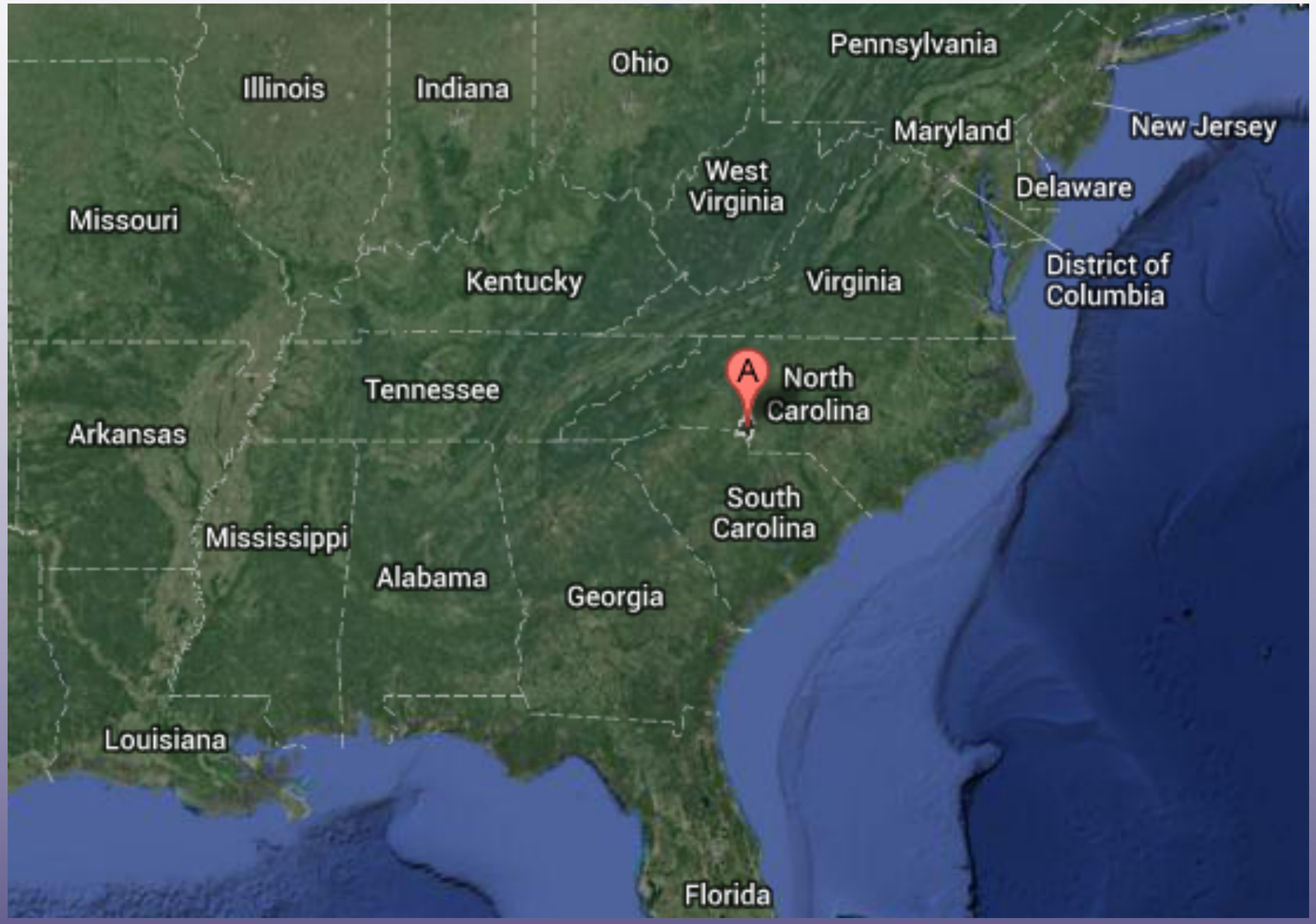
function*

ES6, generators, and all that

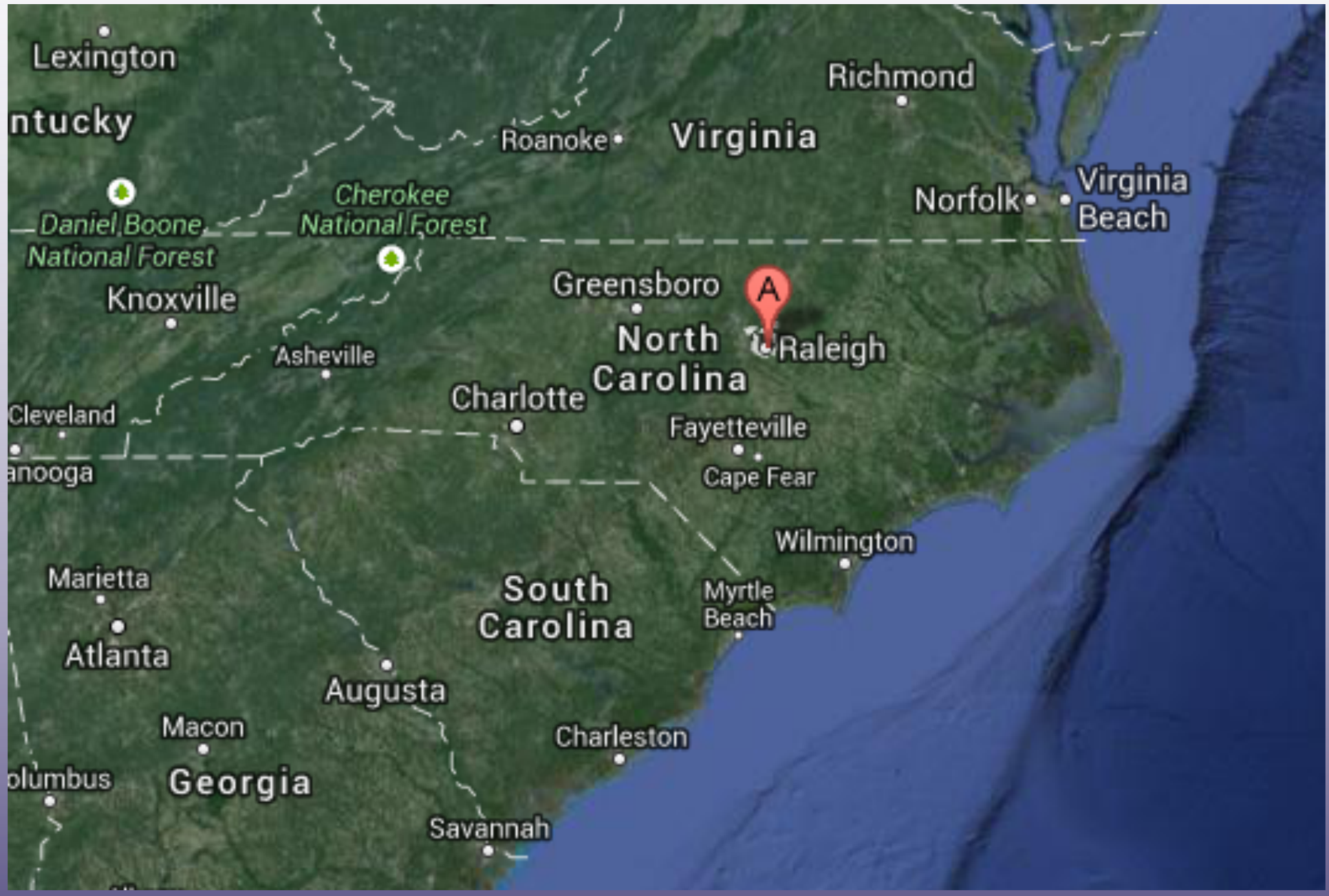
JS Romandie February 2014

Andy Wingo

youth


Illinois
Indiana
Ohio
Pennsylvania
Maryland
New Jersey
West Virginia
Delaware
Missouri
Kentucky
Virginia
District of Columbia
A
North Carolina
Tennessee
Arkansas
South Carolina
Mississippi
Alabama
Georgia
Louisiana
Florida

uni


Lexington
ntucky
Roanoke
Virginia
Richmond
Daniel Boone
National Forest
Cherokee
National Forest
Norfolk
Virginia
Beach
Knoxville
Greensboro
A
North
Carolina
Raleigh
Asheville
Charlotte
Cleveland
Fayetteville
anooga
Cape Fear
Wilmington
Marietta
South
Carolina
Myrtle
Beach
Atlanta
Augusta
Macon
Charleston
olumbus
Georgia
Savannah

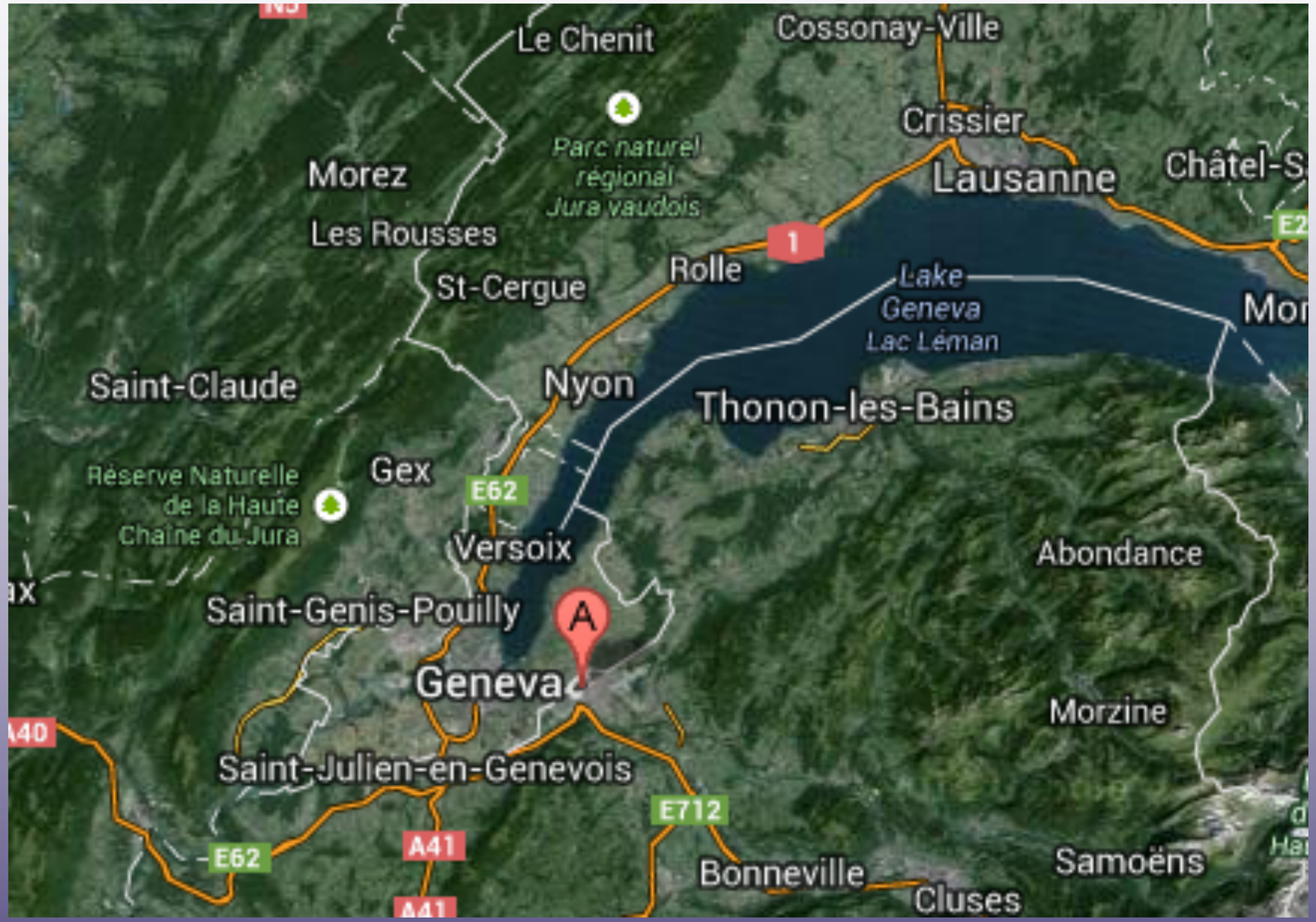
“erasmus”

2002


DR Congo
Rwanda
Burundi
Tanzania
Angola
Zambia
Malawi
A
Mozambique
Zimbabwe
Namibia
Botswana
Swaziland
Lesotho
South Africa

2005

2012


Le Chenit
Cossonay-Ville
Crissier
Parc naturel
régional
Jura vaudois
Morez
Lausanne
Châtel-S
Les Rousses
St-Cergue
Rolle
Lake
Geneva
Lac Léman
Saint-Claude
Nyon
Thonon-les-Bains
Réserve Naturelle
de la Haute
Chaîne du Jura
Gex
E62
Versoix
Abondance
Saint-Genis-Pouilly
A
Geneva
Morzine
Saint-Julien-en-Genevois
E712
A41
E62
Bonneville
Cluses
Samoëns

wingo@igalia.com

Hacking compiler tech at Igalia since 2011

Recently: ES6 generators in V8, SpiderMonkey (sponsored by Bloomberg)

Scheme migrant worker

So let’s talk about functions

♥ ♥ ♥

JS: the good part

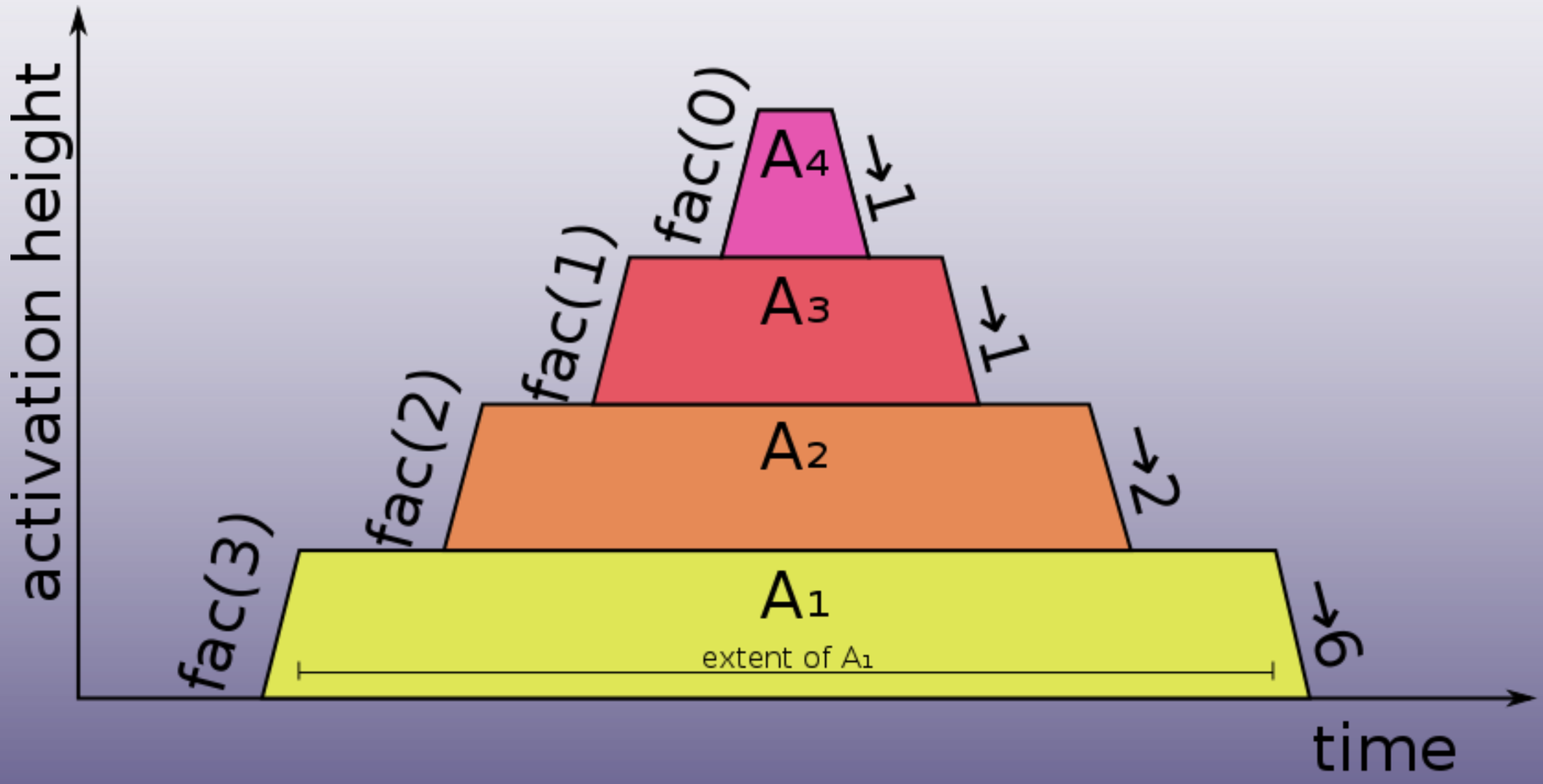
Elegant, clear local reasoning

Program modularity via procedural decomposition

♥ ♥ ♥

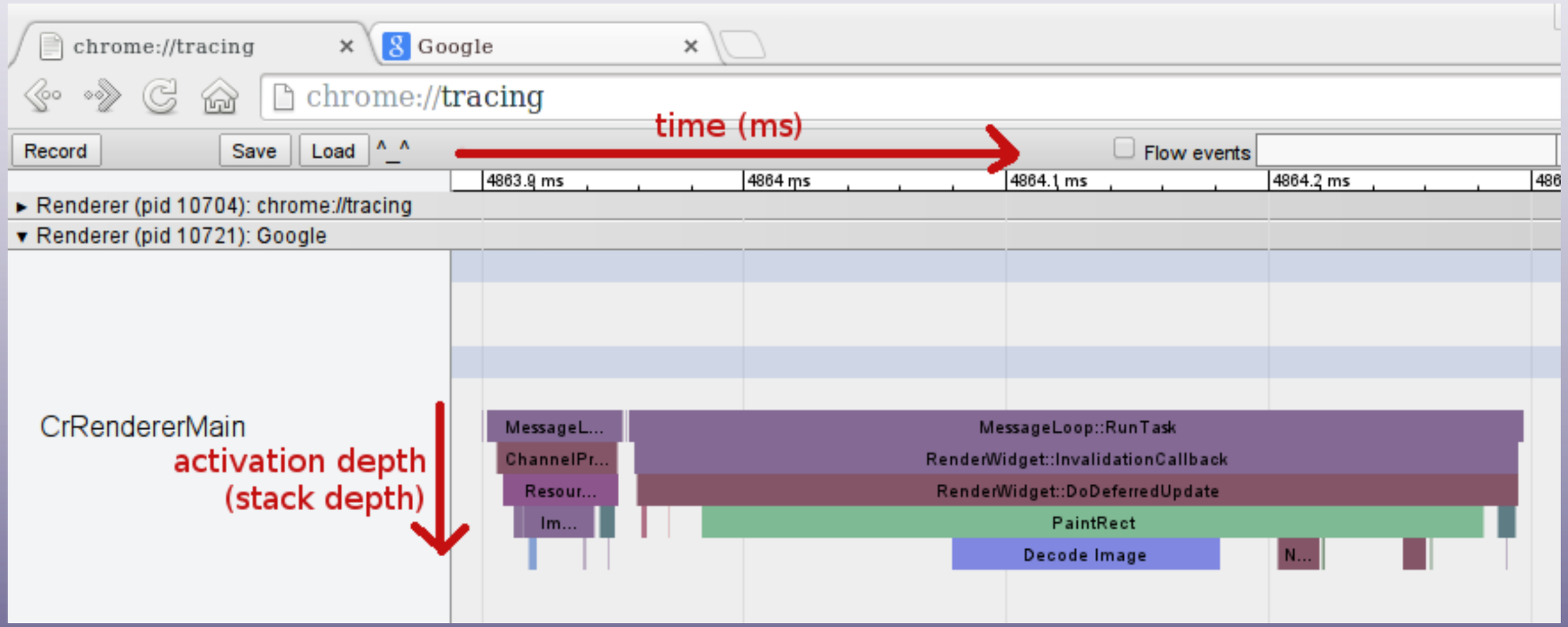
Function activations

```
function fac(n) { return n ? n * fac(n-1) : 1 }
fac(3)
```



Activation extents

Extent: the period of time that a function call is active



Extents in JS

Calling a JS function creates a new activation

- begins with a call
- ends with a return
- extends through time

JS function activations have linear extent

(Contrast to Scheme, Prolog)

Time constraints

Node

- servicing 100 clients/s: 10 ms/client

Browser

- 60 frames/s: 16 ms/frame

One disk seek is 10 ms

Professional deformation

Constraints on time

Constraints on activation extents

Constraints on functions

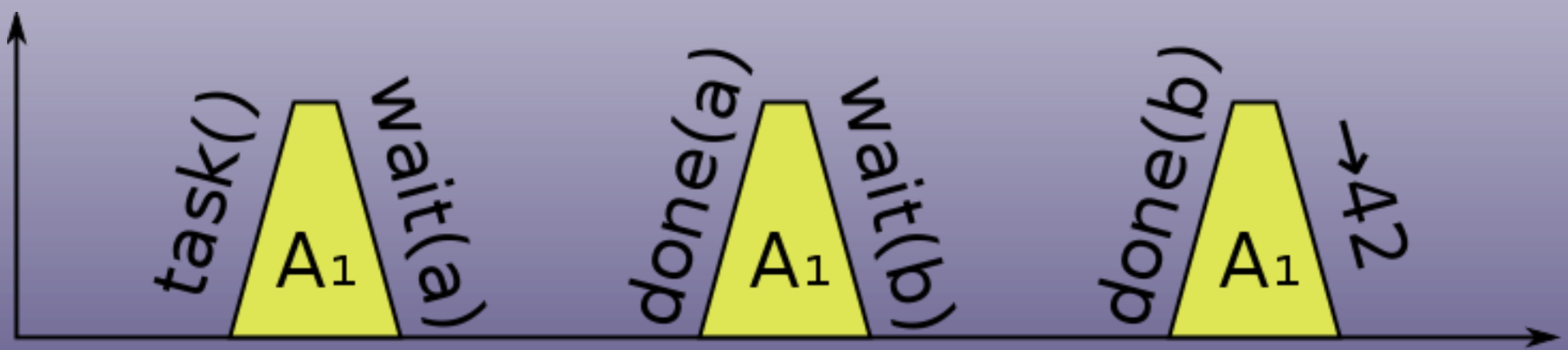
Deformation of programs

Fraction-of-an-action callback/errback hell

Long-extent activations desirable

Asynchronous tasks (XHR, Node)

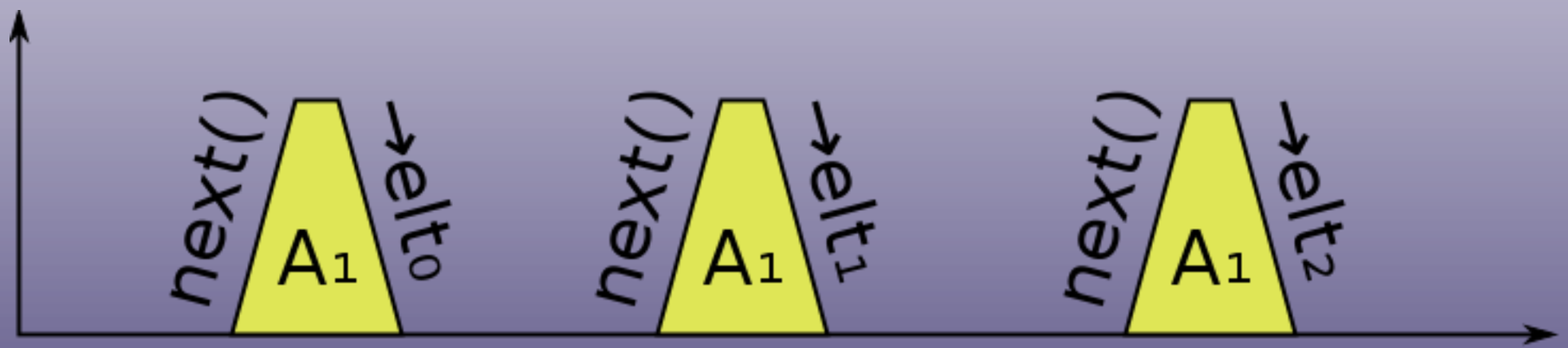
```
function^ task(x) {
  await baz(await bar(x));
  return 42;
}
```



Long-extent activations desirable

Iteration, lazy streams

```
function foreach(f, iterable) {
   for (var elt of iterable)
      f(elt);
}
```



Generators

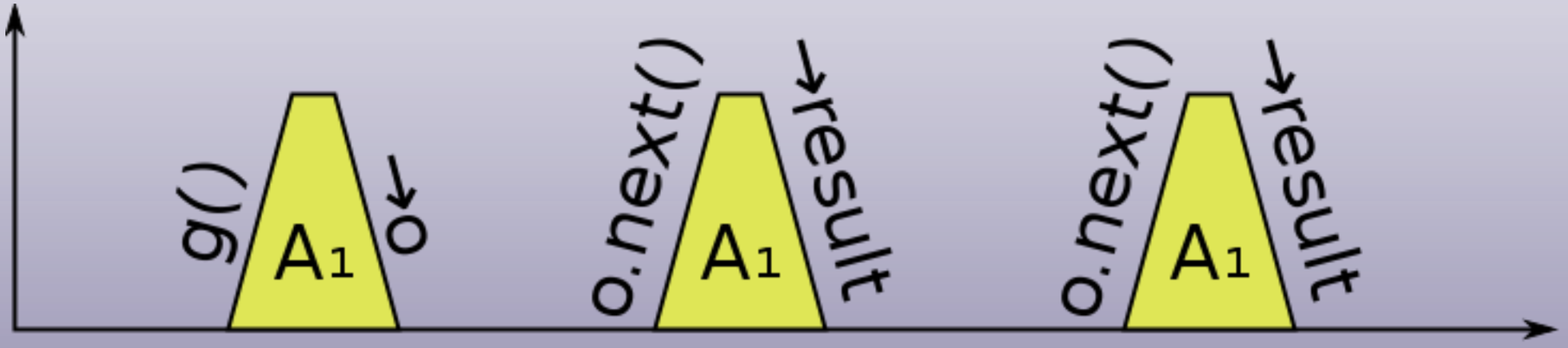
Functions whose activations can suspend

```
function* g() {}
function* g() {
  for (let x = 0; ; x++)
    yield x
}
```

(`yield` valid only in `function*`)

Generator basics

```
function* g() { yield 42; return 10 }
var o = g();
o.next() → { value: 42, done: false }
o.next() → { value: 10, done: true }
```



Functions and objects

Terminology confusing

Generator function: `function* g() {}`

Generator object: `var o = g()`

Generator objects are iterators: `'next' in o`

Objects, instances of functions: `o instanceof g`

Yield expressions

Not just a statement!

```
function* g() { return yield 42 }
var o = g();
o.next() → { value: 42, done: false }
o.next('hai') → { value: 'hai', done: true }
```

Argument to `next` becomes value of corresponding `yield`

Throwing into generators

Generator objects also have `throw` methods

```
function* g() {
  try { yield 10 }
  catch (e) { return e }
}

var o = g()
o.next() → { value: 10, done: false }
o.throw(42) → { value: 42, done: true }
```

Applications

Asynchronous tasks

Iteration

(Lazy streams)

Asynchronous tasks

With promises

```
function process(url, f) {
  function request(url) { foo }
  function update(url, updated) { bar }
  function handleError(e) { baz }
  return request(url)
    .then(data => update(url, f(data)))
    .then(_ => true, handleError);
}
```

Asynchronous tasks

With generators

```
Q.async(function* process(url, f) {
  try {
    var data = yield foo;
    var updated = f(data)
    yield bar;
    return true;
  } catch (e) {
    baz;
  }
})
```

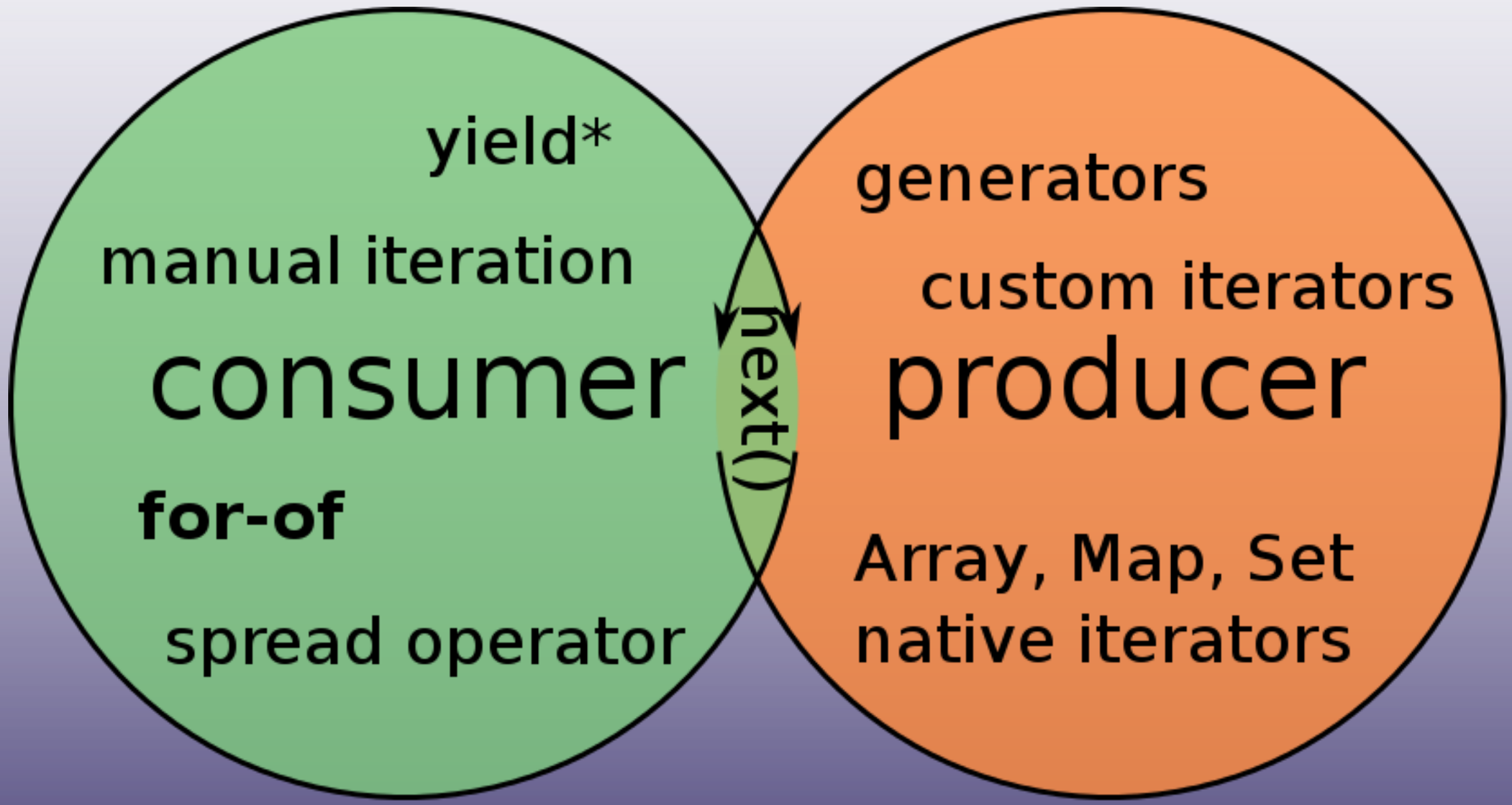
Reclaiming “JS: the good part”

State in local variables

Native JS control flow

The right number of names

Iteration is a form of concurrency



Iteration

Iterables: `@@iterator in o`

`@@iterator` is a “well-known symbol”; not a string

Getting iterator from iterable: `o[@@iterator]()`

Iterables: `Array`, `Map`, `Set`, generators

All iterators are also iterables

for-of

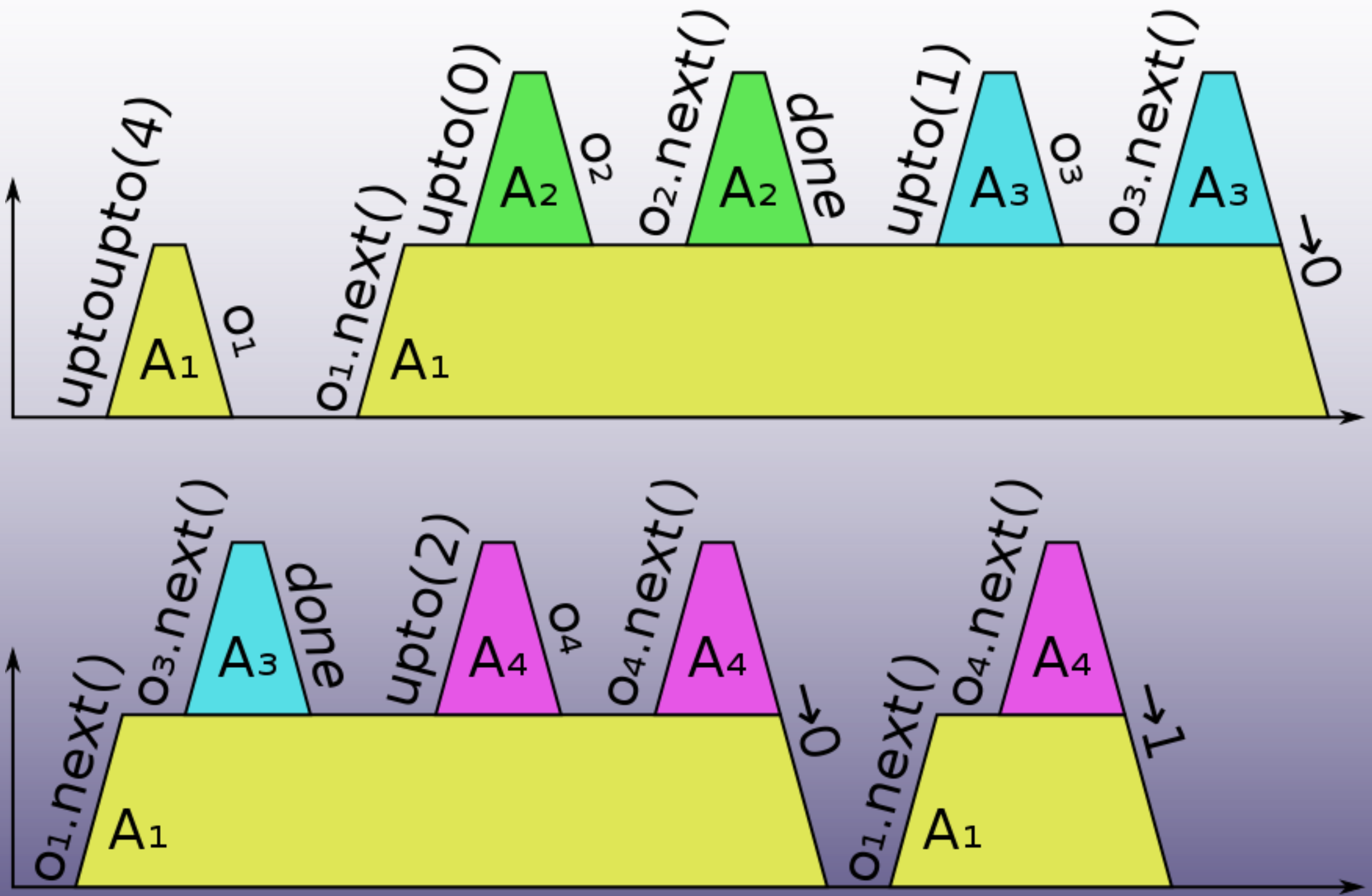
```
for (elt of iterable) body
for (elt of [1, 2, 3]) print(elt);
function* upto(n) {
  for (var x = 0; x < n; x++)
    yield x
}

for (elt of upto(5)) print(elt);

[for (x of upto(5)) x]
  // → [ 0, 1, 2, 3, 4 ]
```

yield*: generator composition

```
function* uptoupto(n) {
  for (let x = 0; x < n; x++)
    yield* upto(x);
}
```

```
[for (x of uptoupto(3)) x]
  // → [ 0, 0, 1, 0, 1, 2 ]
```

```
[for (x of uptoupto(3)) x]
```



Iteration over custom data structures

```
Trie.prototype[@@iterator] = iterateTrie;
for (var elt of trie) { ... }
```

http://wingolog.org/archives/2013/10/07/es6-generators-and-iteration-in-spidermonkey

Availability

Firefox (stable)

Chrome (with experimental flag)

Node.js (with experimental flag)

`@@iterator` story is complex, see my blog

Regenerator: `http://facebook.github.io/regenerator/`

Traceur: `http://es6fiddle.net/`

Asynchrony and promises

Many libraries; I used Q

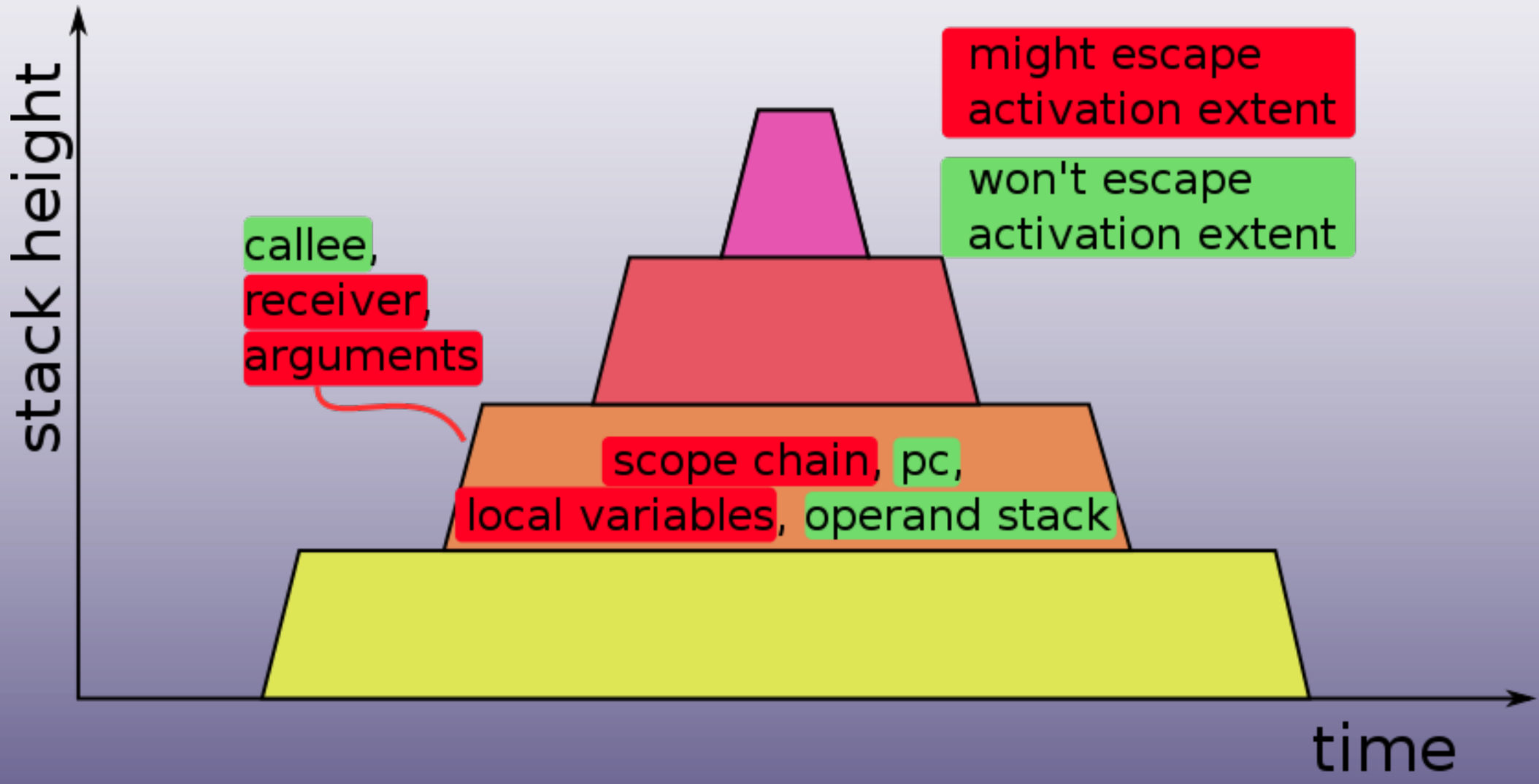
Talk by Forbes Lindesay: `http://www.youtube.com/watch?v=qbKWsbJ76-s`

ES6 things left to implement

Generator comprehensions (Firefox has them with the old syntax, not on by default)

Generator methods

How functions are implemented



Take advantage of linear extent

State that does not escape the extent of an activation can be implemented more efficiently

Example: If a local doesn't escape, it doesn't need to be on the heap

Example: If no locals escape, no scope chain node need be created

Nested closures, `with`, direct `eval`, some arguments access can cause escape

Generators have nonlinear extent

Flag all locals as “escaped” so they are allocated on scope chain

To suspend, package up additional state (pc, callee, scope chain, operand stack) in heap object

To restore, splat it back on the stack

Generator objects are shallow delimited continuations

v8:src/objects.h

```
kFunctionOffset = JSObject::kHeaderSize;
kContextOffset = kFunctionOffset + kPointerSize;
kReceiverOffset = kContextOffset + kPointerSize;
kContinuationOffset = kReceiverOffset + kPointerSize;
kOperandStackOffset = kContinuationOffset + kPointerSize;
kStackHandlerIndexOffset = kOperandStackOffset + kPointerSize;
kSize = kStackHandlerIndexOffset + kPointerSize;
```

StackHandlerIndex is a V8 wart

Q & A

Questions?

wingo@igalia.com

http://wingolog.org/