Immutability,

or Putting the Dream Machine to Work


cognitect

The Dream Machine.

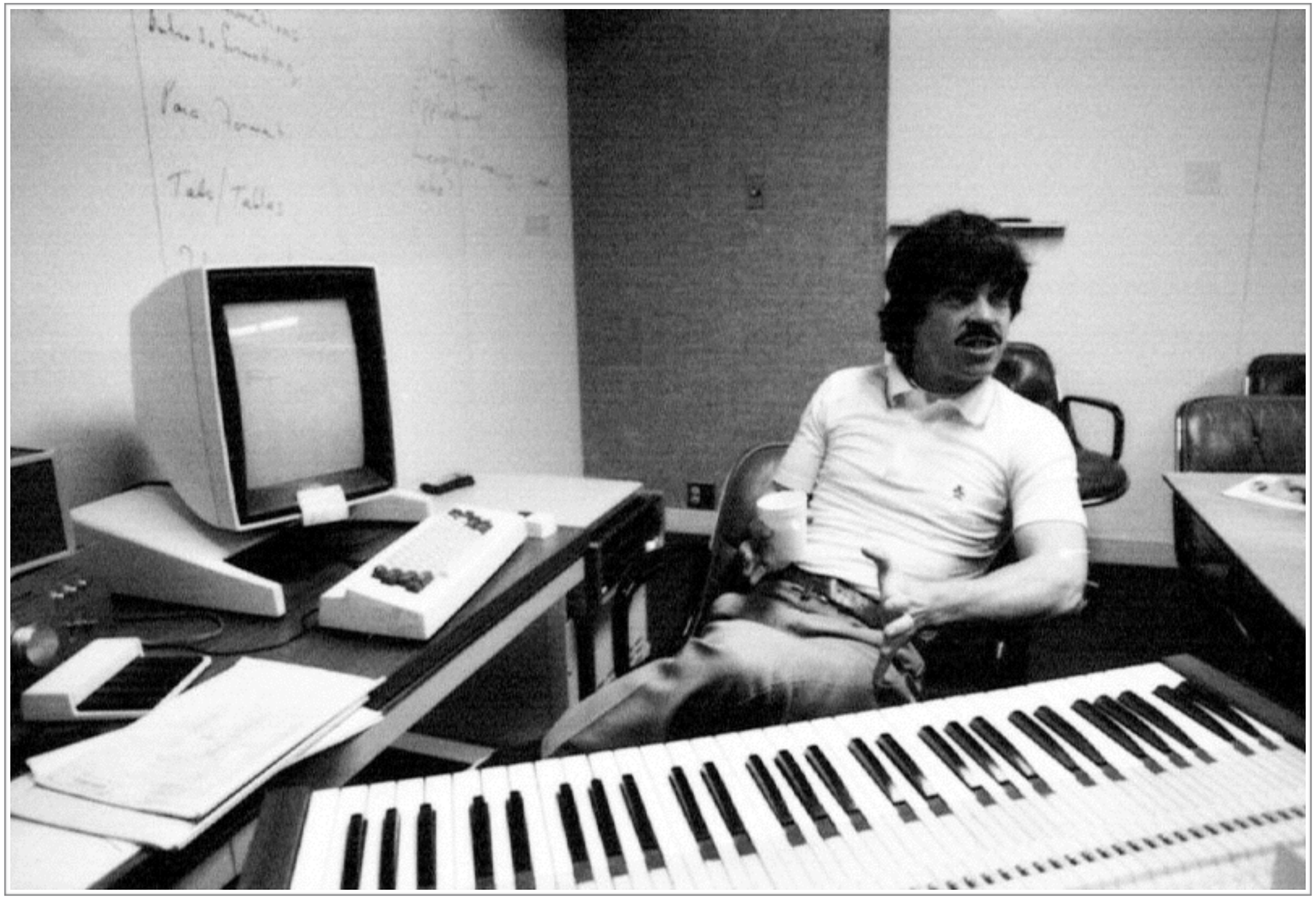
J. C. R. Licklider and the Revolution That Made Computing Personal.

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



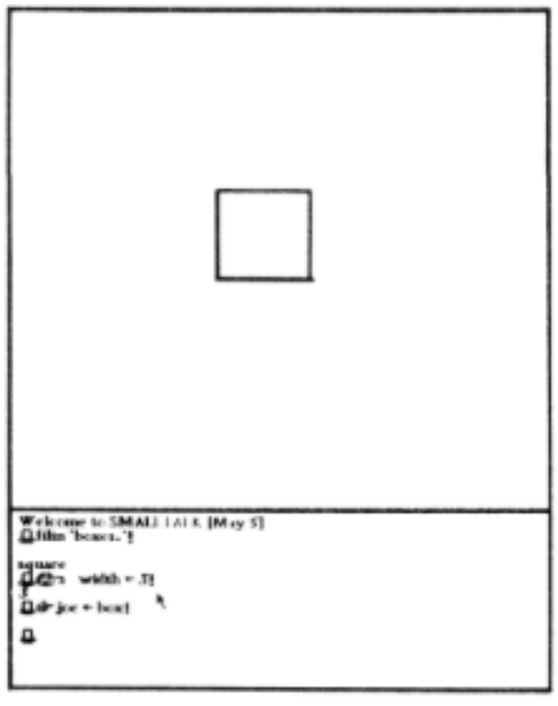
Learning About Smalltalk

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.



These boxes could turn on their axes,

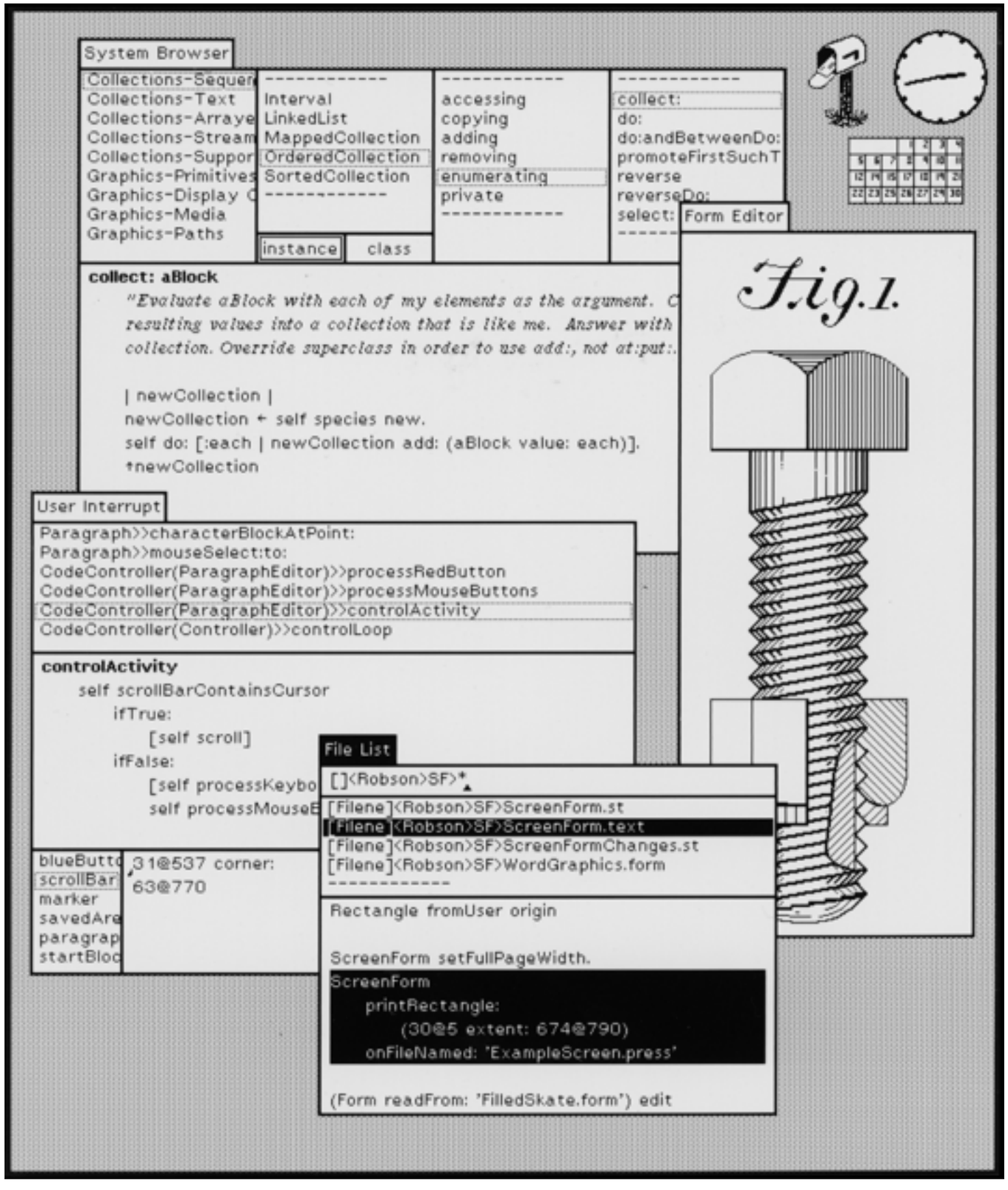
and shrink.

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.

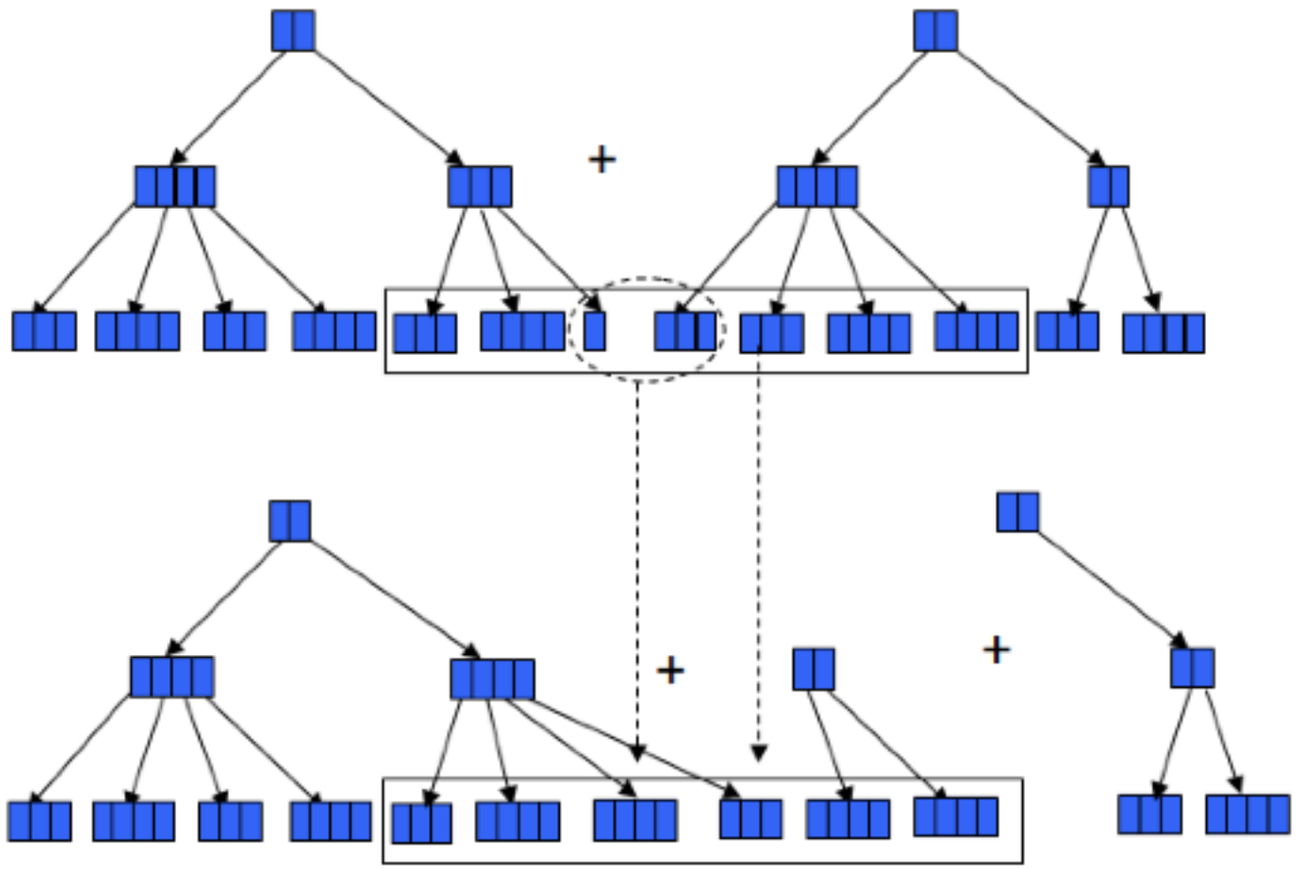
You would point with the mouse to the box that contained the shape you wanted to draw with, then press the top mouse button. Now the shape would be a paint


System Browser
Collections-Sequer
Collections-Text
Collections-Arraye
Collections-Stream
Collections-Suppor
Graphics-Primitives
Graphics-Display C
Graphics-Media
Graphics-Paths
Interval
LinkedList
MappedCollection
OrderedCollection
SortedCollection
instance
class
accessing
copying
adding
removing
enumerating
private
collect:
do:
do:andBetweenDo:
promoteFirstSuchT
reverse
reverseDo:
select:
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
User Interrupt
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 processMouseE
blueButto
scrollBar
marker
savedAre
paragrap
startBloc
31@537 corner:
63@770
File List
[]<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
Form Editor
Fig.1.

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*

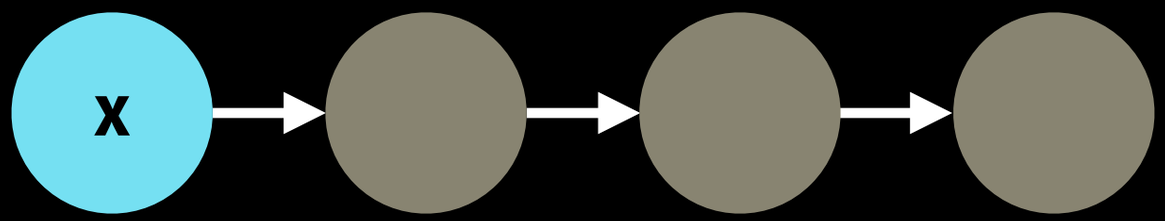



HACKER
SCHOOL
2012

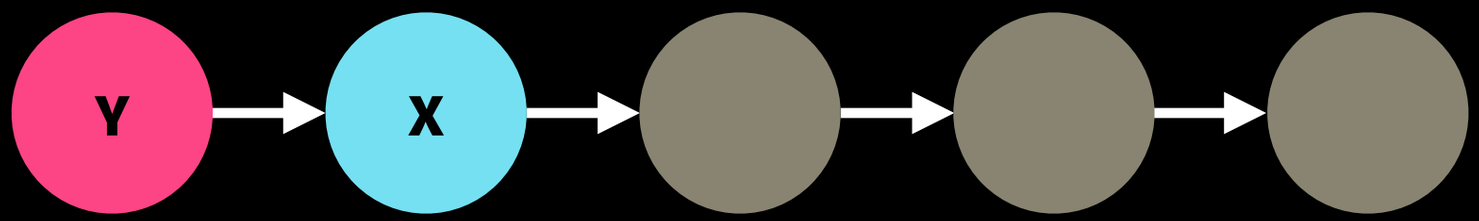
Functional Programming and Data

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

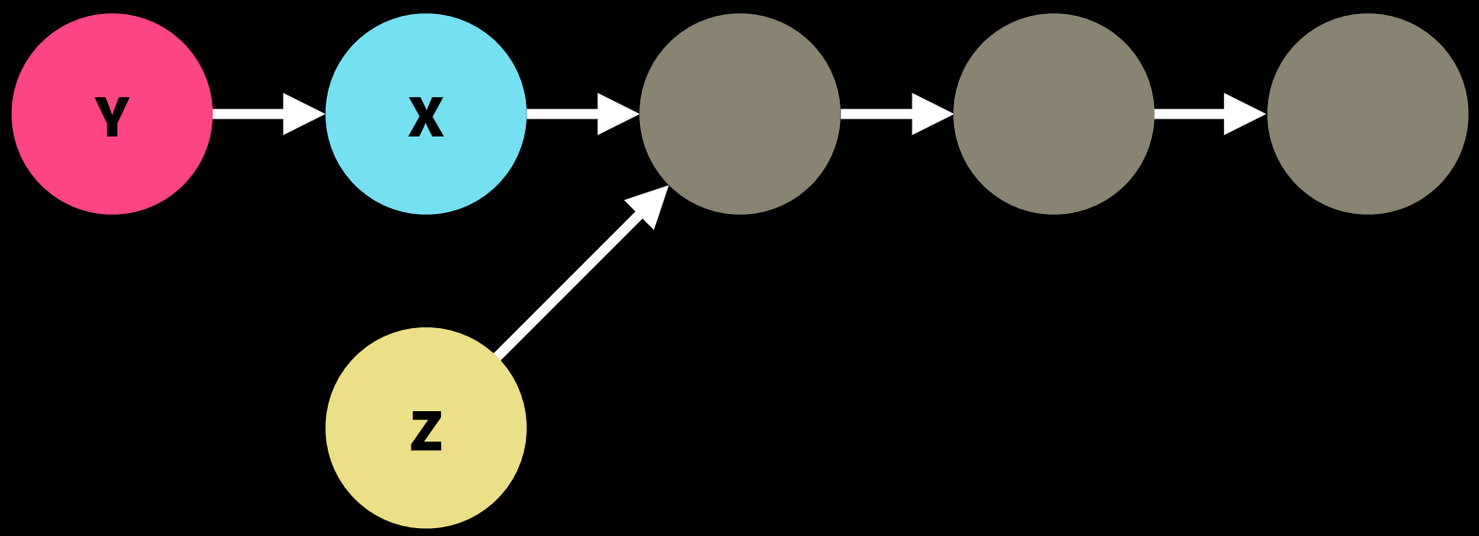
Simple example: Linked List



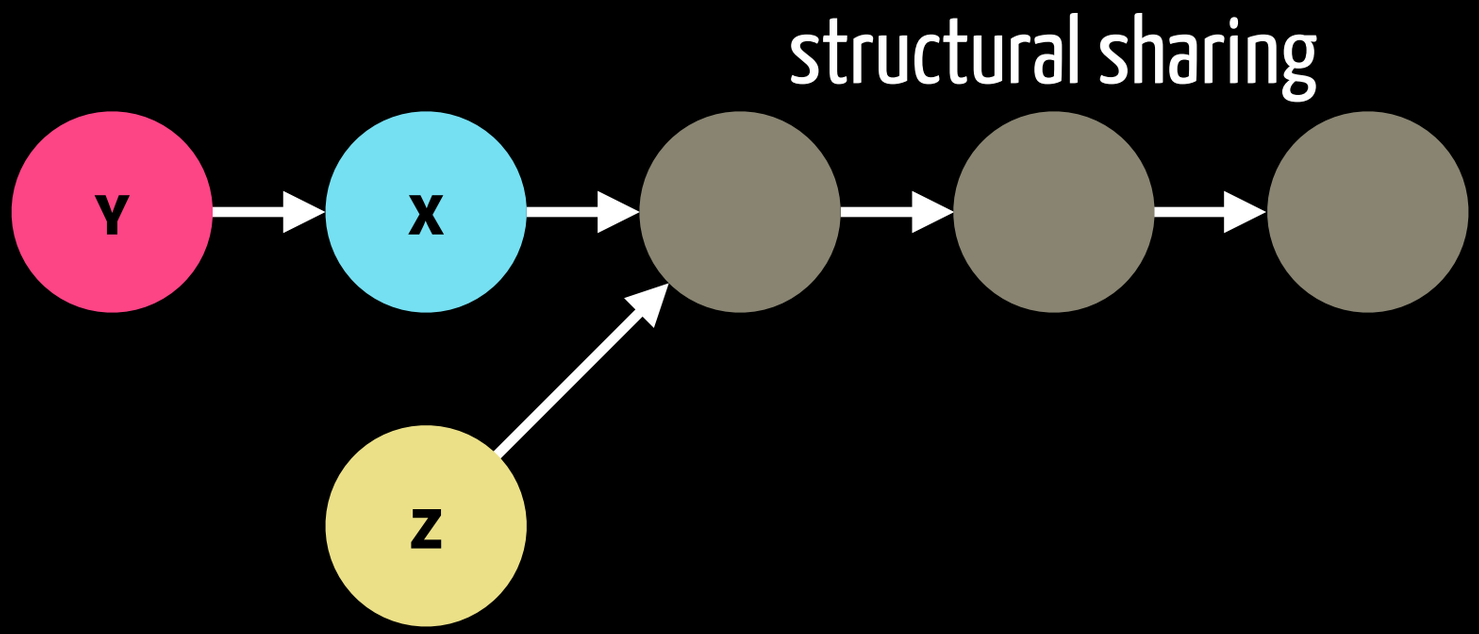
Simple example: Linked List



Simple example: Linked List



Simple example: Linked List



Sharing structure

- space efficiency
- computational efficiency – avoids copying

Phil Bagwell

- Array Mapped Trie
- Hash Array Mapped Trie

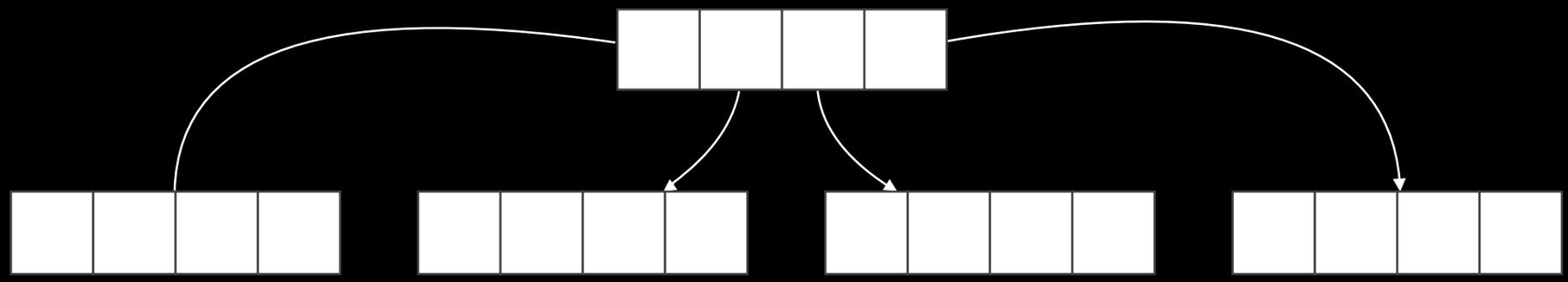
Bitmapped Vector Trie

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

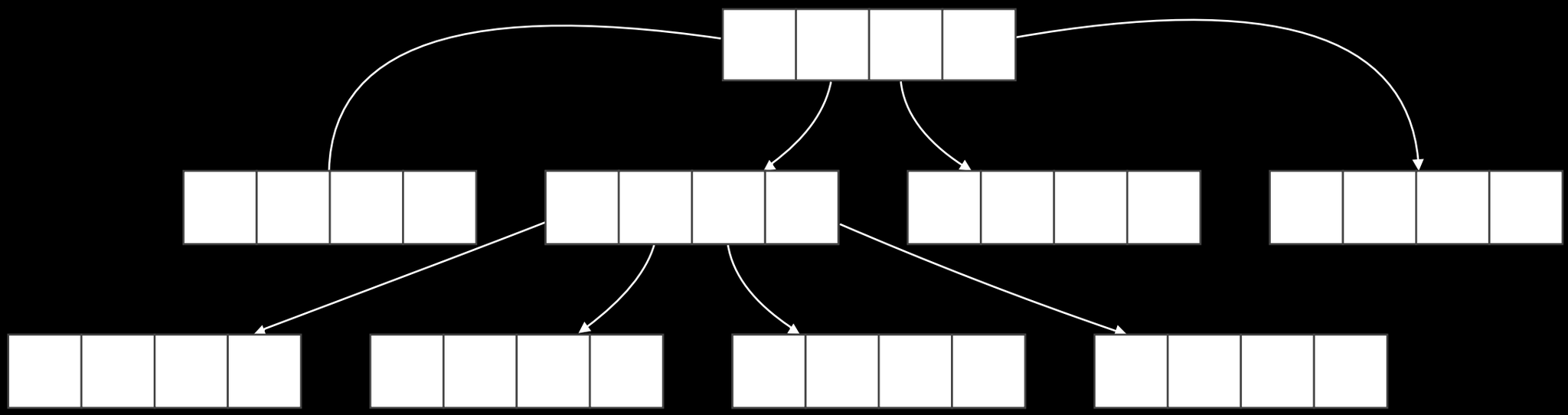
Persistent Vector

Persistent Vector

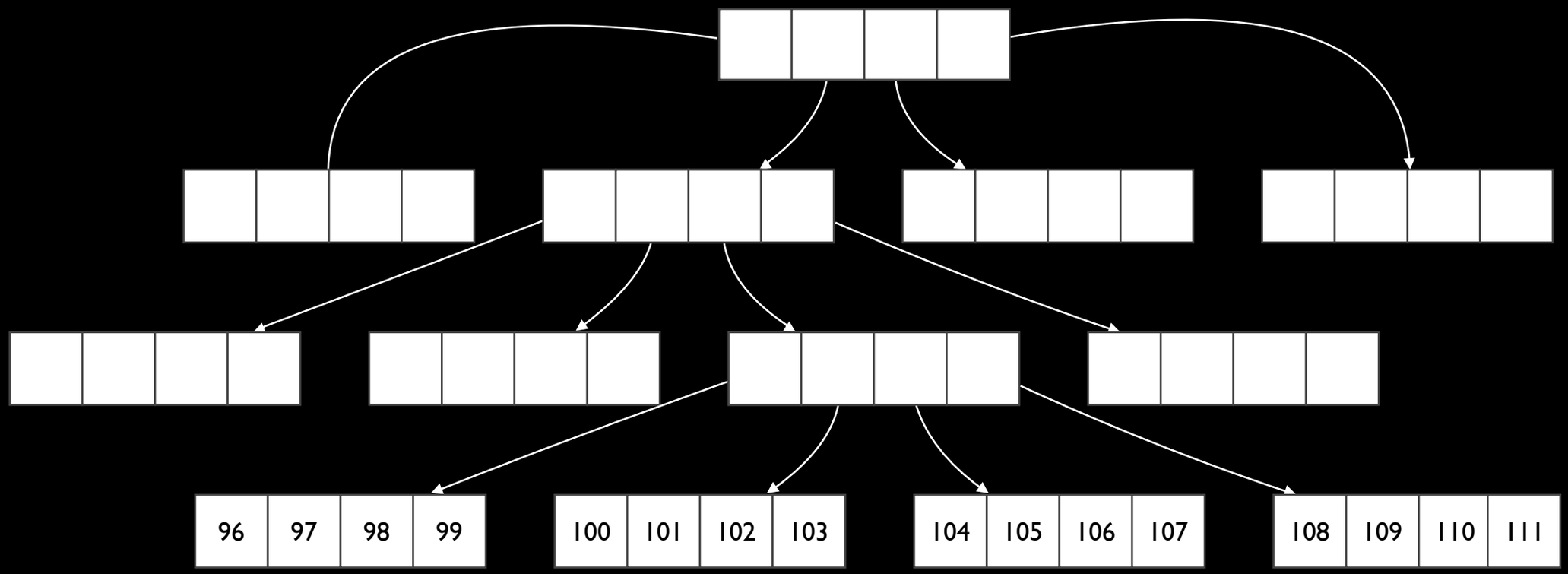
Persistent Vector



Persistent Vector



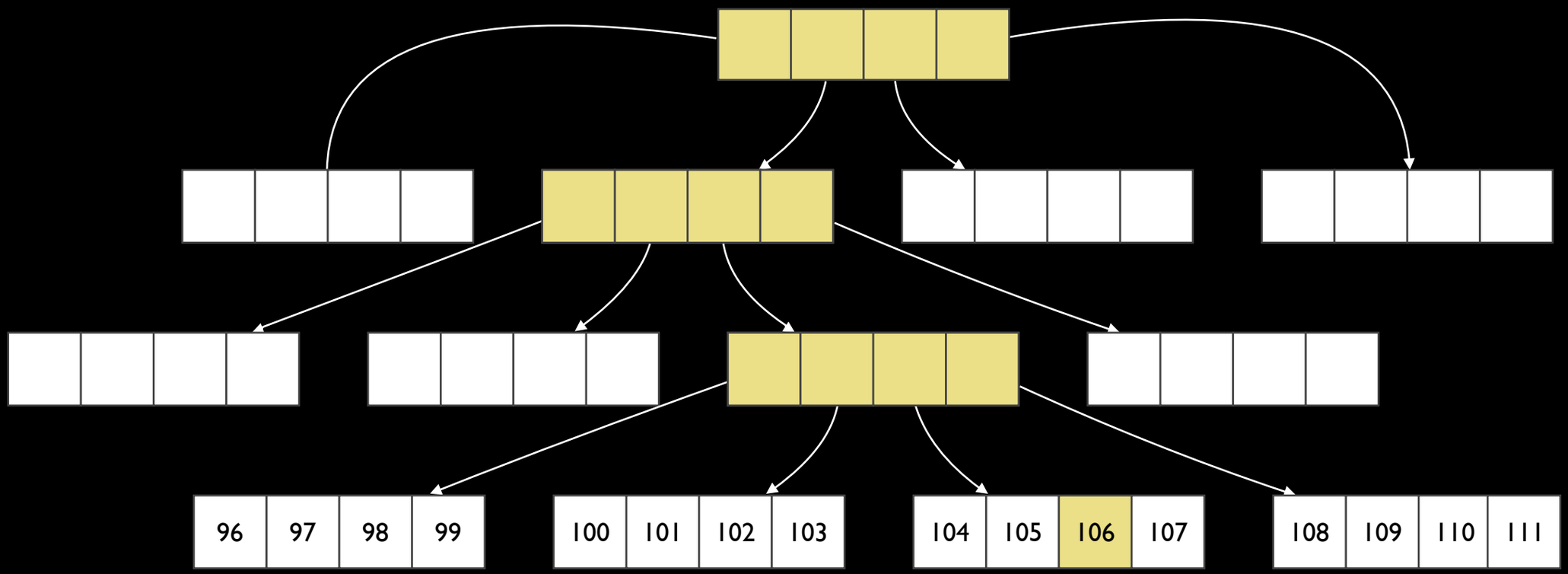
Persistent Vector



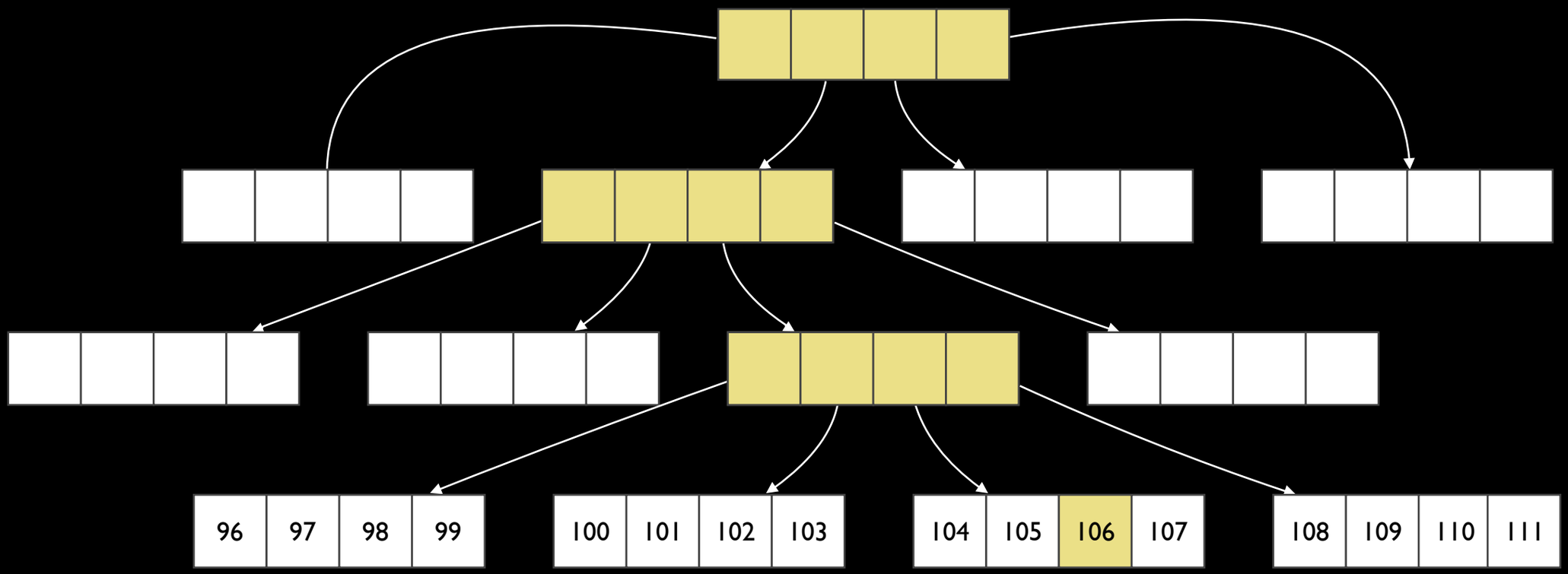
Persistent Vector

`getindex`

Persistent Vector

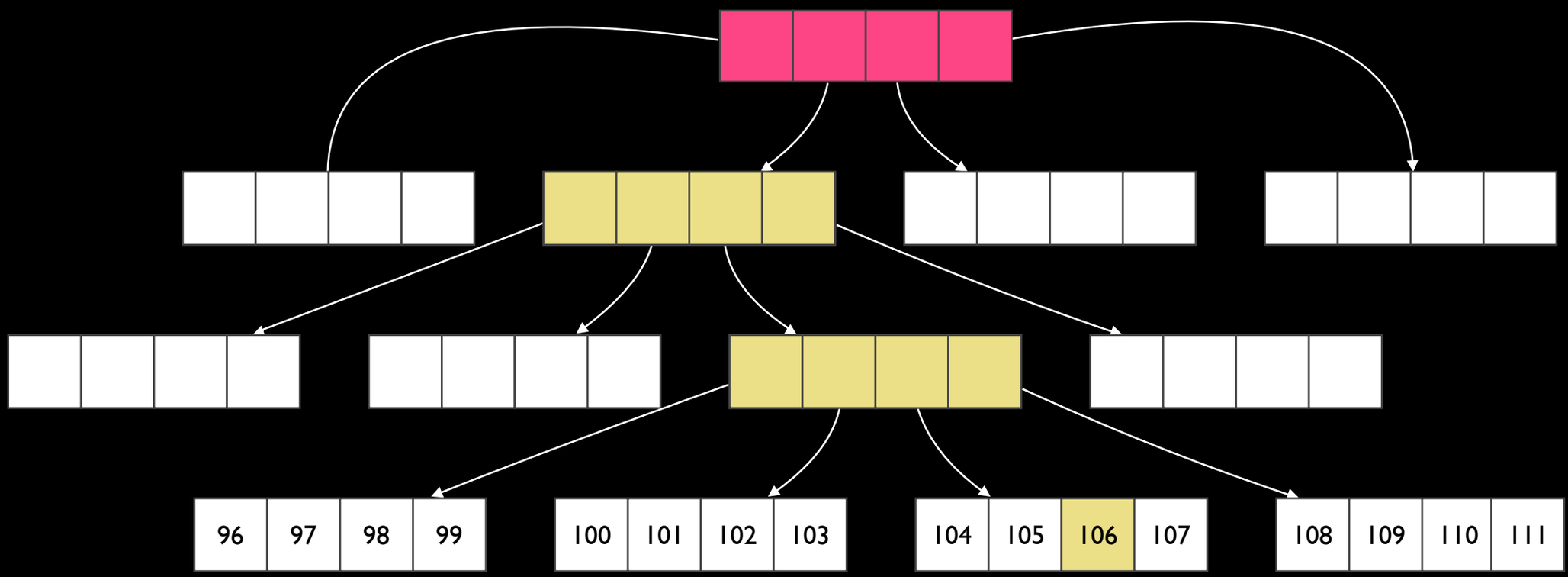


Persistent Vector



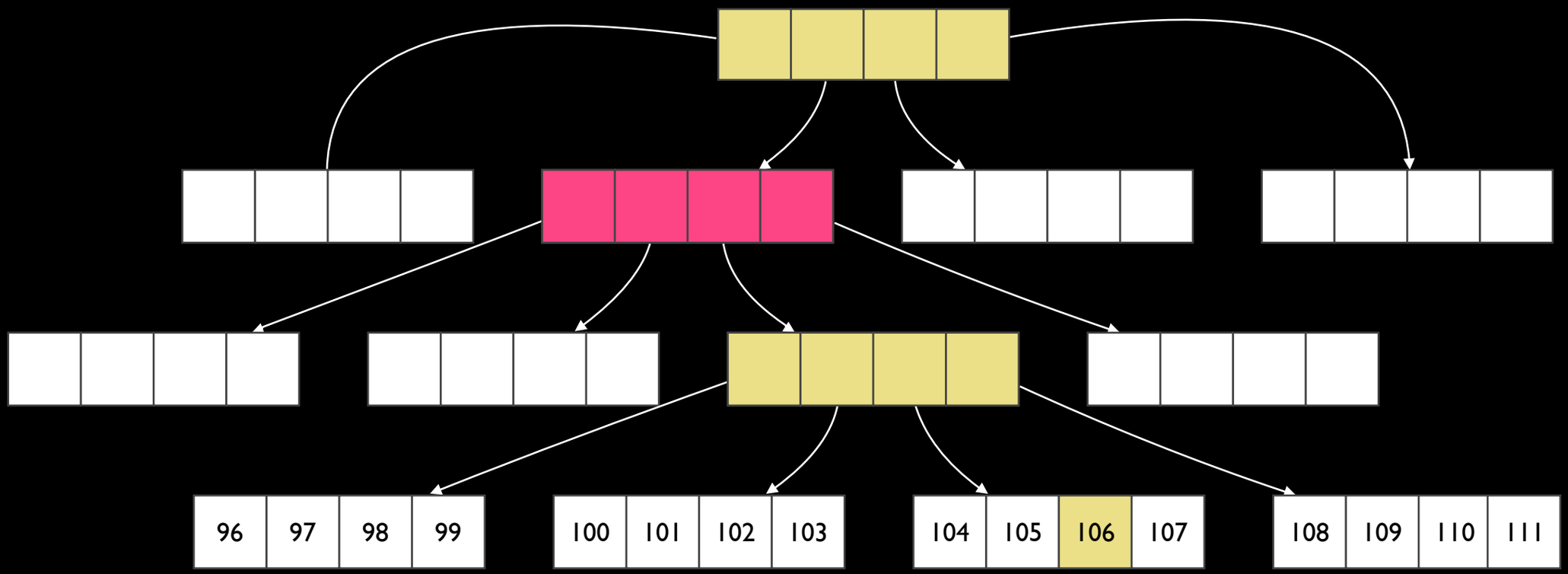
0b01101010

Persistent Vector



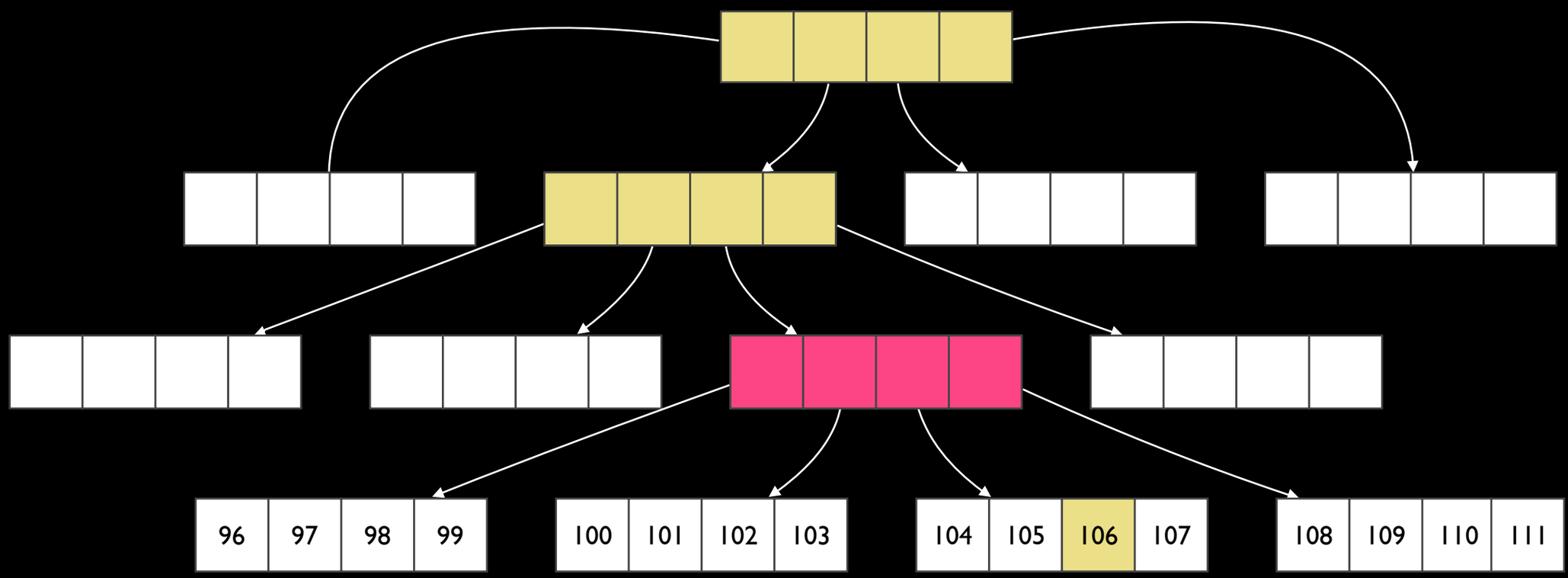
0b01101010

Persistent Vector



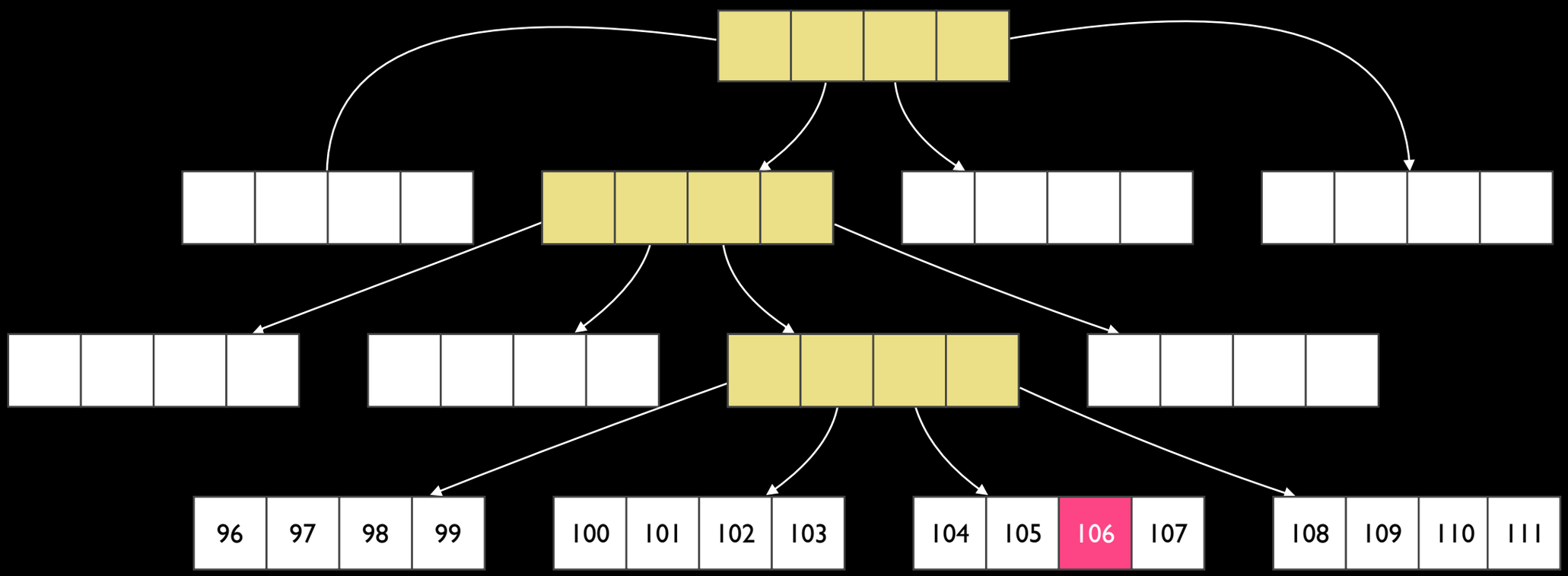
0b01101010

Persistent Vector



0b01101010

Persistent Vector

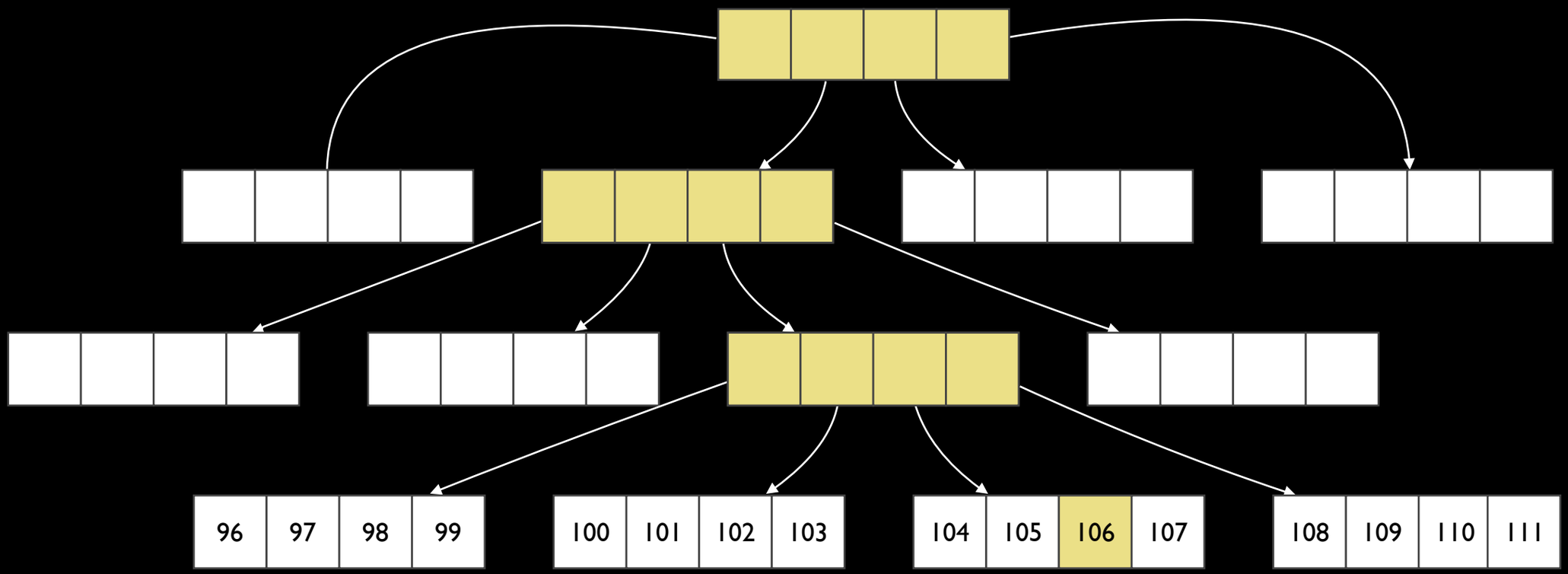


0b01101010

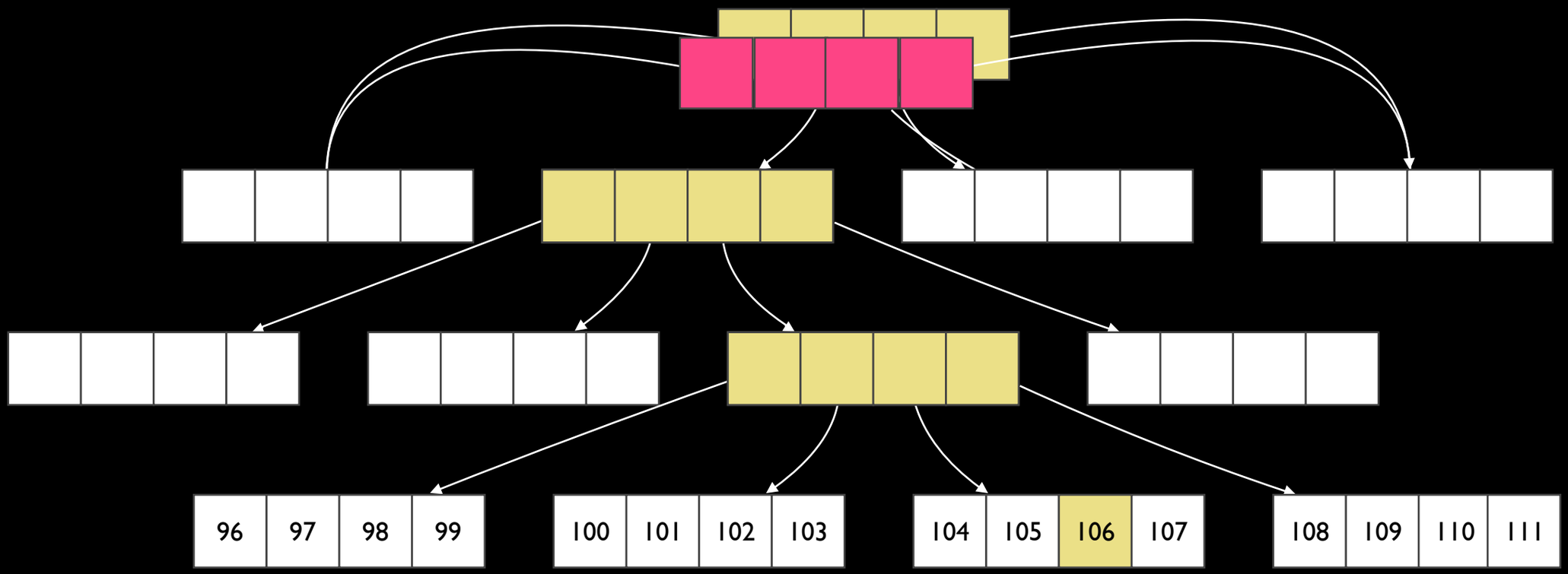
Persistent Vector

assoc

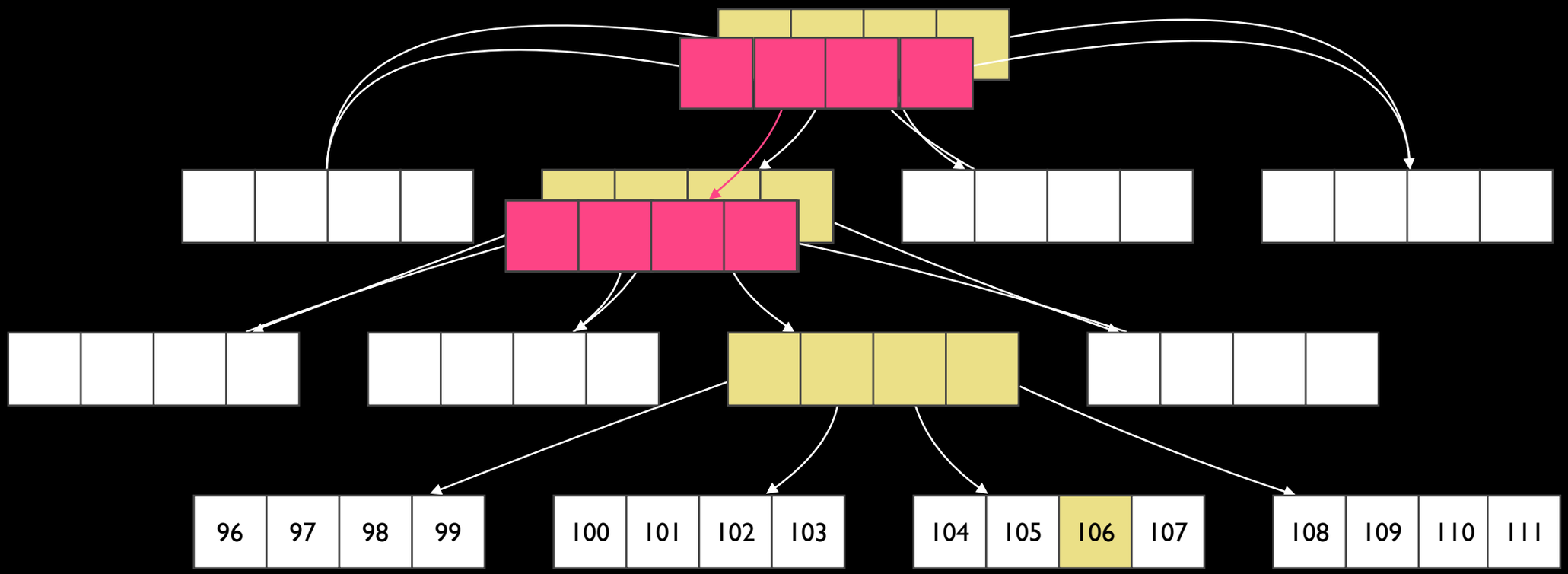
Persistent Vector



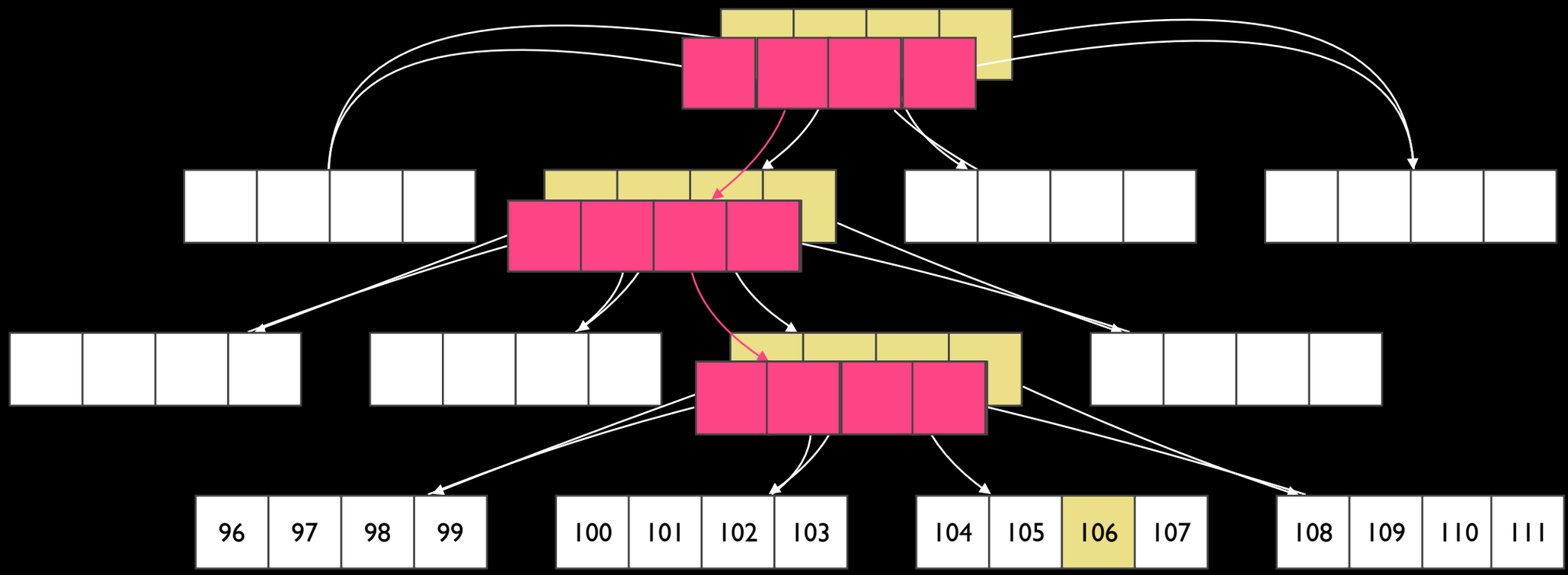
Persistent Vector



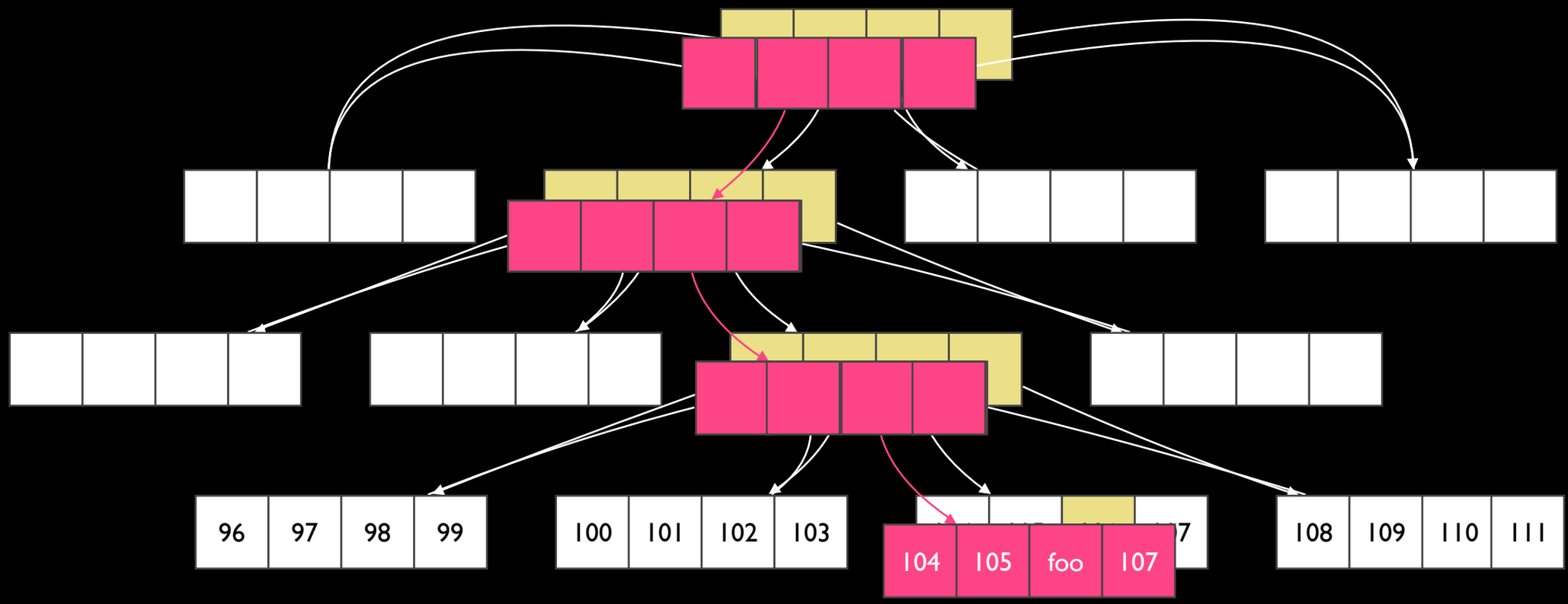
Persistent Vector



Persistent Vector



Persistent Vector



Persistent Vector

Length 4 internal vectors?

Persistent Vector

32

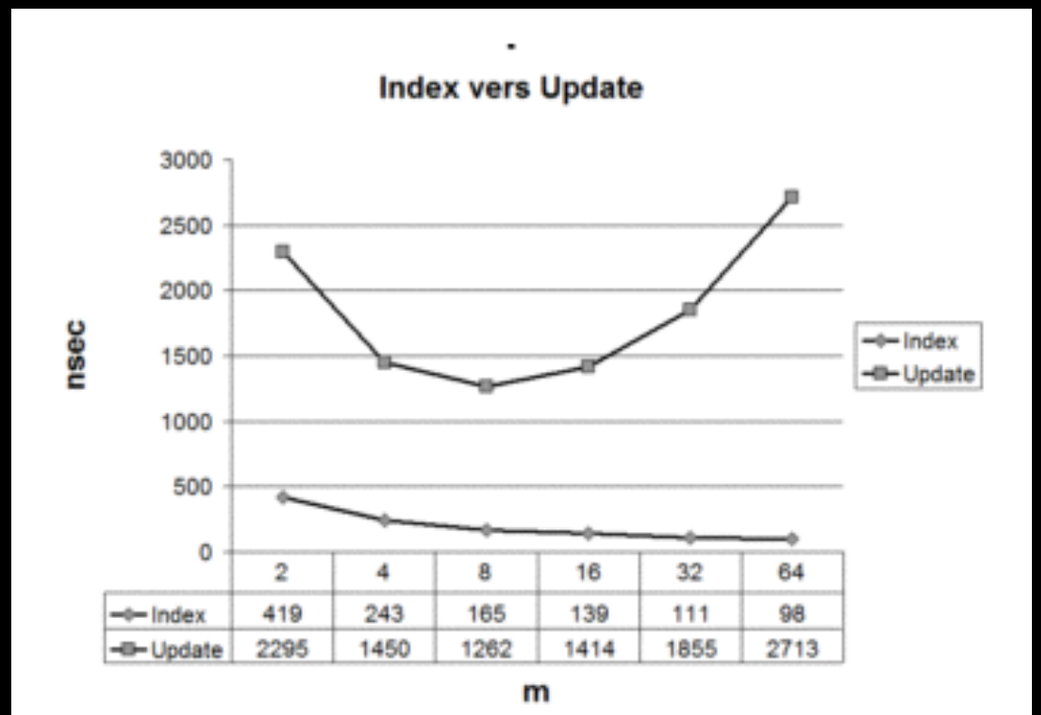


Figure 2. Time for index and update, depending on m

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

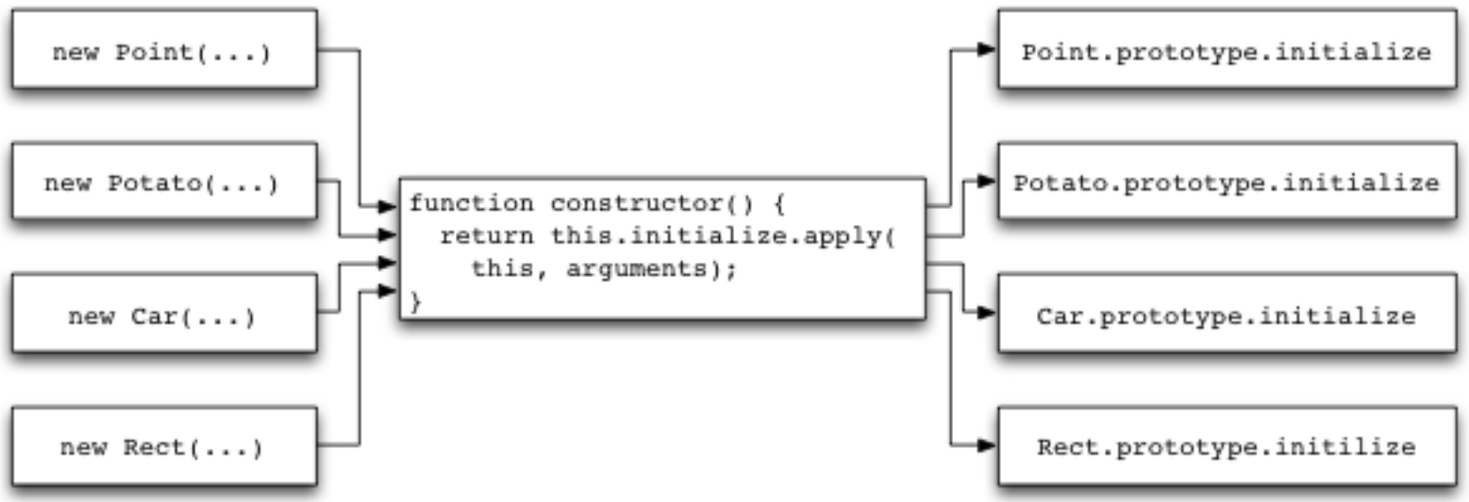
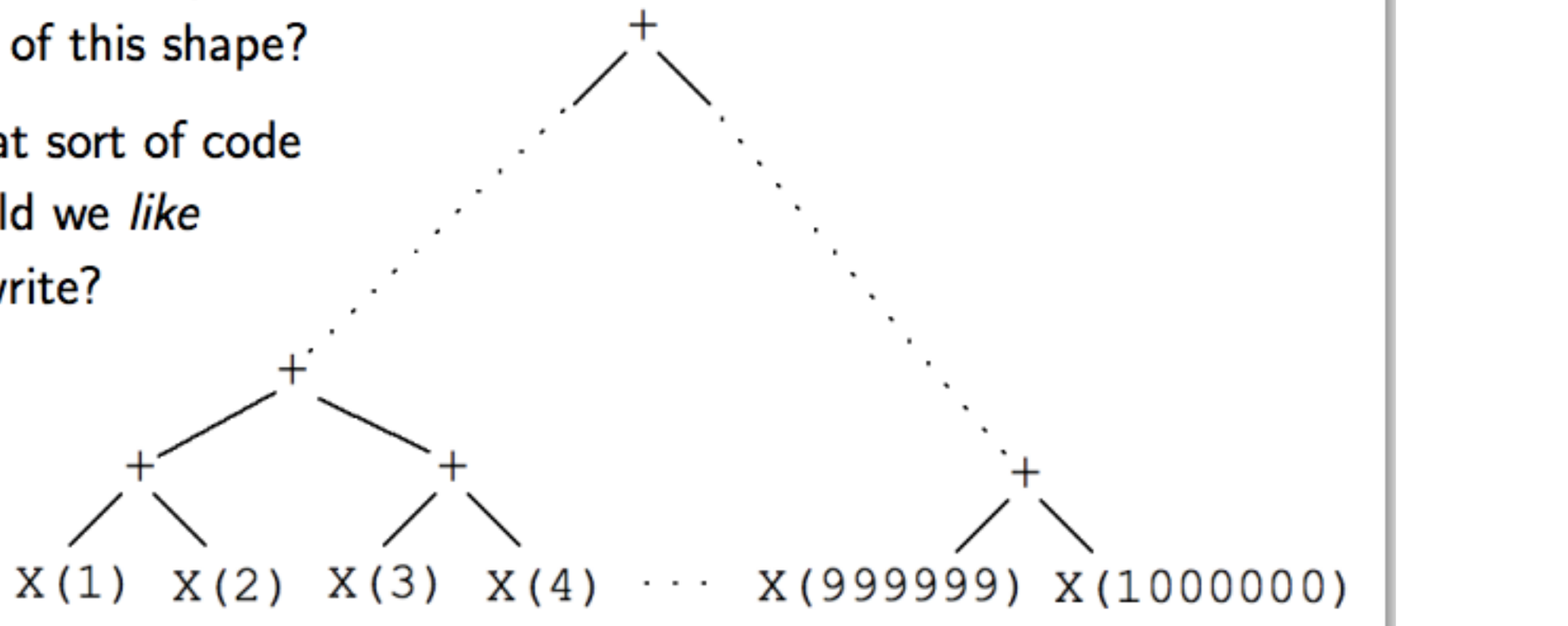


Figure 9. JS idioms such as the inheritance.js or Prototype will funnel execution through helpers, such as the object constructor in this figure. This causes the helper to appear polymorphic. Note that it would not be polymorphic if it was inlined: inlining `constructor` at the `new Point(...)` callsite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

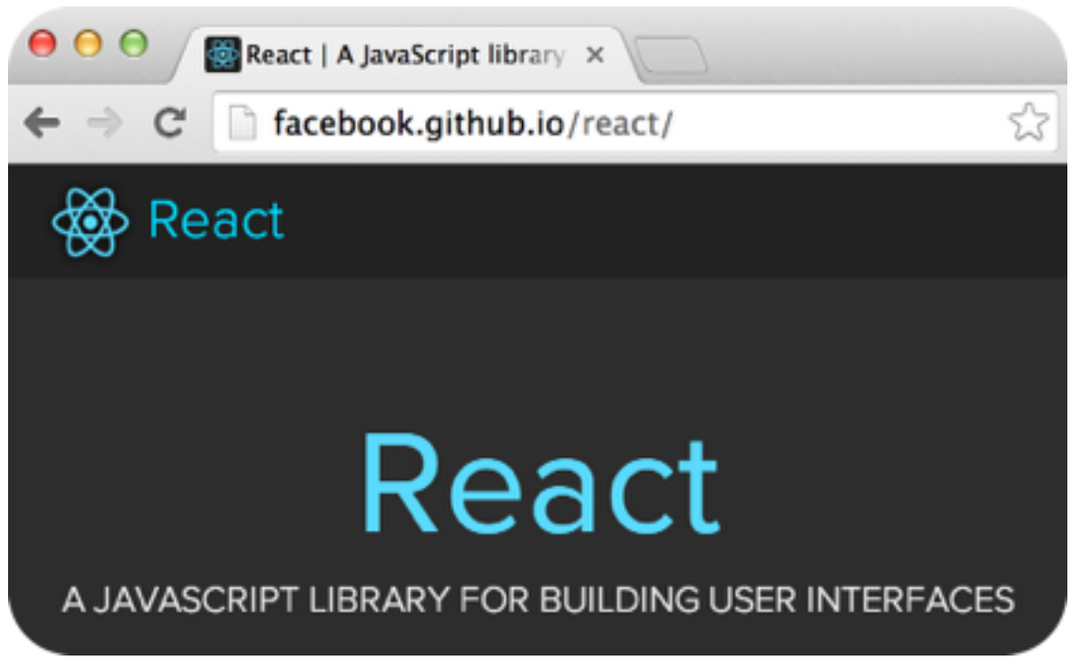
Parallel Computation Tree

What sort of code
should we write
to get a computation
tree of this shape?

What sort of code
would we *like*
to write?



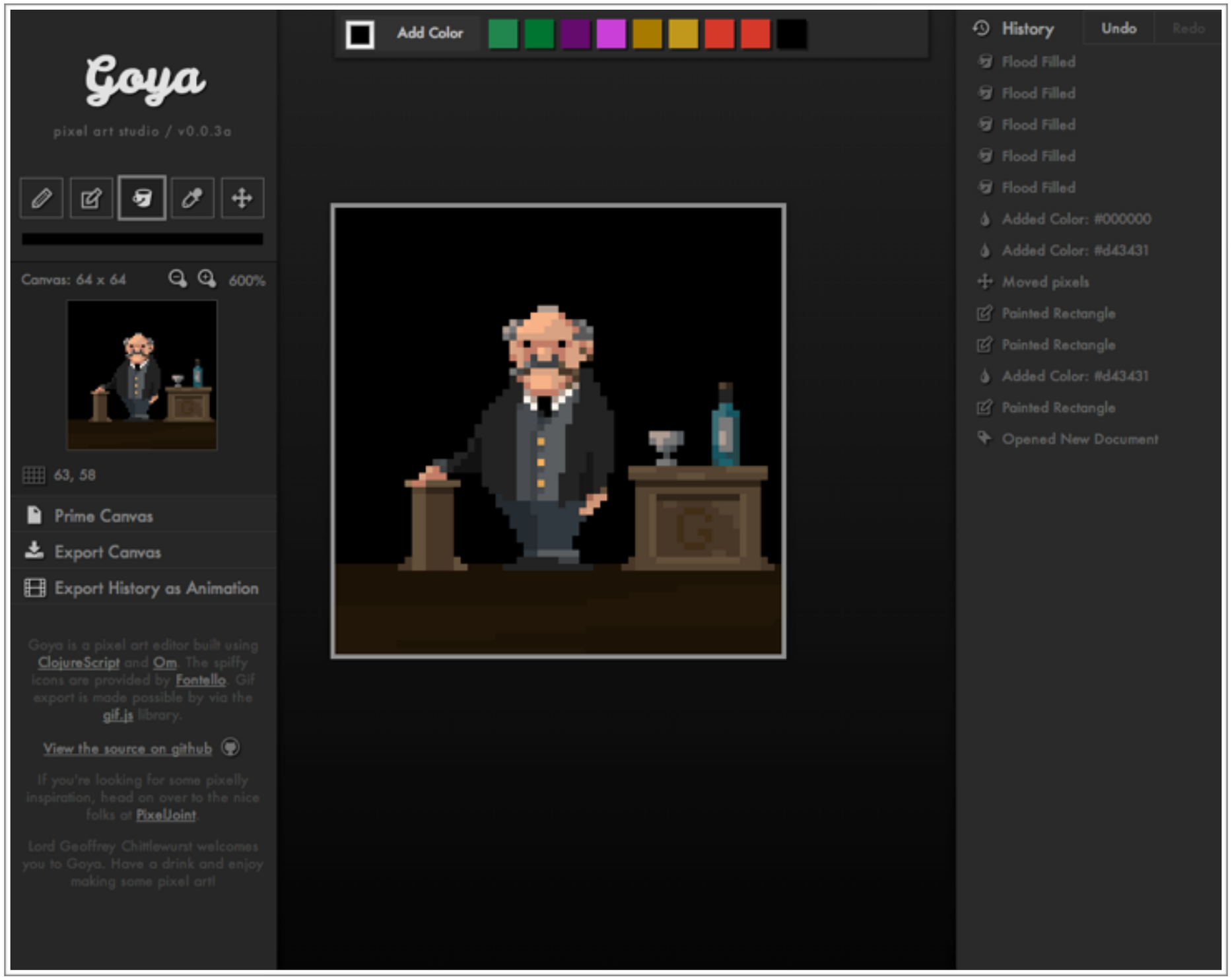
Om


React | A JavaScript library
facebook.github.io/react/
React
React
A JAVASCRIPT LIBRARY FOR BUILDING USER INTERFACES

$$f(D_0) = V_0$$

$$f(D_1) = V_1$$

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


Goya
pixel art studio / v0.0.3a
Add Color
History
Undo
Redo
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
Canvas: 64 x 64
600%
63, 58
Prime Canvas
Export Canvas
Export History as Animation
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

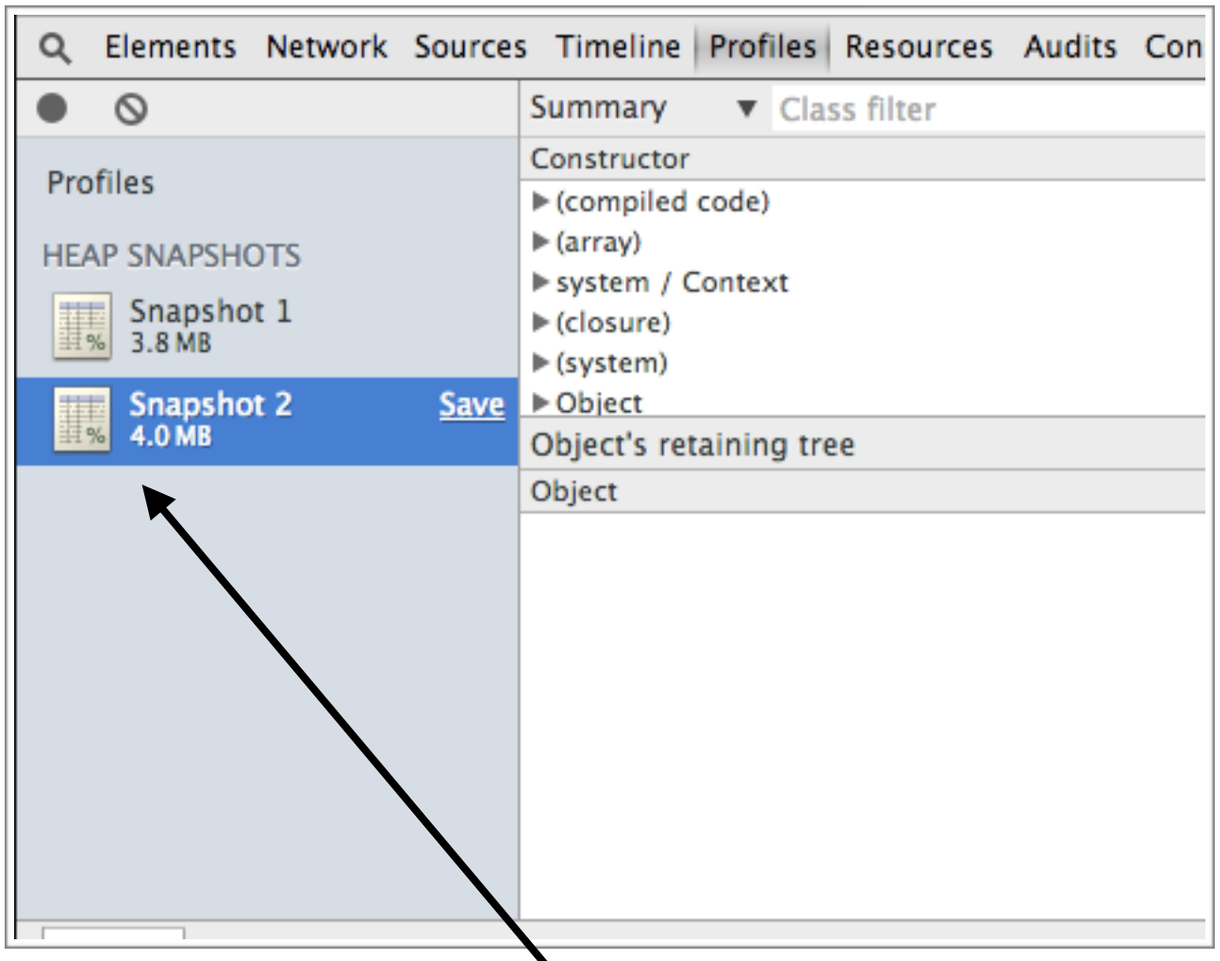
branch: master · goya / src / cljs / goya / timemachine.cljs

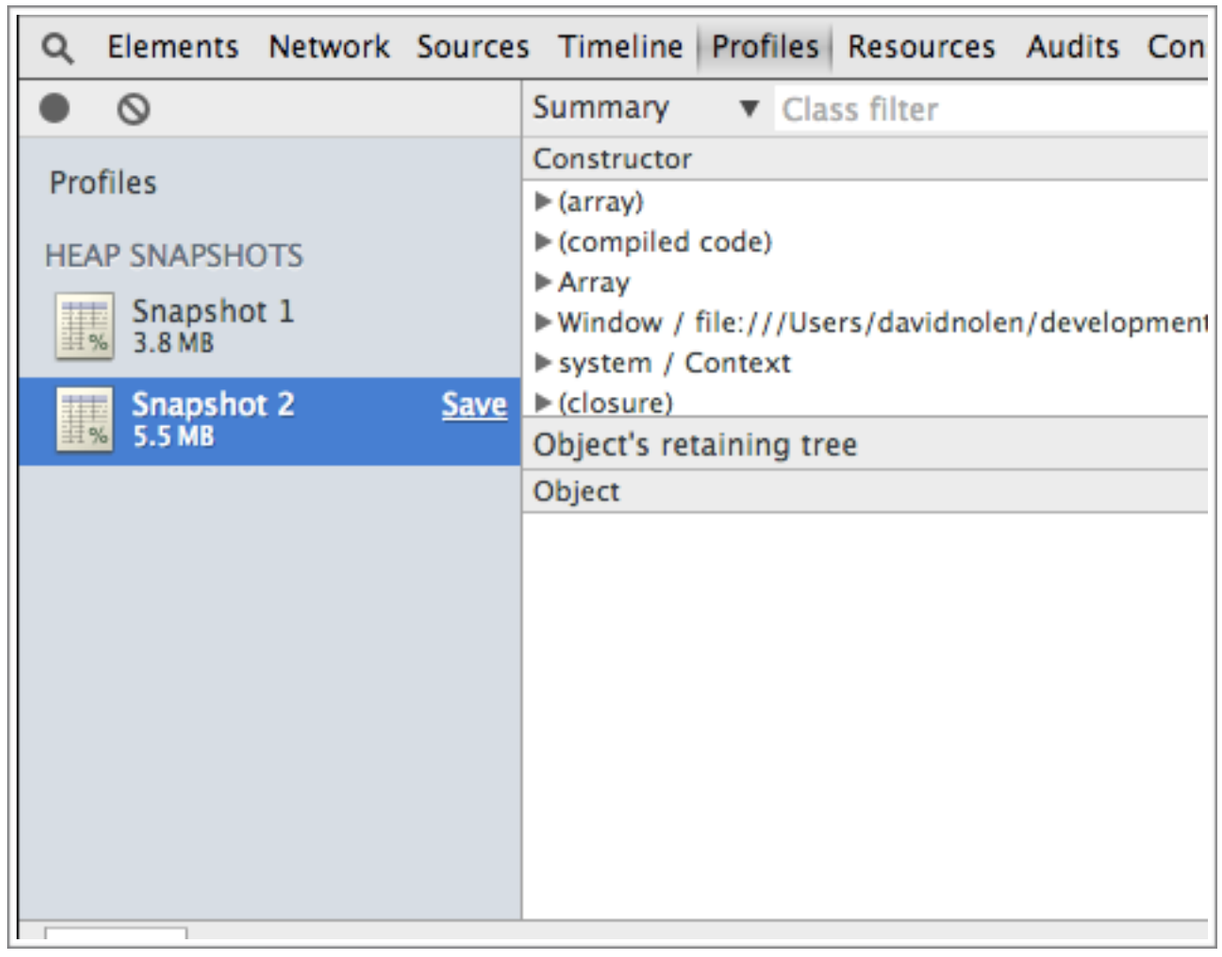
swannodette 13 days ago Project layout refactor, better production settings

1 contributor

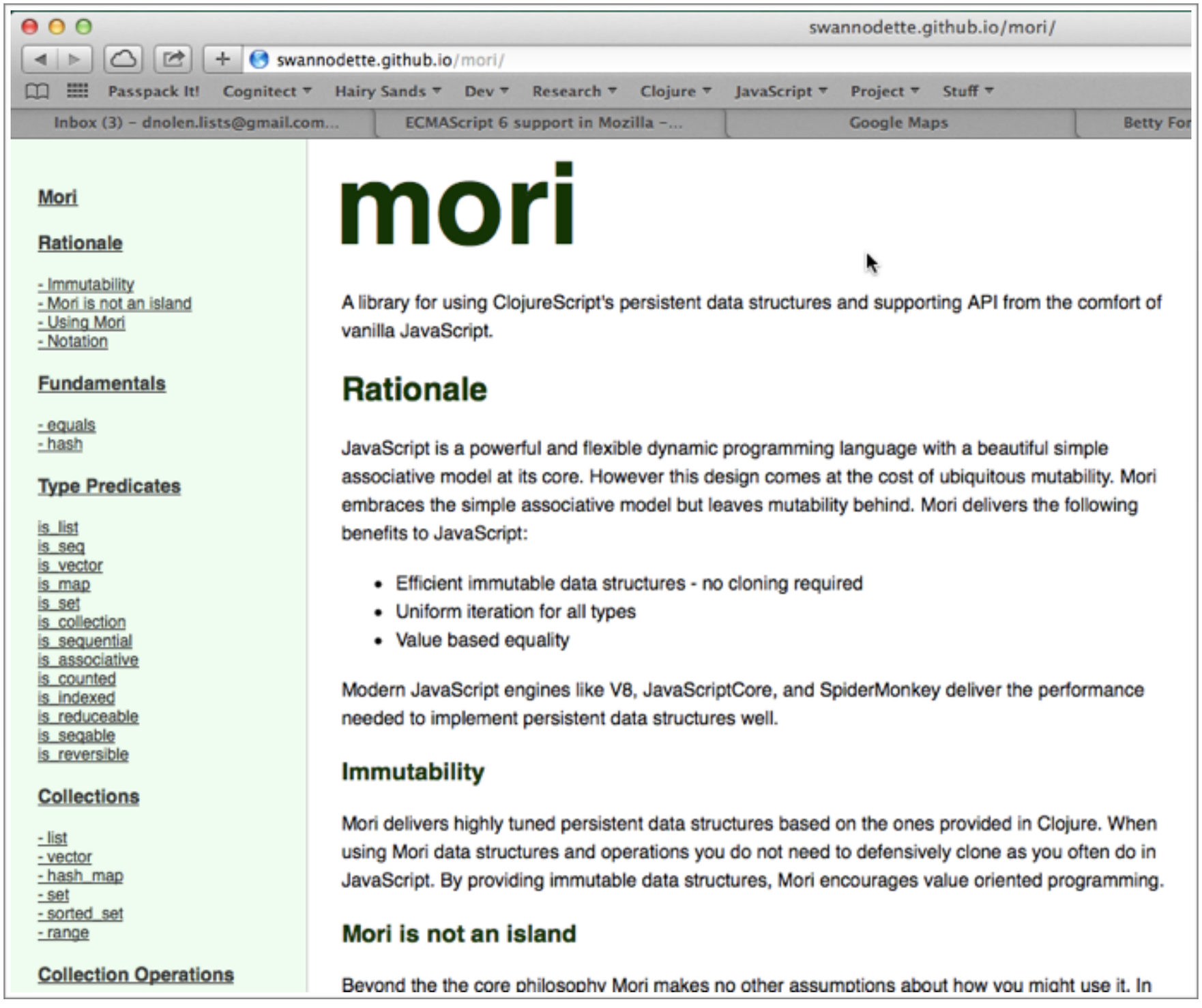
file 62 lines (41 sloc) 1.85 kb — Open | Edit | Raw | Blame | History | Delete

```
(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))))
```





Persistent Data Structures ... ROCK


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
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

Questions?