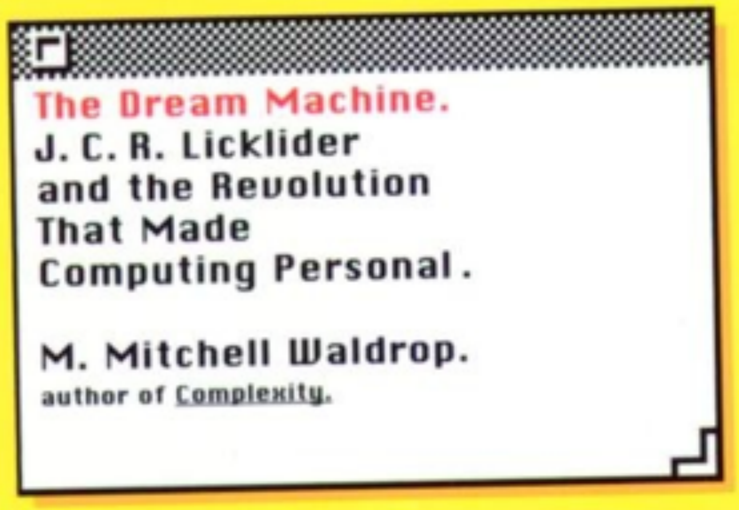
Immutability,

or Putting the Dream Machine to Work

cognitect



The Dream Machine.

J. C. R. Licklider and the Revolution That Made Computing Personal.

M. Mitchell Waldrop.

author of Complexity.

"Waldrop's account of [Licklider's] and many others' world-transforming contributions is compelling."
—John Allen Paulos, The New York Times Book Review

The trie memory scheme is inefficient for small memories, but it be- comes increasingly efficient in using available storage space as memory size increases. The attractive features of the scheme are these: 1) The retrieval process is extremely simple. Given the argument, enter the standard ini- tial register with the first character, and pick up the address of the second. Then go to the second register, and pick up the address of the third, etc. 2) If two arguments have initial characters in common, they use the same storage space for those characters.

-J.C.R. Licklider, "Man-Computer Symbiosis" 1960

Para Format
Tabs/Tables

Learning about Smalltalk (Xerox, Alan Kay, PARC)

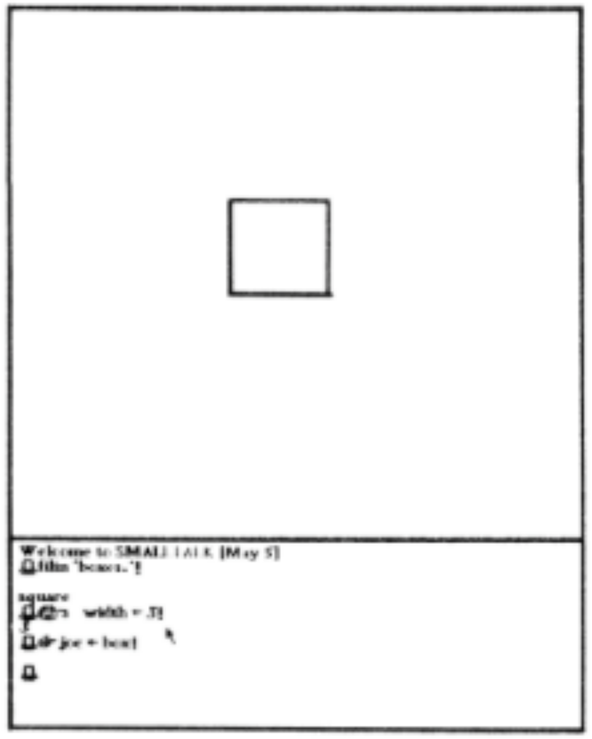
Learning About Smalltalk

by Marian Goldeen

My name is Marian Goldeen. I'm an eighth grade student at Jordan Junior High School in Palo Alto, California, and I would like to tell you about how I got started working with computers at Xerox and the class I taught.

It all started in December, 1973 when I was in the seventh grade. There were four people in my class who were interested in taking a course about the computer language Smalltalk at Xerox.

When we first started we were shown how to start the machines up, and file in our one file, which had already been written onto our disks. These files contained some programs that would draw boxes like this.



```
Welcome to SMALLTALK [May 5]
filin 'boxes.'!

square
do   width ← 3!

do joe ← box!

```

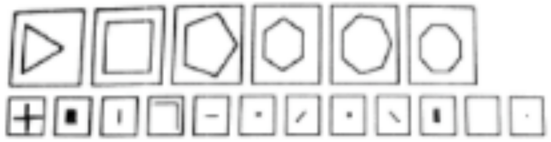
These boxes could turn on their axes,

and shrink.

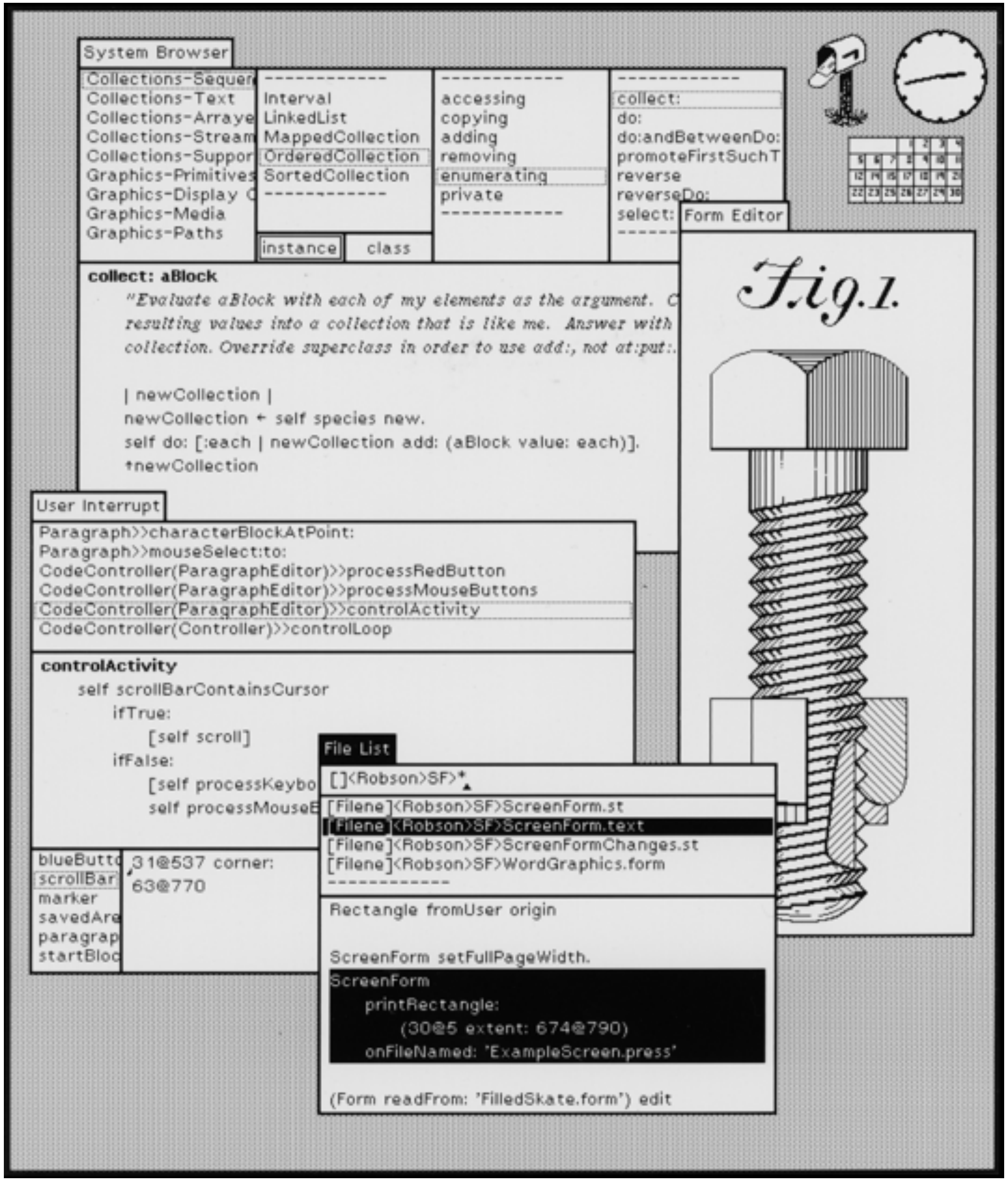
Later on we learned how to change the programs which had been created and drew these boxes so that we could do different things with them, for instance, move them to different places on the screen.

There was a little rectangular object to the side of the keyboard, called the mouse. When you moved the mouse around a corresponding pointer on the screen moved around too. We learned how to make the boxes follow the mouse pointer.

After we had learned just about everything there was to know about boxes we were able to create our own programs (Gulp). I don't know what the two boys in the class did, but Colleen and I created a painting program. It was fairly complicated. To run it you first had to set up the menu.



You would point with the mouse to the box that contained the shape you wanted to draw with, then press the top mouse button. Now the shape would be a paint



```
collect: aBlock
    "Evaluate aBlock with each of my elements as the argument. C
    resulting values into a collection that is like me.  Answer with
    collection. Override superclass in order to use add:, not at:put:.

    | newCollection |
    newCollection ← self species new.
    self do: [:each | newCollection add: (aBlock value: each)].
    ↑newCollection
```

```
Paragraph>>characterBlockAtPoint:
Paragraph>>mouseSelect:to:
CodeController(ParagraphEditor)>>processRedButton
CodeController(ParagraphEditor)>>processMouseButtons
CodeController(ParagraphEditor)>>controlActivity
CodeController(Controller)>>controlLoop
```

```
controlActivity
    self scrollBarContainsCursor
        ifTrue:
            [self scroll]
        ifFalse:
            [self processKeybo
            self processMouseB
```

```
31@537 corner:
63@770
```

```
[]<Robson>SF>*
[Filene]<Robson>SF>ScreenForm.st
[Filene]<Robson>SF>ScreenForm.text
[Filene]<Robson>SF>ScreenFormChanges.st
[Filene]<Robson>SF>WordGraphics.form
------------
```

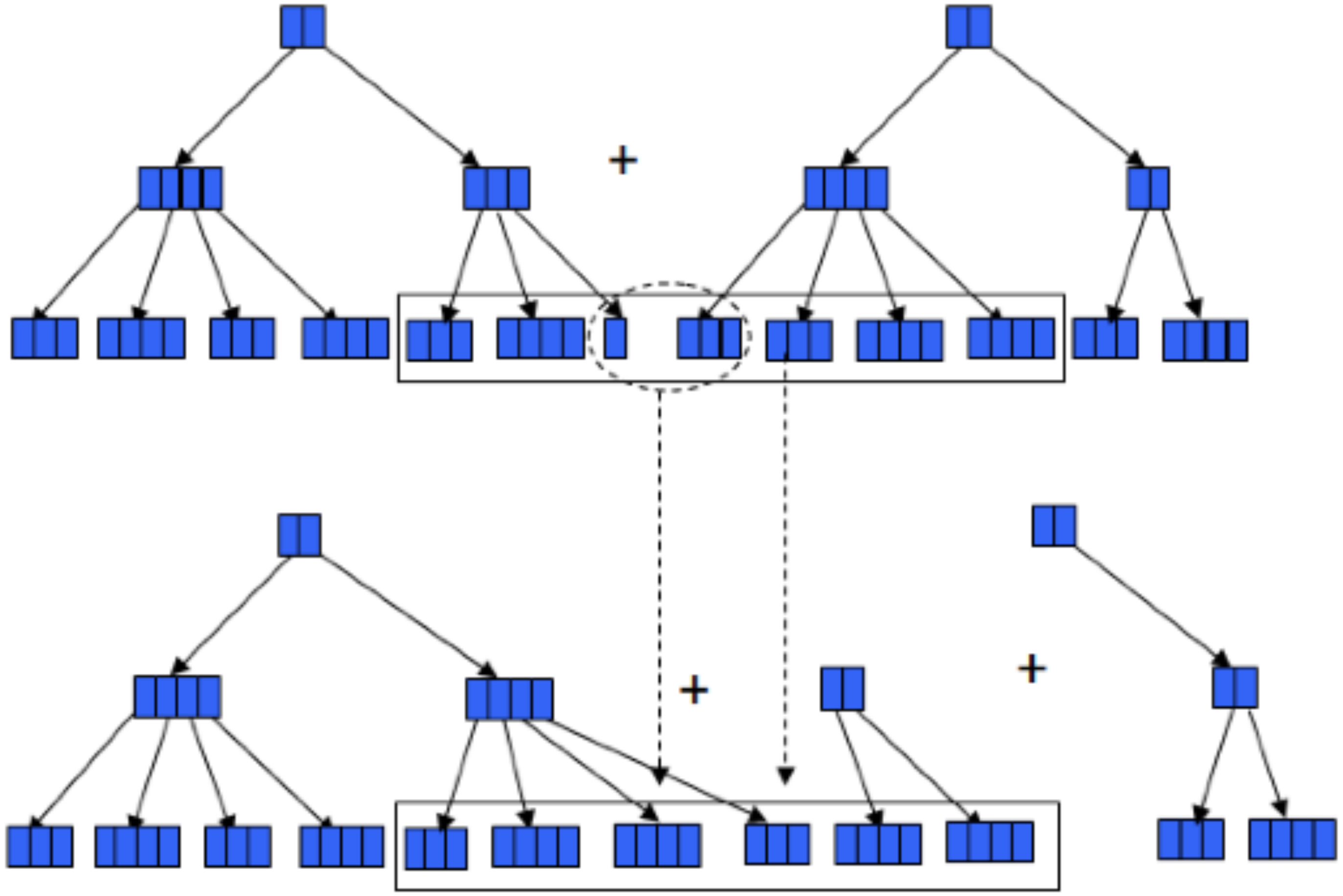
```
Rectangle fromUser origin

ScreenForm setFullPageWidth.
ScreenForm
    printRectangle:
        (30@5 extent: 674@790)
    onFileNamed: 'ExampleScreen.press'

(Form readFrom: 'FilledSkate.form') edit
```

Model-View-Controller

- first formulated by Trygve Reenskaug Adele Goldberg and others at Xerox PARC in 1979

- long shadow, the basic concepts still prevalent today.

- At a very abstract level MVC is a sound separation of concerns
- Implementations leave much to be desired
    - *Stateful objects everywhere*

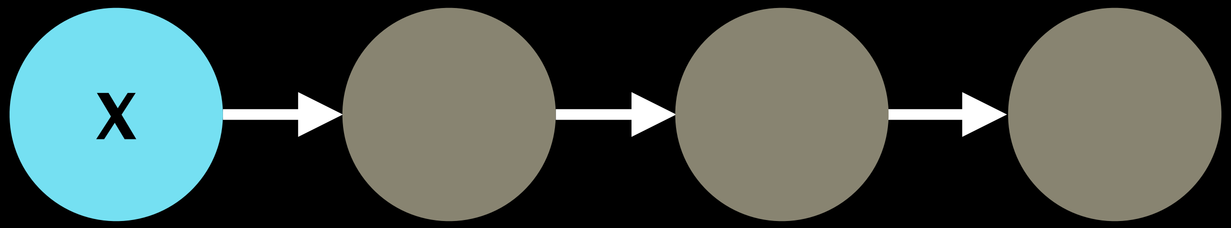


HACKER
SCHOOL
2012

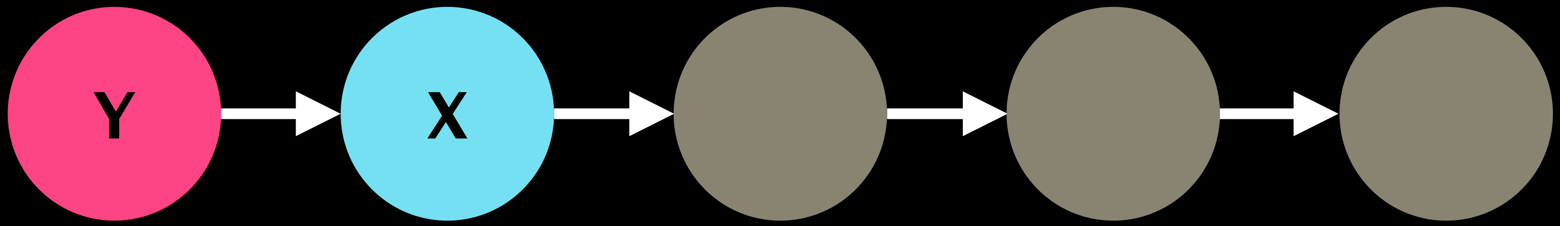
Functional

- immutable values, not mutable objects
- “change” returns a new value, leaving the old one unmodified
- they’re persistent
- they’re fast

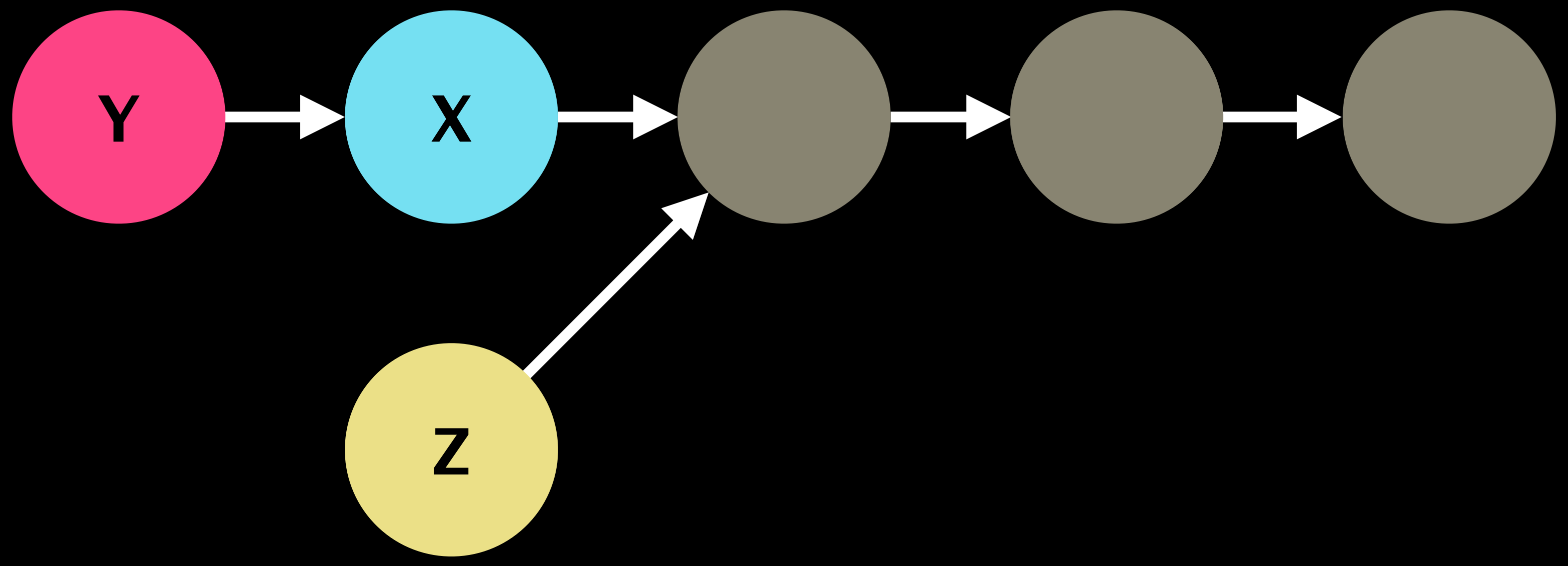
Simple example:



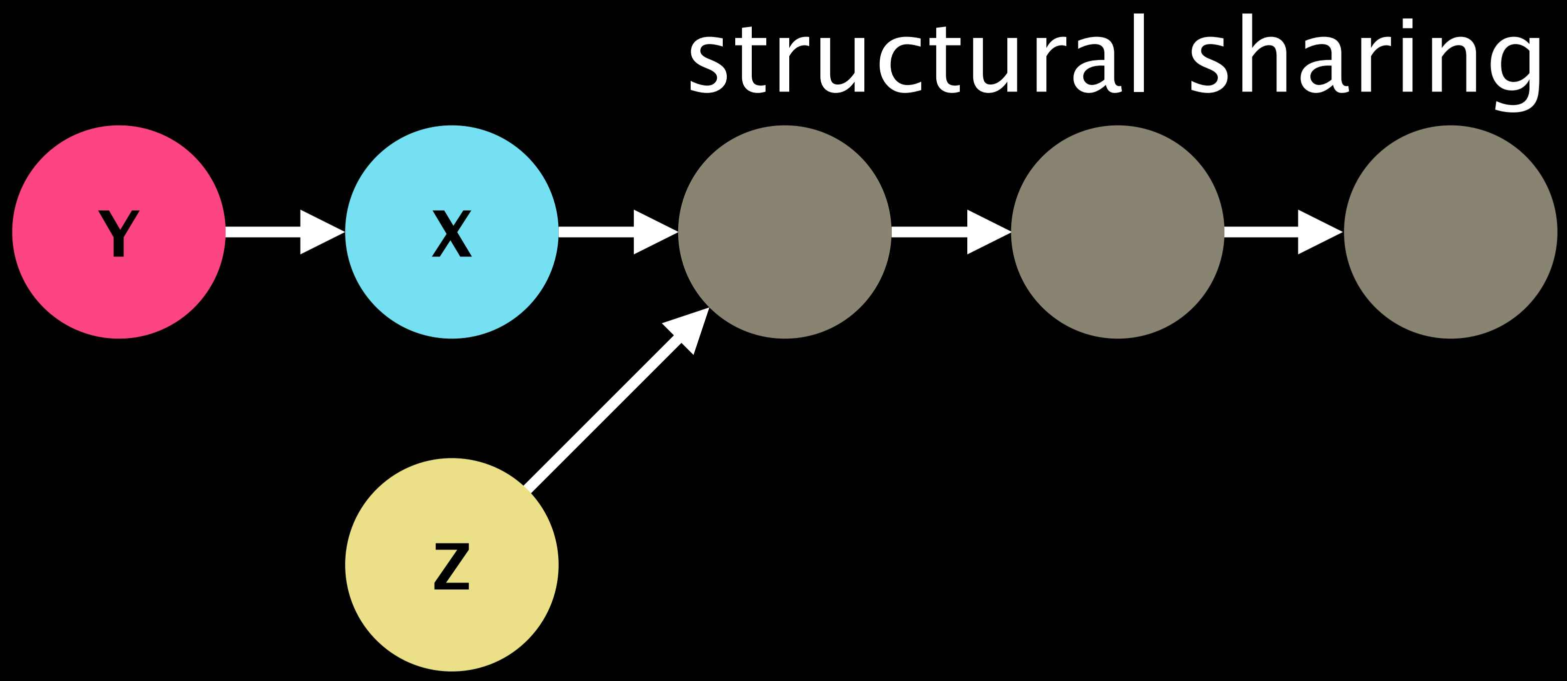
Simple example:



Simple example:



Simple example:



Sharing structure

- space efficiency
- computational efficiency – avoids copying

Phil Bagwell

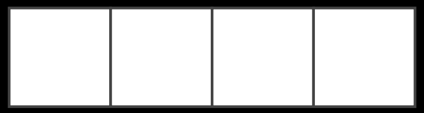
- Array Mapped Trie
- Hash Array Mapped Trie

Bitmapped Vector Trie

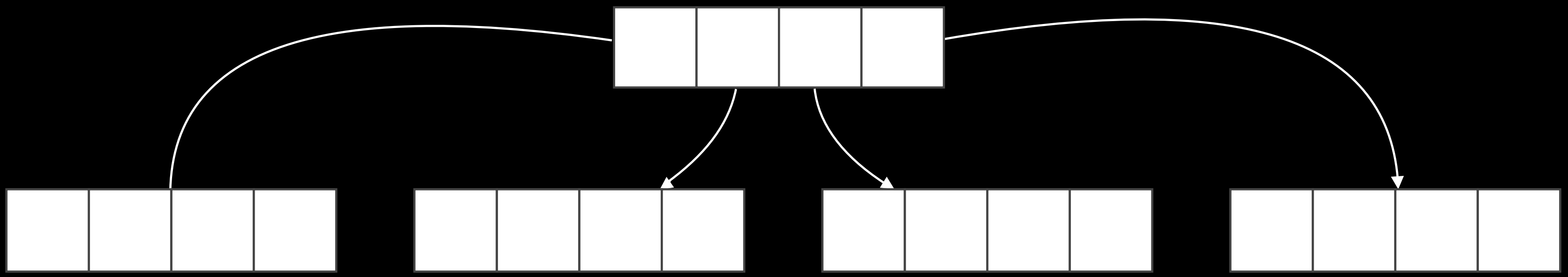
- data lives in the leaves
- e.g. prefix tree used for string lookup
- bitwise trie

Persistent Vector

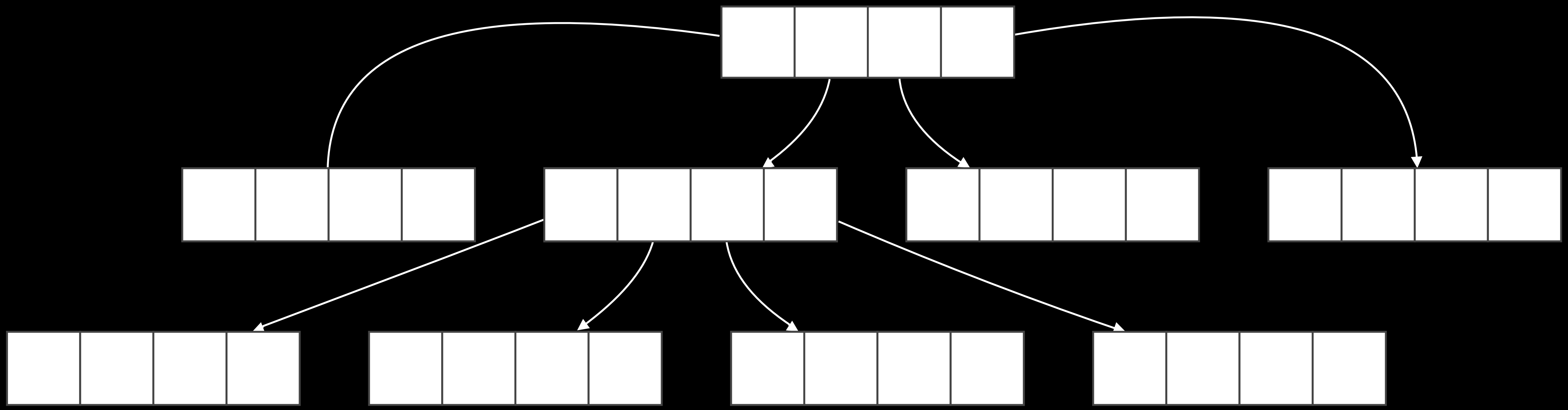
Persistent Vector



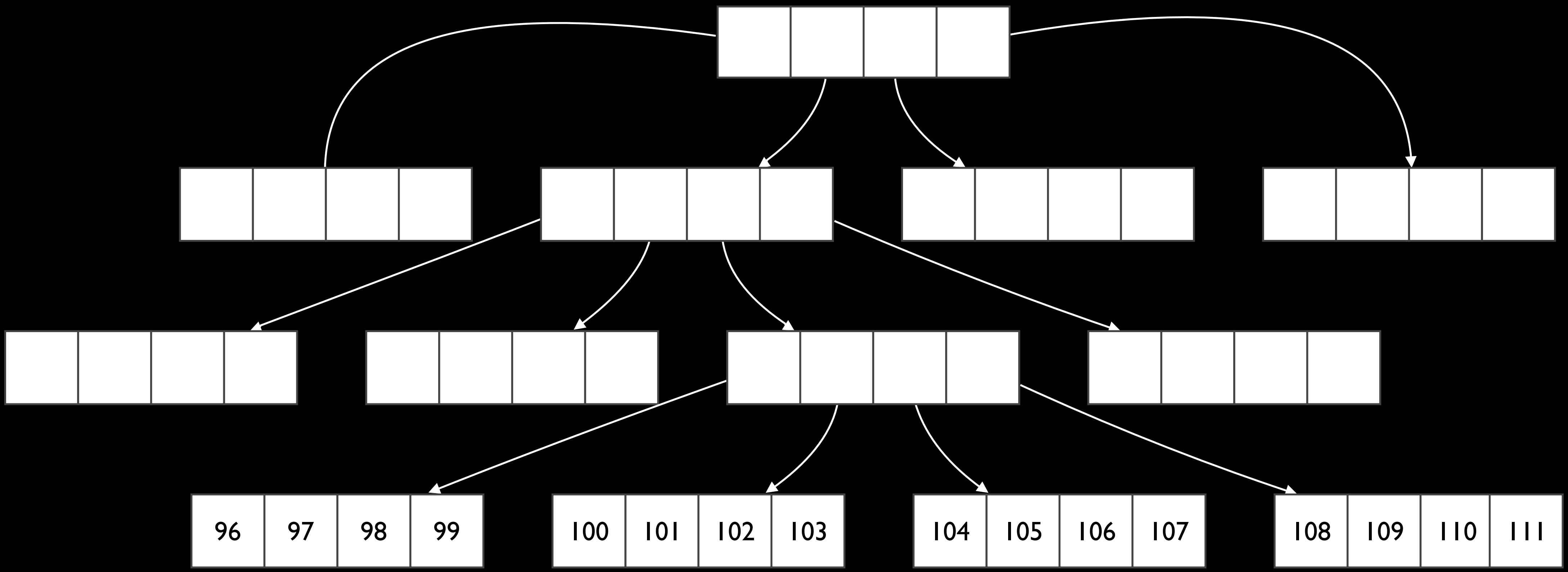
Persistent Vector



Persistent Vector



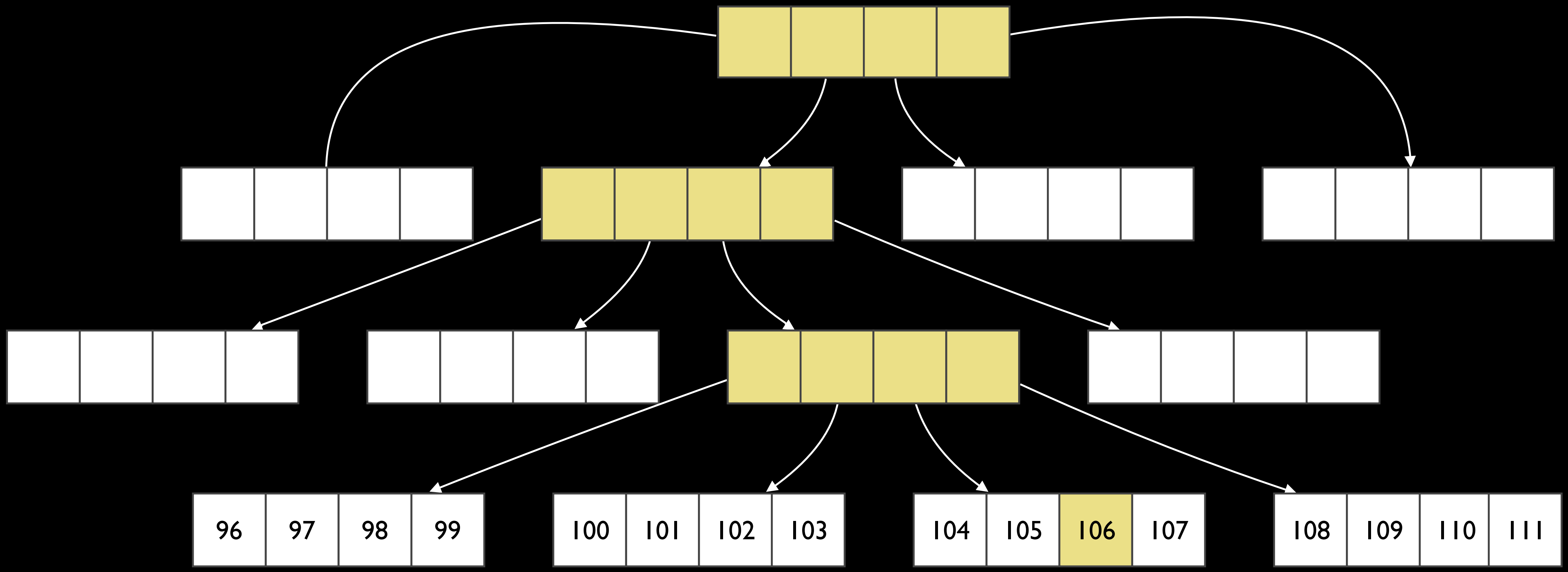
Persistent Vector



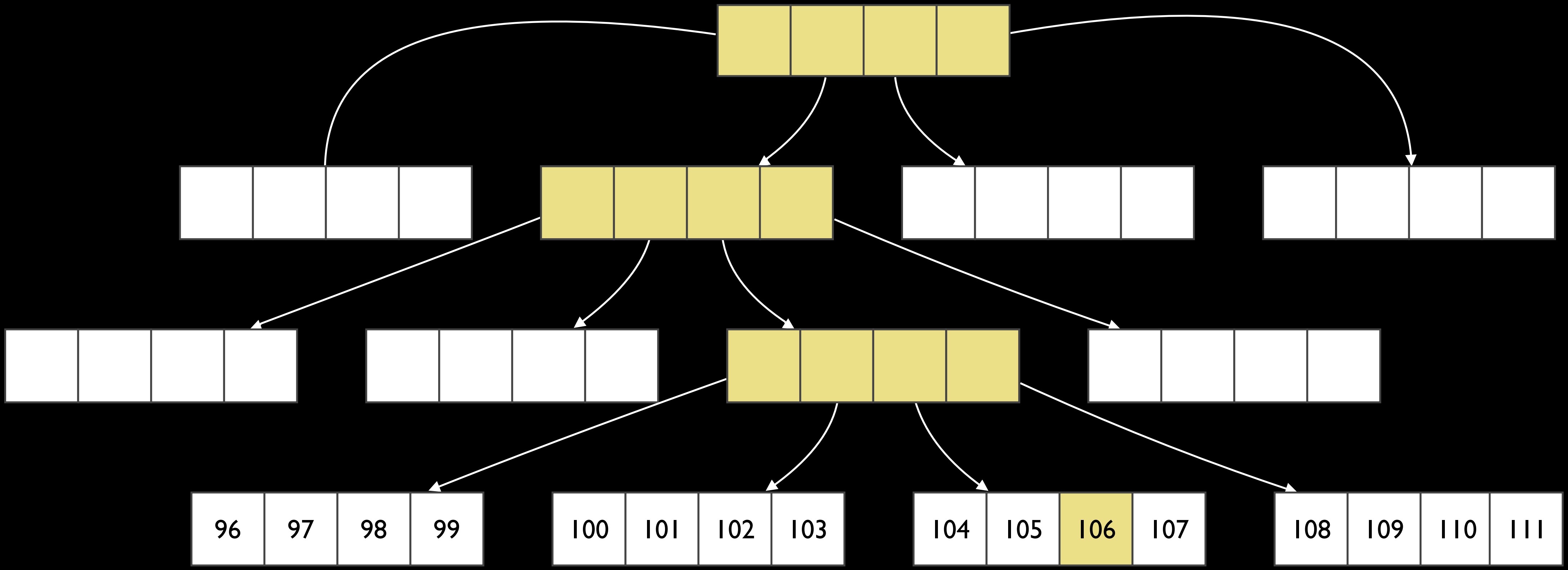
Persistent Vector

getindex

Persistent Vector

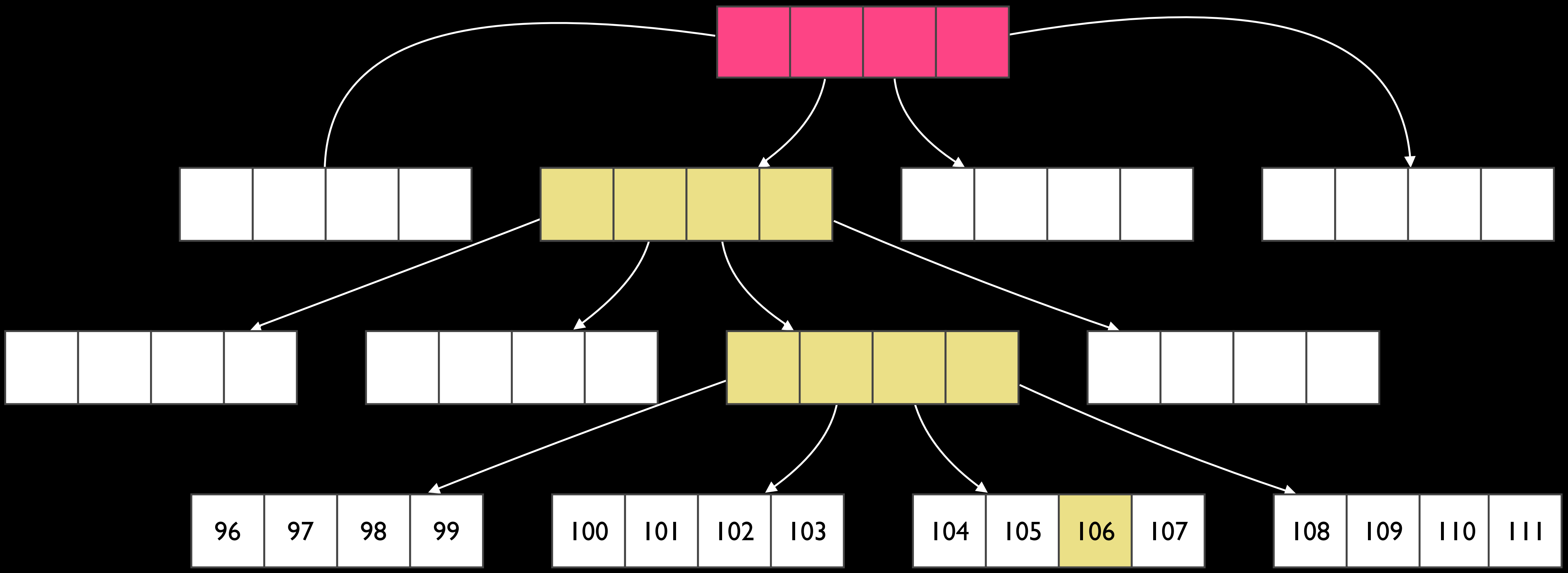


Persistent Vector



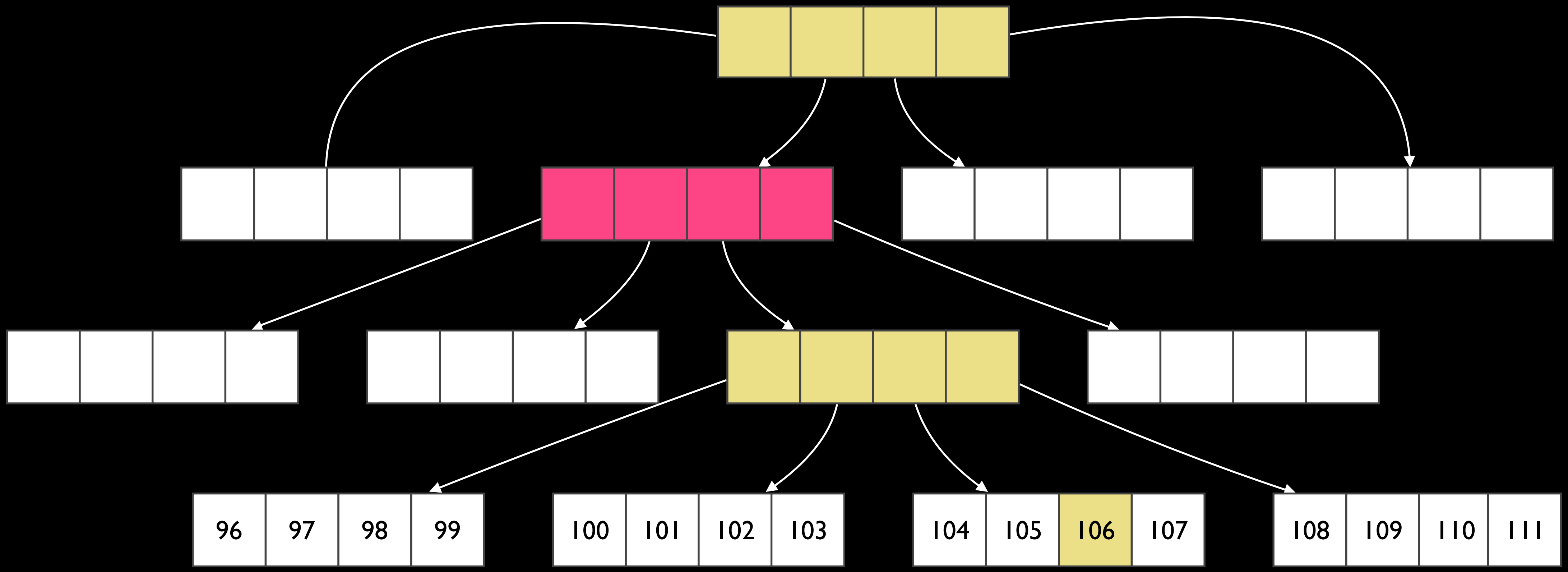
0b01101010

Persistent Vector



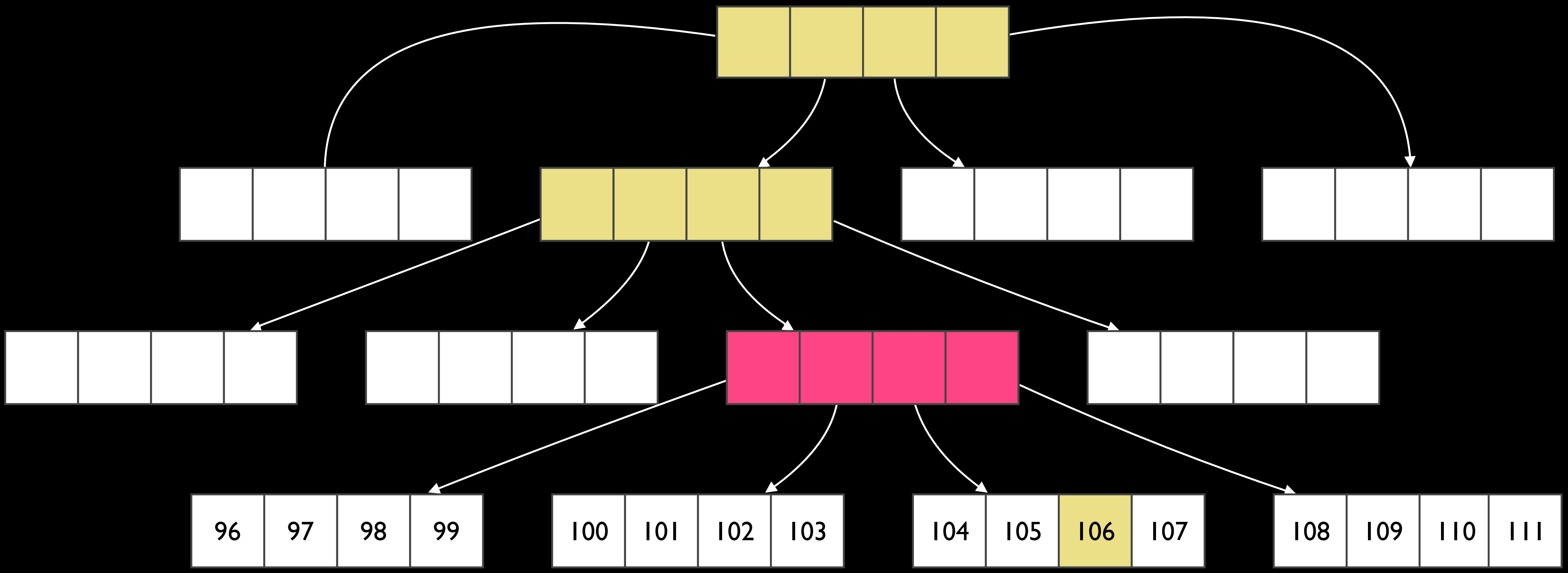
0b01101010

Persistent Vector



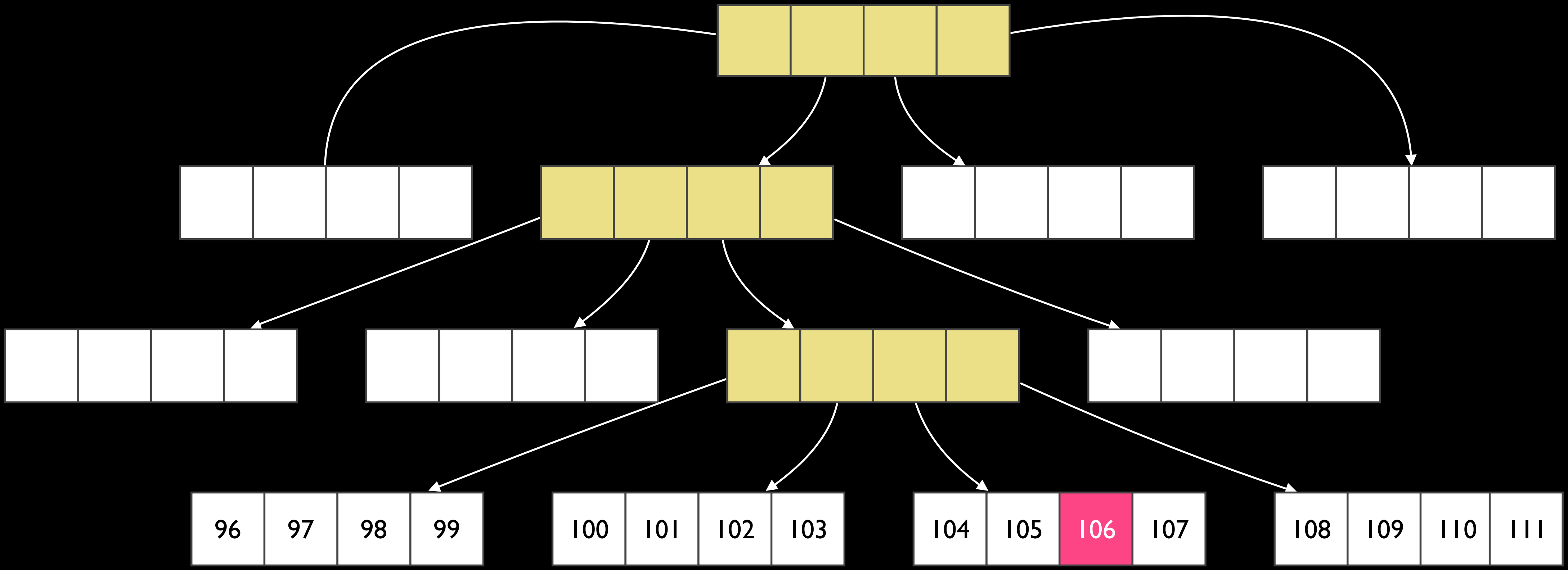
0b01101010

Persistent Vector



0b01101010

Persistent Vector

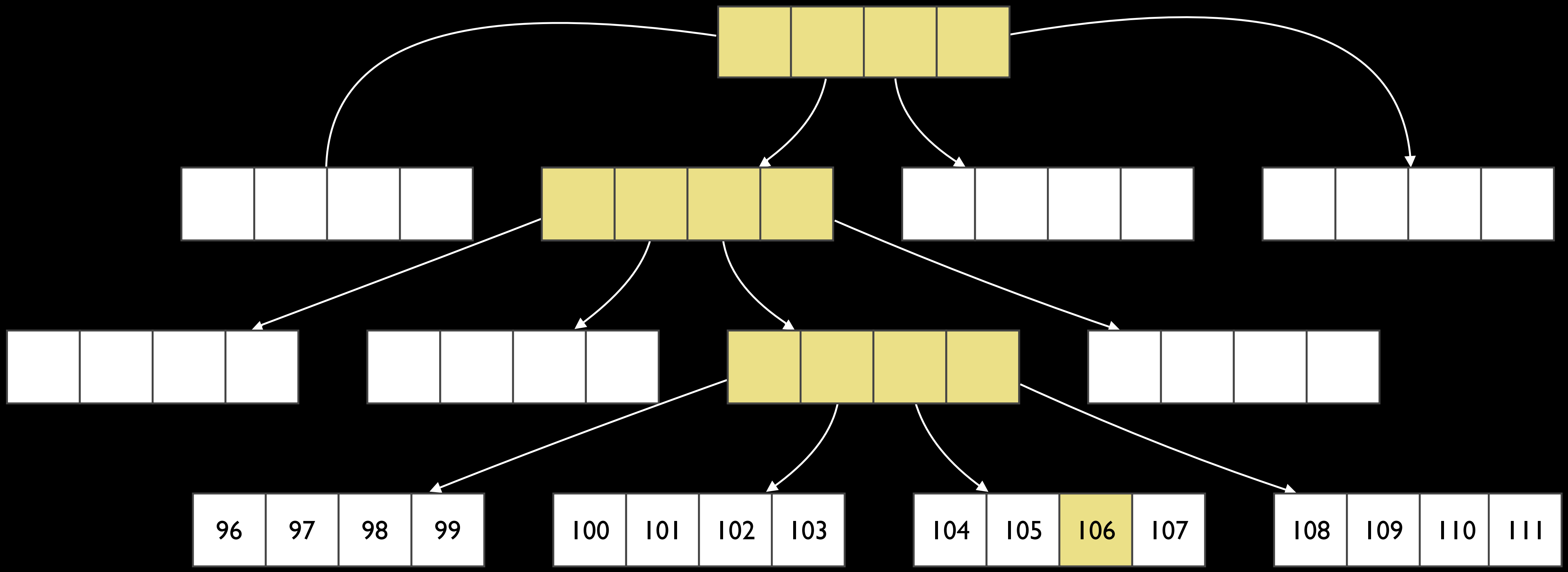


0b0110101010

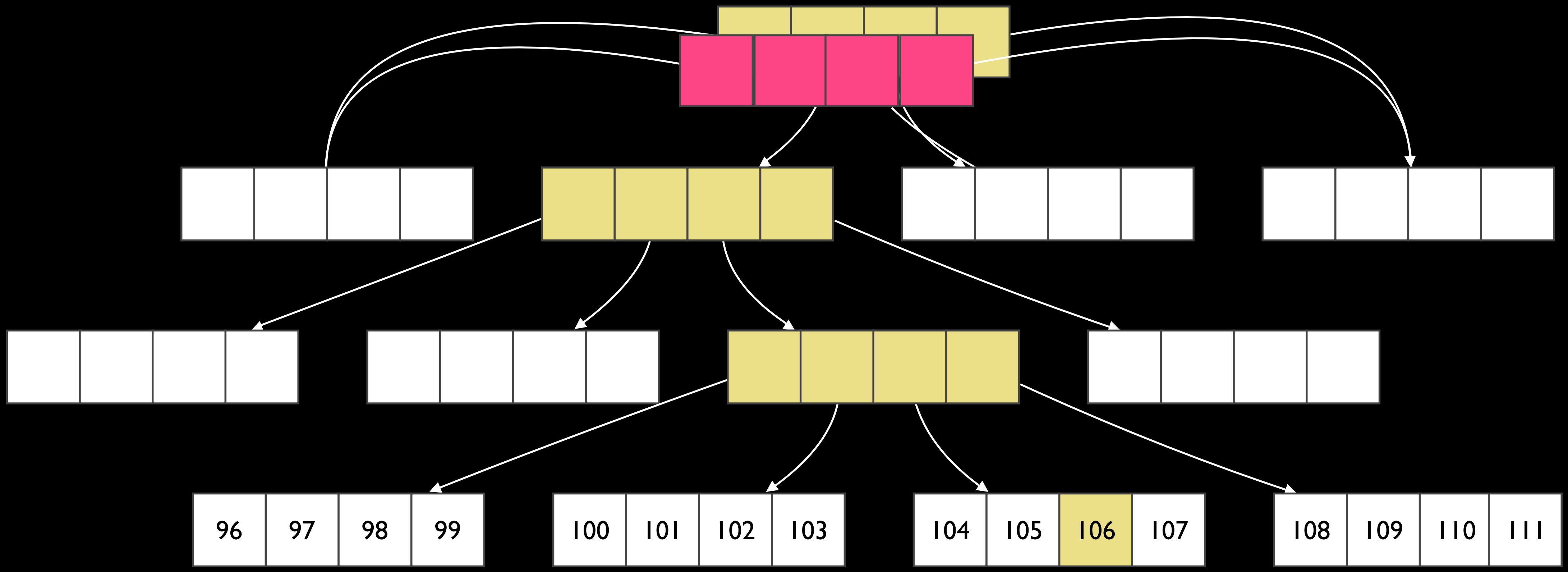
Persistent Vector

assoc

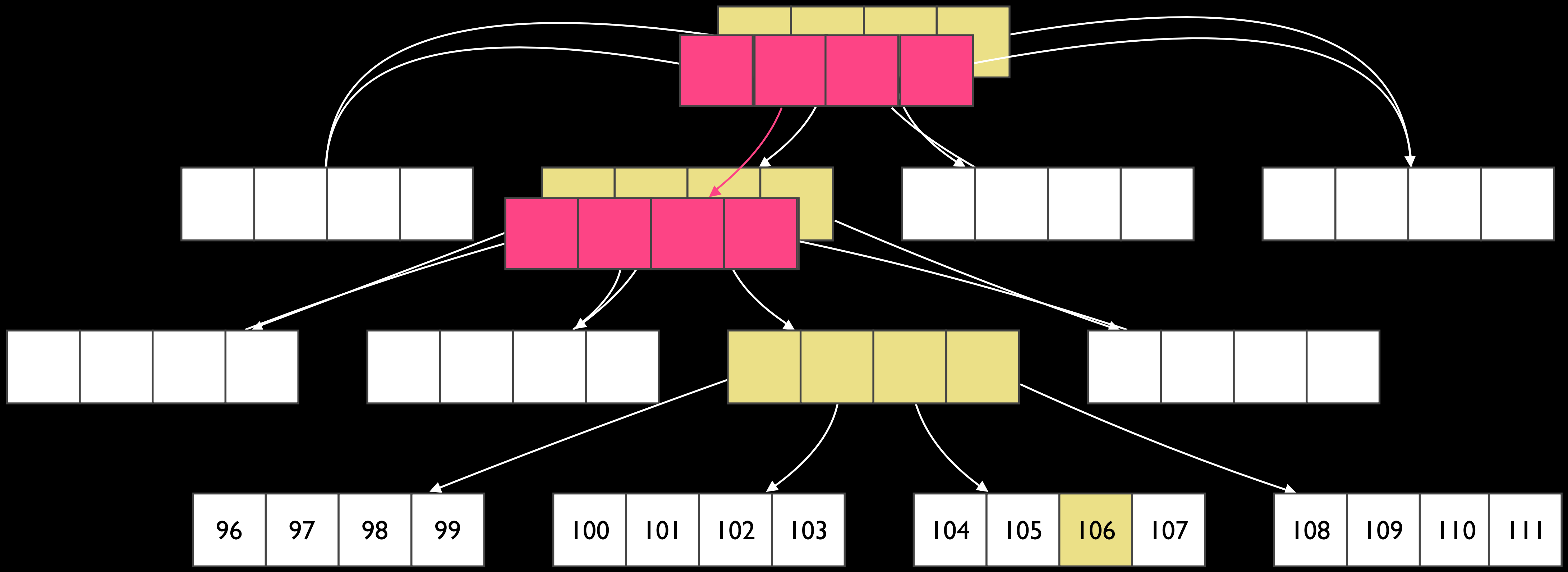
Persistent Vector



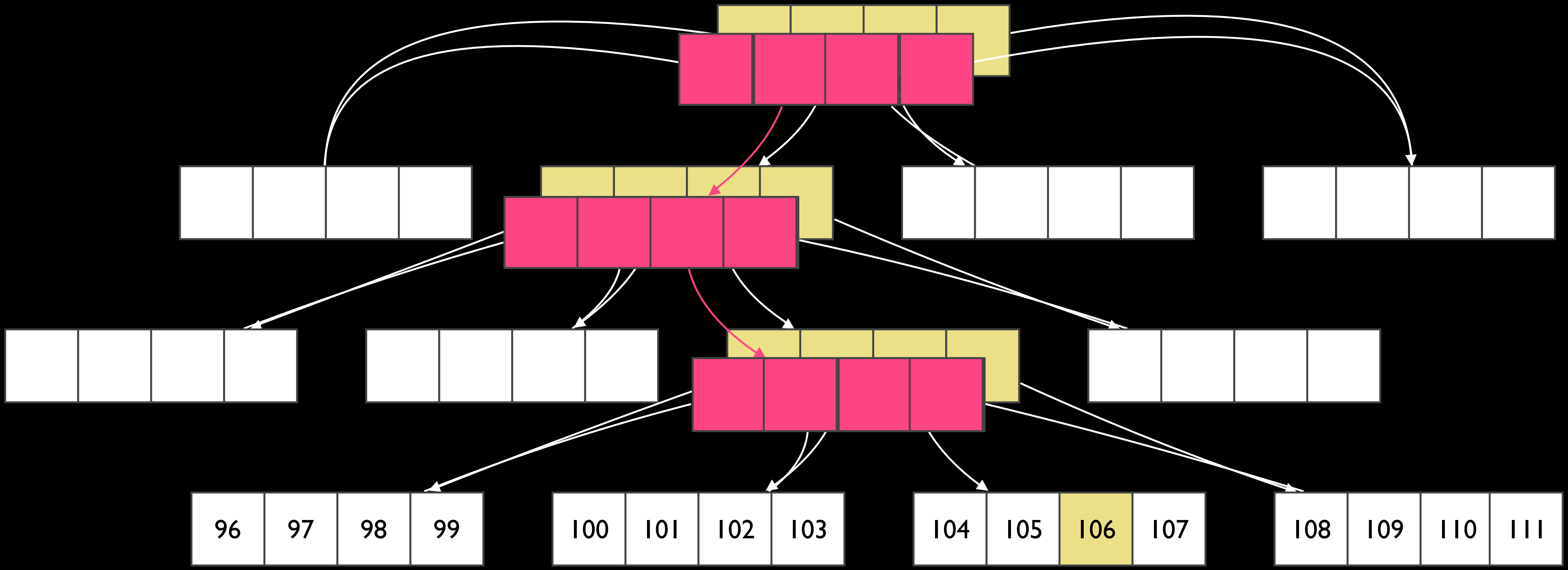
Persistent Vector



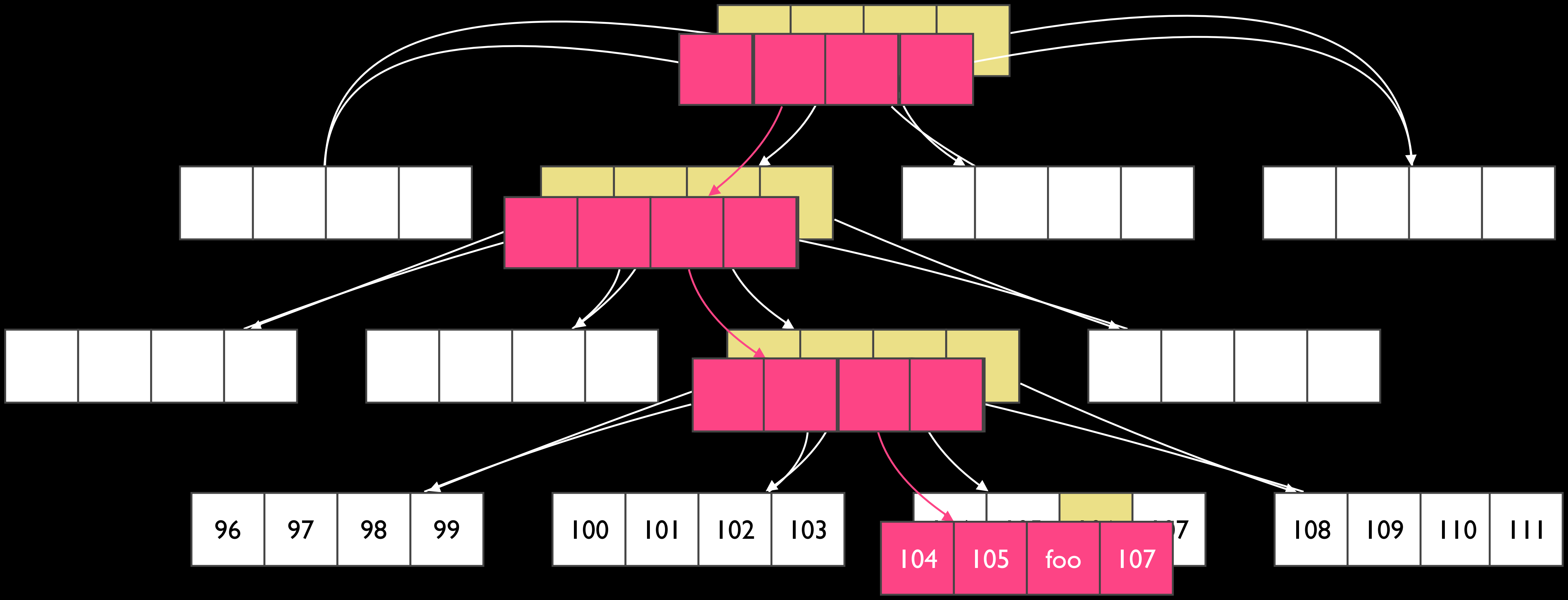
Persistent Vector



Persistent Vector



Persistent Vector

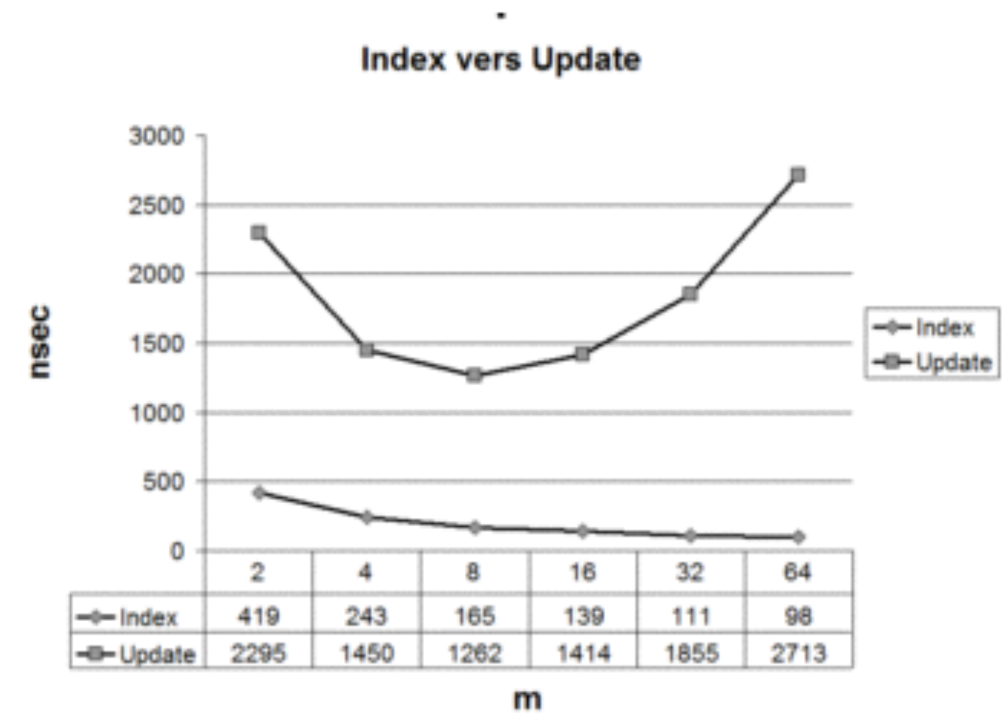


Persistent Vector

Length 4 internal vectors?

Persistent Vector

32



m	2	4	8	16	32	64
Index	419	243	165	139	111	98
Update	2295	1450	1262	1414	1855	2713

Figure 2. Time for index and update, depending on m

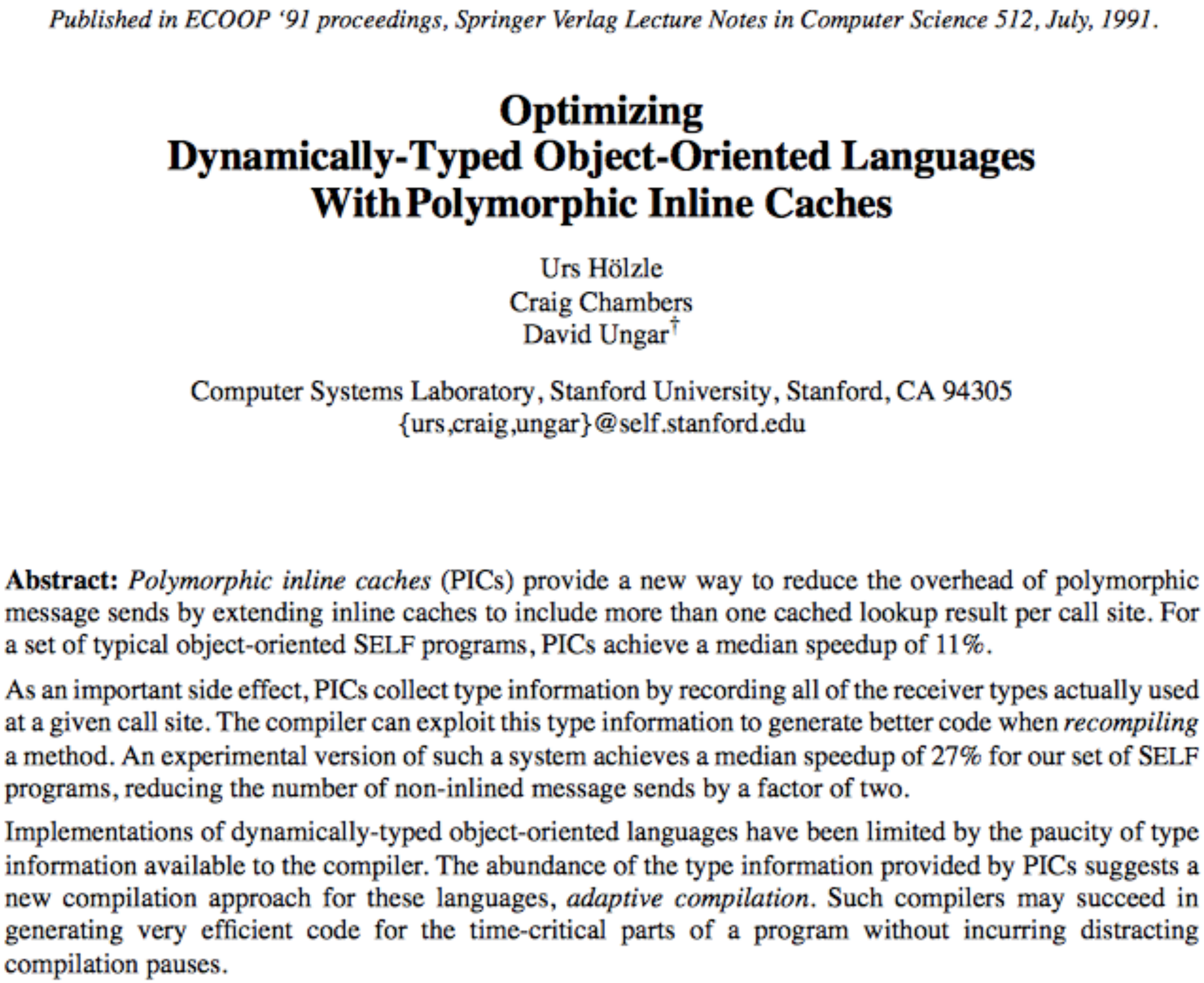
From Bagwell, Rompf 2011

$$32^7$$

34,359,738,368

elements

demo



Published in ECOOP '91 proceedings, Springer Verlag Lecture Notes in Computer Science 512, July, 1991.

Optimizing Dynamically-Typed Object-Oriented Languages With Polymorphic Inline Caches

Urs Hölzle
Craig Chambers
David Ungar[†]

Computer Systems Laboratory, Stanford University, Stanford, CA 94305
{urs,craig,ungar}@self.stanford.edu

Abstract: *Polymorphic inline caches* (PICs) provide a new way to reduce the overhead of polymorphic message sends by extending inline caches to include more than one cached lookup result per call site. For a set of typical object-oriented SELF programs, PICs achieve a median speedup of 11%.

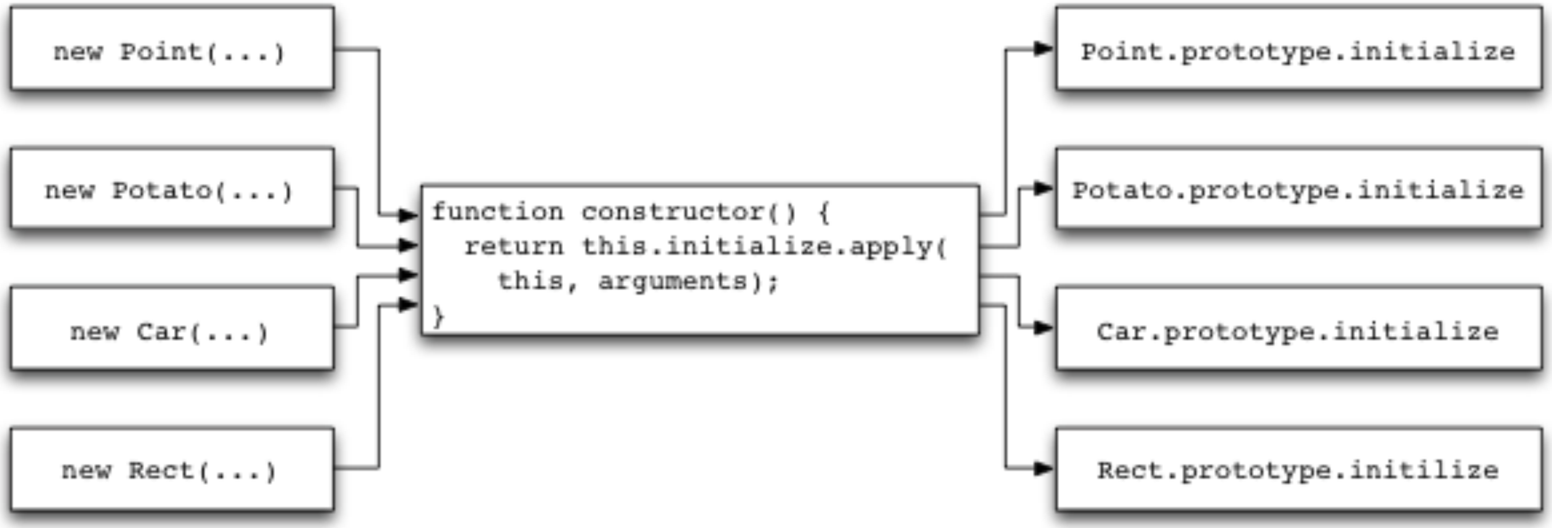
As an important side effect, PICs collect type information by recording all of the receiver types actually used at a given call site. The compiler can exploit this type information to generate better code when *recompiling* a method. An experimental version of such a system achieves a median speedup of 27% for our set of SELF programs, reducing the number of non-inlined message sends by a factor of two.

Implementations of dynamically-typed object-oriented languages have been limited by the paucity of type information available to the compiler. The abundance of the type information provided by PICs suggests a new compilation approach for these languages, *adaptive compilation*. Such compilers may succeed in generating very efficient code for the time-critical parts of a program without incurring distracting compilation pauses.

FTL-specific high-level optimizations

So far this post has given details on how we integrated with LLVM and managed to leverage its low-level optimization capabilities without losing the capabilities of our DFG JIT. But adding a higher-tier JIT also empowers us to do optimizations that would have been impossible if our tiering strategy ended with the DFG. The sections that follow show two capabilities that a fourth tier makes possible, that aren't specific to LLVM.

Polyvariant Devirtualization



```
function constructor() {
  return this.initialize.apply(
    this, arguments);
}
```

Figure 9. JS idioms such as the inheritance.js or Prototype will funnel execution through helpers, such as the object constructor in this figure. This causes the helper to appear polymorphic. Note that it would not be polymorphic if it was inlined: inlining `constructor` at the `new Point(...)` callsite causes the call to `initialize` to always call `Point`'s `initialize` method.

Om

React

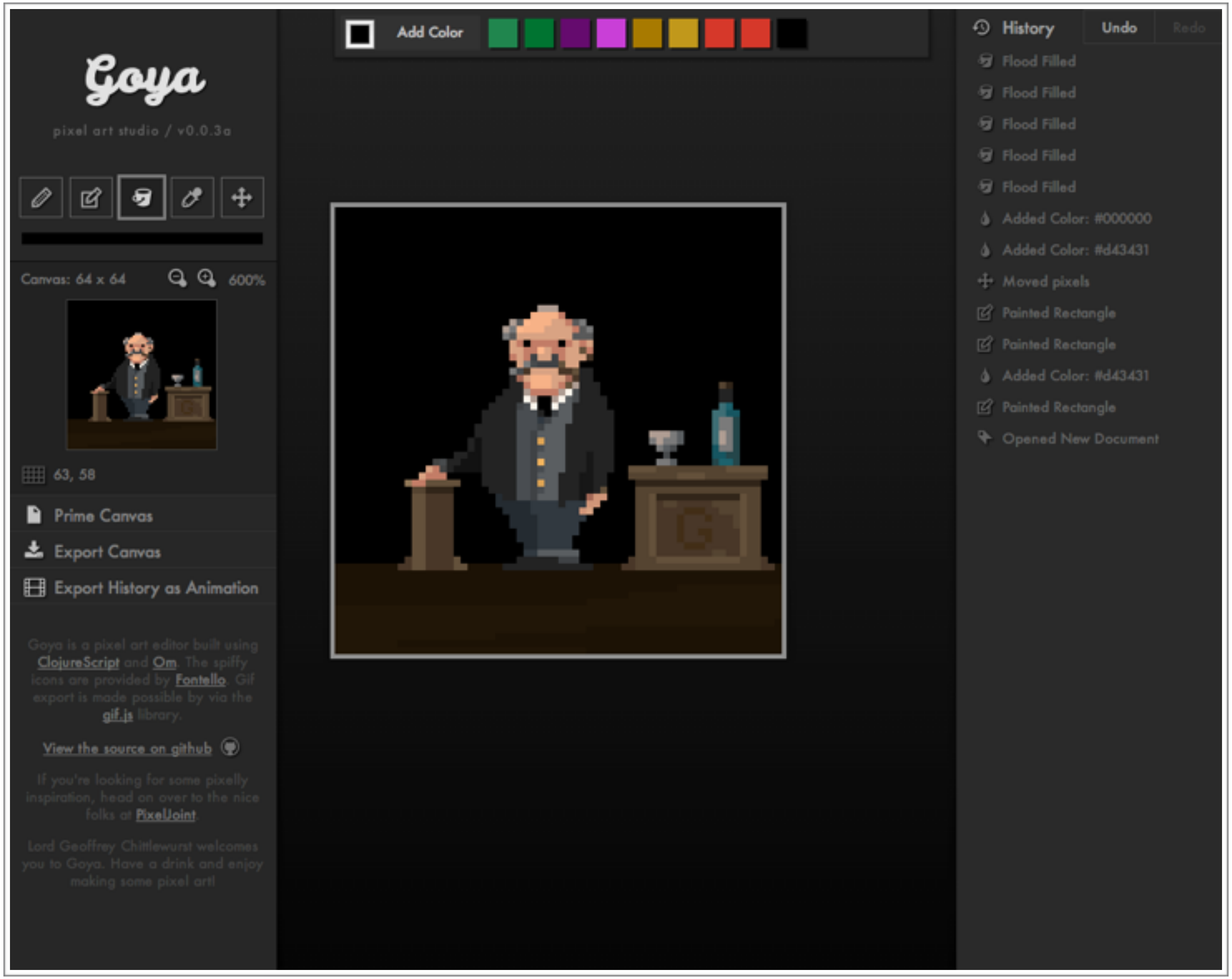
React

A JAVASCRIPT LIBRARY FOR BUILDING USER INTERFACES

$$\mathrm{f}(\mathrm{D}_0) = \mathrm{V}_0$$

$$\mathrm{f}(\mathrm{D}_1) = \mathbf{V}_1$$

$$\mathrm{diff}(\mathrm{V}_0,\mathrm{V}_1) = \mathrm{CHANGES}$$



Flood Filled

Flood Filled

Flood Filled

Flood Filled

Flood Filled

Added Color: #000000

Added Color: #d43431

Moved pixels

Painted Rectangle

Painted Rectangle

Added Color: #d43431

Painted Rectangle

Opened New Document

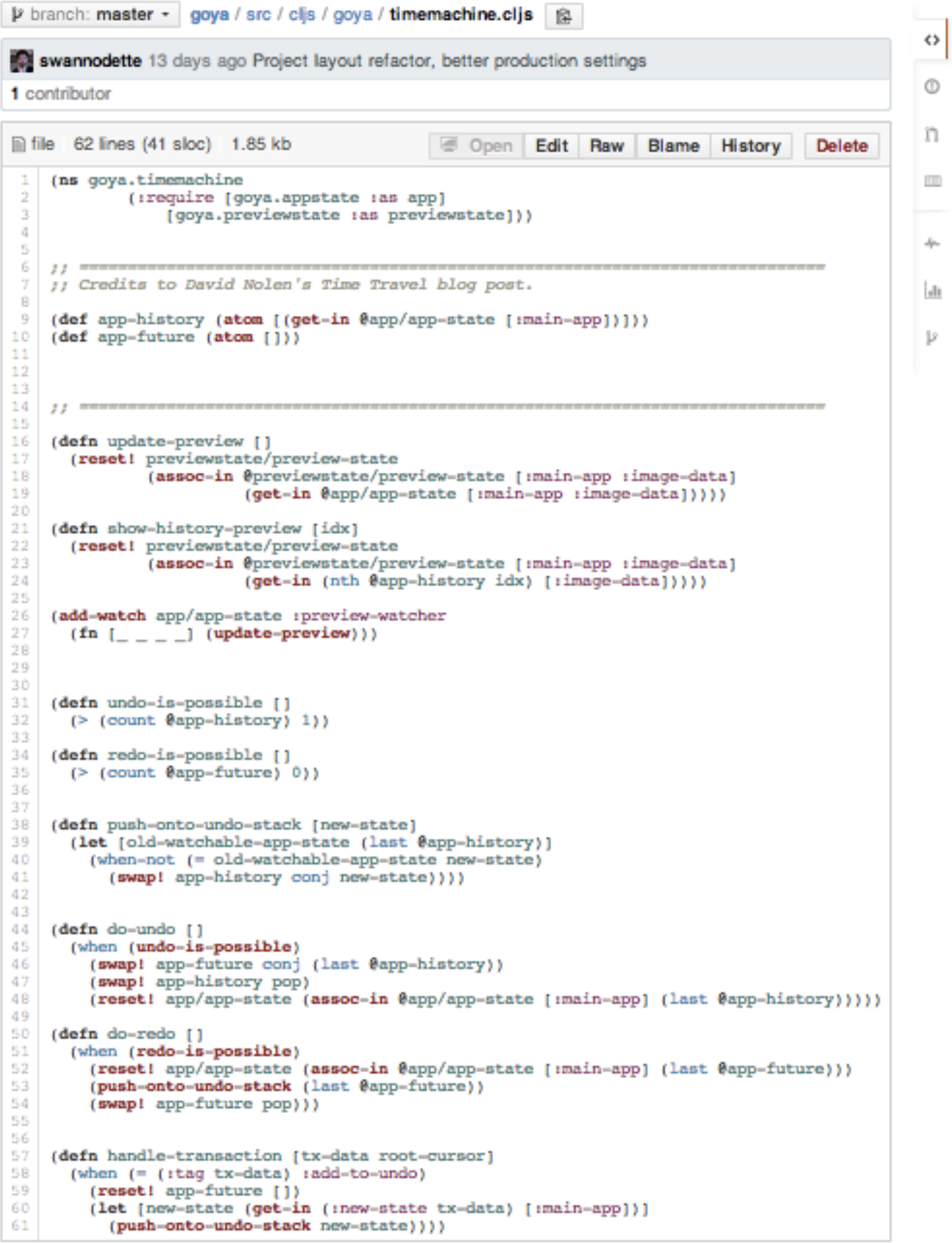
Goya is a pixel art editor built using ClojureScript and Om. The spiffy icons are provided by Fontello. Gif export is made possible by via the gif.js library.

View the source on github

If you're looking for some pixelly inspiration, head on over to the nice folks at PixelJoint.

Lord Geoffrey Chittlewurst welcomes you to Goya. Have a drink and enjoy making some pixel art!

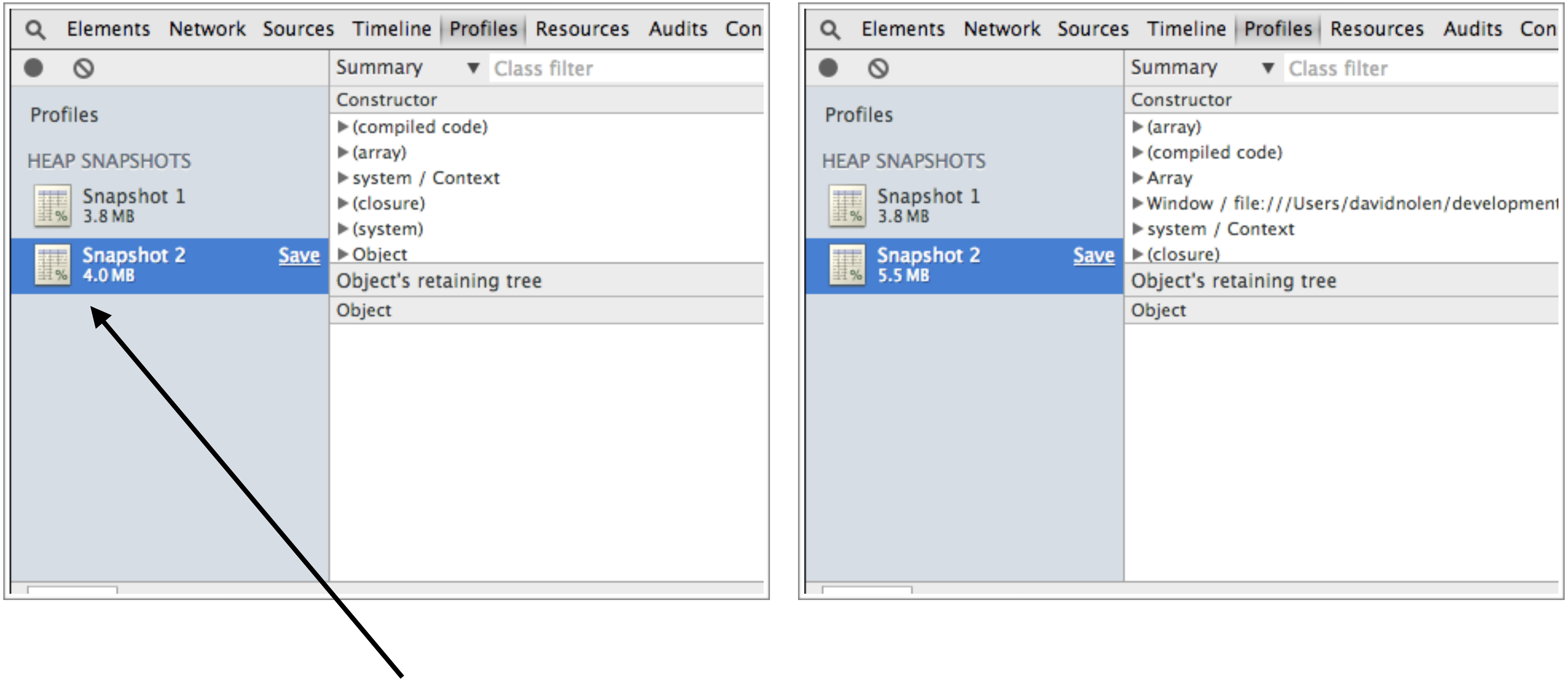
demo



swannodette 13 days ago Project layout refactor, better production settings

1 contributor

```clojure
(ns goya.timemachine
        (:require [goya.appstate :as app]
            [goya.previewstate :as previewstate]))


;; ==============================================================================
;; Credits to David Nolen's Time Travel blog post.

(def app-history (atom [(get-in @app/app-state [:main-app])]))
(def app-future (atom []))



;; ==============================================================================

(defn update-preview []
  (reset! previewstate/preview-state
          (assoc-in @previewstate/preview-state [:main-app :image-data]
                    (get-in @app/app-state [:main-app :image-data]))))

(defn show-history-preview [idx]
  (reset! previewstate/preview-state
          (assoc-in @previewstate/preview-state [:main-app :image-data]
                    (get-in (nth @app-history idx) [:image-data]))))

(add-watch app/app-state :preview-watcher
  (fn [_ _ _ _] (update-preview)))



(defn undo-is-possible []
  (> (count @app-history) 1))

(defn redo-is-possible []
  (> (count @app-future) 0))


(defn push-onto-undo-stack [new-state]
  (let [old-watchable-app-state (last @app-history)]
    (when-not (= old-watchable-app-state new-state)
      (swap! app-history conj new-state))))


(defn do-undo []
  (when (undo-is-possible)
    (swap! app-future conj (last @app-history))
    (swap! app-history pop)
    (reset! app/app-state (assoc-in @app/app-state [:main-app] (last @app-history)))))

(defn do-redo []
  (when (redo-is-possible)
    (reset! app/app-state (assoc-in @app/app-state [:main-app] (last @app-future)))
    (push-onto-undo-stack (last @app-future))
    (swap! app-future pop)))


(defn handle-transaction [tx-data root-cursor]
  (when (= (:tag tx-data) :add-to-undo)
    (reset! app-future [])
    (let [new-state (get-in (:new-state tx-data) [:main-app])]
      (push-onto-undo-stack new-state))))
```


Elements Network Sources Timeline Profiles Resources Audits Con
Summary ▼ Class filter

Profiles

HEAP SNAPSHOTS

Snapshot 1
3.8 MB

Snapshot 2 Save
4.0 MB

Constructor
- ▶(compiled code)
- ▶(array)
- ▶system / Context
- ▶(closure)
- ▶(system)
- ▶Object

Object's retaining tree

Object

Elements Network Sources Timeline Profiles Resources Audits Con
Summary ▼ Class filter

Profiles

HEAP SNAPSHOTS

Snapshot 1
3.8 MB

Snapshot 2 Save
5.5 MB

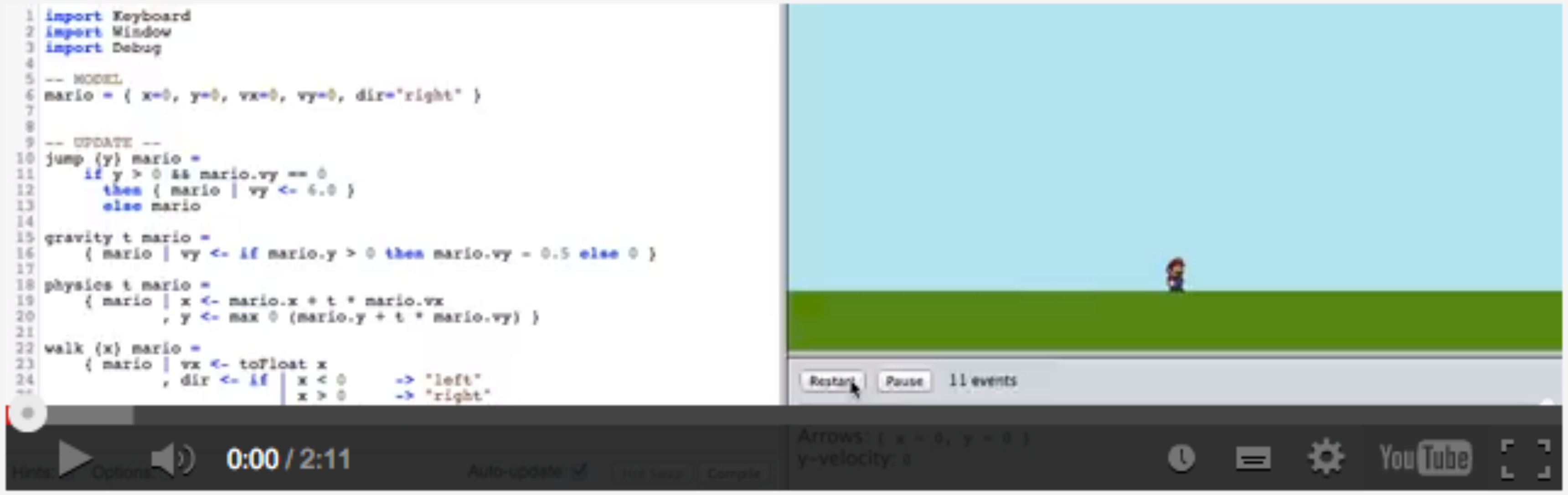
Constructor
- ▶(array)
- ▶(compiled code)
- ▶Array
- ▶Window / file:///Users/davidnolen/developmen
- ▶system / Context
- ▶(closure)

Object's retaining tree

Object

Persistent Data Structures ... ROCK

Now that we know the basic tools of the debugger, we turn to a slightly more complicated animation of Mario with a simple bug. Something is wrong in our code such that Mario can double jump! In this case we actually modify the `jump` and `gravity` functions to see how that changes the program.

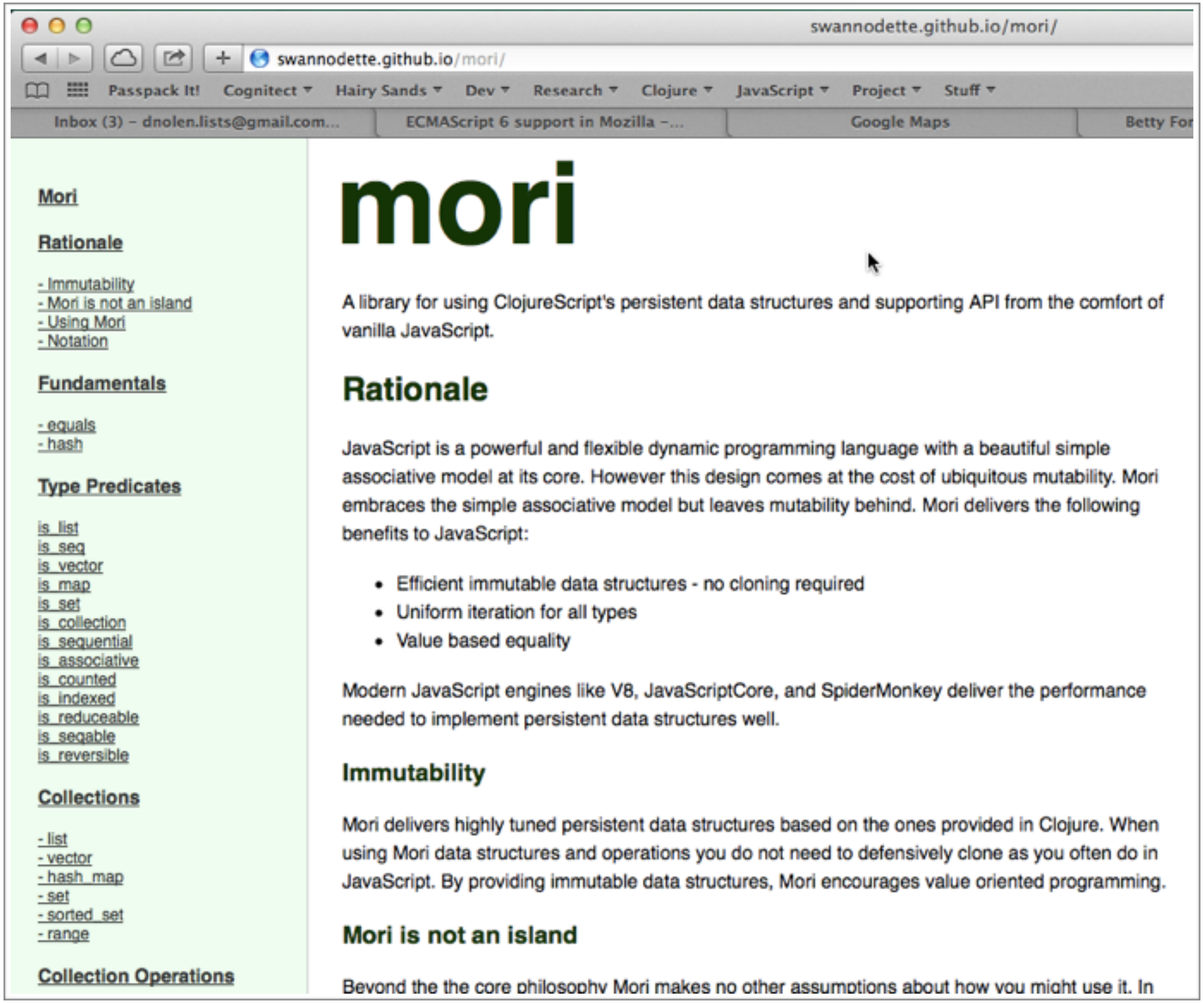


```
import Keyboard
import Window
import Debug

-- MODEL
mario = { x=0, y=0, vx=0, vy=0, dir="right" }


-- UPDATE --
jump {y} mario =
    if y > 0 && mario.vy == 0
      then { mario | vy <- 6.0 }
      else mario

gravity t mario =
    { mario | vy <- if mario.y > 0 then mario.vy - 0.5 else 0 }

physics t mario =
    { mario | x <- mario.x + t * mario.vx
            , y <- max 0 (mario.y + t * mario.vy) }

walk {x} mario =
    { mario | vx <- toFloat x
            , dir <- if | x < 0     -> "left"
                        | x > 0     -> "right"
```

Try it!

At about 30 seconds in, we begin tracing Mario's path through time. This trace is crucial to visualizing the meaning of our program. To explore the double jump bug, we need to see how changing our code changes Mario's path. Laszlo introduced this ability with the following function:

```
Debug.trace : String -> Element -> Element
```


swannodette.github.io/mori/
swannodette.github.io/mori/
Passpack It!
Cognitect
Hairy Sands
Dev
Research
Clojure
JavaScript
Project
Stuff
Inbox (3) – dnolen.lists@gmail.com...
ECMAScript 6 support in Mozilla –...
Google Maps
Betty For

Mori

Rationale

- Immutability
- Mori is not an island
- Using Mori
- Notation

Fundamentals

- equals
- hash

Type Predicates

is_list
is_seq
is_vector
is_map
is_set
is_collection
is_sequential
is_associative
is_counted
is_indexed
is_reduceable
is_seqable
is_reversible

Collections

- list
- vector
- hash_map
- set
- sorted_set
- range

Collection Operations

mori

A library for using ClojureScript's persistent data structures and supporting API from the comfort of vanilla JavaScript.

Rationale

JavaScript is a powerful and flexible dynamic programming language with a beautiful simple associative model at its core. However this design comes at the cost of ubiquitous mutability. Mori embraces the simple associative model but leaves mutability behind. Mori delivers the following benefits to JavaScript:

- Efficient immutable data structures - no cloning required
- Uniform iteration for all types
- Value based equality

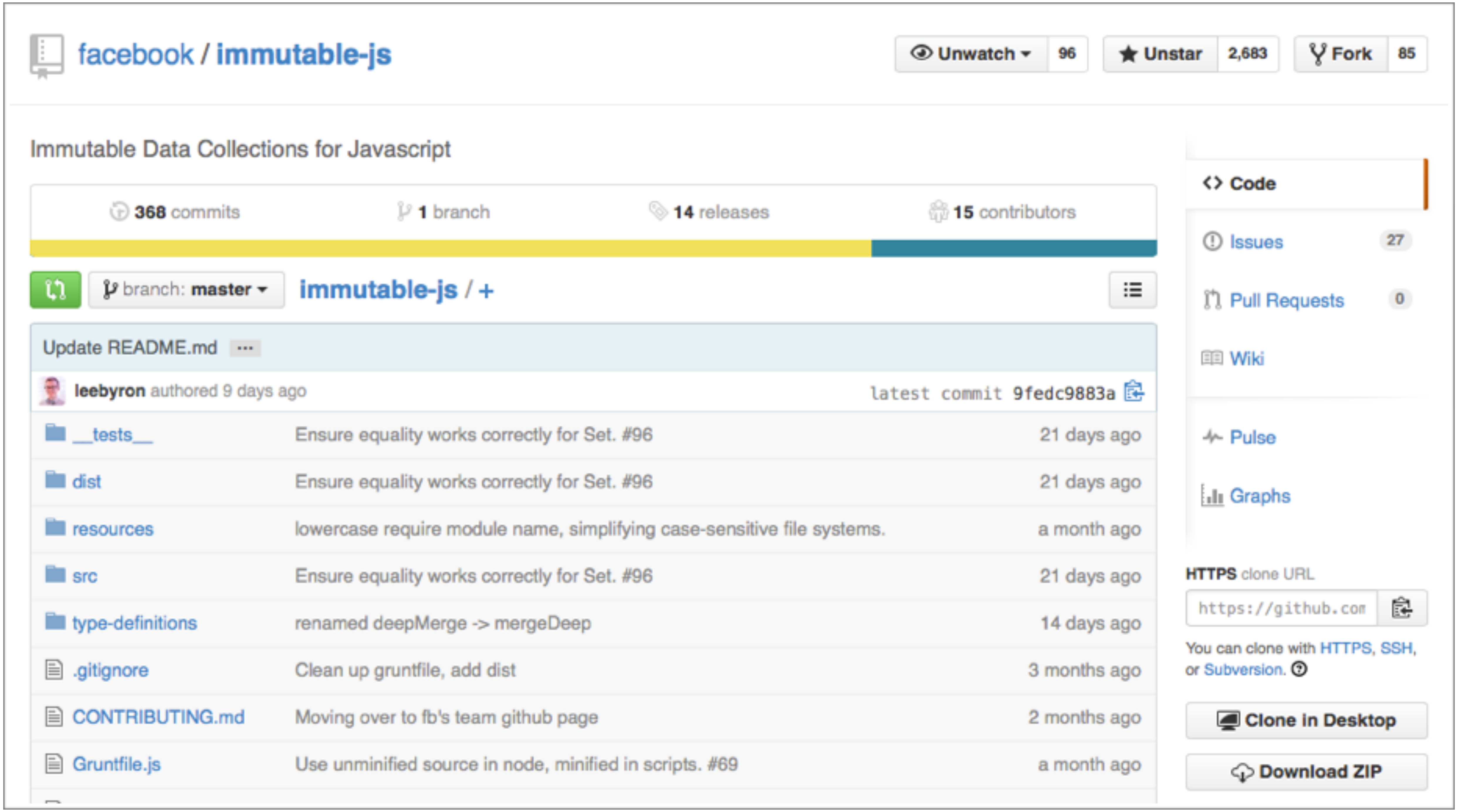
Modern JavaScript engines like V8, JavaScriptCore, and SpiderMonkey deliver the performance needed to implement persistent data structures well.

Immutability

Mori delivers highly tuned persistent data structures based on the ones provided in Clojure. When using Mori data structures and operations you do not need to defensively clone as you often do in JavaScript. By providing immutable data structures, Mori encourages value oriented programming.

Mori is not an island

Beyond the the core philosophy Mori makes no other assumptions about how you might use it. In



Immutable Data Collections for Javascript

368 commits 1 branch 14 releases 15 contributors

Update README.md ···

leebyron authored 9 days ago latest commit 9fedc9883a

__tests__	Ensure equality works correctly for Set. #96	21 days ago
dist	Ensure equality works correctly for Set. #96	21 days ago
resources	lowercase require module name, simplifying case-sensitive file systems.	a month ago
src	Ensure equality works correctly for Set. #96	21 days ago
type-definitions	renamed deepMerge -> mergeDeep	14 days ago
.gitignore	Clean up gruntfile, add dist	3 months ago
CONTRIBUTING.md	Moving over to fb's team github page	2 months ago
Gruntfile.js	Use unminified source in node, minified in scripts. #69	a month ago

HTTPS clone URL

https://github.com

You can clone with HTTPS, SSH, or Subversion.

100%
IMMUTABLE MODELS

Questions?