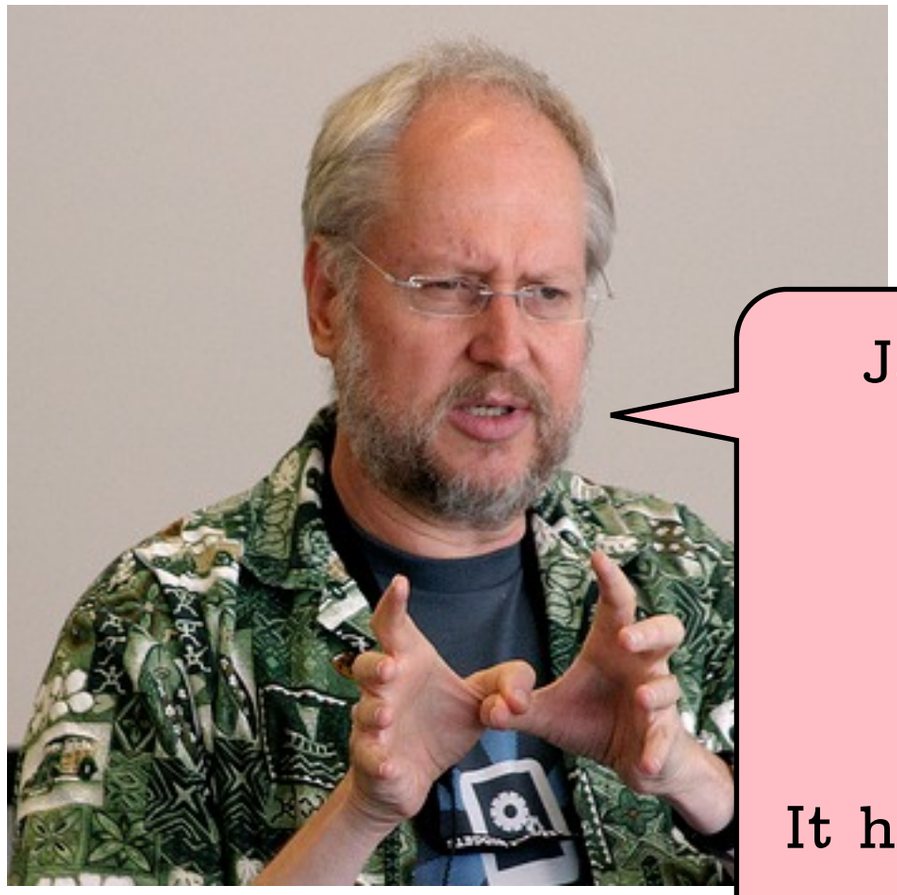

2
Mix TAPE
EQ-70μs
BIAS-HIGH
Made in Japan
RT-60XA
Noise
Reduction
IN
OUT
C-60
universe

TargetWhy?Script

Primacy of Semantics

I also regard syntactical problems as essentially irrelevant to programming languages at their present stage … the urgent task in programming languages is to explore the field of semantic possibilities.

— Christopher Strachey - “Fundamental Concepts in Programming Languages



JavaScript has much in common with Scheme.

It is a dynamic language.

It has a flexible datatype (arrays) that can easily simulate s-expressions.

And most importantly, functions are lambdas.

http://javascript.crockford.com/little.html

Power Points

- Reach
  - browser, shell, mobile, databases
- First-class functions
- Prototypal inheritance
- Closures
- Literal syntax

Pain Points

- Everything else

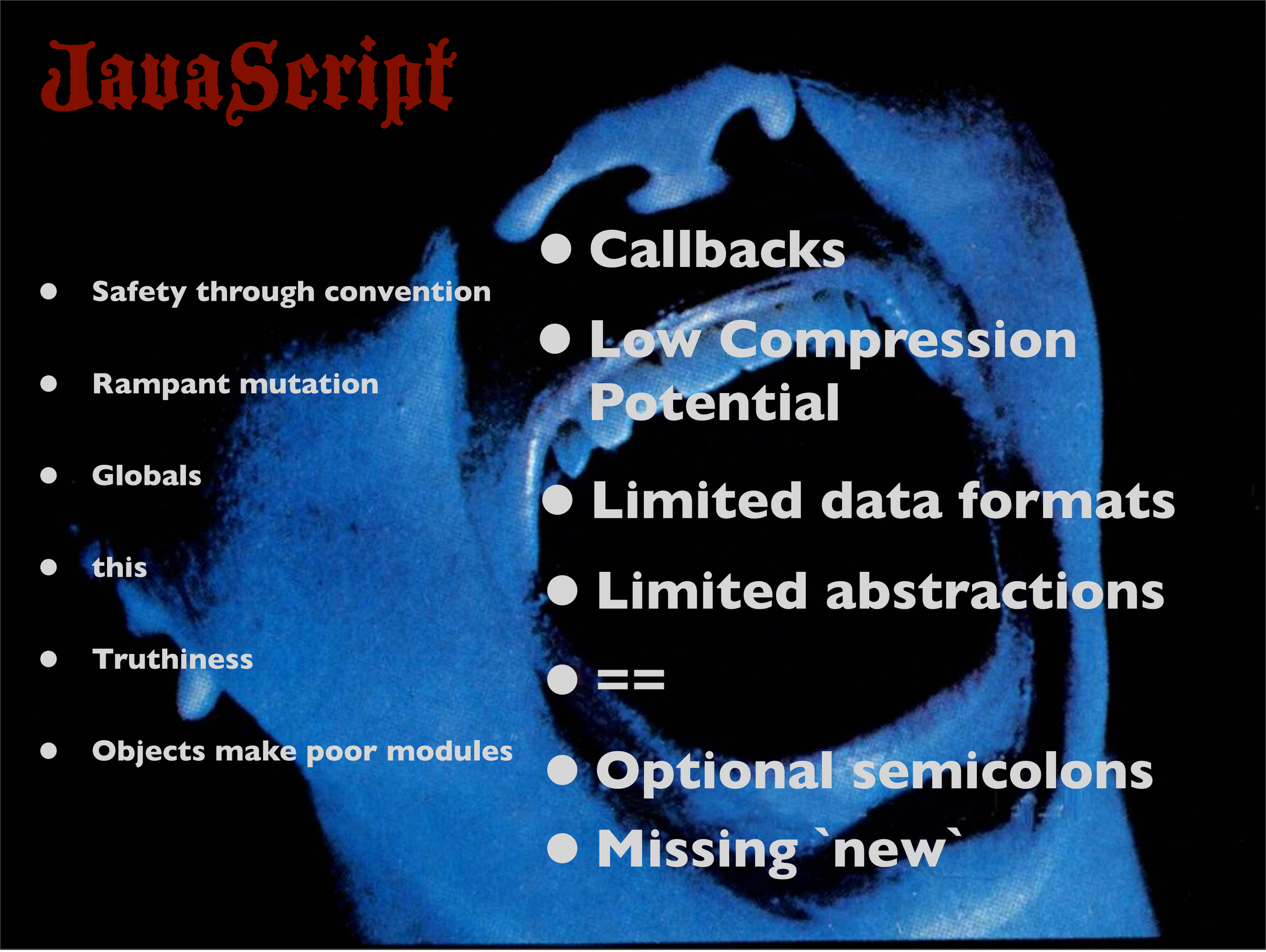
PLT HULK
@PLT_Hulk

JAVASCRIPT IS SCHEME IN THE BROWSER WITHOUT THE PARENTHESES OR THE SCHEME.

JavaScript

- Safety through convention
- Rampant mutation
- Globals
- this
- Truthiness
- Objects make poor modules

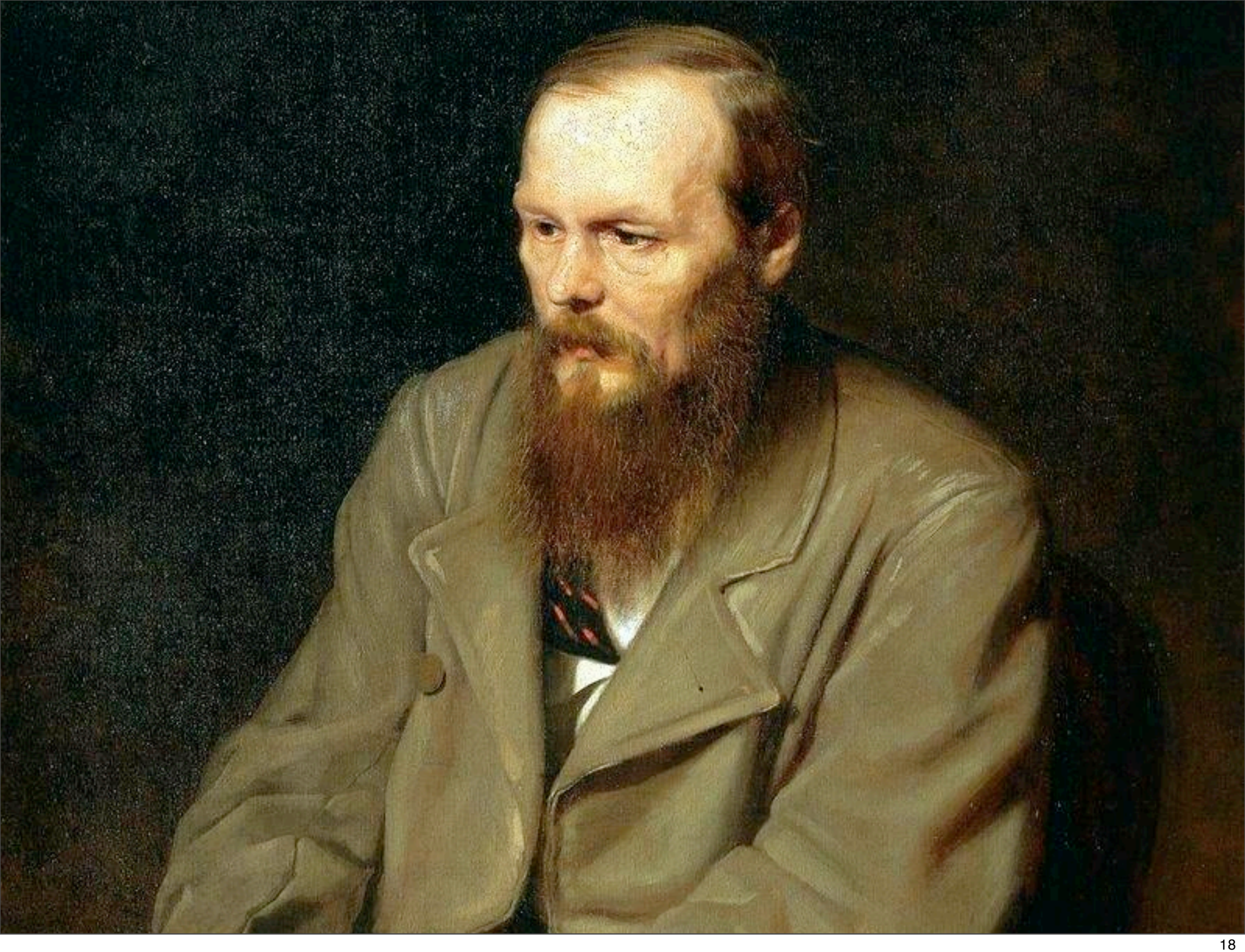
JavaScript



- Safety through convention
- Rampant mutation
- Globals
- this
- Truthiness
- Objects make poor modules

- Callbacks
- Low Compression Potential
- Limited data formats
- Limited abstractions
- ==
- Optional semicolons
- Missing `new`


OH, THE HUGE MANATEE!



```
my = my || {};
my.awesome = my.awesome || {};
my.awesome.lib = my.awesome.lib || {};

my.awesome.lib = (function(){
  var that = {};

  function init(){
    /* do something */
  };

  that.foo = true;
  that.init = init;

  return that;
}());

my.awesome.lib.frob = (function($){
  function privateFun(){
    /* do something */
  }

  return function() {
    /* do something with $ */
  };
}(baz));
```

```
my = my || {};
my.awesome = my.awesome || {};
my.awesome.lib = my.awesome.lib || {};

my.awesome.lib = (function(){
  var that = {};

  function init(){
    /* do something */
  };

  that.foo = true;
  that.init = init;

  return that;
}());

my.awesome.lib.frob = (function($){
  function privateFun(){
    /* do something */
  }

  return function() {
    /* do something with $ */
  };
}(baz));
```

```
my = my || {};
my.awesome = my.awesome || {};
my.awesome.lib = my.awesome.lib || {};

my.awesome.lib = (function(){
  var that = {};

  function init(){
    /* do something */
  };

  that.foo = true;
  that.init = init;

  return that;
}());

my.awesome.lib.frob = (function($){
  function privateFun(){
    /* do something */
  }

  return function() {
    /* do something with $ */
  };
}(baz));
```

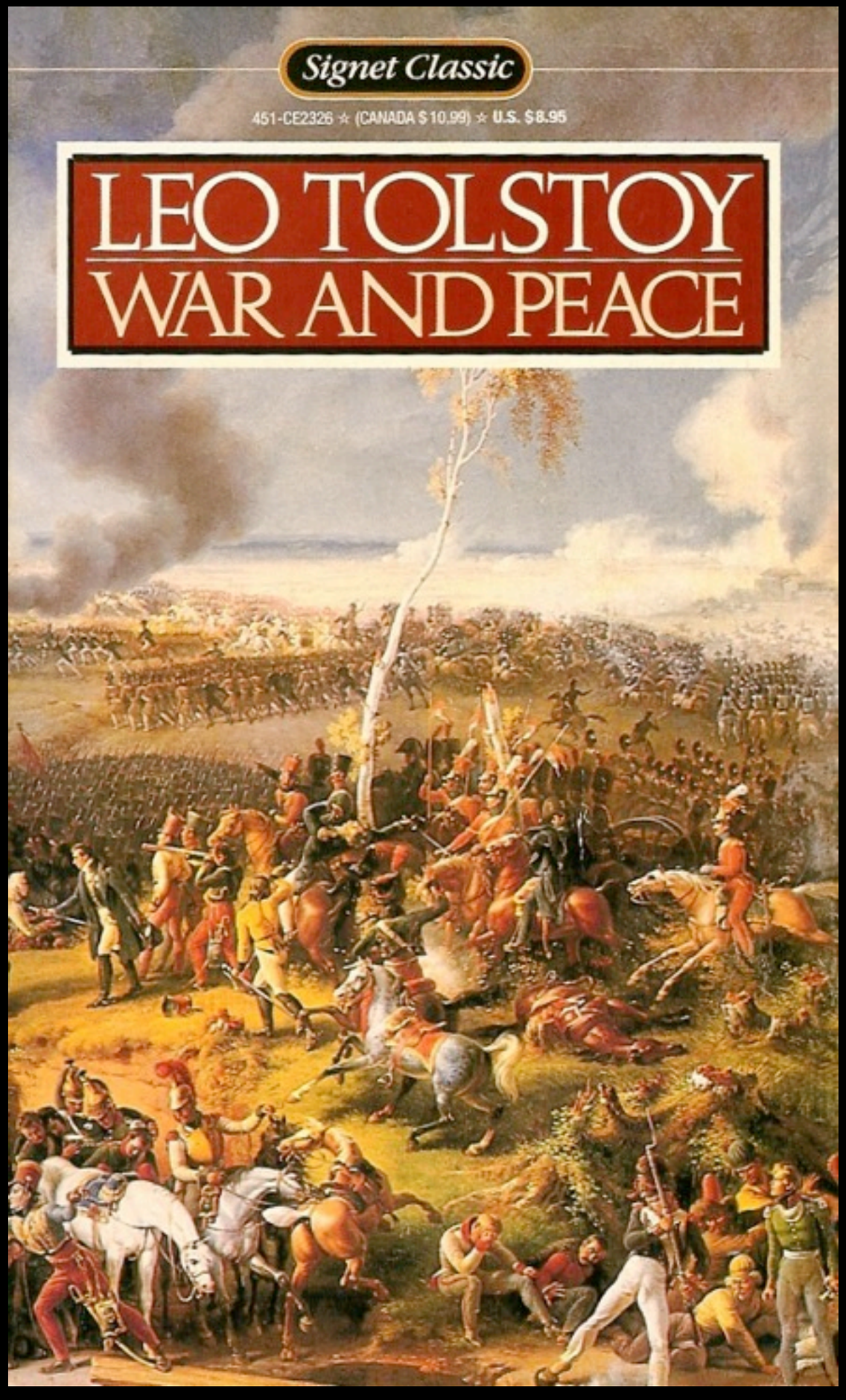
```
my = my || {};
my.awesome = my.awesome || {};
my.awesome.lib = my.awesome.lib || {};

my.awesome.lib = (function(){
  var that = {};

  function init(){
    /* do something */
  };

  that.foo = true;
  that.init = init;

  return that;
}());

my.awesome.lib.frob = (function($){
  function privateFun(){
    /* do something */
  }

  return function() {
    /* do something with $ */
  };
}(baz));
```

```
my = my || {};
my.awesome = my.awesome || {};
my.awesome.lib = my.awesome.lib || {};

my.awesome.lib = (function(){
  var that = {};

  var init = function(){
    /* do something */
  };

  that.foo = true;
  that.init = init;

  return that;
}());

my.awesome.lib.frob = (function($){
  function privateFun(){
    /* do something */
  }

  return function() {
    /* do something with $ */
  };
}(baz));
```

```
my = my || {};
my.awesome = my.awesome || {};
my.awesome.lib = my.awesome.lib || {};

my.awesome.lib = (function(){
  var that = {};

  var init = function(){
    /* do something */
  };

  that.foo = true;
  that.init = init;

  return that;
}());

my.awesome.lib.frob = (function($){
  function privateFun(){
    /* do something */
  }

  return function() {
    /* do something with $ */
  };
}(baz));
```

```
my = my || {};
my.awesome = my.awesome || {};
my.awesome.lib = my.awesome.lib || {};

my.awesome.lib = (function(){
  var that = {};

  var init = function(){
    /* do something */
  };

  that.foo = true;
  that.init = init;

  return that;
}());

my.awesome.lib.frob = (function($){
  function privateFun(){
    /* do something */
  }

  return function() {
    /* do something with $ */
  };
}(baz));
```

```
my = my || {};
my.awesome = my.awesome || {};
my.awesome.lib = my.awesome.lib || {};

my.awesome.lib = (function(){
  var that = {};

  var init = function(){
    /* do something */
  };

  that.foo = true;
  that.init = init;

  return that;
}());

my.awesome.lib.frob = (function($){
  function privateFun(){
    /* do something */
  }

  return function() {
    /* do something with $ */
  };
}(baz));
```

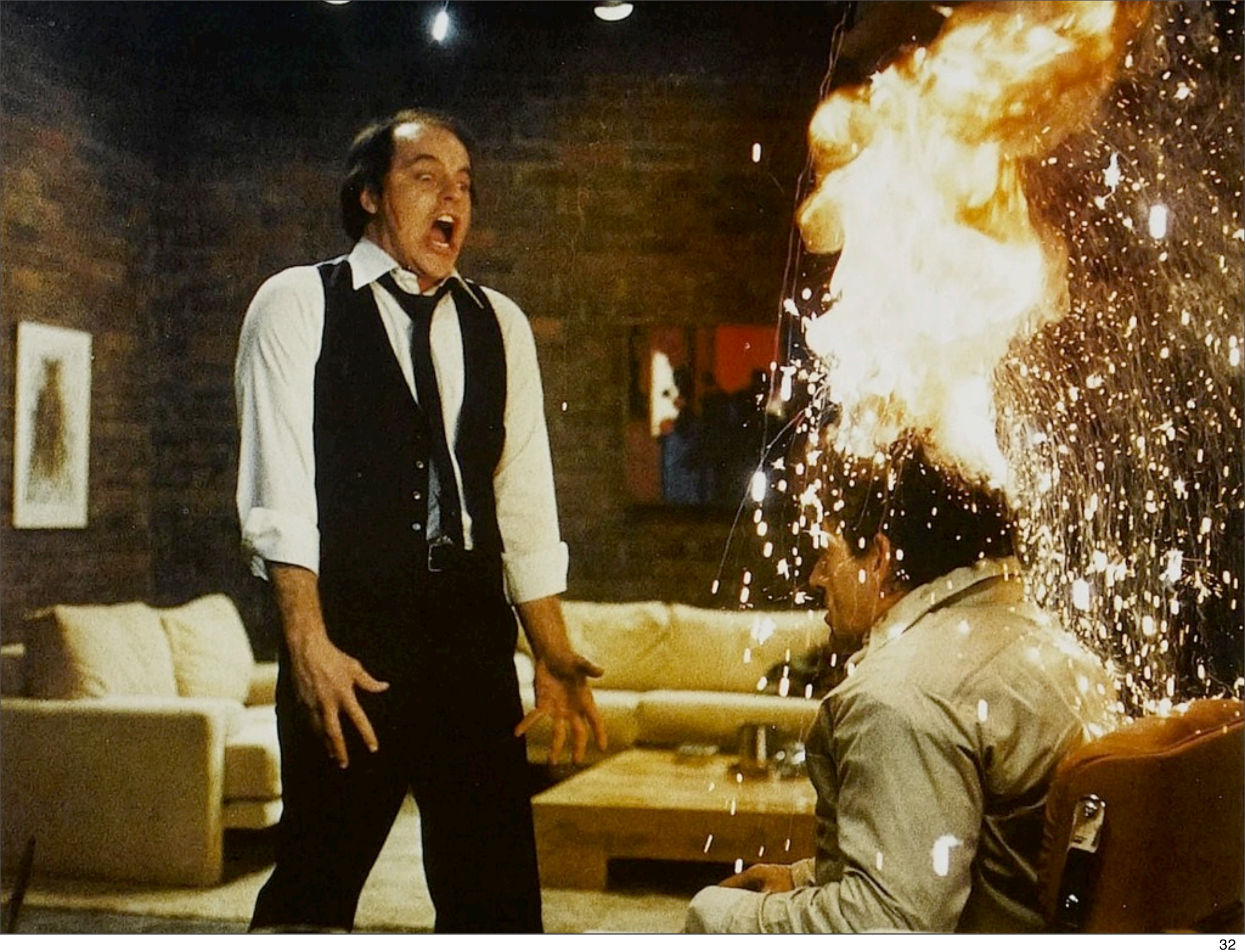

Signet Classic
451-CE2326 ☆ (CANADA $10.99) ☆ U.S. $8.95
LEO TOLSTOY
WAR AND PEACE

Google Closure

- Standard library
- Dependency system
- Compiler/minifier (GClosure)
- Templating language
- 1 kajillion man-hours of effort

Google Closure

- Standard library
- Dependency system
- Compiler/minifier (GClosure)
- Templating language
- 1 beellion man-hours of effort




by xrq


convention
convention
convention
convention
convention

Primacy of Syntax

one could say that all semantics is being represented as syntax
... semantics has vanished entirely to be replaced with pure syntax.
— John Shutt - “Primacy of Syntax”

CoffeeScript

the web be hostile

Namespaces

ECMAScript.Next

JSModule

nodules

dojo

modules.js

curl.js

Namespaces

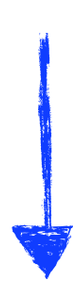
```
fogus.my.lib.frobnicate();
```

Namespaces

```
fogus = fogus || {};

fogus.my = fogus.my || {};

fogus.my.lib = fogus.my.lib || {};
```

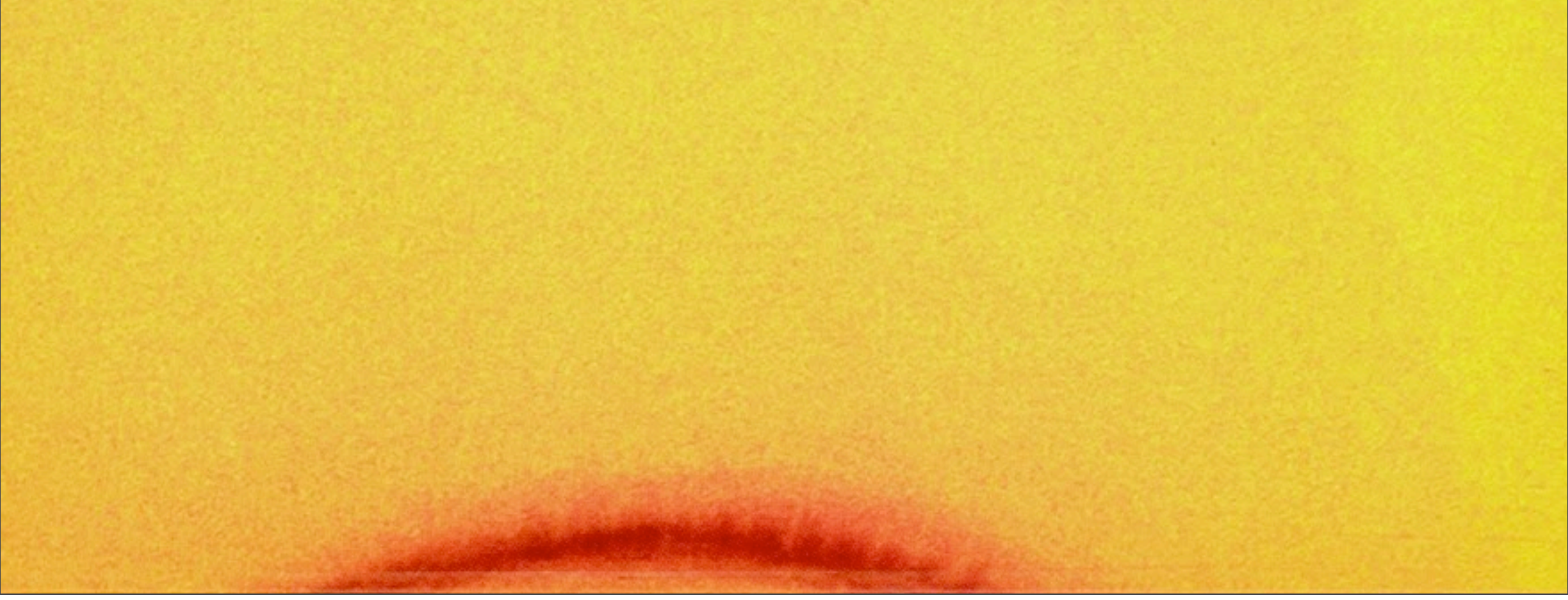
Namespaces

```
(ns fogus.my.lib)

(defn frobnicate [] :frobbed!)
```



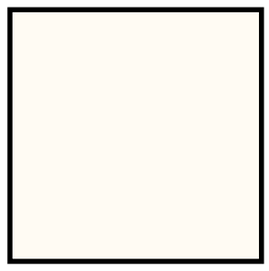
```
goog.provide('fogus.my.lib');

fogus.my.lib = function() {
  return "□ :frobbed!"
};
```

Compression Potential

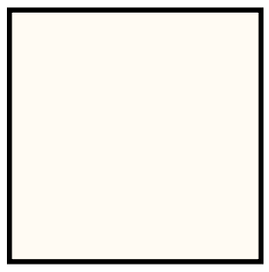


Take your pick

```
(ns fogus.my.lib)
```



```
goog.provide('fogus.my.lib');
```

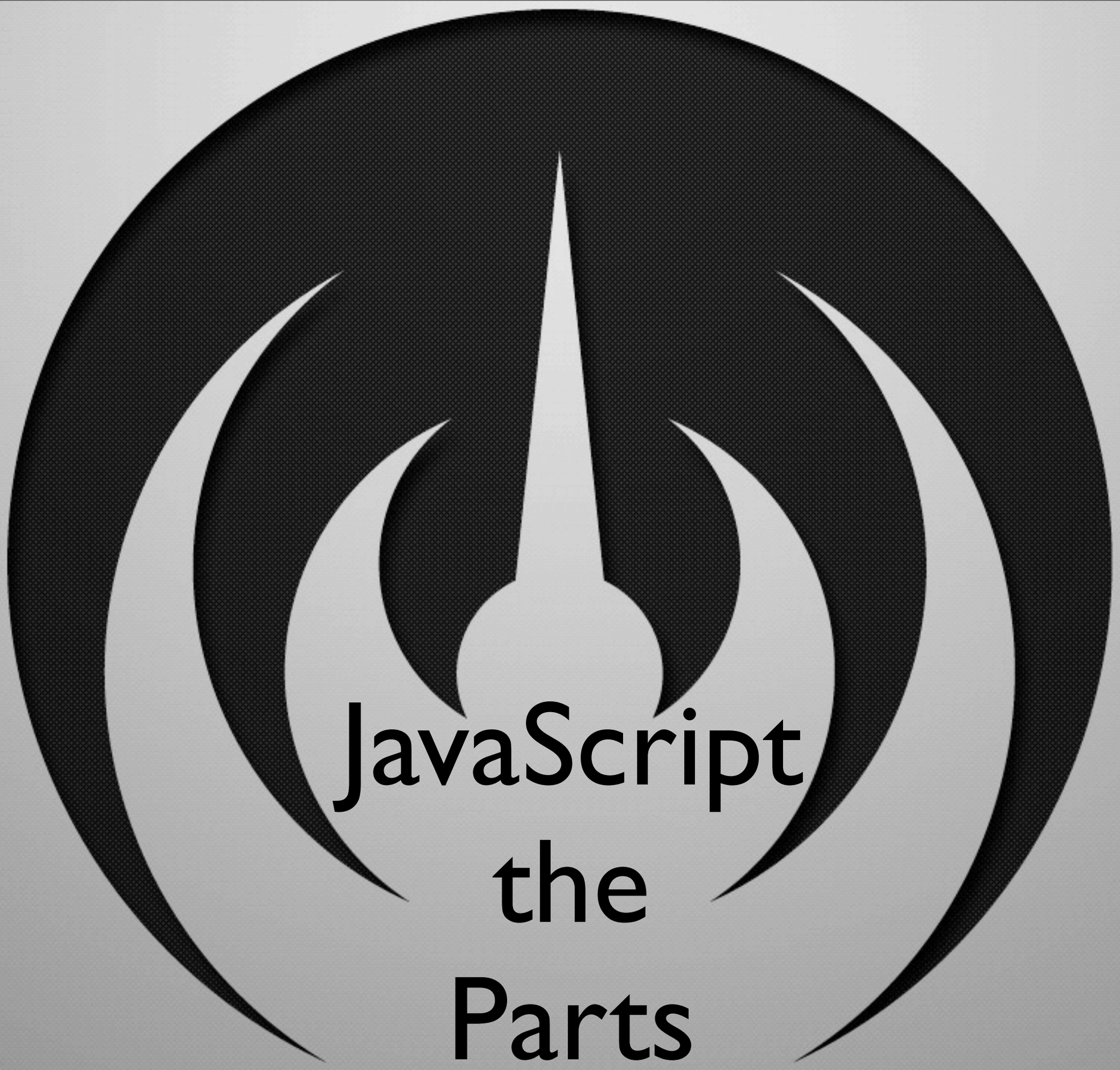


```
fogus = fogus || {};

fogus.my = fogus.my || {};

fogus.my.lib = fogus.my.lib || {};
```

Abstractions

Abstractions

- Protocols at the bottom
  - `ISeq`, `IMap`, `ILookup`, `IDeref`, etc.
- Realizing powerful immutable types
  - `Map`, `List`, `Vector`, etc.
- Extended to JavaScript
  - strings, null, arrays, etc.



JavaScript the Parts

Functional Inheritance Pattern

- Private data
- Access to super methods
- Missing `new` safe

Functional Inheritance Pattern

```
var person = function(spec) {
  var that = {};

  that.getName = function() {
    return spec.name;
  };
  return that;
};

var man = function(spec) {
  var that = person(spec);
  var super_getName = that.superior('getName');

  that.getName = function() {
    return "Mr. " + super_getName();
  };
  return that;
};
```

```
var me = person({'name' : 'Fogus'});
me.getName()
//=> 'Fogus'
```

```
var me = man({'name' : 'Fogus'});
me.getName()
//=> 'Mr. Fogus'
```

Functional Inheritance Pattern Dangers

- Harder to minify
- Larger memory footprint
- Cannot update instances
- Eliminates `instanceof`
- Mucks with `Function.prototype` and `Object.prototype` in an unsafe way

Pseudo-classical Inheritance Pattern

```
Person = function(name) {
  this._name = name;
};

Person.prototype.getName = function() {
  return this._name;
};


Man = function(name) {
  Person.call(this, name);
};

inherits(Man, Person);

Man.prototype.getName = function() {
  return "Mr. "
         + Man._super.getName.call(this);
};
```

```
var me = new Person('Fogus');
me.getName()
//=> 'Fogus'
```

```
var me = new Man('Fogus');
me.getName()
//=> 'Mr. Fogus'
```

Pseudo-classical Inheritance Pattern

- ~~Harder~~ Easier to minify
- ~~Larger~~ Smaller memory footprint
- ~~Cannot~~ Can update instances
- ~~Eliminates~~ `instanceof`
- ~~Mucks with~~ Clean `Function.prototype` and `Object.prototype`

Safe Extension

```
String.prototype.cljs$core$first = ...

Object.prototype.my$lib$first = ...
```

me._name
//=> ‘Fogus’

Minification

- Inlining
- Renaming
- Dead-code elimination

Inlining

Functional Inheritance Inlining

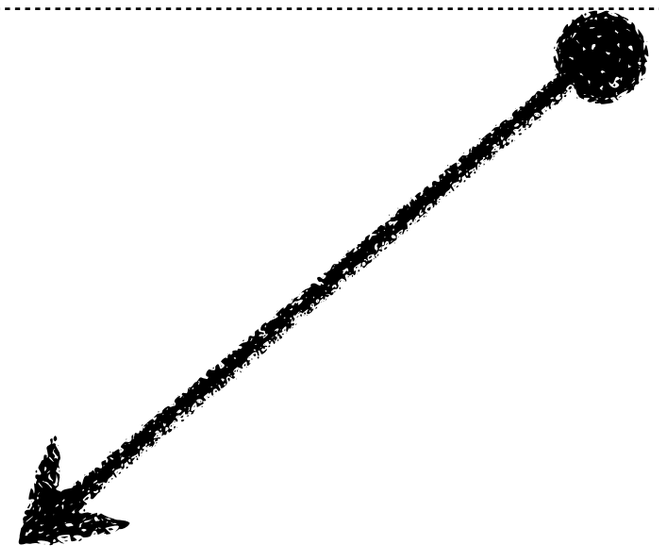
```
var person = function(spec) {
  var that = {};

  that.getName = function() {
    return spec.name;
  };
  return that;
};
```

```
var myName = me.getName();
var sig = signature({'name' : myName});
sig.toString();
//=> 'Sent from teh Fogus iPhone'
```

Functional Inheritance Inlining

```
var person = function(spec) {
  var that = {};

  that.getName = function() {
    return spec.name;
  };
  return that;
};
```

```
var myName = me.getName();
var sig = signature({'name' : myName});
sig.toString();
//=> 'Sent from teh Fogus iPhone'
```

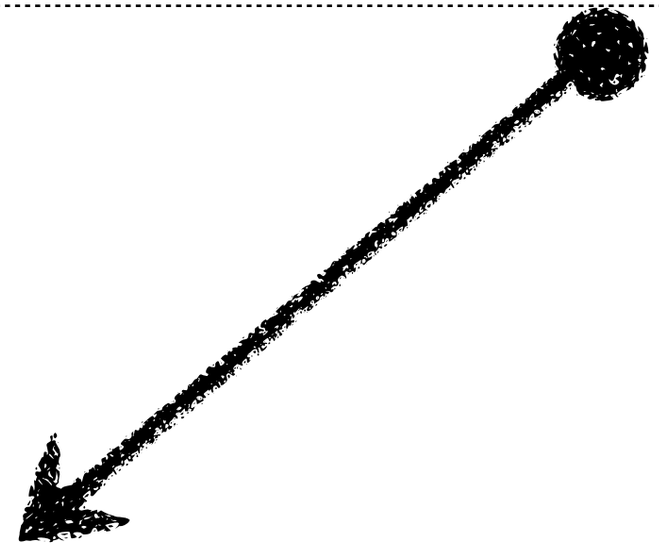


?

Functional Inheritance Inlining

```
var person = function(spec) {
  var that = {};

  that.getName = function() {
    return spec.name;
  };
  return that;
};
```

```
var myName = me.getName();
var sig = signature({'name' : myName});
sig.toString();
//=> 'Sent from teh Fogus iPhone'
```



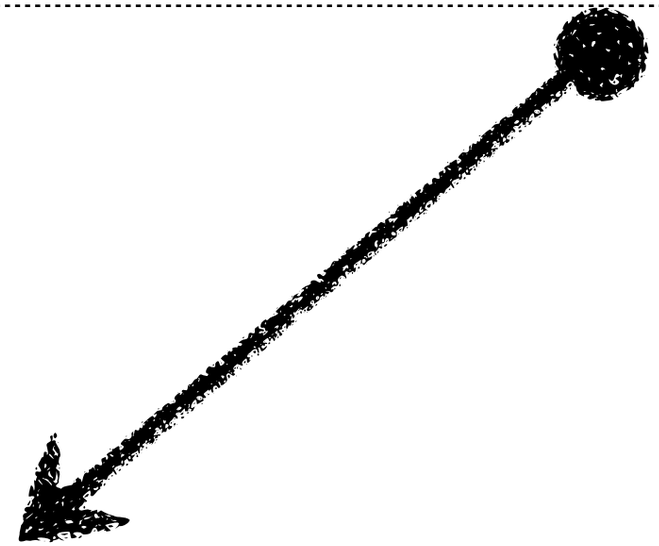
```
var sig = signature({'name' : me.getName()});
sig.toString();
//=> 'Sent from teh Fogus iPhone'
```

Inlining

Pseudo-classical Inlining

```
Person = function(name) {
  this._name = name;
};

Person.prototype.getName = function() {
  return this._name;
};
```

```
var myName = me.getName();
var sig = new Signature(myName);
sig.toString();
//=> 'Sent from teh Fogus iPhone'
```

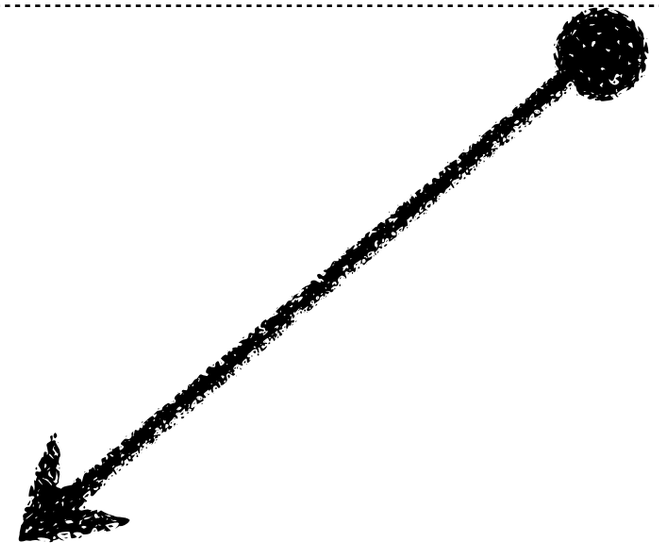


?

Pseudo-classical Inlining

```
Person = function(name) {
  this._name = name;
};

Person.prototype.getName = function() {
  return this._name;
};
```

```
var myName = me.getName();
var sig = new Signature(myName);
sig.toString();
//=> 'Sent from teh Fogus iPhone'
```



```
var sig = new Signature(me._name);
sig.toString();
//=> 'Sent from teh Fogus iPhone'
```

Renaming

Functional Inheritance Renaming

```
var person = function(spec) {
  var that = {};

  that.getName = function() {
    return spec.name;
  };
  return that;
};

var man = function(spec) {
  var that = person(spec);
  var super_getName = that.superior('getName');

  that.getName = function() {
    return "Mr. " + super_getName();
  };
  return that;
};
```

Functional Inheritance Renaming

```
var person = function(spec) {
  var that = {};

  that.getName = function() {
    return spec.name;
  };
  return that;
};

var man = function(spec) {
  var that = person(spec);
  var super_getName = that.superior('getName');

  that.getName = function() {
    return "Mr. " + super_getName();
  };
  return that;
};
```

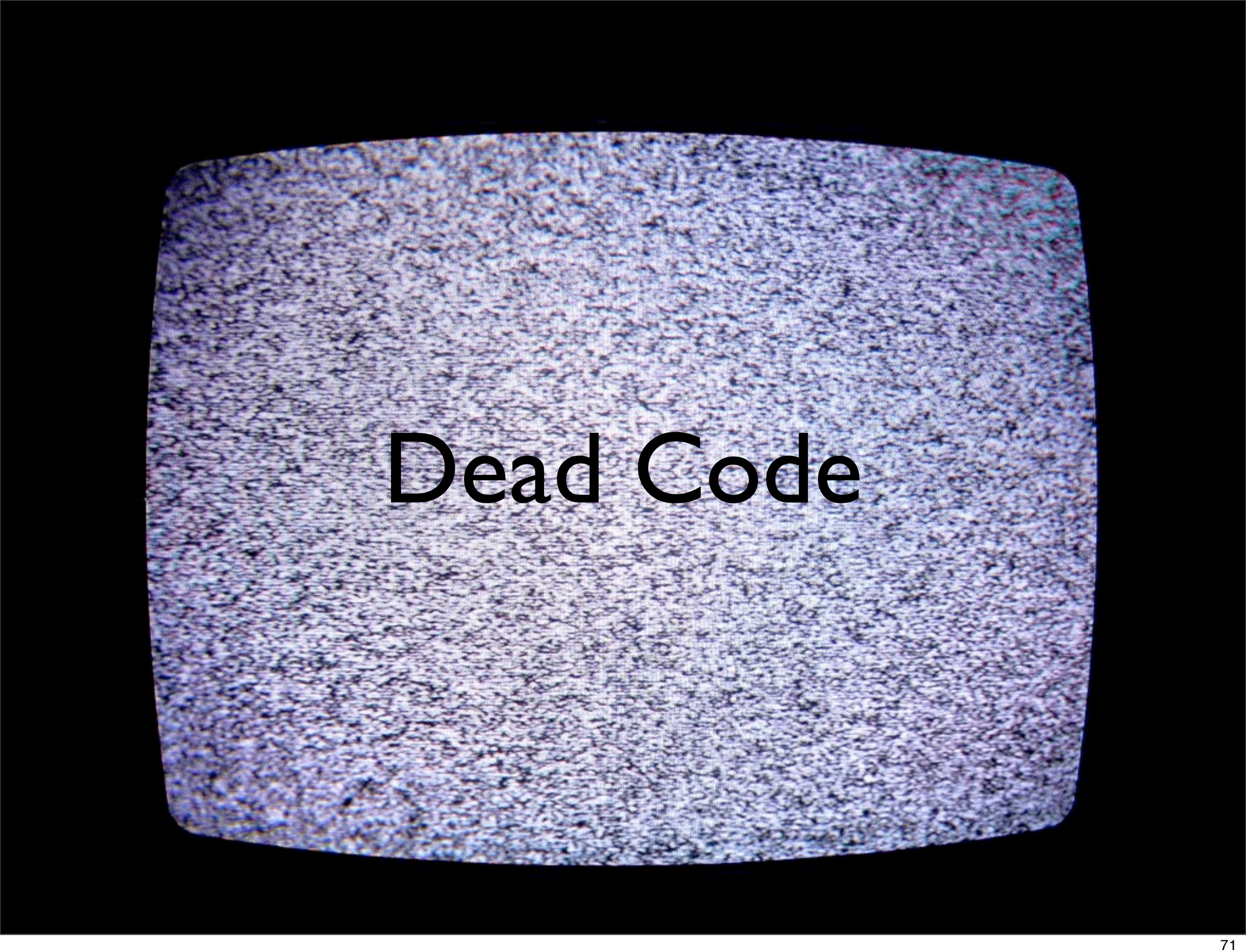

Renaming

Pseudo-classical Renaming

```
Person = function(name) {
  this._name = name;
};

Person.prototype.getName = function() {
  return this._name;
};


Man = function(name) {
  Person.call(this, name);
};

inherits(Man, Person);

Man.prototype.getName = function() {
  return "Mr. "
         + Man._super.getName.call(this);
};
```

Pseudo-classical Renaming

```
Person = function(name) {
  this._name = name;
};

Person.prototype.o = function() {
  return this._name;
};


Man = function(name) {
  Person.call(this, name);
};

inherits(Man, Person);

Man.prototype.getName = function() {
  return "Mr. "
         + Man._super.o.call(this);
};
```

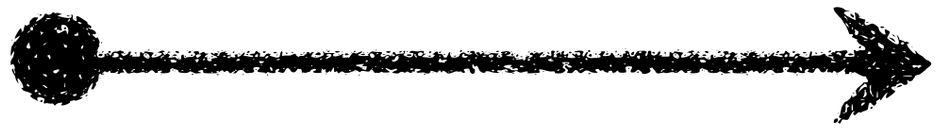

Dead Code

Dead-code Elimination Intro

```
(let [a 1
      b 2
      a b]
  a)

;=> 2
```

Dead-code Elimination Intro

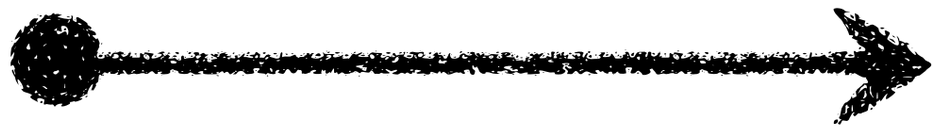
```
(let [a 1
      b 2
      a b]
  a)

;=> 2
```



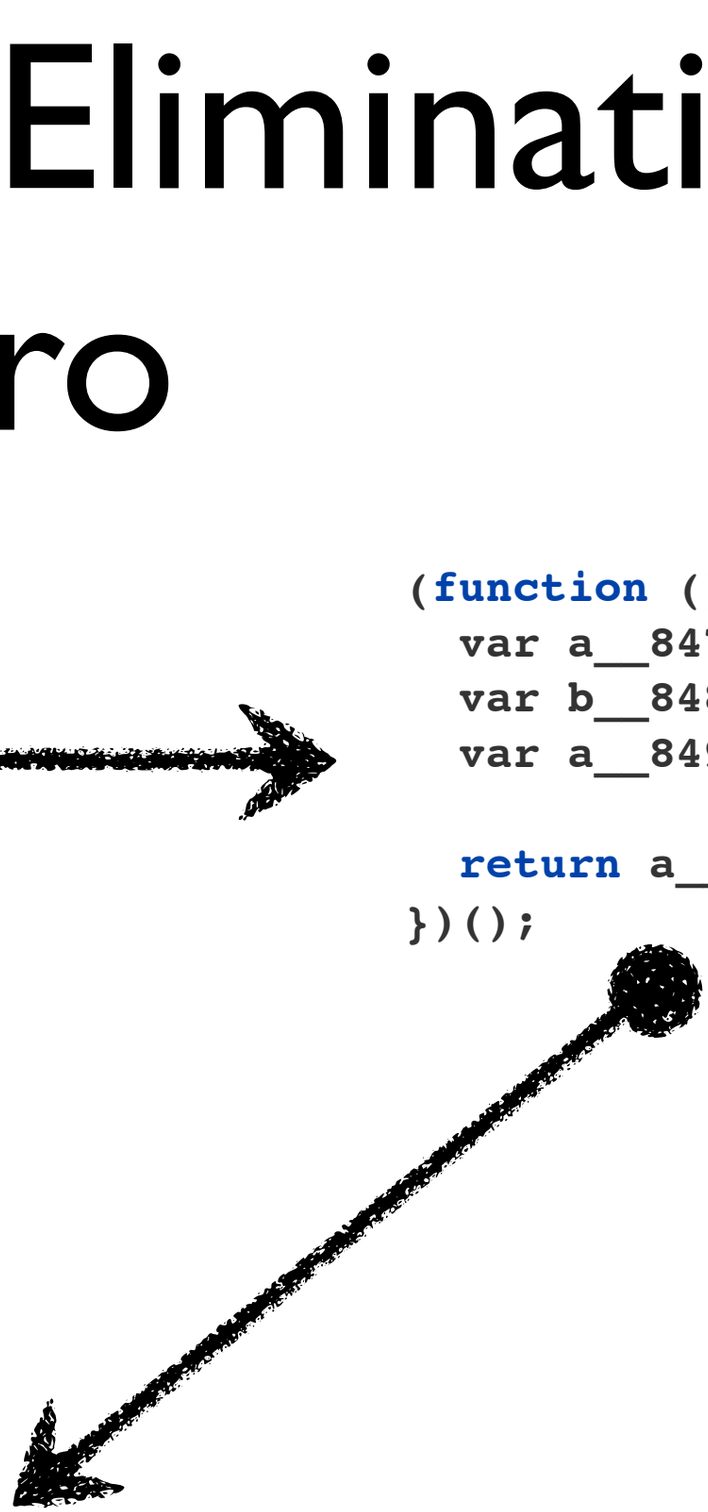
```
(function (){
  var a__847 = 1;
  var b__848 = 2;
  var a__849 = b__848;

  return a__849;
})();
```

Dead-code Elimination Intro

```
(let [a 1
      b 2
      a b]
  a)

;=> 2
```



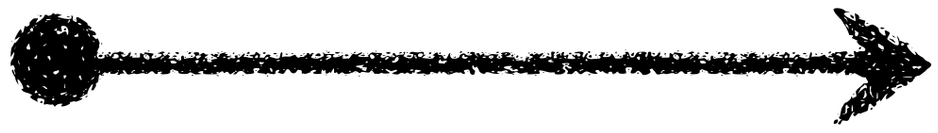
```
(function (){
  var a__847 = 1;
  var b__848 = 2;
  var a__849 = b__848;

  return a__849;
})();
```



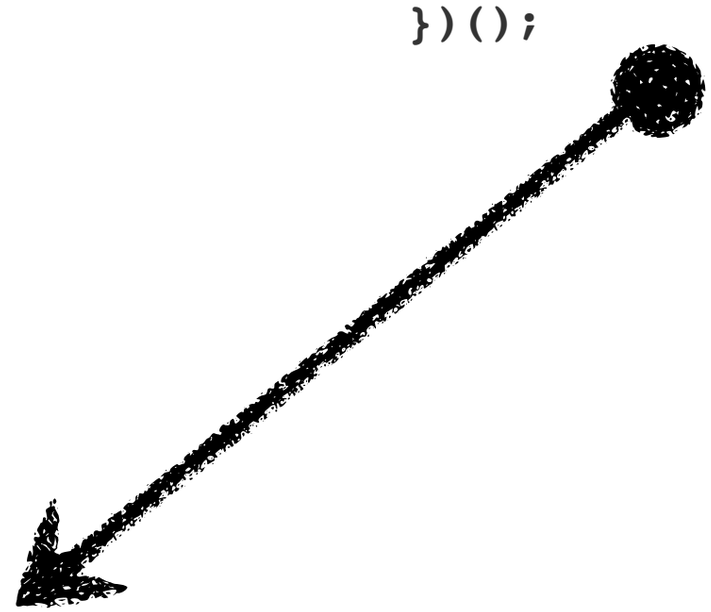
?

Dead-code Elimination Intro

```
(let [a 1
      b 2
      a b]
  a)

;=> 2
```



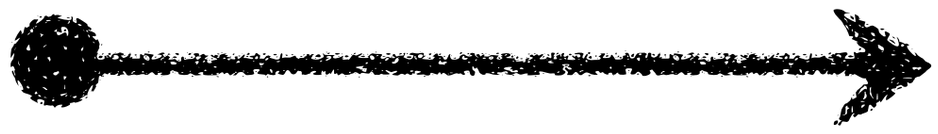
```
(function (){
  var a__847 = 1;
  var b__848 = 2;
  var a__849 = b__848;

  return a__849;
})();
```



2

Dead-code Elimination Intro

```
(let [a 1
      b 2
      a b]
  a)

;=> 2
```



```
(function (){
  var a__847 = 1;
  var b__848 = 2;
  var a__849 = b__848;

  return a__849;
})();
```

Functional Inheritance Dead-code

```
var person = function(spec) {
  var that = {};
  var hasBeard = false;

  that.getName = function() {
    return spec.name;
  };
  return that;
};

var man = function(spec) {
  var that = person(spec);
  var super_getName = that.superior('getName');

  that.getName = function() {
    return "Mr. " + super_getName();
  };
  return that;
};
```

Functional Inheritance Dead-code

```
var person = function(spec) {
  var that = {};
  var hasBeard = false;

  that.getName = function() {
    return spec.name;
  };
  return that;
};

var man = function(spec) {
  var that = person(spec);
  var super_getName = that.superior('getName');

  that.getName = function() {
    return "Mr. " + super_getName();
  };
  return that;
};
```

All good Right?

Functional Inheritance Dead-code

```
var person = function(spec) {
  var that = {};
  var hasBeard = false;

  that.getName = function() {
    return spec.name;
  };
  return that;
};

var man = function(spec) {
  var that = person(spec);
  var super_getName = that.superior('getName');

  that.getName = function() {
    return "Mr. " + super_getName();
  };
  return that;
};
```

Cursed Closures!


Dead Code

Pseudo-classical
Dead-code

```
Person = function(name) {
  this._name = name;
  this._hasBeard = false;
};

Person.prototype.getName = function() {
  return this._name;
};


Man = function(name) {
  Person.call(this, name);
};

inherits(Man, Person);

Man.prototype.getName = function() {
  return "Mr. "
         + Man._super.getName.call(this);
};
```

Pseudo-classical Dead-code

```
Person = function(name) {
  this._name = name;
  this._hasBeard = false;
};

Person.prototype.getName = function() {
  return this._name;
};


Man = function(name) {
  Person.call(this, name);
};

inherits(Man, Person);

Man.prototype.getName = function() {
  return "Mr. "
         + Man._super.getName.call(this);
};
```

Pseudo-classical
Dead-code

```
Person = function(name) {
  this._name = name;
};

Person.prototype.getName = function() {
  return this._name;
};


Man = function(name) {
  Person.call(this, name);
};

inherits(Man, Person);

Man.prototype.getName = function() {
  return "Mr. "
         + Man._super.getName.call(this);
};
```

kthxbai

There's more to life than this

Memory Footprint

- Crockford
  - Closure per-method, per-type
  - Leaky (holds a reference to all variables)
- Pseudo-classical
  - No closures needed

Live

Instance Updating

- Problem
  - How to update every instance in the system with new behavior?

Functional Inheritance Updating

```
var person = function(spec) {
  var that = {};

  that.getName = function() {
    return spec.name;
  };
  return that;
};

var me  = person({'name' : 'Fogus'});
var you = person({'name' : 'Cersei'});
```

How can we add **`hasBeard`** to every instance *now and in the future*?

Functional Inheritance Updating

```
var person = function(spec) {
  var that = {};

  that.getName = function() {
    return spec.name;
  };
  return that;
};

var me  = person({'name' : 'Fogus'});
var you = person({'name' : 'Cersei'});
```

How can we add **`hasBeard`** to every instance *now and in the future*?

One at a time... every single time


ClojureScript
Cares.

ClojureScript Extending

```
(defprotocol Catable
  (cat [this other]))

(def sound1 "Meow")
(def sound2 "Mao")

(cat sound1 sound2)
; ASPLODE!
```

How can we allow **`cat`** to work with every instance *now and in the future*?

ClojureScript Extending

```
(extend-type string
  Catable
  (cat [this other]
    (str this other)))

(cat sound1 sound2)
;=> “MeowMao”

(cat sound1 "ZZZZZ")
;=> "MeowZZZZZ"
```

Pseudo-classical underpinnings FTW

Pseudo-classical Updating

```
Person = function(name) {
  this._name = name;
};

Person.prototype.getName = function() {
  return this._name;
};

var me  = new Person('Fogus');
var you = new Person('Cercei');
```

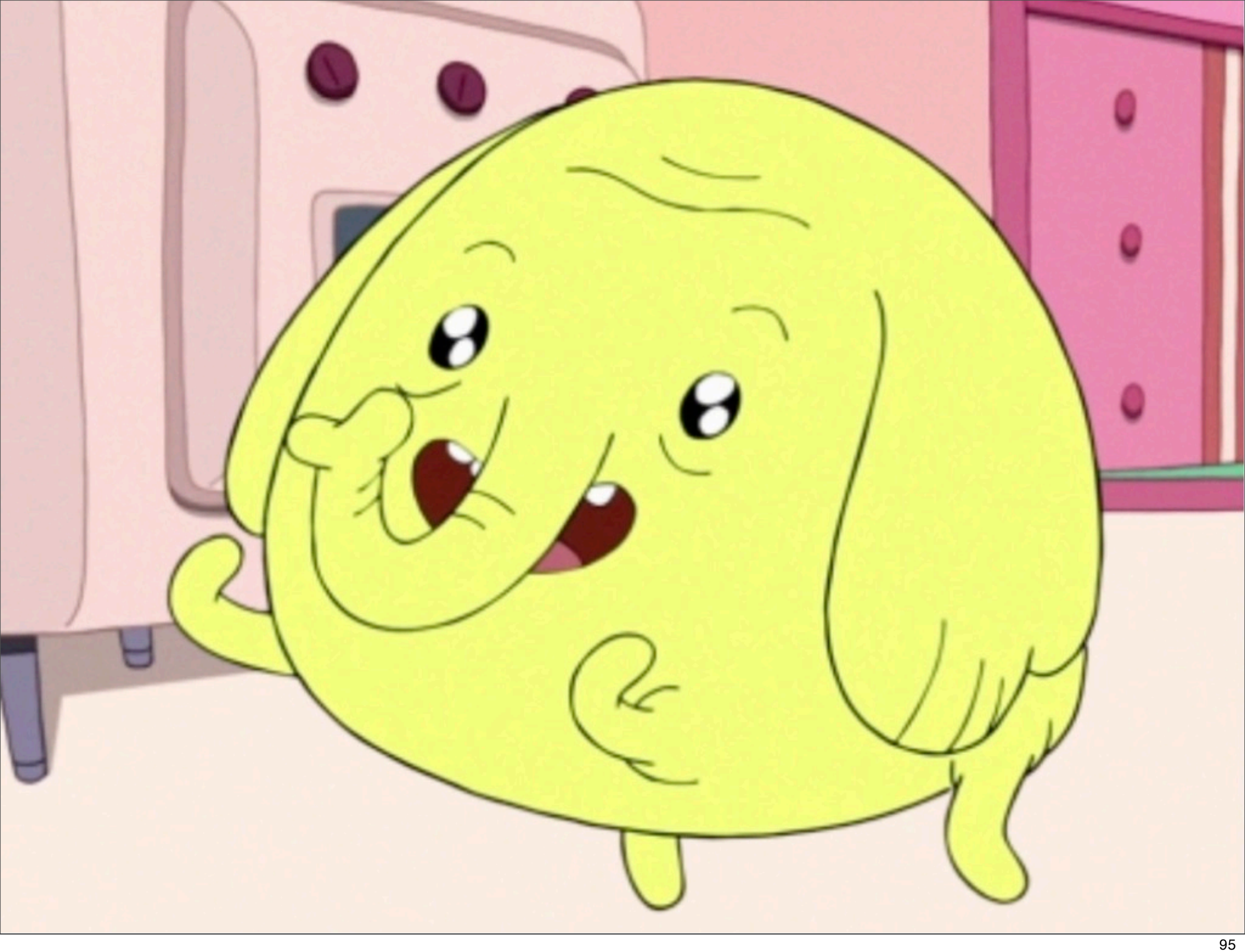
How can we add **`hasBeard`** to every instance *now and in the future*?

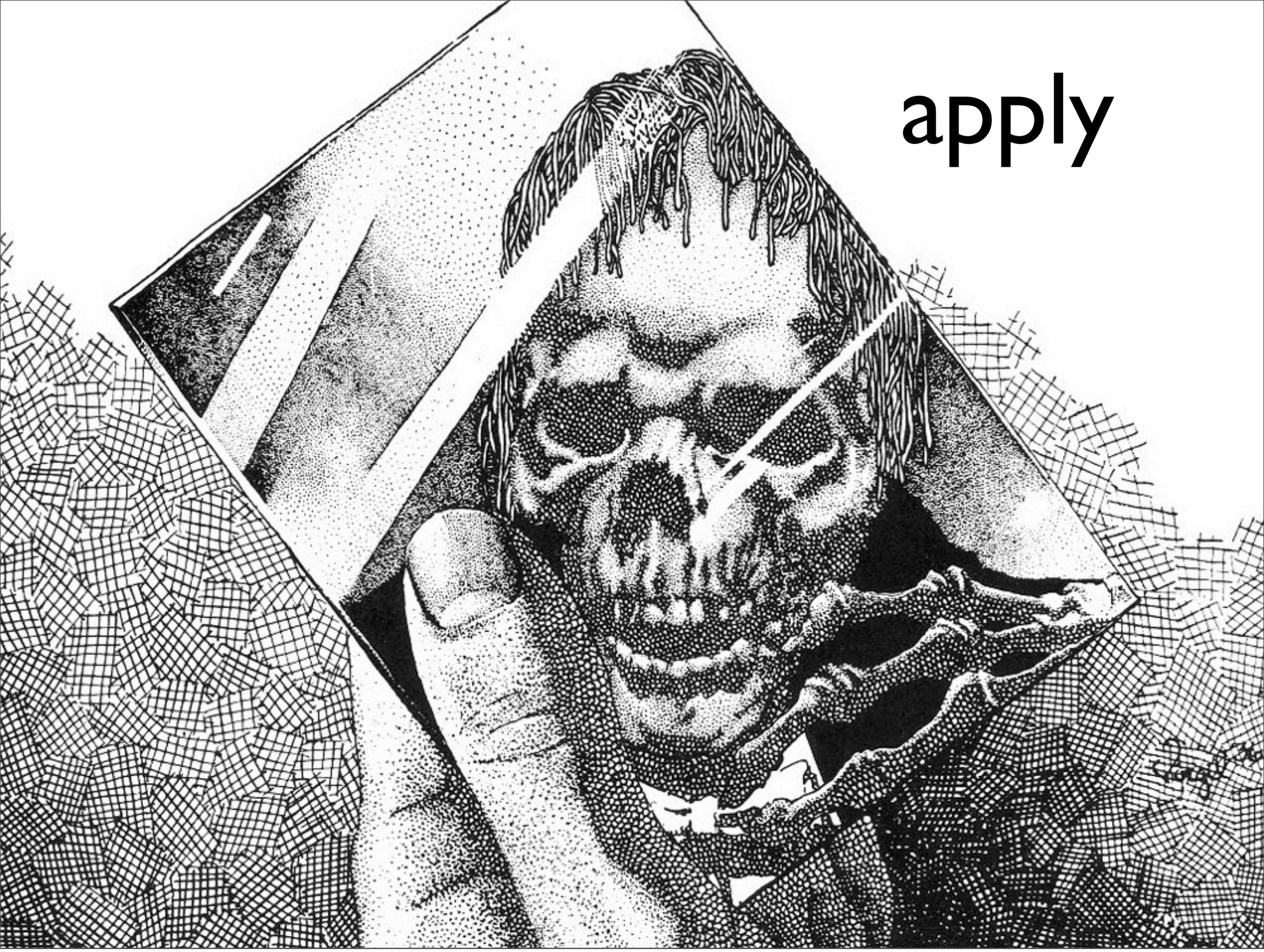
Pseudo-classical Updating

```
Person = function(name) {
  this._name = name;
};

Person.prototype.getName = function() {
  return this._name;
};

var me  = new Person('Fogus');
var you = new Person('Cercei');
```

How can we add **`hasBeard`** to every instance *now and in the future*?

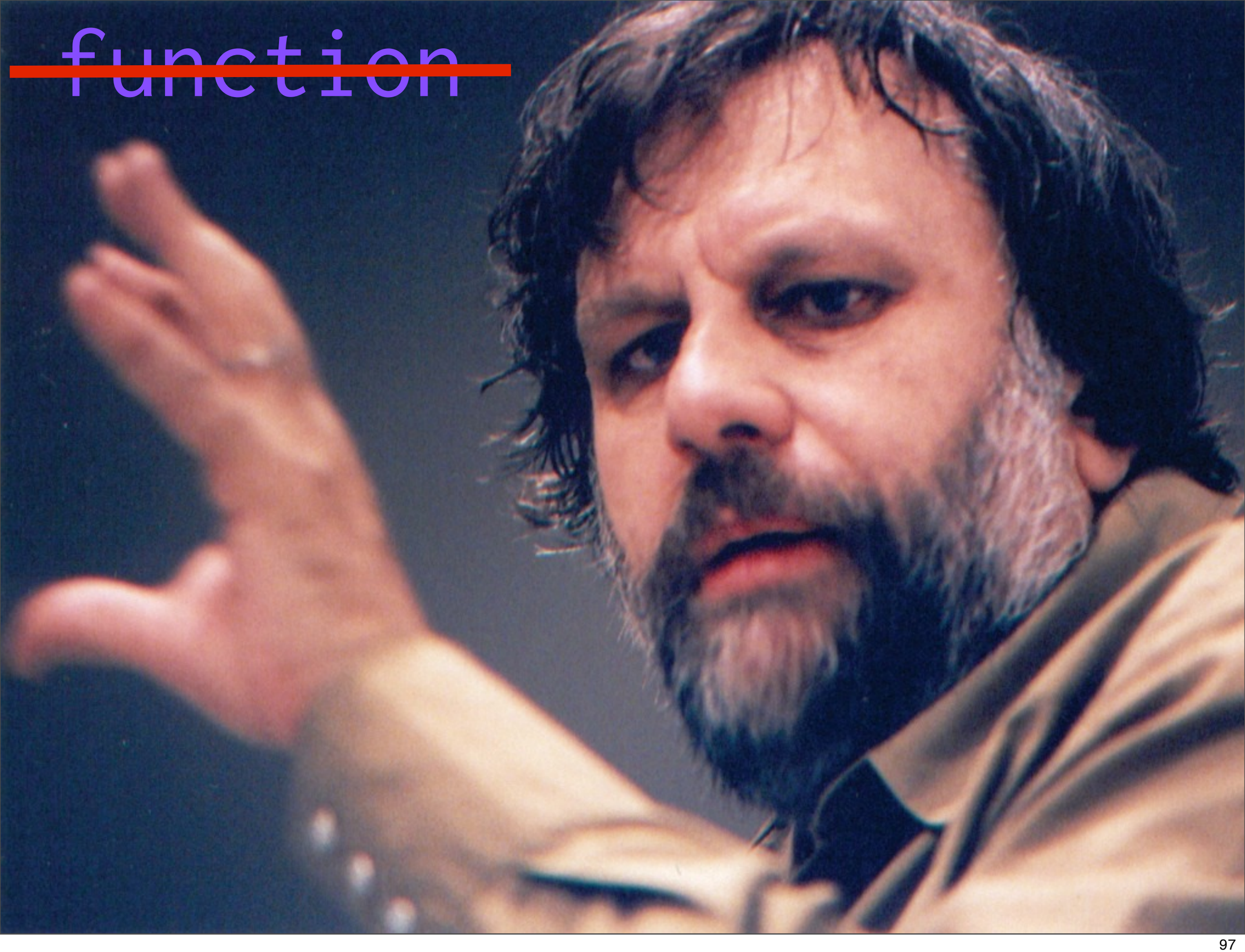
```
Person.prototype.hasBeard = function() { return false };
```





apply

~~function~~



Var Args

JavaScript

```
arguments
```

ClojureScript

```
& stuff
```

Var Args

JavaScript

`arguments`

An object that holds arguments

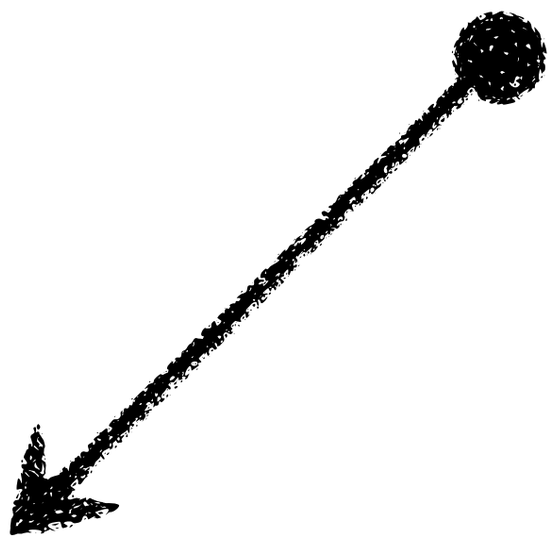
ClojureScript

`& stuff`

a seq

ClojureScript apply

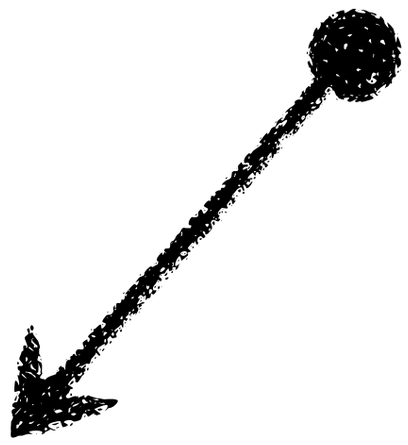
```
(defn add-first-two
  [& args]
  (+ (first args)
     (second args)))
```



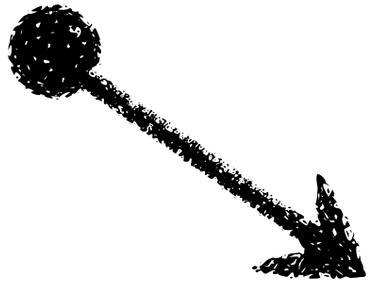
```
function addFirstTwo() {
  var args = // that slice thing
  return cljs.first(args) +
         cljs.second(args);
}
```

ClojureScript apply

```
(defn add-first-two
  [& args]
  (+ (first args)
     (second args)))
```



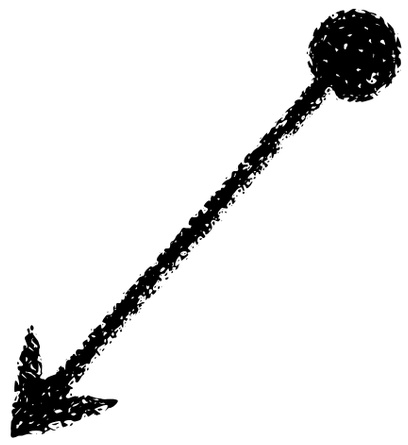
```
function addFirstTwo() {
  var args = // that slice thing
  return cljs.first(args) +
         cljs.second(args);
}
```



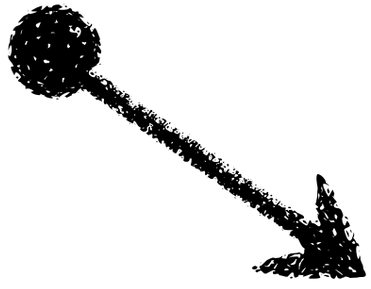
```
(add-first-two 1 2 3)
;=> 3
```

ClojureScript `apply`

```
(defn add-first-two
  [& args]
  (+ (first args)
     (second args)))
```



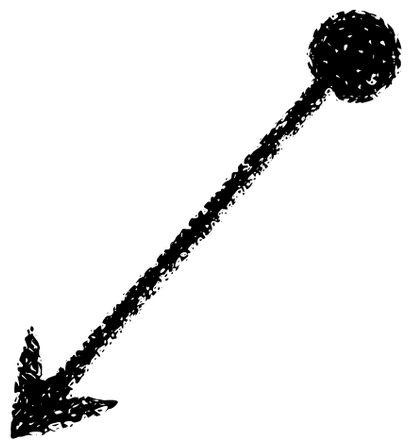
```
function addFirstTwo() {
  var args = // that slice thing
  return cljs.first(args) +
         cljs.second(args);
}
```



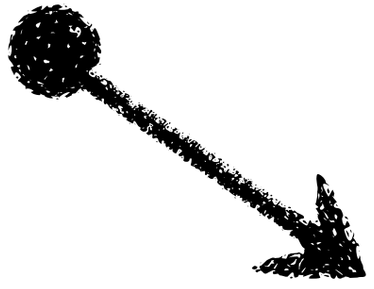
```
(apply add-first-two [1 2 3])
```

ClojureScript `apply`

```
(defn add-first-two
  [& args]
  (+ (first args)
     (second args)))
```



```
function addFirstTwo() {
  var args = // that slice thing
  return cljs.first(args) +
         cljs.second(args);
}
```



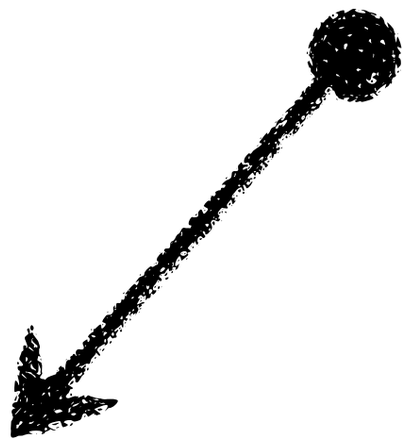
```
(apply add-first-two [1 2 3])
```

ClojureScript apply

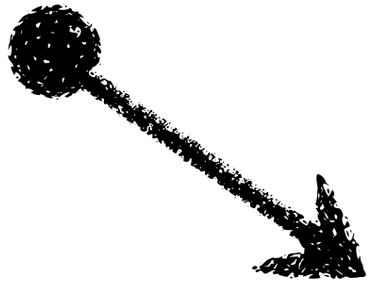
```
(defn apply
  [f arg-seq]
  (.apply f (to-array arg-seq)))
```

ClojureScript `apply`

```clojure
(defn add-first-two
  [& args]
  (+ (first args)
     (second args)))
```



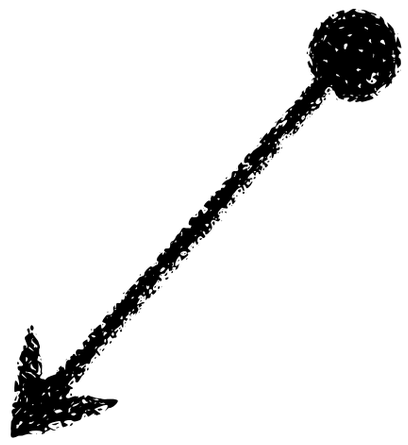
```javascript
function addFirstTwo() {
  var args = // that slice thing
  return cljs.first(args) +
         cljs.second(args);
}
```



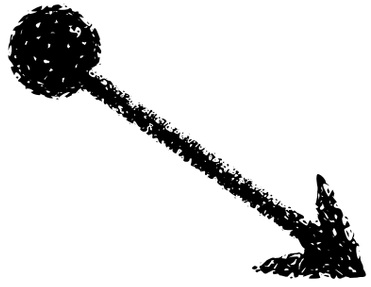
```clojure
(apply add-first-two [1 2 3])
;=> 3
```

ClojureScript apply

```
(defn add-first-two
  [& args]
  (+ (first args)
     (second args)))
```



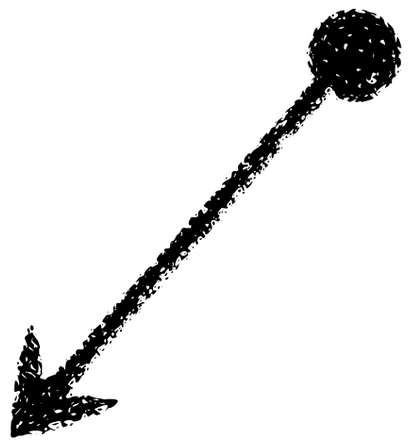
```
function addFirstTwo() {
  var args = // that slice thing
  return cljs.first(args) +
         cljs.second(args);
}
```



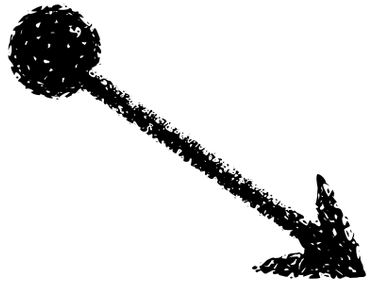
```
(apply add-first-two (iterate inc 1))
;=> ?
```

ClojureScript apply

```
(defn add-first-two
  [& args]
  (+ (first args)
     (second args)))
```



```
function addFirstTwo() {
  var args = // that slice thing
  return cljs.first(args) +
         cljs.second(args);
}
```



```
(apply add-first-two (iterate inc 1))
;=> 
```

ClojureScript apply

```
(defn apply
  [f arg-seq]
  (.apply f (to-array arg-seq)))
```


STOP

ClojureScript `applyTo`

- A function stored on ClojureScript functions
  - Only if they have var args
- Compiled to only deal with the args seq at the maximum fixed arity
  - This ensures that an infinite seq is not realized

ClojureScript apply

```
(defn apply
  [f arg-seq]
  (if (.applyTo f)
    (if (<= bounded-count fixed-arity)
      (.apply f (to-array arg-seq))
      (.applyTo f args))
     (.apply f (to-array arg-seq))))
```

ClojureScript apply

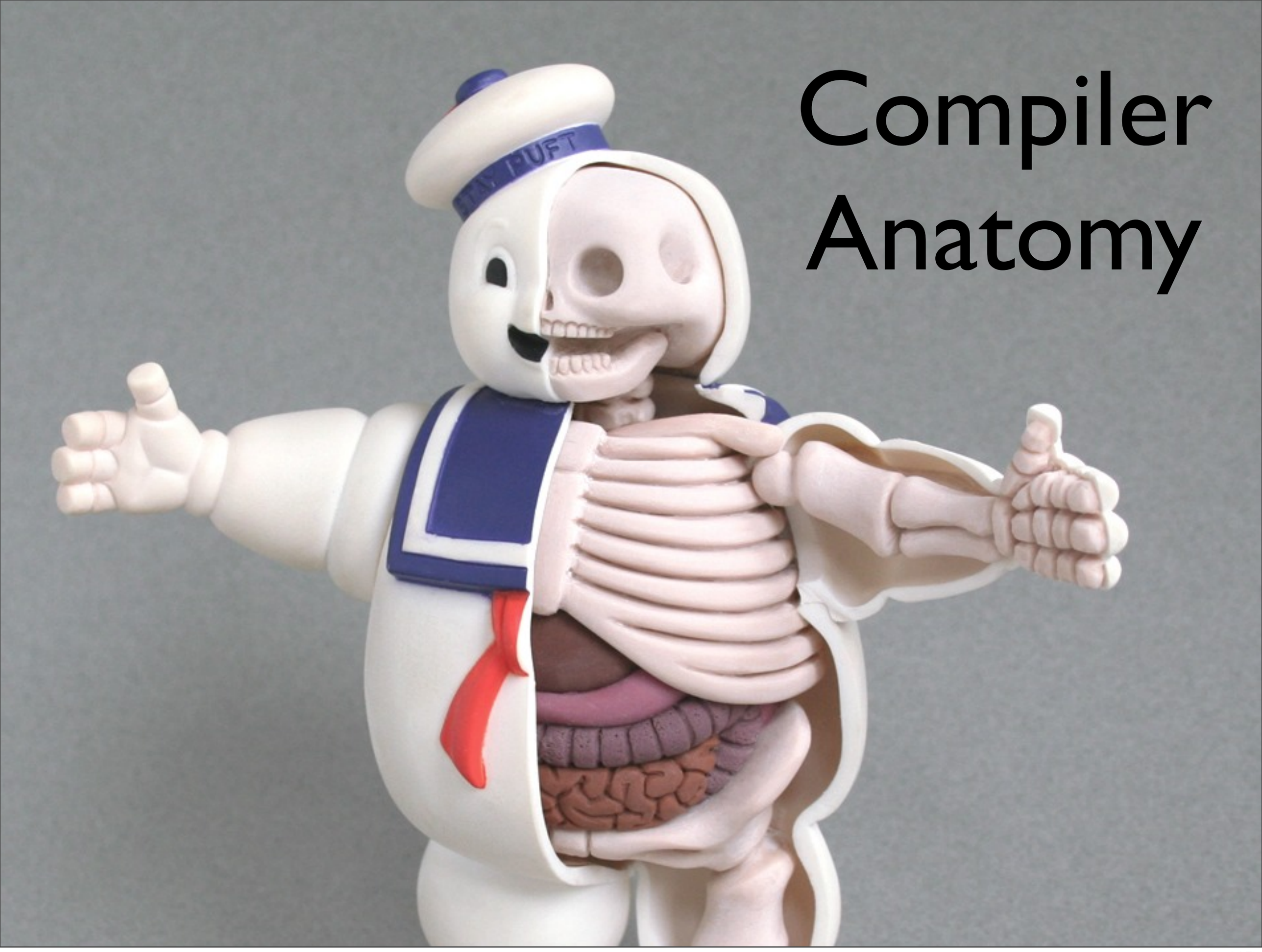
```
(defn apply
  [f arg-seq]
  (if (.applyTo f)
    (if (< bounded-count fixed-arity)
      (.apply f (to-array arg-seq))
      (.applyTo f args))
     (.apply f (to-array arg-seq))))
```

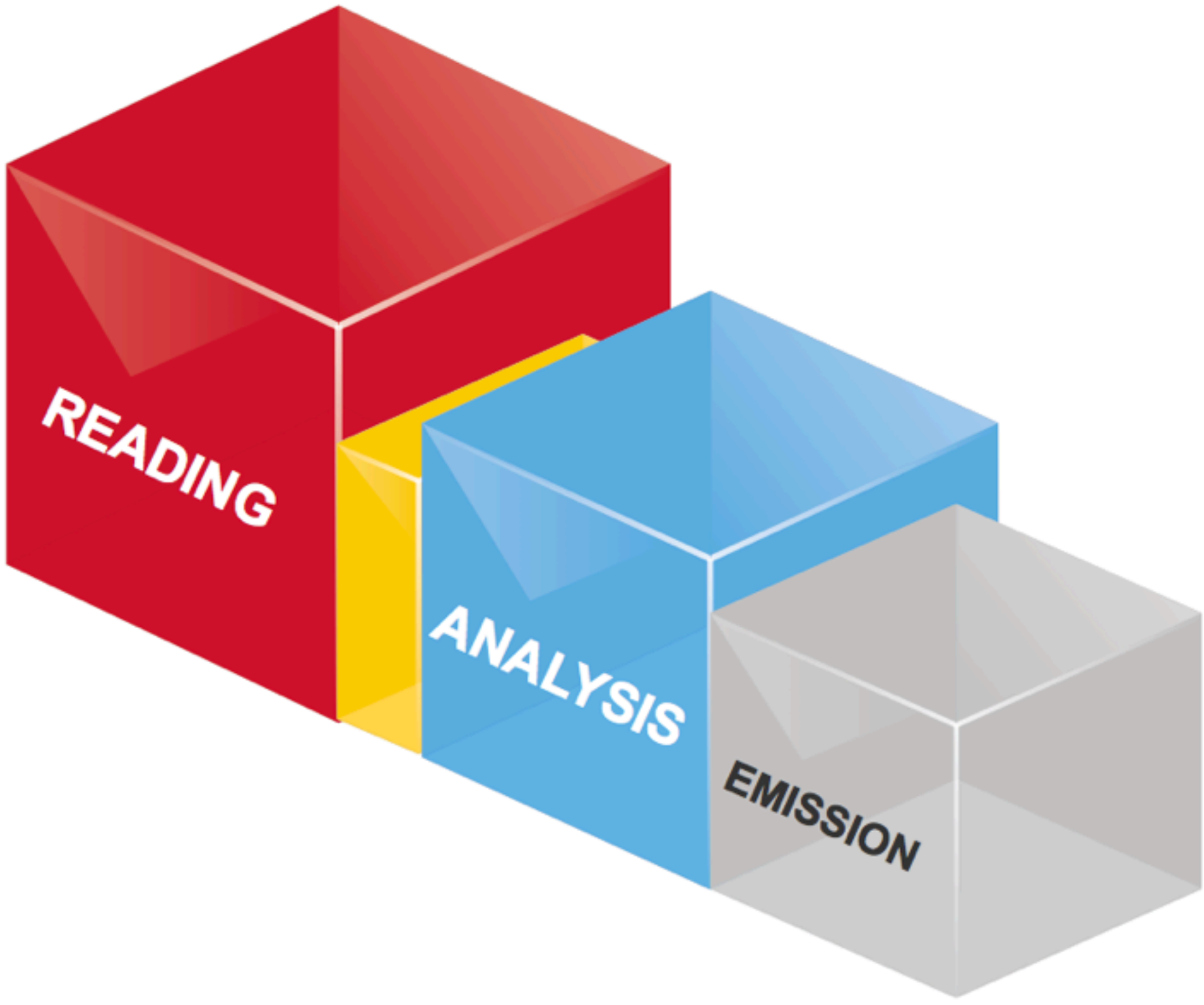
ClojureScript apply

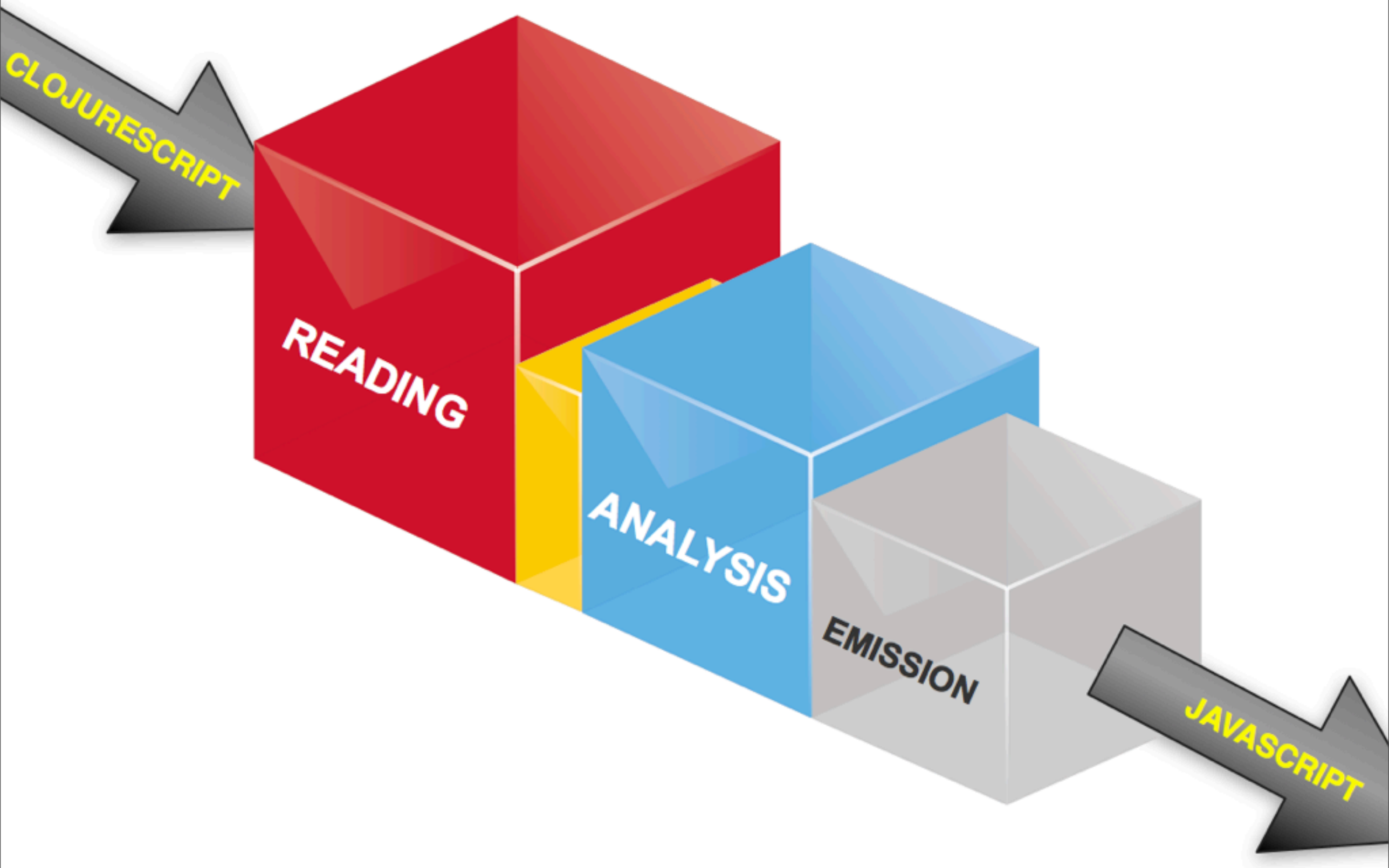
```
(defn apply
  [f arg-seq]
  (if (.applyTo f)
    (if (<= bounded-count fixed-arity)
      (.apply f (to-array arg-seq))
      (.applyTo f args))
     (.apply f (to-array arg-seq)))
```

ClojureScript apply

```
(defn apply
  [f arg-seq]
  (if (.applyTo f)
    (if (<= bounded-count fixed-arity)
      (.apply f (to-array arg-seq))
      (.applyTo f args))
      (.apply f (to-array arg-seq)))
```


Compiler
Anatomy

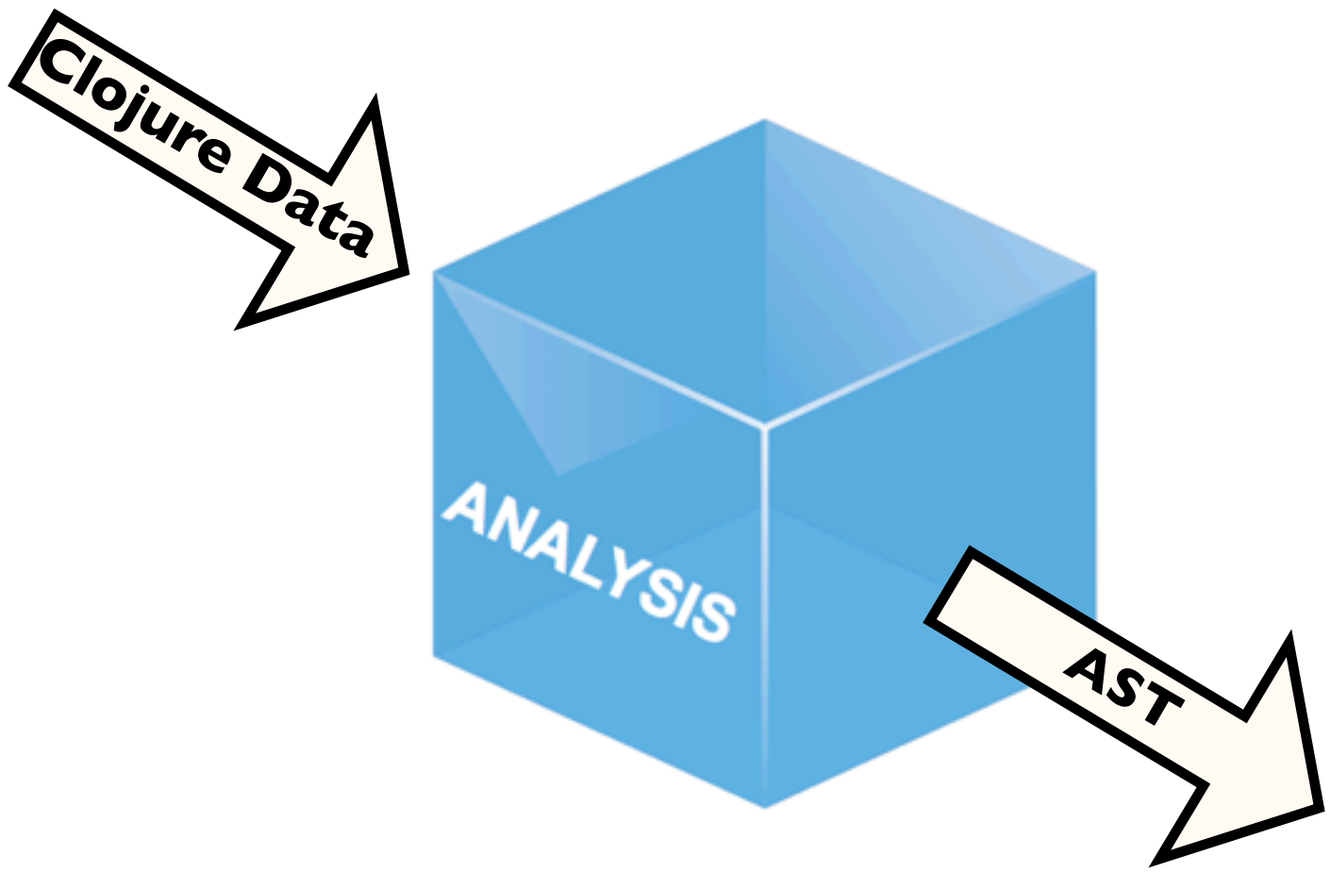

READING
ANALYSIS
EMISSION

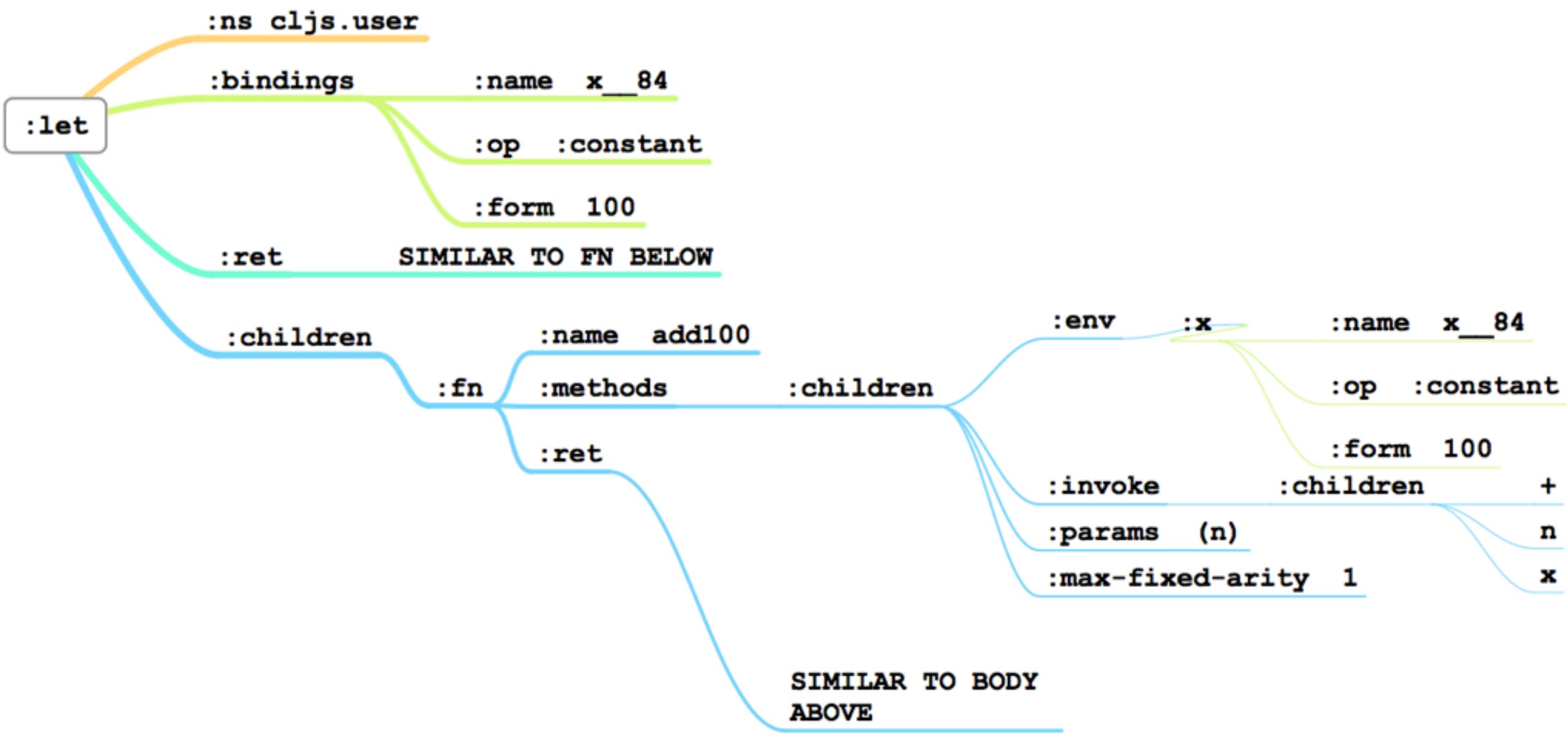

CLOJURESCRIPT
READING
ANALYSIS
EMISSION
JAVASCRIPT


READING

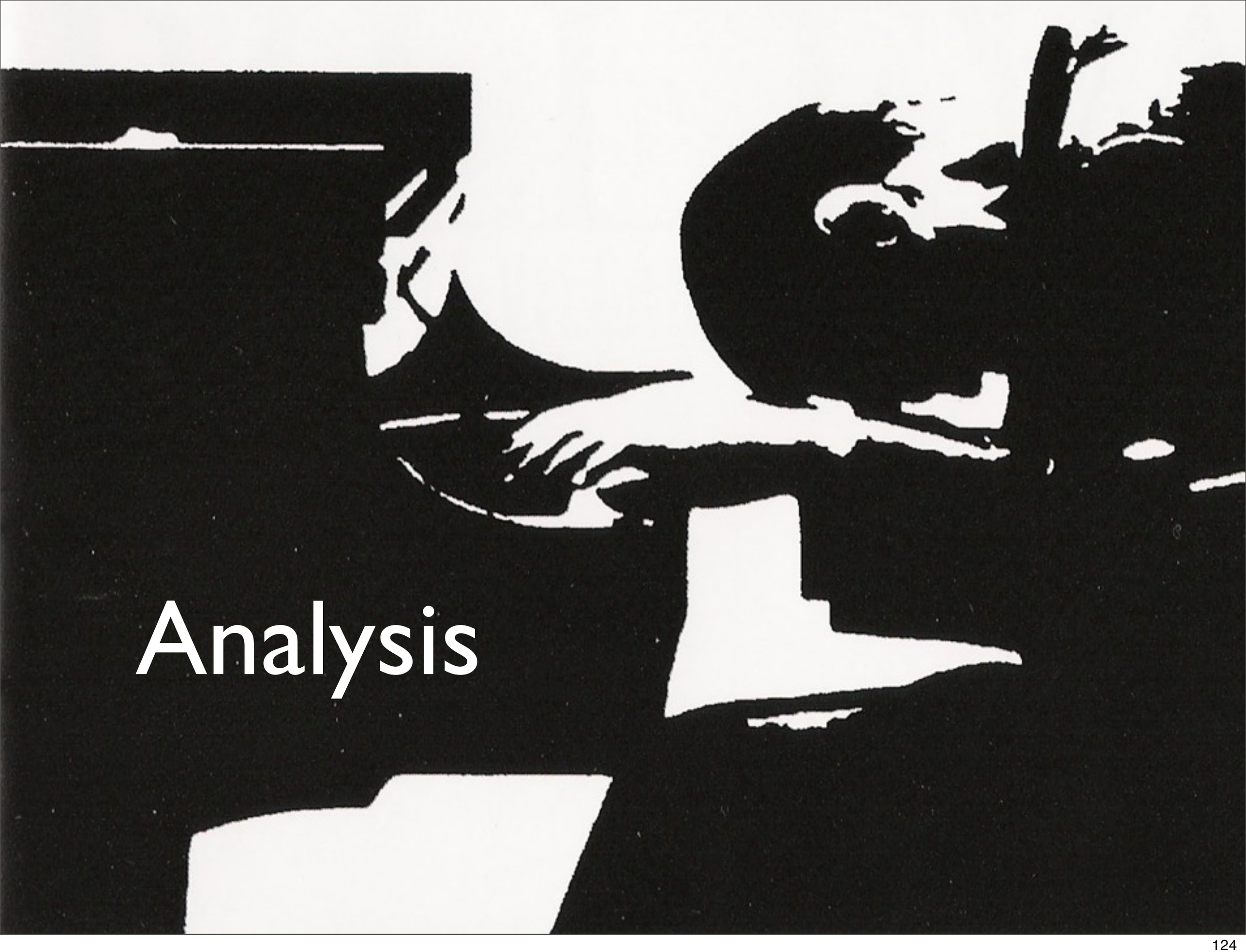

MACROS


ANALYSIS


Clojure Data
ANALYSIS
AST


:let
:ns cljs.user
:bindings
:name x__84
:op :constant
:form 100
:ret
SIMILAR TO FN BELOW
:children
:fn
:name add100
:methods
:children
:env
:x
:name x__84
:op :constant
:form 100
:invoke
:children
+
n
x
:params (n)
:max-fixed-arity 1
:ret
SIMILAR TO BODY ABOVE

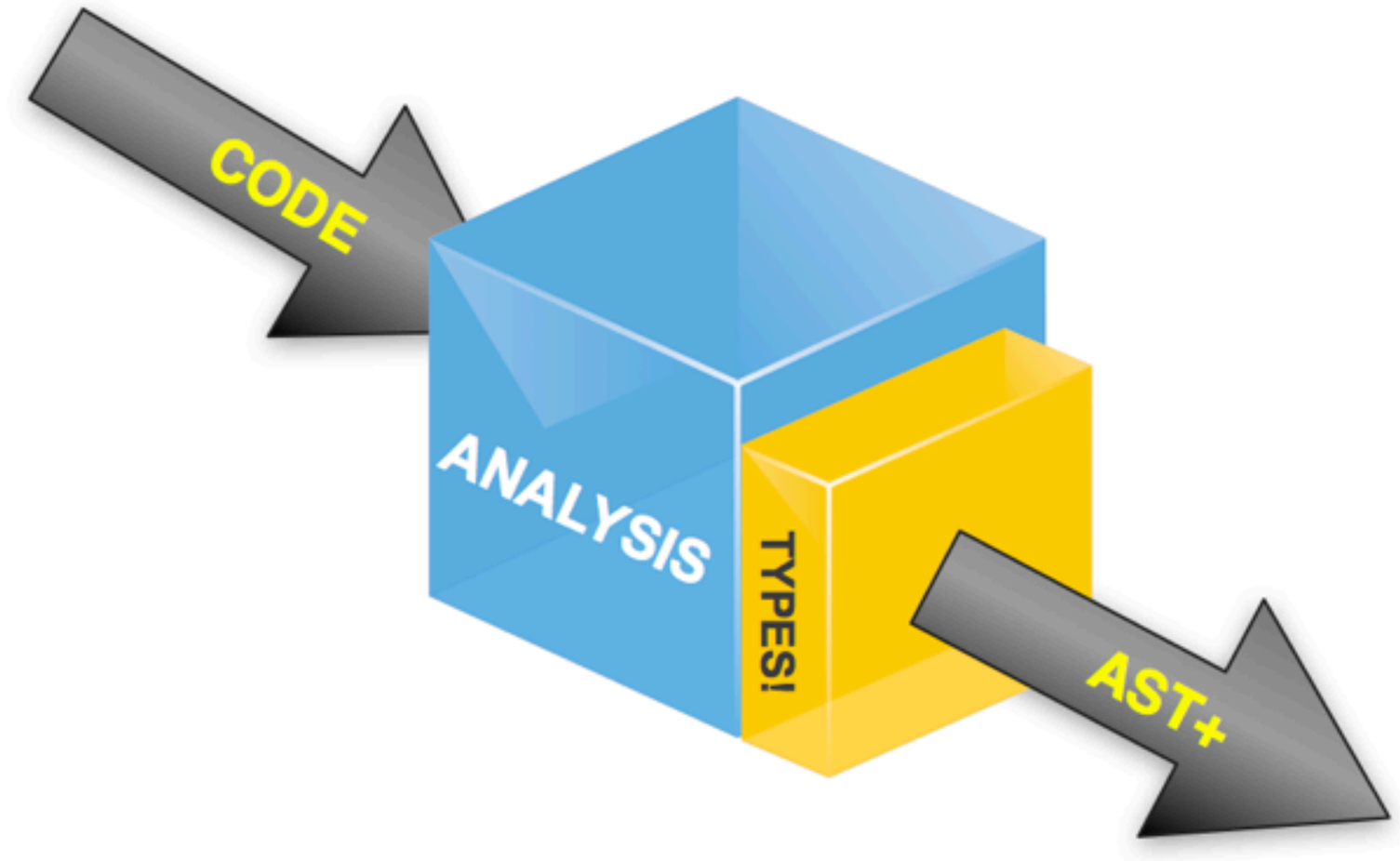

EMISSION

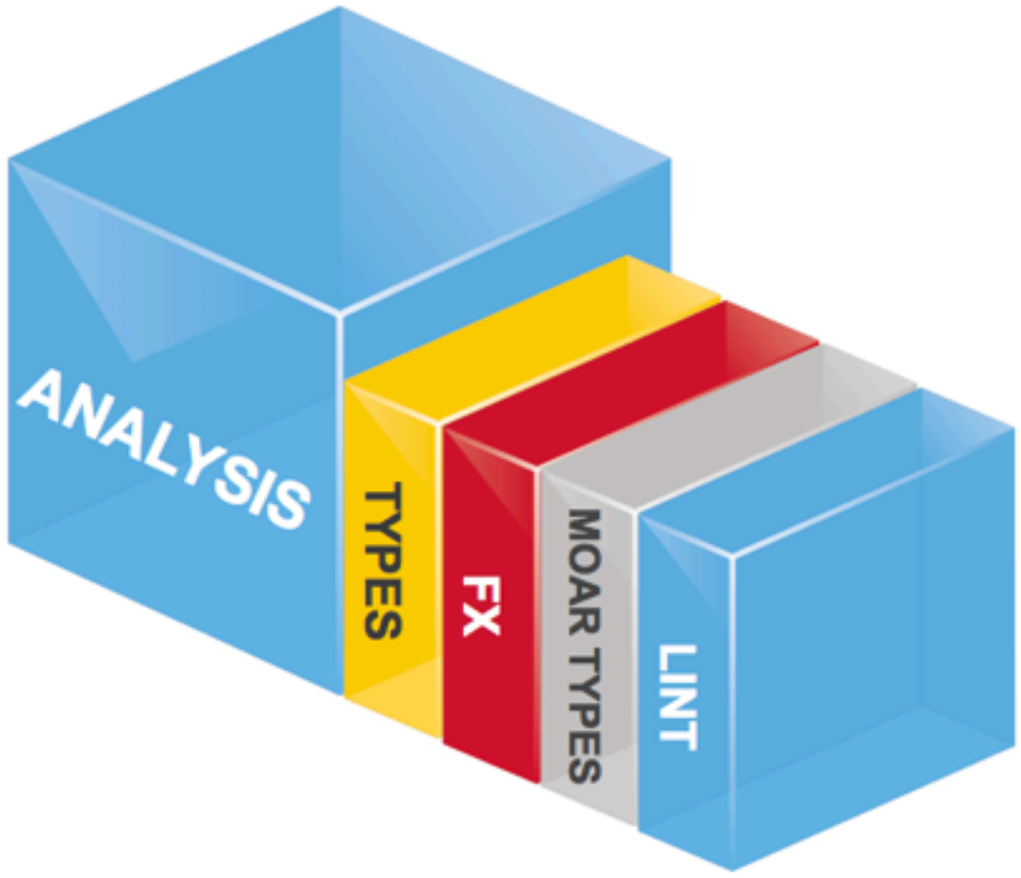


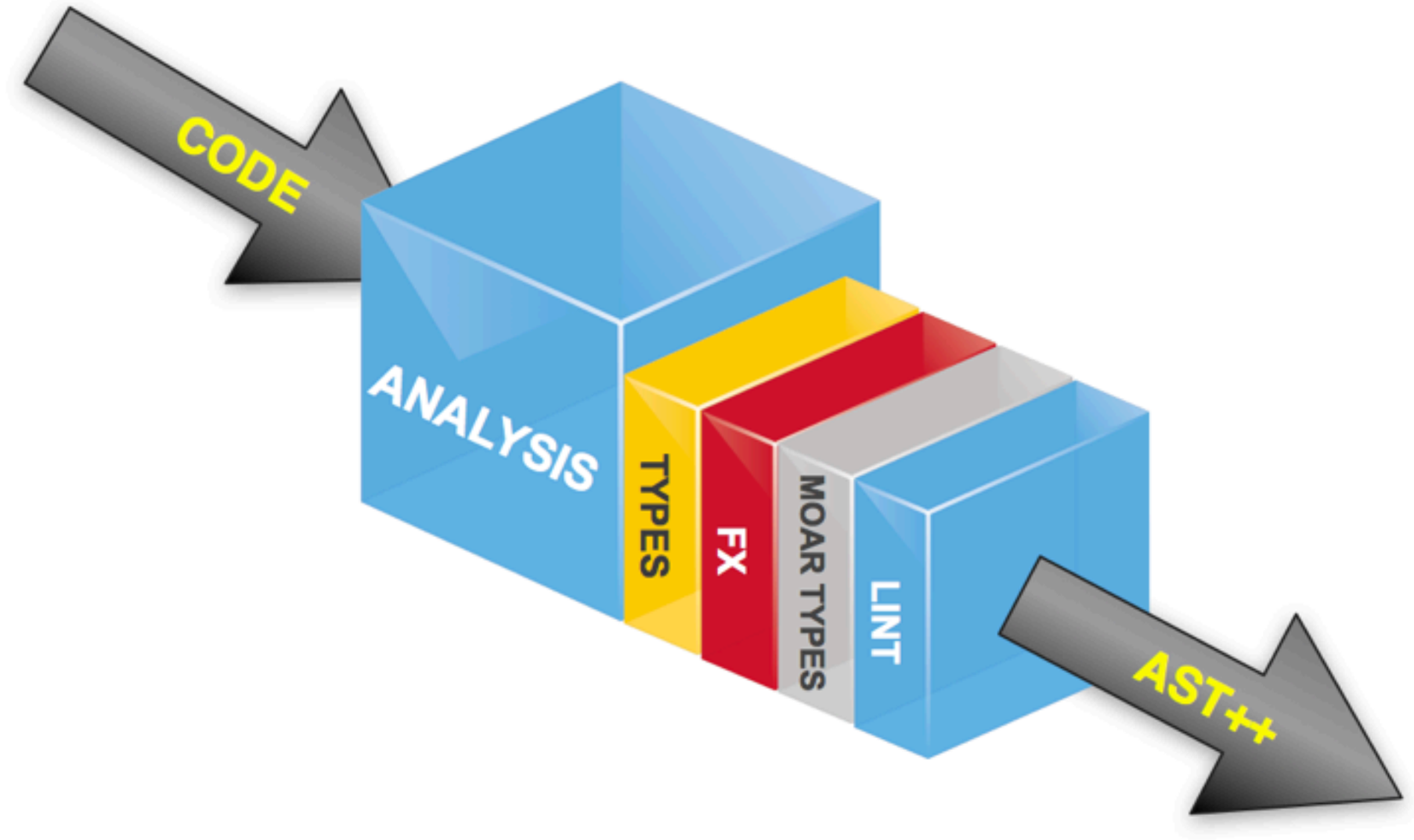
Analysis


ANALYSIS

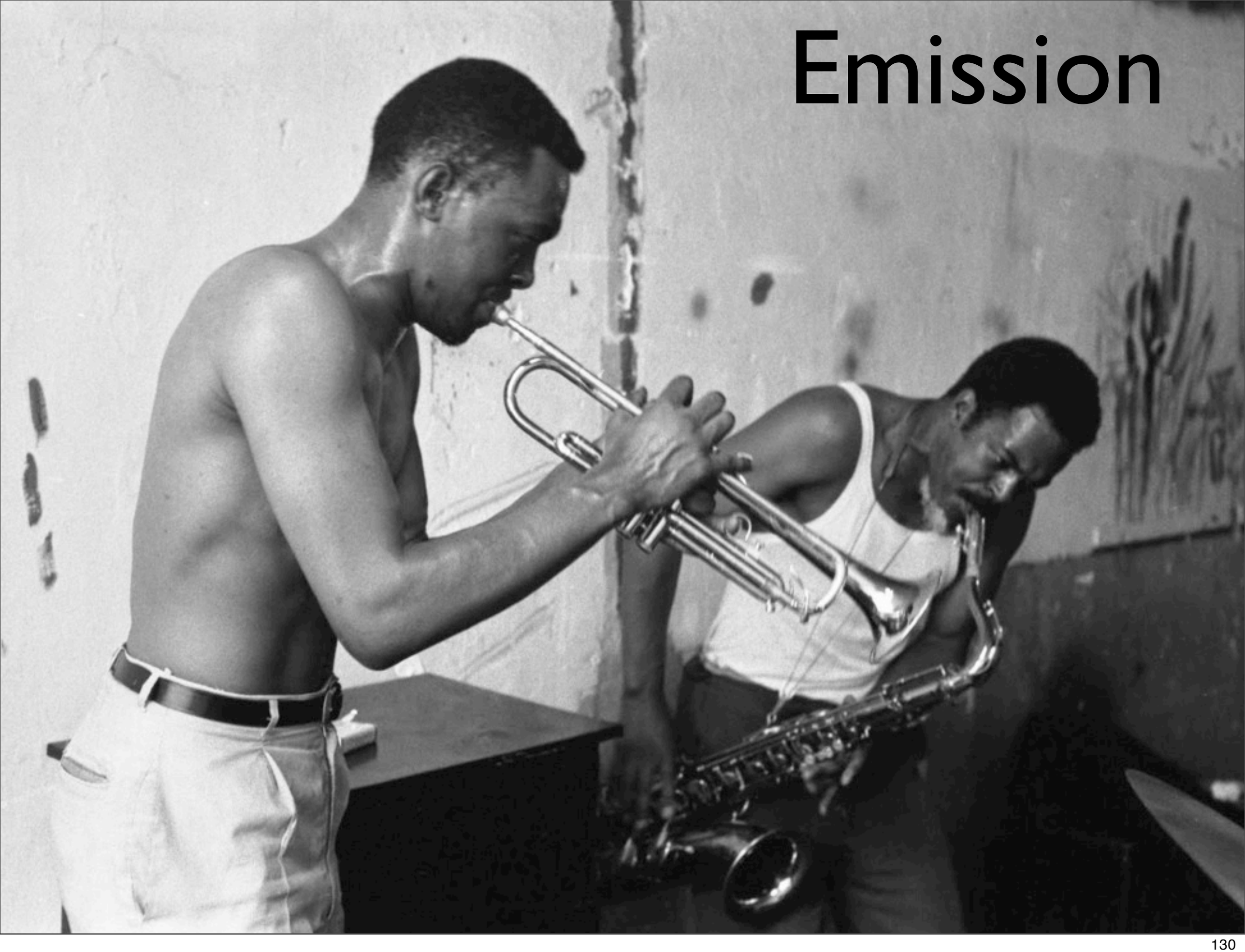

ANALYSIS
TYPES!

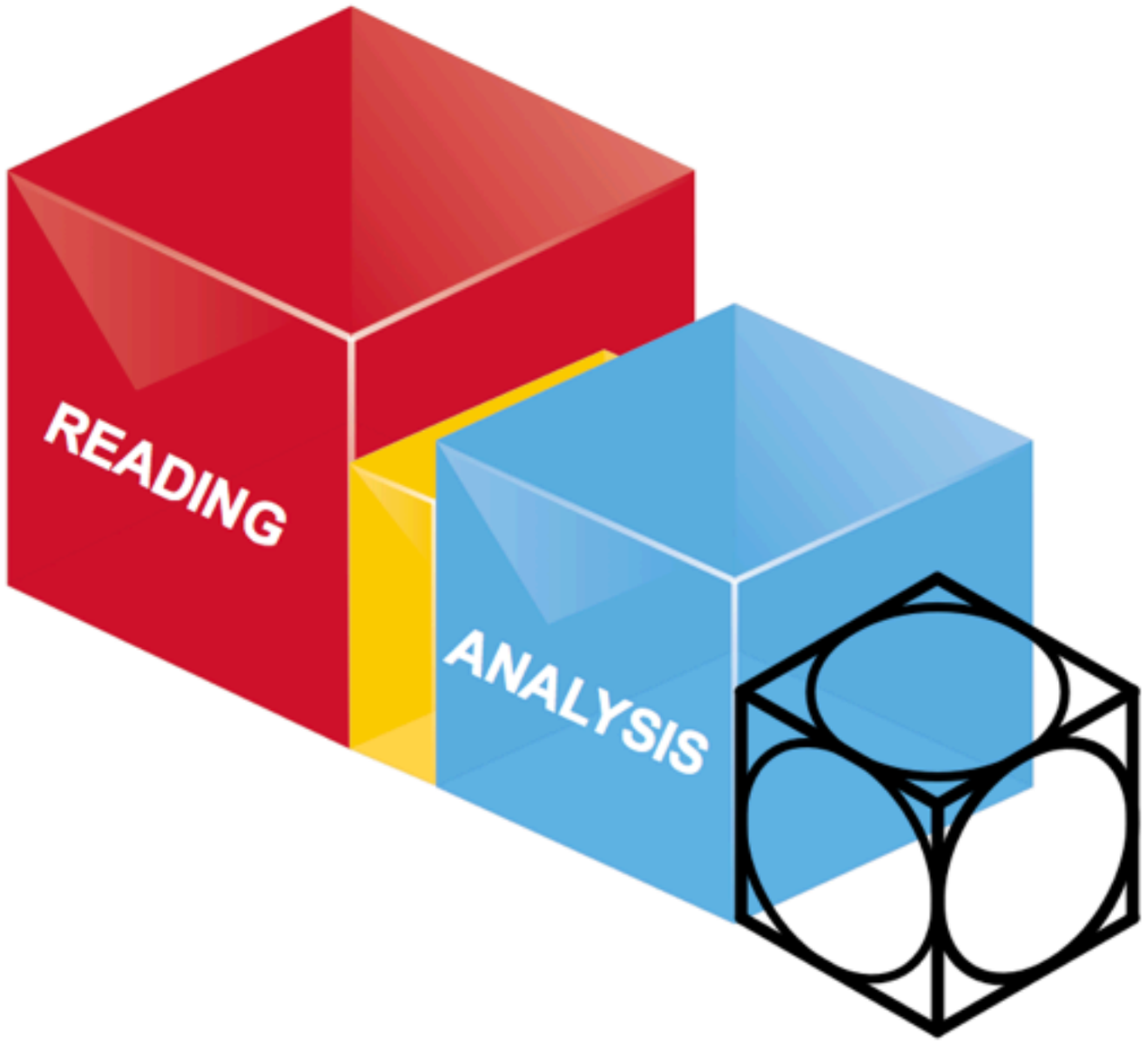

CODE
ANALYSIS
TYPES!
AST+

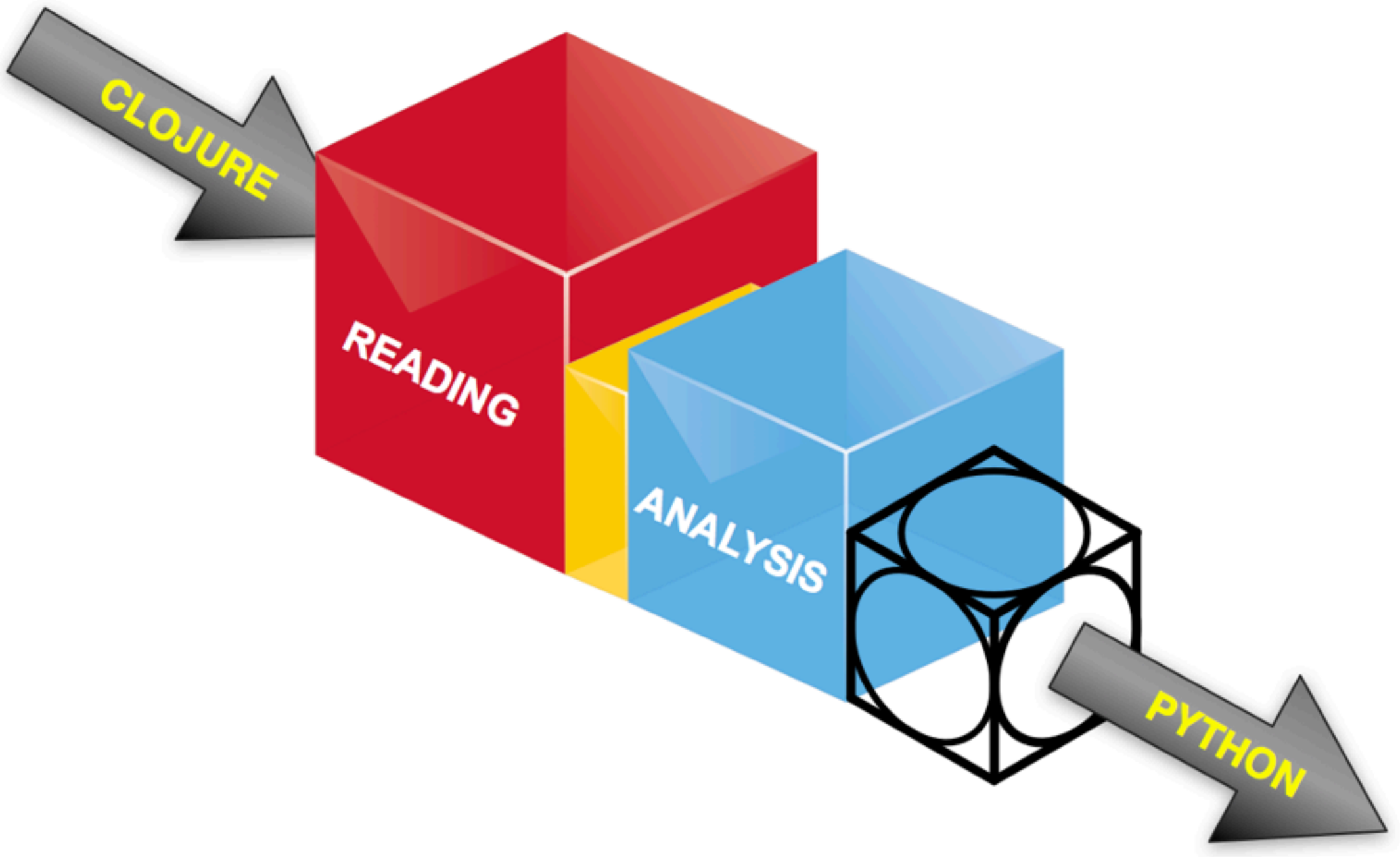

ANALYSIS
TYPES
FX
MOAR TYPES
LINT


CODE
ANALYSIS
TYPES
FX
MOAR TYPES
LINT
AST++

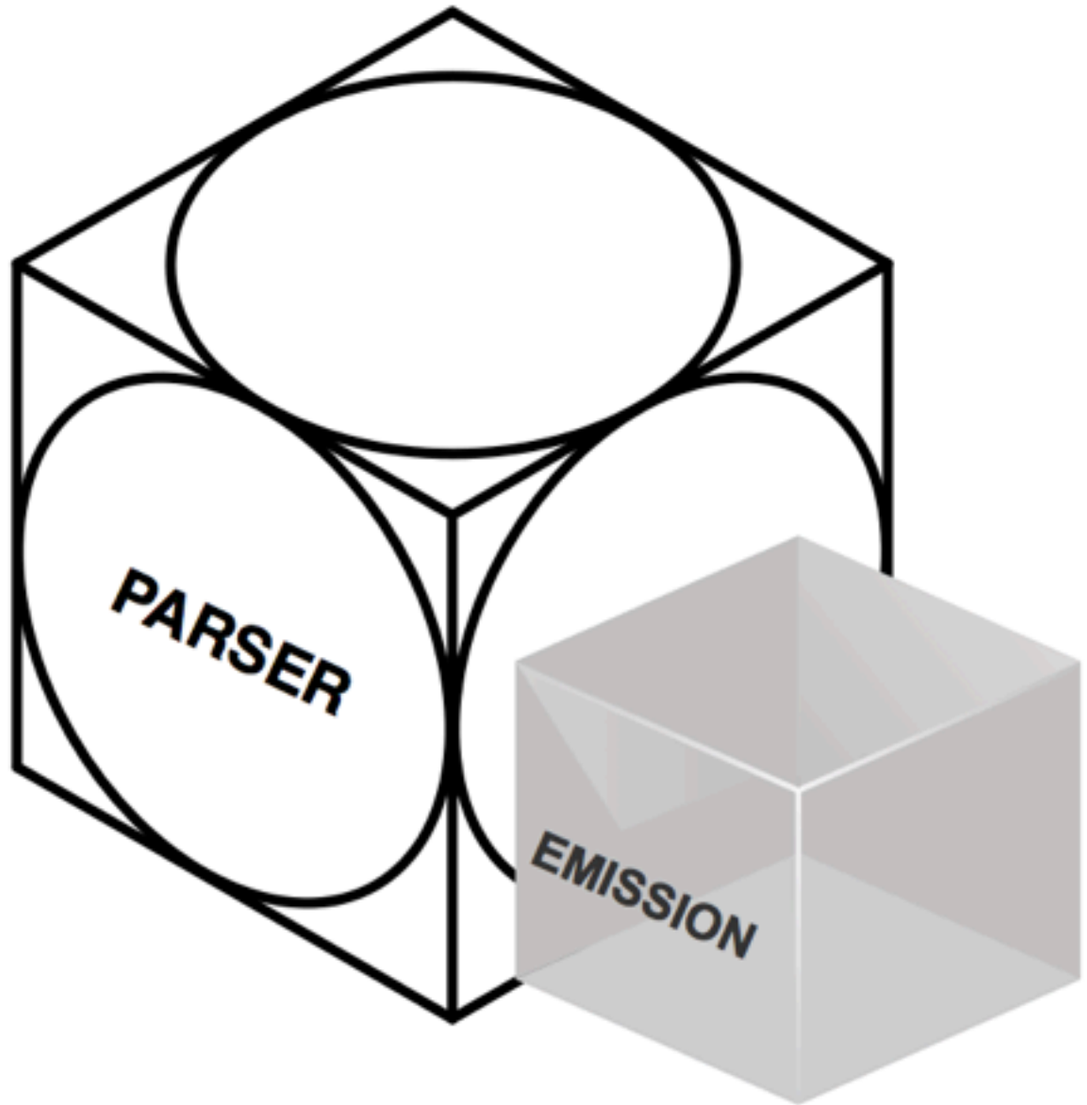
Emission

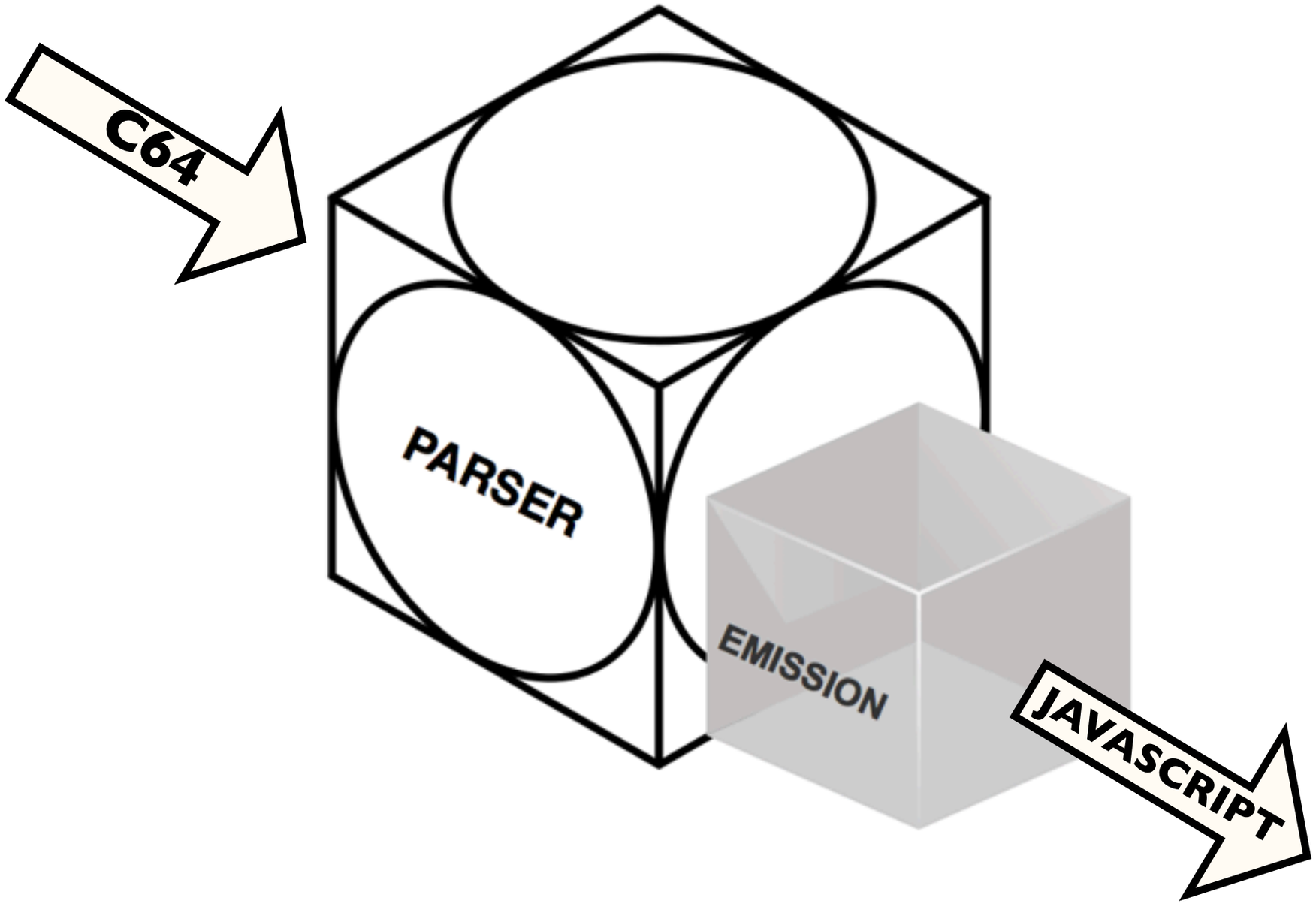


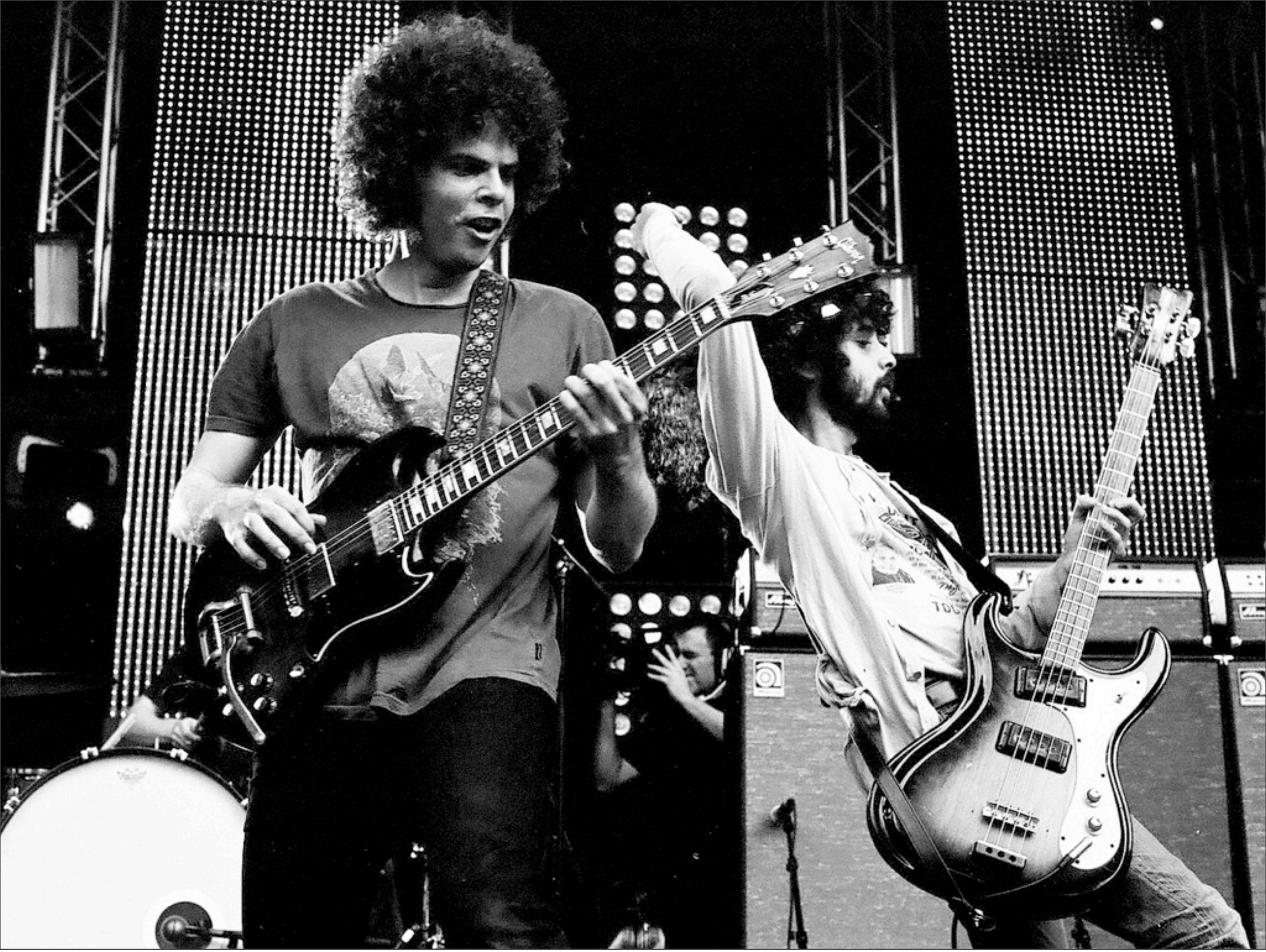

READING
ANALYSIS

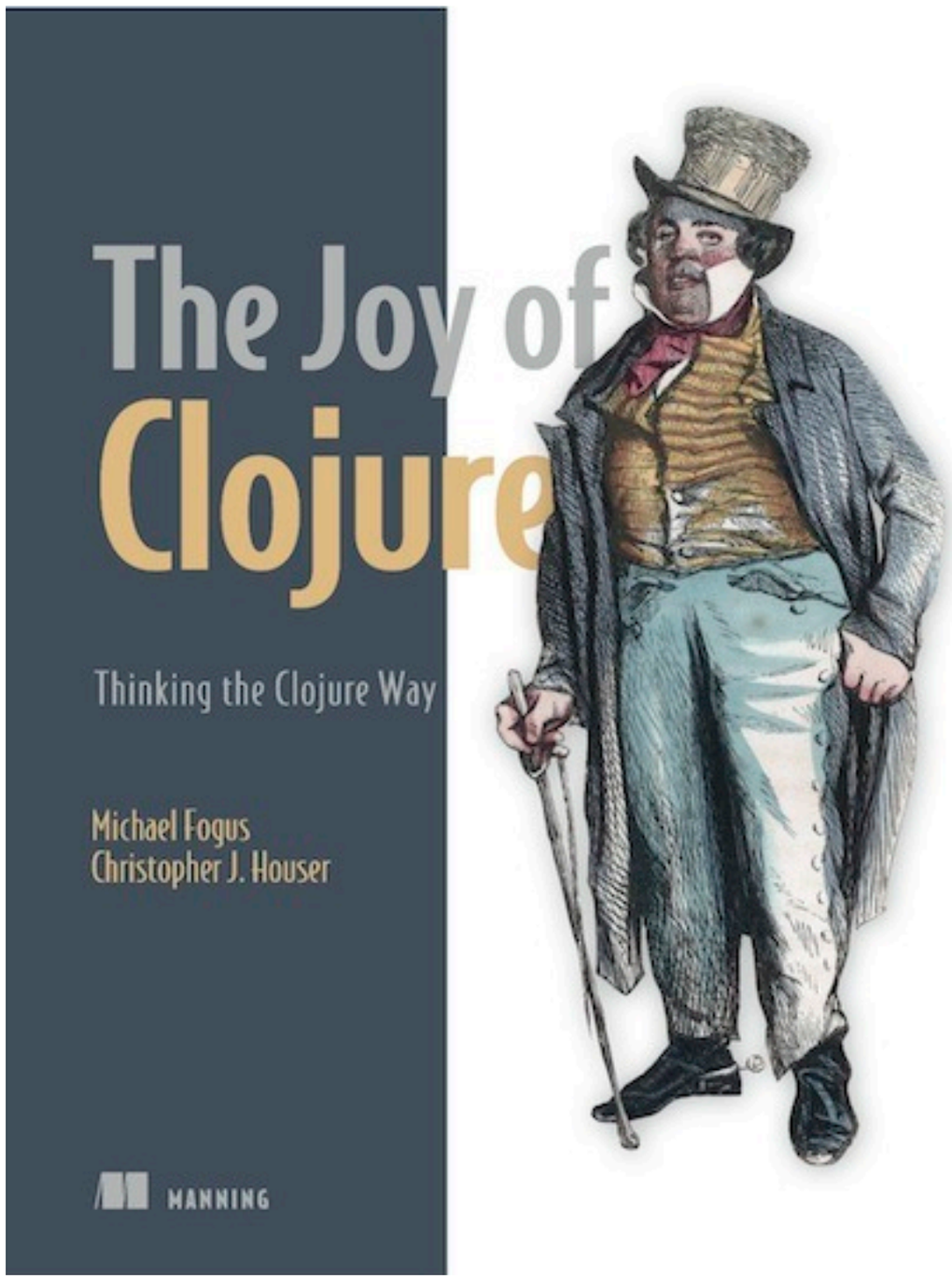

CLOJURE
READING
ANALYSIS
PYTHON


EMISSION


PARSER
EMISSION


C64
PARSER
EMISSION
JAVASCRIPT



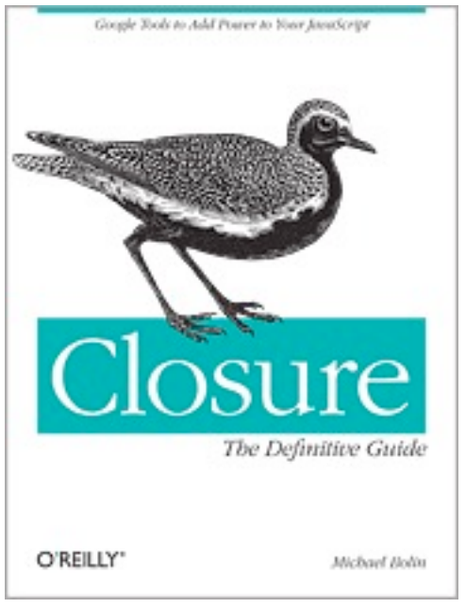


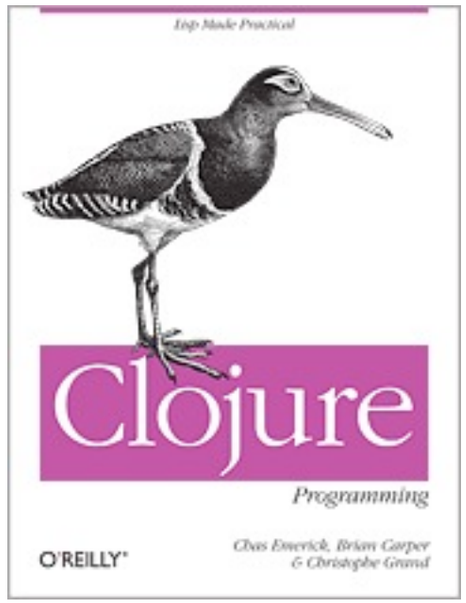
Thank You

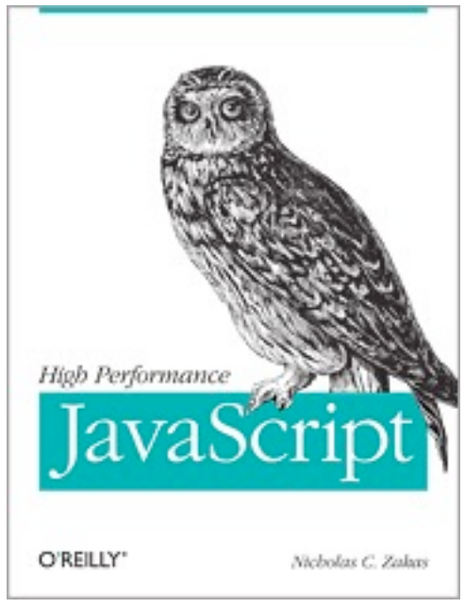
http://joyofclojure.com

@fogus

Learn More







- http://bit.ly/cljs-intro
- http://boss-level.com/?p=102
- http://blog.fogus.me/tag/clojurescript
- http://github.com/clojure/clojurescript

Crockford Hates

`instanceof`

Functional Inheritance
`instanceof`

```
var me  = person({'name' : 'Fogus'});

me instanceof person;
//=> false
```

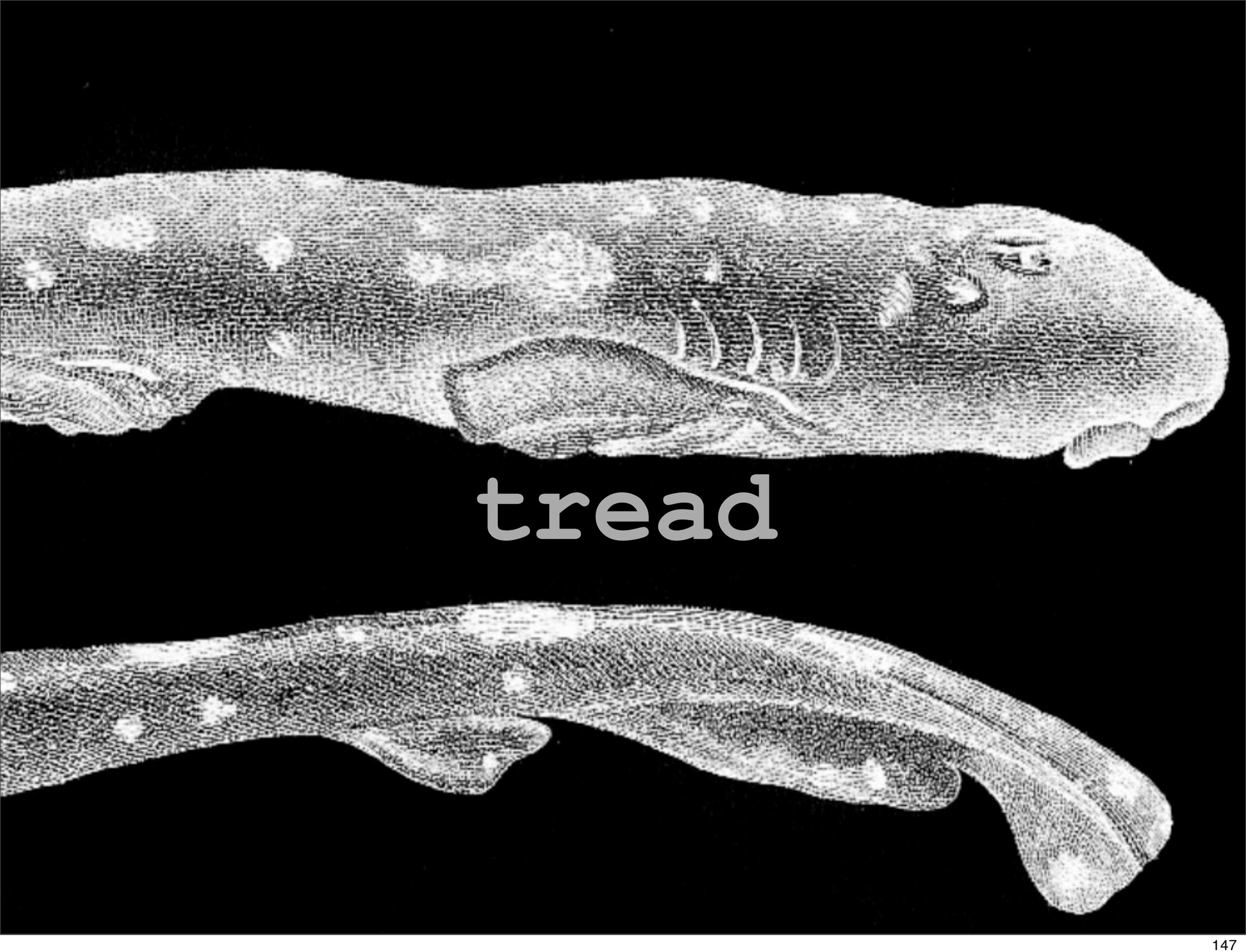
ClojureScript Loves `instanceof`

Pseudo-classical `instanceof`

```
var me  = new Person('Fogus');

me instanceof Person;
//=> true
```

ClojureScript satisfies?

```
(defprotocol Catable
  (cat [this other]))

(deftype Cons [H T]
  Catable
  (cat [_ new-tail] nil))

(satisfies? Catable (Cons. 1 2))
;=> true
```



tread

Don’t Tread on Me

```
Function.prototype.method = function(name, f) {
  this.prototype[name] = f;
  return this;
};

Object.method('superior', function(name) {
  var that = function;
  var method = that[name];

  return function() {
    return method.apply(that, arguments);
  };
});
```

ClojureScript Gently Caresses `Object` & `Function` as One Might Caress the finest Fabergé Egg

Look with your eyes, not with your hands

```
var inherits = function(child, parent) {
  function tmp() {};
  tmp.prototype = parent.prototype;

  child._super = parent.prototype;
  child.prototype = new tmp();
  child.prototype.constructor = child;
};
```

Var Args

JavaScript

```
function stuffToArray() {
  // create an array of stuff
}

stuffToArray(1,2,3,4);
//=> [1,2,3,4]
```

ClojureScript

```
(defn stuff->seq
  [& stuff]
  stuff)

(stuff->seq 1 2 3 4)
;=> (1 2 3 4)
```

Var Args

JavaScript

```
function stuffToArray() {
  // create an array of stuff
}

stuffToArray(1,2,3,4);
//=> [1,2,3,4]
```

ClojureScript

```
(defn stuff->seq
  [& stuff]
  stuff)

(stuff->seq 1 2 3 4)
;=> (1 2 3 4)
```

Var Args

JavaScript

```
function stuffToArray() {
  return Array.prototype.slice.call(arguments);
}

stuffToArray(1,2,3,4);
//=> [1,2,3,4]
```

ClojureScript

```
(defn stuff->seq
  [& stuff]
  stuff)

(stuff->seq 1 2 3 4)
;=> (1 2 3 4)
```

Compiler Macros

- Operate in the compiler only
  - Analysis phase
- Written in Clojure
- Allows shadowing of fns
  - Operates according to a fix point