A BRIEF HISTORY OF PROTOTYPES

[KAT MARCHÁN  - NORDIC.JS 2017]

PROTOTYPE-BASED OBJECT-ORIENTED PROGRAMMING

(PROTOTYPE OOP)

("POOP"? 💩)

CLASSES ARE BLUEPRINTS

```ruby
class Dog
  def initialize(name)
    @name = name
  end
  def bark
    puts "#{@name} goes Woof!"
  end
end

dogen = Dog.new('Dogen')
dogen.bark # prints: Dogen goes Woof!
```

CLASSES ARE BLUEPRINTS

```
class Pug < Dog
  # Override the bark method in the Dog class.
  def bark
    puts "#{@name} goes Yip! Yip!"
  end
end

butch = Pug.new('Butch')
butch.bark # prints: Butch goes Yip! Yip!
```

PROTOTYPES ARE EXEMPLARS

```
// The "prototypical" dog is a plain JS object.
const dog = {}

// Give it a name and breed
dog.name = 'a dog'
dog.breed = 'unknown'

// and teach it how to bark
dog.bark = function () {
  console.log(this.name, 'goes Woof!')
}

// Now we have a dog that barks!
dog.bark() // prints: 'a dog goes Woof!'
```

PROTOTYPES ARE EXEMPLARS

```
// *Clone* the "prototypical" dog from before
const dogen = Object.create(dog)

// Specialize only the name
dogen.name = 'Dogen'

// Reuse the `bark` method from the other dog.
dogen.bark() // prints: 'Dogen goes Woof!'

// Breed is delegated, too!
console.log(dogen.breed) // prints: 'unknown'
```

JAVASCRIPT IS A PROTOTYPE-BASED LANGUAGE

JAVASCRIPT HAS NO CLASSES

```
// Good ol' constructor-based JS
function Dog (name) {
  this.name = name
}

// Change the prototype of the constructor!
Dog.prototype = dog

const charlie = new Dog('Charlie')
charlie.bark() // prints: 'Charlie goes Woof!'
```

JAVASCRIPT HAS NO CLASSES

```
// Shiny new class syntax!
class ClassyDog extends Dog {
  constructor (name) {
    this.name = name
  }
}

// ClassyDog has a full-fledged prototype!
ClassyDog.prototype.bark() // prints: 'a dog goes Woof!'

const butch = new ClassyDog('Butch')
butch.bark() // prints: 'Butch goes Woof!'

Object.create(ClassyDog.prototype) // This still works!
```

PROTOTYPES

ARE ABOUT INTERACTION

ARE ABOUT FLEXIBILITY

ARE ABOUT PLAY 

THIS TALK IS ABOUT HISTORY

THIS TALK IS ABOUT THE FUTURE

THIS STORY STARTS 40+ YEARS AGO.

DIRECTOR (1976)

TAKING ACTION ON THE ACTOR MODEL

THE ACTOR MODEL


HELLO, MIKE
imgflip.com


HERE
NO, THIS IS PATRICK.


THE REBELLION BEGINS
Erlang
The Movie
ERLANG SOLUTIONS PRESENTS
AN ETC PRODUCTION "ERLANG - THE MOVIE" MIKE WILLIAMS JOE ARMSTRONG ROBERT VIRDING BJARNE DÄCKER FEATURING FUNCTIONAL PROGRAMMING
BASED ON A NOVEL BY AGNER KRARUP ERLANG SPECIAL EFFECTS BY ERICSSON SCREENPLAY BY SYNAPSE MOBILE NETWORKS PRODUCTION DESIGN BY KLARNA DIRECTOR OF PRODUCTION MOBILE ARTS EXECUTIVE PRODUCER TAIL-F SYSTEMS
PRODUCED BY CORELATUS CREATIVE DESIGNER PROCESSONE SET MANAGER SJÖLAND & THYSELIUS TESTING SUPERVISOR QUVIQ
ERLANG
tail-f
coreletus
ERICSSON
klarna
process one
Sjöland&Thyselius
O'REILLY
Mobile Apps
QuviQ
synapse mobile networks
Erlang

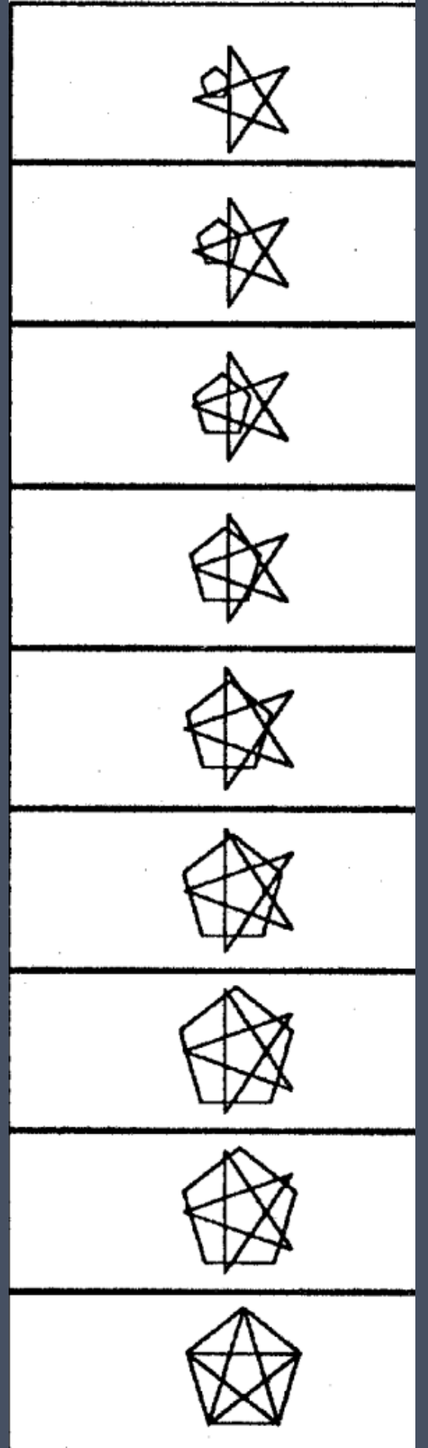
DIRECTOR'S LISP-BASED "ACTORS"

- EARLY WORK IN COMPUTER GRAPHICS
- MESSAGE-PASSING WITH `(ask ...)`
- OPTIONAL ASYNCHRONOUS EVENT LOOP
- DELEGATION-BASED INHERITANCE

MAKING A MOVIE

```
(ask movie make my-first-film) ;; clone movie
(ask default-clock set your frames-per-second to 2)

(ask star plan next do at speed 25 move forward 100)
(ask star plan after 2 seconds grow 50)
(ask star plan after 4 seconds turn right 18)

(ask pentagon plan next gradually grow 300)
(ask pentagon plan next do in 4 seconds move back 200)

(ask my-first-film film the next 4 seconds) ;; record scene
(ask my-first-film project) ;; play back scene
```

CREATES AN ANIMATION
AND PLAYS IT BACK:



OBJECTLISP (1985)

YES. KAT LIKES LISP A LOT

(BEEP) (BOOP)

- USED FOR LISP MACHINE UI
- CALLED ITSELF OBJECT-ORIENTED
- MORE INFLUENCED BY CLASS-BASED OOP

```
(setq icon (make-obj))

(defobfun (goto-xy icon) (x y)
  ;; Do stuff
  (undraw-self x-coord y-coord)
  (draw-self x y)
  ;; Record the new position for next time.
  (setq x-coord x)
  (setq y-coord y))

;; Clone the `icon` prototype
(setq icon1 (kindof icon))

;; Use `ask`, like in Director
(ask icon1 (goto-xy 100 100))
```

```
(setq icon (make-obj))

(defobfun (goto-xy icon) (x y)
  ;; Do stuff
  (undraw-self x-coord y-coord)
  (draw-self x y)
  ;; Record the new position for next time.
  (setq x-coord x)
  (setq y-coord y))

;; Clone the `icon` prototype
(setq icon1 (kindof icon))

;; Use `ask`, like in Director
(ask icon1 (goto-xy 100 100))
```

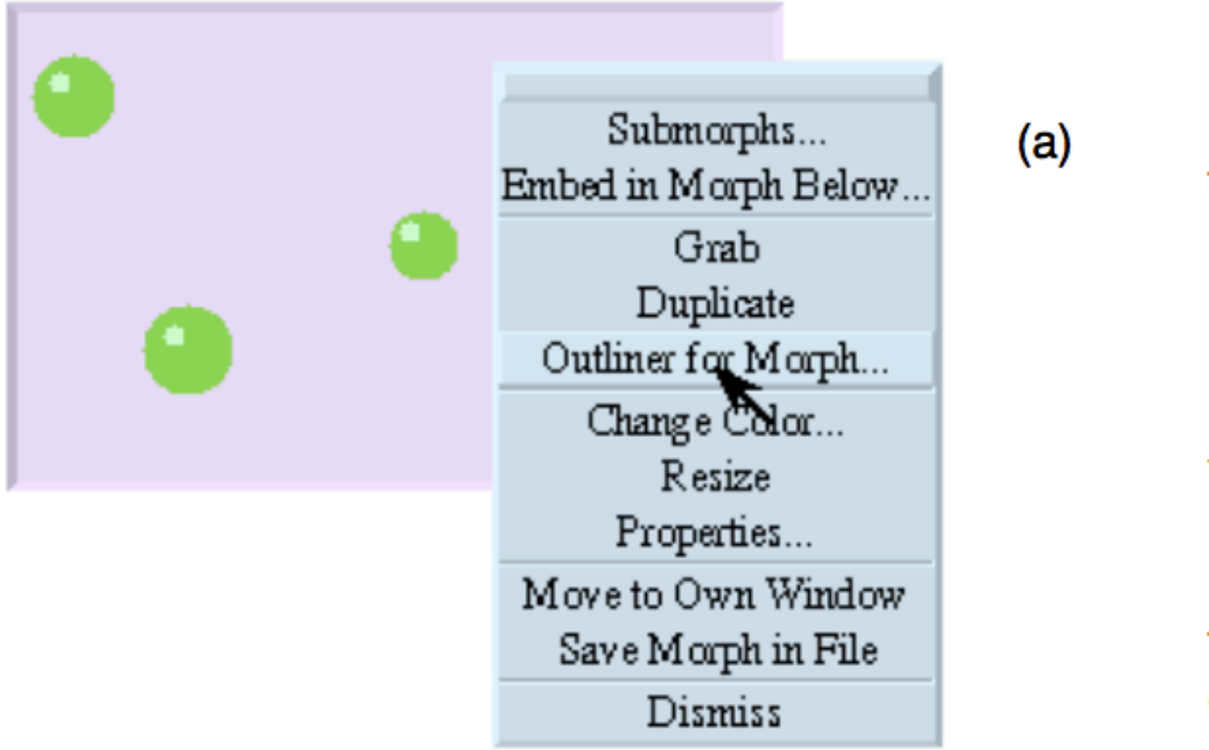
```
var icon = new GraphicsObject()

icon.gotoXY = function (x, y) {
  // Do stuff
  this.undraw(this.x, this.y)
  this.draw(x, y)
  // Record the new position for next time
  this.x = x
  this.y = y
}

// Make a clone
var icon1 = Object.create(icon)

// Call the method
icon1.gotoXY(100, 100)
```

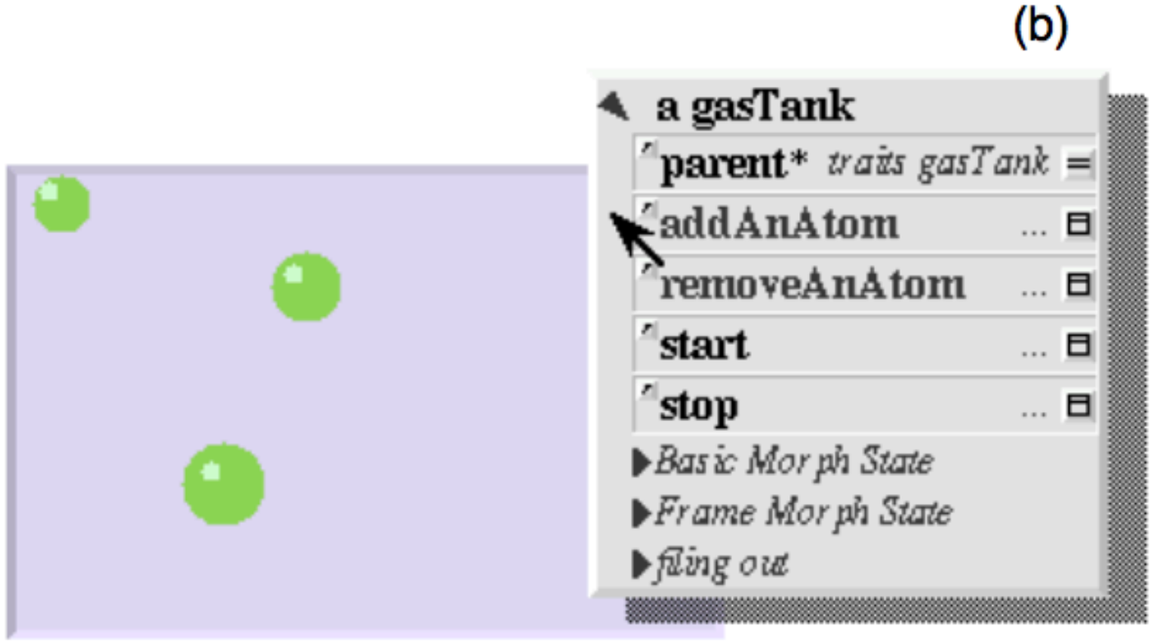
SELF (1987)

PROTOTYPES ALL THE WAY DOWN 🐢

PROGRAMMING AS AN EXPERIENCE

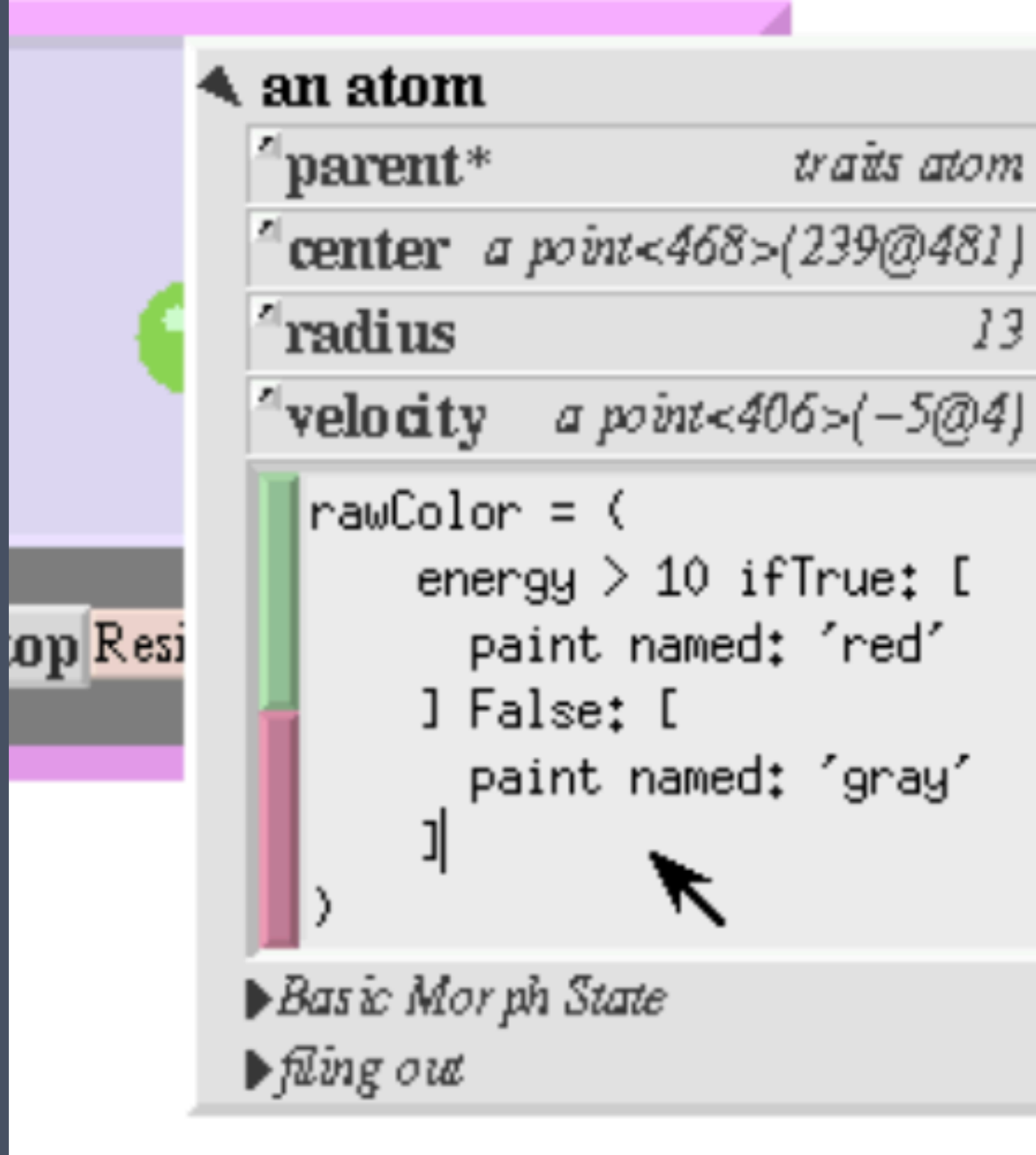
- XEROX PARC RESEARCH
- SMALLTALK WITH PROTOTYPES
- GRAPHICAL INTERFACE TO CODE
- MORE TANGIBLE OBJECTS
- LOTS OF UI/UX WORK





PROGRAMMING AS AN EXPERIENCE

- XEROX PARC RESEARCH
- SMALLTALK WITH PROTOTYPES
- GRAPHICAL INTERFACE TO CODE
- MORE TANGIBLE OBJECTS
- LOTS OF UI/UX WORK



PROTOTYPES. BUT FAST

- INLINE CACHES
- HIDDEN CLASSES ("MAPS")
- ADAPTIVE OPTIMIZATION
- HIDE COMPLEXITY FROM USER

LANGUAGE INNOVATION

- INHERITANCE
- SLOT SEMANTICS
- TONS OF PUBLISHED PAPERS
- XEROX PARC IS COOL
- BIGGEST INFLUENCE ON JS

Retrospective

- **Programming as an Experience: The Inspiration for Self**

Language

- **Self: The Power of Simplicity**
- **Parents are Shared Parts: Inheritance and Encapsulation in Self**
- **Organizing Programs Without Classes**

Implementation

- **Object Storage and Inheritance for Self**
- **Customization: Optimizing Compiler Technology for Self, a Dynamically-Typed Ob**
- **An Efficient Implementation of Self, a Dynamically-Typed Object-Oriented Languag**
- **Iterative Type Analysis and Extended Message Splitting: Optimizing Dynamically-T**
- **Making Pure Object-Oriented Languages Practical**
- **Optimizing Dynamically-Typed Object-Oriented Programming Languages with Poly**
- **The Design and Implementation of the Self Compiler, an Optimizing Compiler for C**
- **Debugging Optimized Code with Dynamic Deoptimization**
- **Object, Message, and Performance: How They Coexist in Self**
- **A Fast Write Barrier for Generational Garbage Collectors**
- **Optimizing Dynamically-Dispatched Calls with Run-Time Type Feedback**
- **Adaptive optimization for Self: Reconciling High Performance with Exploratory Pro**
- **A Third-Generation Self Implementation: Reconciling Responsiveness with Perfor**

- **Do object-oriented languages need special hardware support?** (ECOOP '95), **Urs H**

User Interface

- **Animation: From Cartoons to the User Interface**
- **Experiencing Self Objects: An Object-Based Artificial Reality**
- **The Use-Mention Perspective on Programming for the Interface**
- **Getting Close to Objects: Object-Focused Programming Environments**

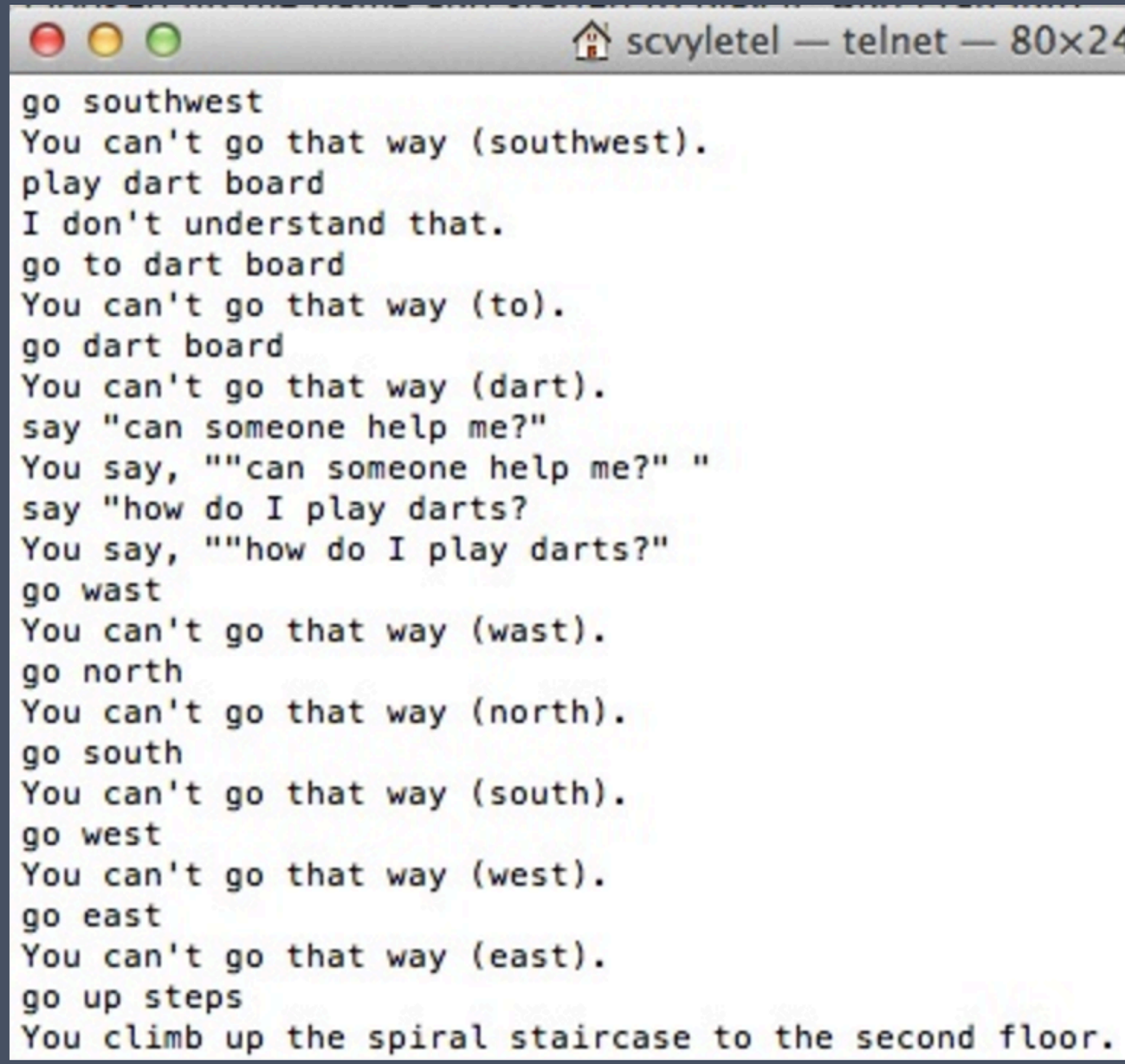
LAMBDAMOO (1990)

PLAY AS PROGRAMMING

PROGRAMMING AS PLAY

MUD: OBJECT-ORIENTED

- TEXT-BASED MUD
- FOUNDED AT XEROX PARC (BY SELF FOLKS)
  - ONLINE PLAYERS BUILDING THE WORLD
    - EARLY VIRTUAL WORLD RESEARCH



```
$ telnet lambdamoo.local 8080
> connect wizard
*** Connected ***
The First Room
This is all there is right now.

> @dig n,north to "Nobelberget"                          <-- "dig" a brand new room to the north!
Nobelberget (#96) created.
Exit from The First Room (#62) to Nobelberget (#96)
via {"n", "north"} created with id #97.

> north                                                  <-- move into the new room
Nobelberget
You see nothing special.

> @describe here as "The Nordic.js conference venue."    <-- set its description
Description set.

> look                                                   <-- behold the fruits of your labor!
Nobelberget
The Nordic.js conference venue.
```

```
> @create $thing called audience                          <-- now create a regular object
You now have audience with object number #98
and parent generic thing (#5).

> @describe audience as "A captive audience full         <-- and set its description
of excellent folks."
Description set.

> look audience                                           <-- let's take a gander, now
A captive audience full of excellent folks.

> @verb audience:entertain this                           <-- add a verb to audience
Verb added (1).

> @edit audience:entertain                                <-- drop into the editor
...(editor stuff)...
player:tell("The audience cheers for you! clap clap!"); <-- write the code and compile it!
...(compile and exit editor)...

> entertain audience
The audience cheers for you! clap clap!                   <-- Wow ilu all 😭💚
```

"LAMBDAMOO IS A NEW KIND OF SOCIETY"

- LAMBDAMOO WELCOME MESSAGE

```
$ telnet lambda.moo.mud.org 8888
```

NEWTONSCRIPT (1993)

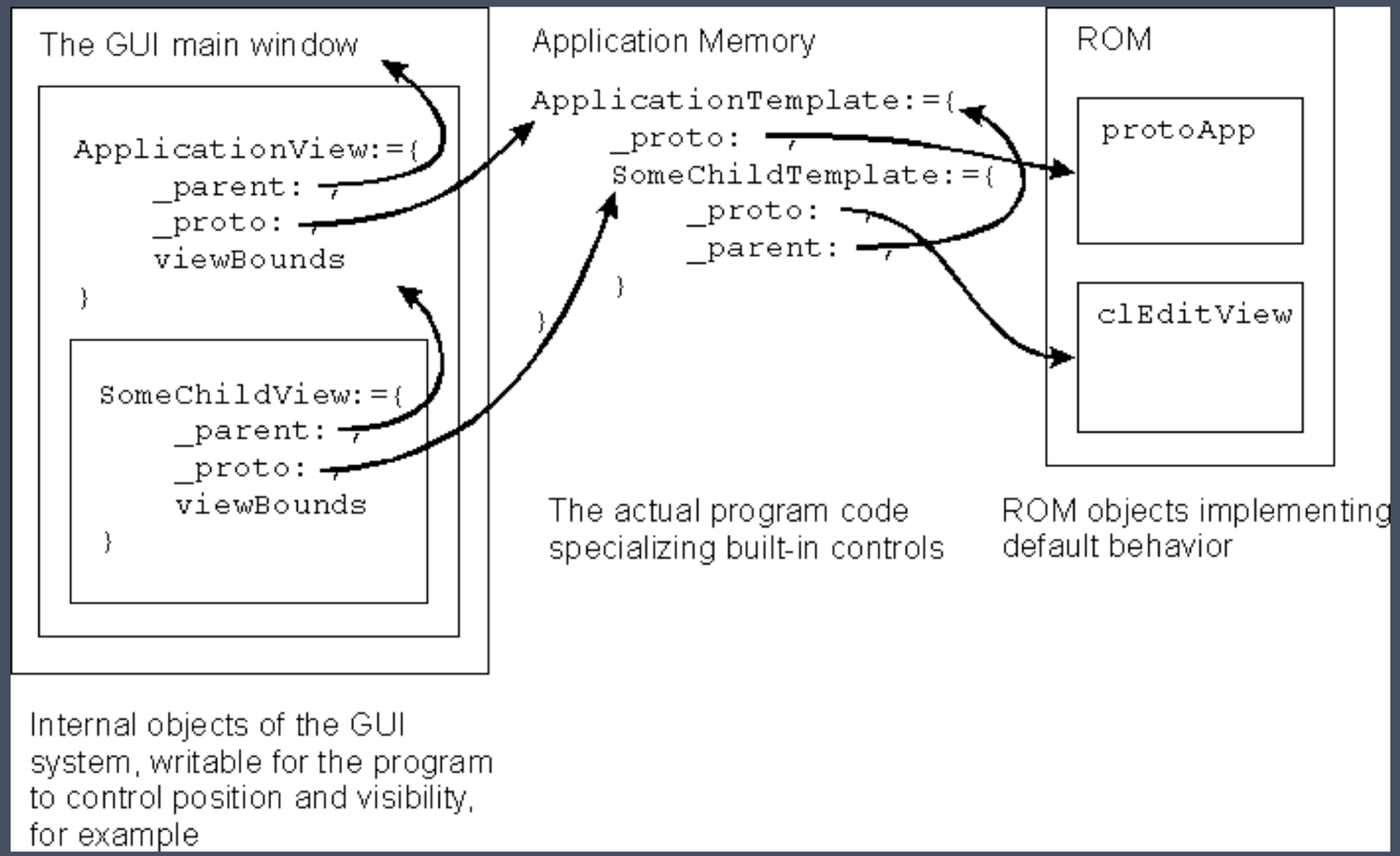
PRICES FOR RAM ARE TOO DAMN HIGH


NEWTON
TECHNOLOGY
Hello,
MessagePad 2100
HUEG
AT&T
1:05 PM
Notes
Hello, Ne...
Today
Jul 7 12:53 PM
Hello, Newton.

EMBEDDED PROTOTYPES

- APPLE NEWTON DEVICES
- 640KB RAM TOTAL (128KB FOR OS)
- MUCH CHEAPER 4MB ROM
- SELF VM STARTED AT 32MB

DELEGATION TO SAVE RAM



LUA (1993)

JAVASCRIPT FOR C++ PEOPLE 😏

CUSTOM SOFTWARE. FAST 🏃🏽‍♀️

- TECGRAF IN RIO DE JANEIRO
- TRADE BARRIERS -> NIH 👷🏽‍♀️🛠️
- TCL -> 😴
- LISP SEMANTICS -> 👍🏽
- LISP SYNTAX -> 👎🏽
- COMPUTER GRAPHICS -> 🚀 + 📲

```lua
local Vector = {}
Vector.__index = Vector

function Vector:new(x, y, z)
  local vec = {x = x, y = y, z = z}
  return setmetatable(vec, Vector)
end

function Vector:magnitude()
  return math.sqrt(self.x^2 +
                   self.y^2 +
                   self.z^2)
end

local vec = Vector:new(0, 1, 0)
print(vec:magnitude())
```

- TABLES (OBJECTS)
- METATABLES (INHERITANCE)
- ACTUALLY QUITE SNAPPY
- POPULAR IN GAMES INDUSTRY

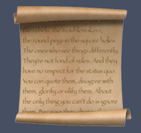
```lua
local Vector = {}
Vector.__index = Vector

function Vector:new(x, y, z)
  local vec = {x = x, y = y, z = z}
  return setmetatable(vec, Vector)
end

function Vector:magnitude()
  return math.sqrt(self.x^2 +
                   self.y^2 +
                   self.z^2)
end

local vec = Vector:new(0, 1, 0)
print(vec:magnitude())
```

```coffee
function Vector (x, y, z) {
  this.x = x
  this.y = y
  this.z = z
}

Vector.prototype.magnitude = function () {
  return Math.sqrt(this.x ** 2 +
                   this.y ** 2 +
                   this.z ** 2)
}

var vec = new Vector(0, 1, 0)
console.log(vec.magnitude())
```

JAVASCRIPT (1995)

OUR BEAUTIFUL PROBLEMATIC CHILD

EVERYONE'S FAVORITE 10-DAY HACK

- MOST WIDELY-USED PROTOTYPE LANGUAGE 🌏
- WEB PLATFORM SCRIPTING 
- EASY TO LEARN 👩🏼‍🏫
- SCHEME👍🏼 + SELF👍🏼 + JAVA'S SYNTAX🤷🏽‍♀️

IT'S ACTUALLY PRETTY GREAT

- PROTOTYPES ARE A GREAT FIT FOR THE DOM
- WEBDEV IS REALLY FUN
- ECMASCRIPT STANDARD AND ES-NEXT

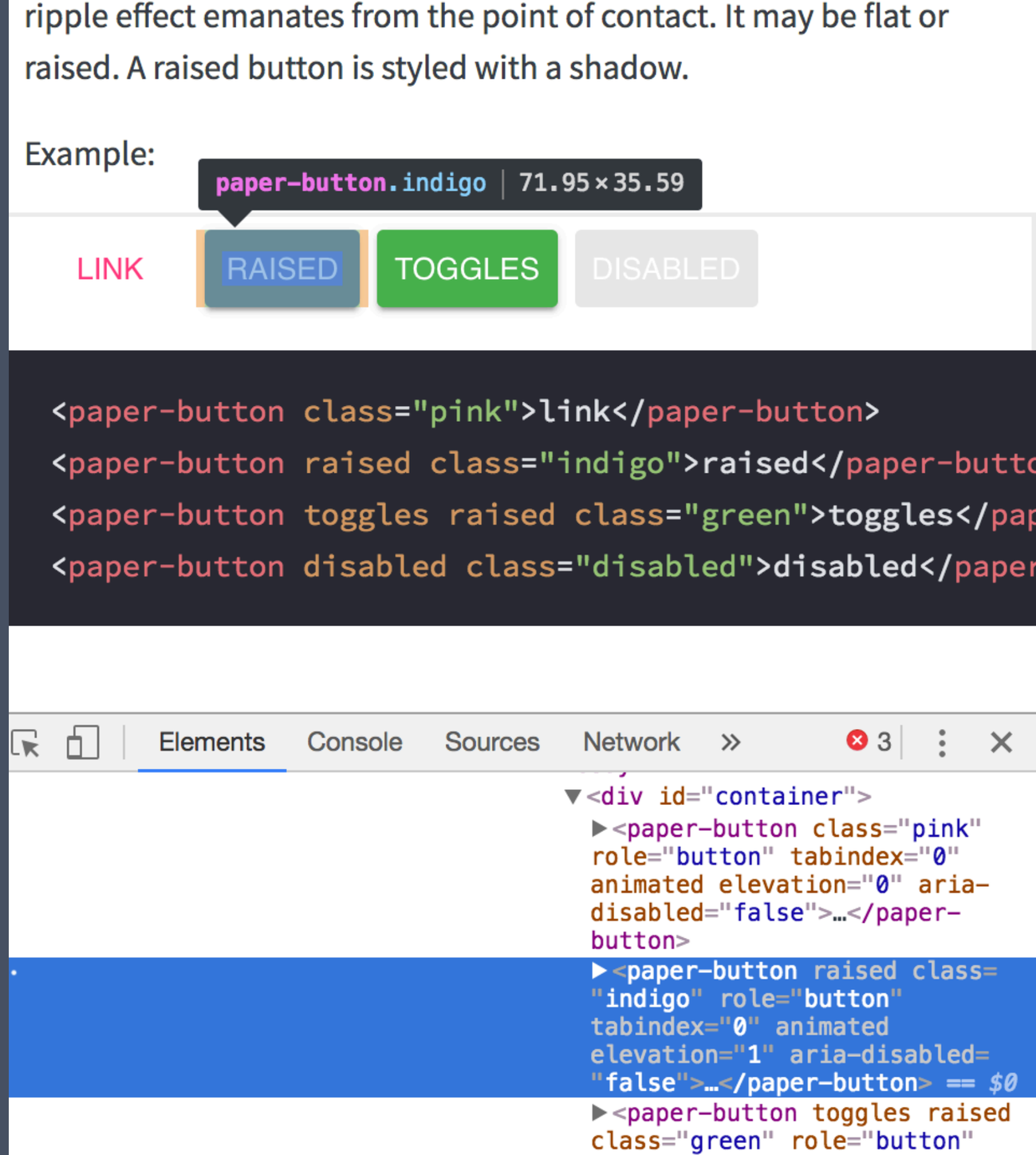
THE FUTURE OF JS

(BASED ON THE PAST )

WEBPACK LESS

HACK MORE

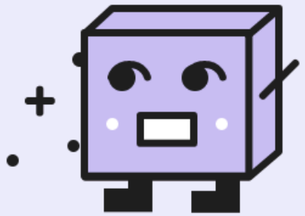
- MORE INTERACTIVE DEV 📲
- KEEP EVOLVING DEVTOOLS ⚒
- WEB COMPONENTS 💚



KEEP JS EASY

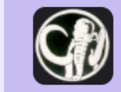
- THE POWER OF SIMPLICITY 👶🏾
- WEB PLATFORM AS A LITTLE OS 👾
- IMMEDIATE RESULTS ARE DELIGHTFUL ☺️
- GLITCH.COM🎏 IS FUN

Handy Bots →



Build helpful tools, meme generators, or Westworld. Your bots have your back.

byronhulcher



tracery-mastodon-bot

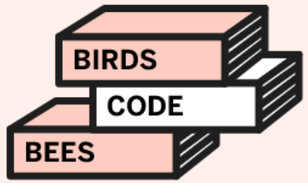
A starter Mastodon bot that generates random toots using Tracery

fourtonfish

twitterb

A template for making fun Tw bots with the Node.js library

Learn to Code →



Learn by doing, then breaking, then doing some more. You got this!

davidtmiller

freelancer-theme

A one page Bootstrap website theme for freelancers

manuelkiess

node-beginne

A beginner's g Node.js and JavaScript. Ge started by sele

GET FANCY SOMETIMES!

- FRP WITH RX.JS / BACON.JS 🥓
- LEARN TO LOVE `Object.create()` 🤓
- COMPILE-TO-JS LETS US EXPLORE 🎮

```
-- DISPLAY

display : (Int, Int) -> Model -> Element
display (w',h') mario =
  let (w,h) = (toFloat w', toFloat h')

      verb = if | mario.y  >  0 -> "jump"
                | mario.vx /= 0 -> "walk"
                | otherwise     -> "stand"

      dir = case mario.dir of
              Left -> "left"
              Right -> "right"

      src  = "imgs/mario/"++ verb ++ "/" ++ dir ++ ".gif"

      marioImage = image 35 35 src

      groundY = 62 - h/2
  in
      collage w' h'
          [ rect w h
              |> filled (rgb 174 238 238)
          , rect w 50
              |> filled (rgb 74 167 43)
              |> move (0, 24 - h/2)
          , marioImage
              |> toForm
              |> Debug.trace "mario"
              |> move (mario.x, mario.y + groundY)
          ]


-- SIGNALS

main : Signal Element
main = lift2 display Window.dimensions (foldp step mario inpu

input : Signal (Float, Keys)
input =
```

THE PAST WAS PRETTY COOL 

AND THE FUTURE IS REALLY EXCITING

HAPPY HACKING!