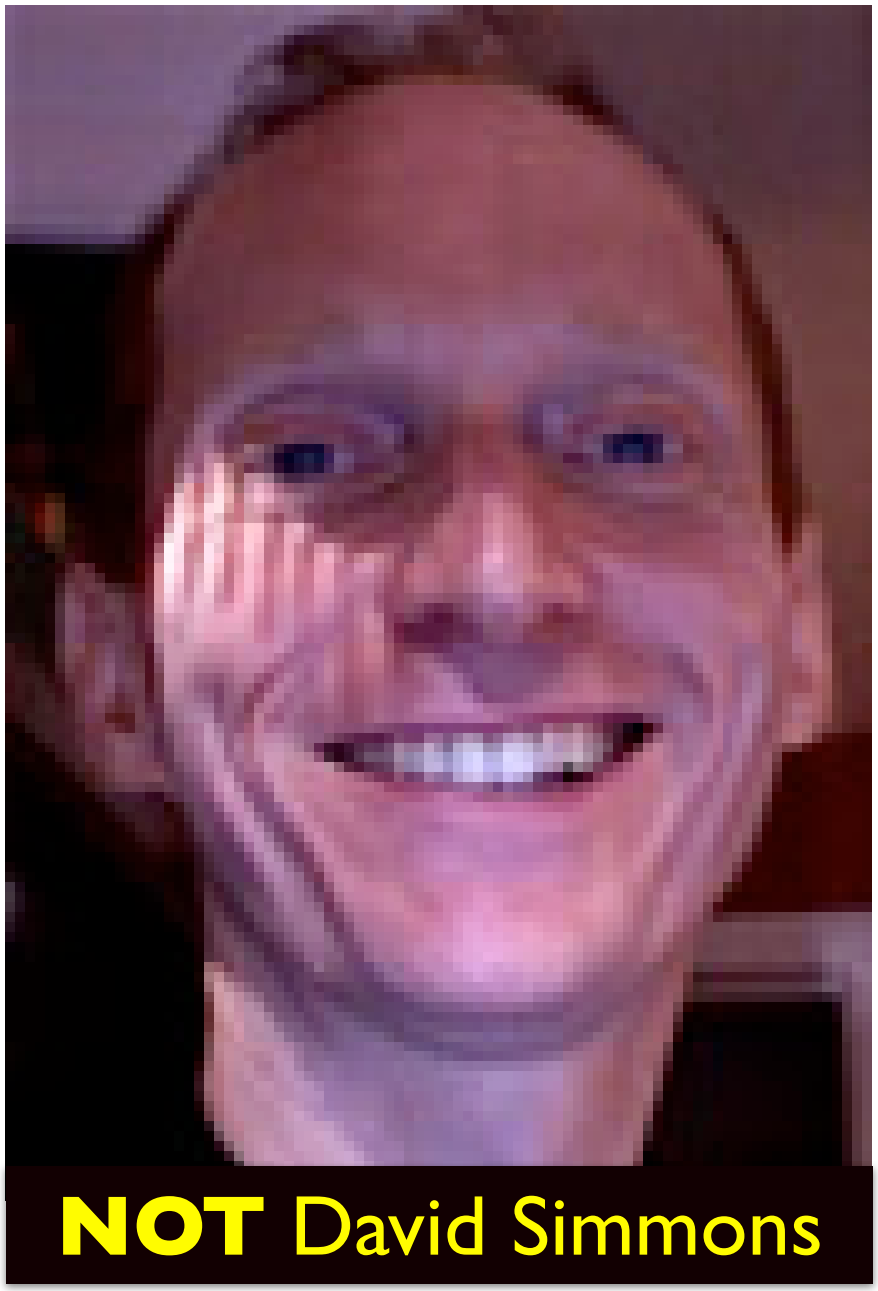
Implementing Programming Languages for Fun and Profit with **OMeta**

Alessandro Warth
Viewpoints Research Institute & UCLA

Who am I?


NOT David Simmons

NOT Billy Idol

programming languages

STEPS

... toward the reinvention of programming

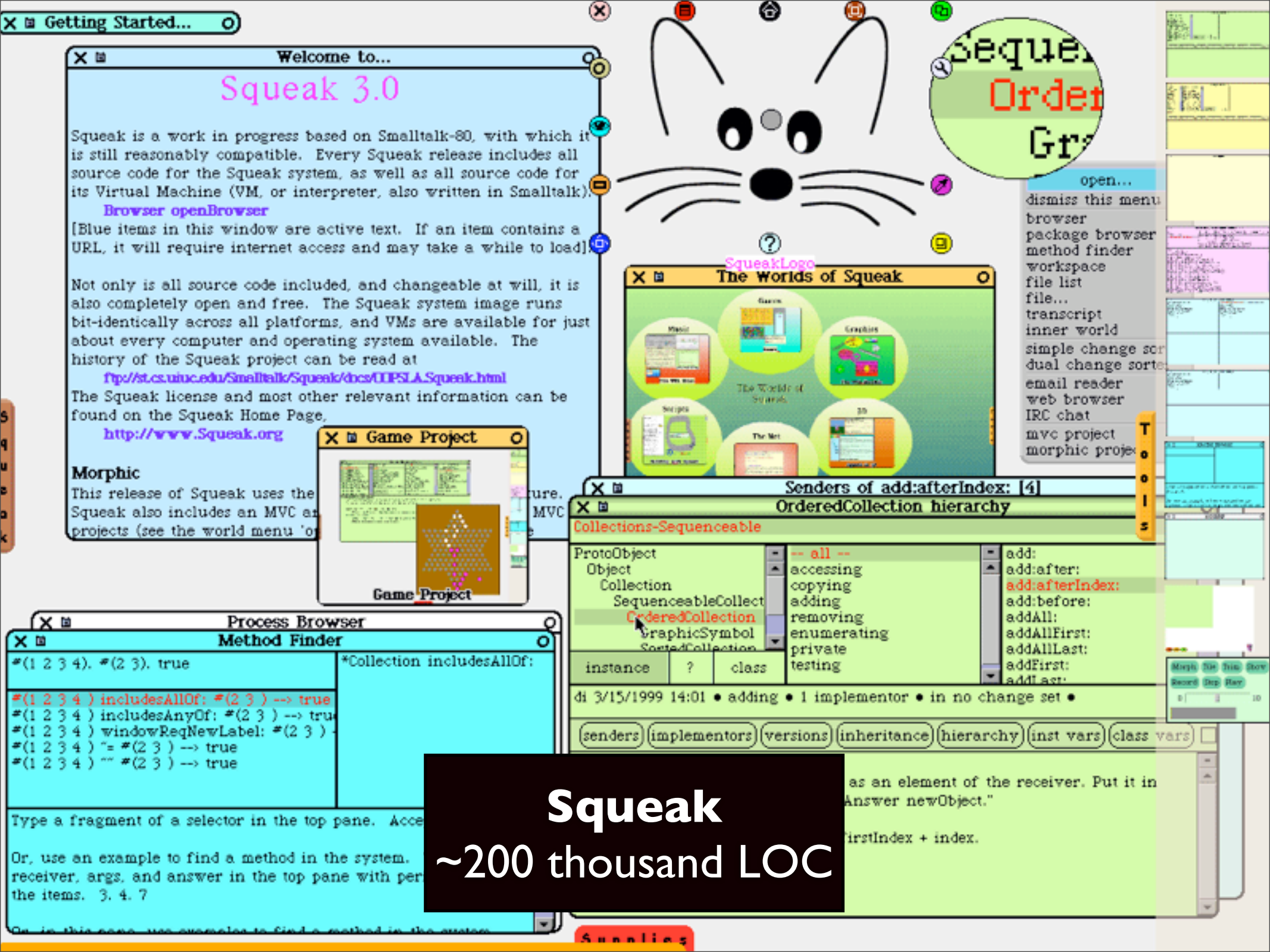
The STEPS Project

- **Goal:** To create a highly useful end-user system including...
  - operating system
  - programming environment
  - "applications"
  - graphics, sound

} *personal computing*


... in under
20,000 LOC!


Picture Tasks
View as a slide show
Order prints online
Print pictures
Copy all items to CD
Shop for pictures online
File and Folder Tasks
Make a new folder
Publish this folder to the Web
Share this folder
Other Places
Shared Pictures
My Pictures
My Computer
My Network Places
Details
Sample Pictures
File Folder
Date Modified: Friday, January 24, 2003, 3:06 AM
Sample Music
Music Tasks
Play all
Beethoven's Symphony No. 9 (Scherzo)
Ludwig van Beethoven, comp...
New Stories (Highway Blues)
Marc Seales, composer. New ...
Spealkin' Out
Control Panel
Switch to Classic View
See Also
Windows Update
Help and Support
Other Control Panel Options
Pick a category
Appearance and Themes
Printers and Other Hardware
Network and Internet Connections
User Accounts
Add or Remove Programs
Date, Time, Language, and Regional Options
Sounds, Speech, and Audio Devices
Accessibility Options
Performance and Maintenance
Security Center
Windows XP
~40 million LOC
start
Sample Pictures
Sample Music
Control Panel
11:28 AM

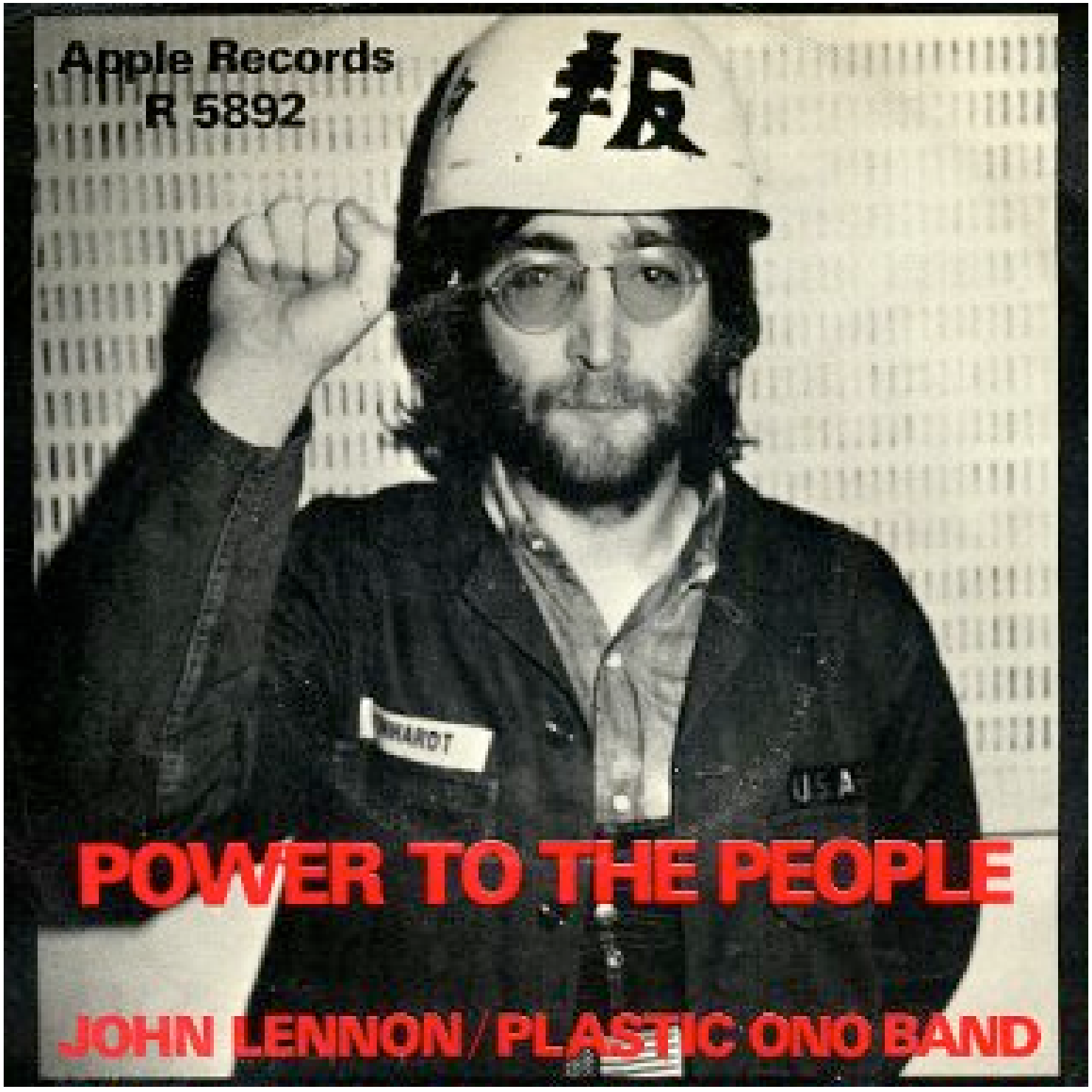


Squeak

~200 thousand LOC

Why?

- To put people in charge of their own SW destinies
  - No single person can understand 40,000,000 LOC (~library)
  - You can "own" 20,000 LOC (~book)


Apple Records
R 5892
反
USA
POWER TO THE PEOPLE
JOHN LENNON / PLASTIC ONO BAND

Why? (cont'd)

- Didactic value
  - a curriculum for university students to learn about powerful ideas
  - may even be good for AP C.S.

Programming Languages

- STEPS: code size, understandability
- Choice of programming language has a big impact on both
- What's the right one for STEPS?


Walt Disney RECORDS PRESENTS
Disneyland
Park
The Official Album

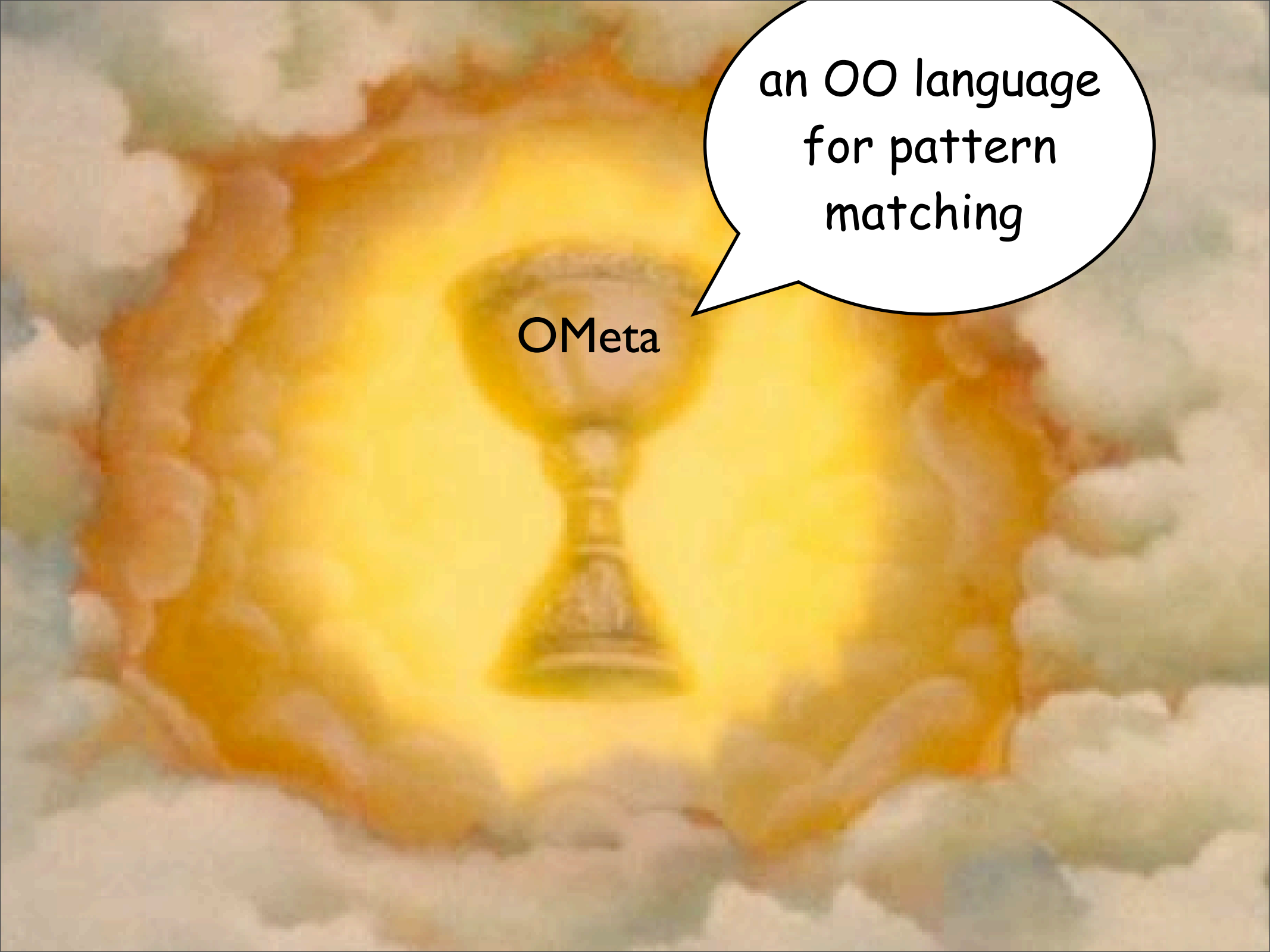
Long Lines...

- It takes lots of time and effort to implement a programming language
  - limits how much experimenting we can do

Big and Bad

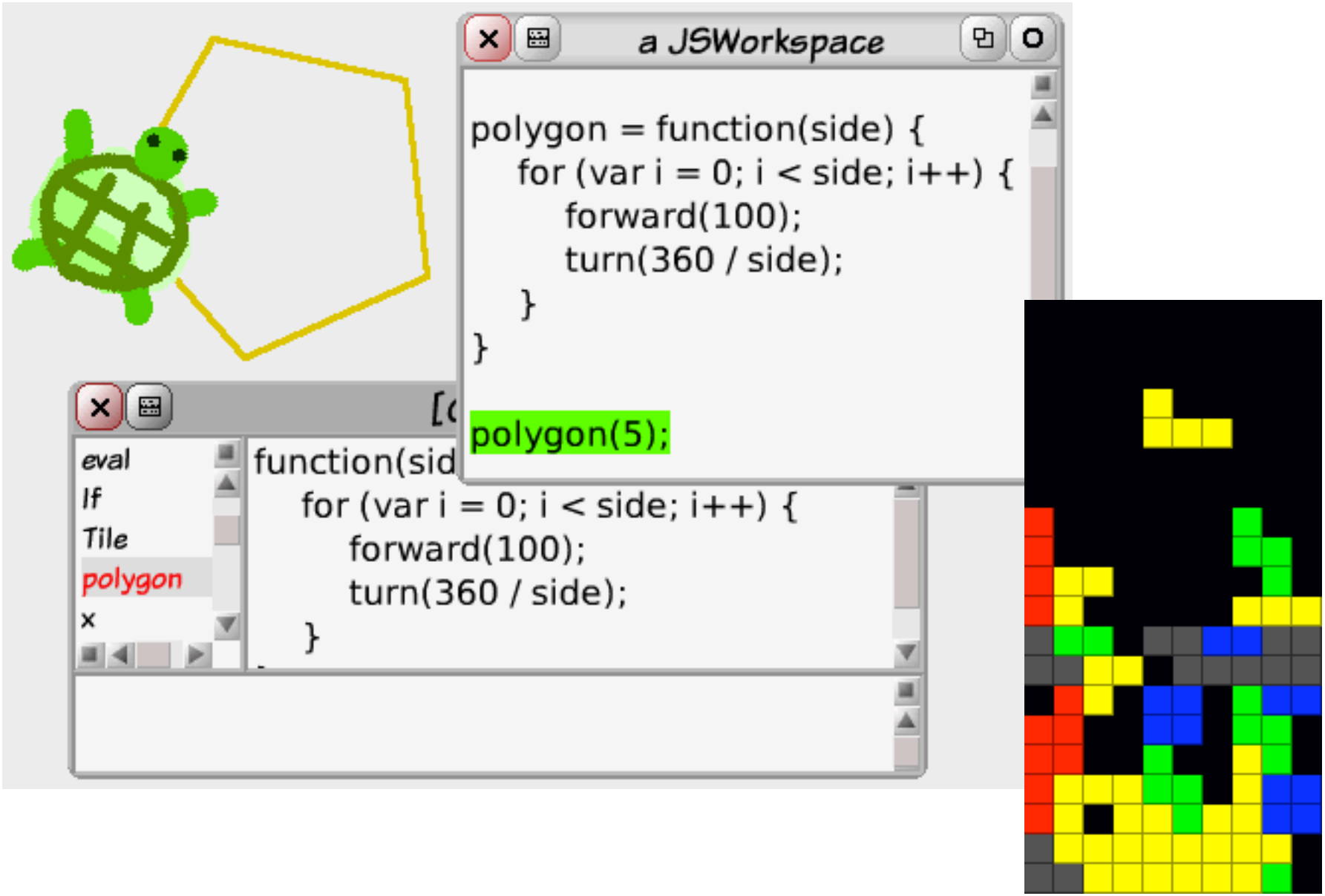
- Traditional PL implementations are **BIG**
  - only have 20k LOC for the whole thing!


OMeta

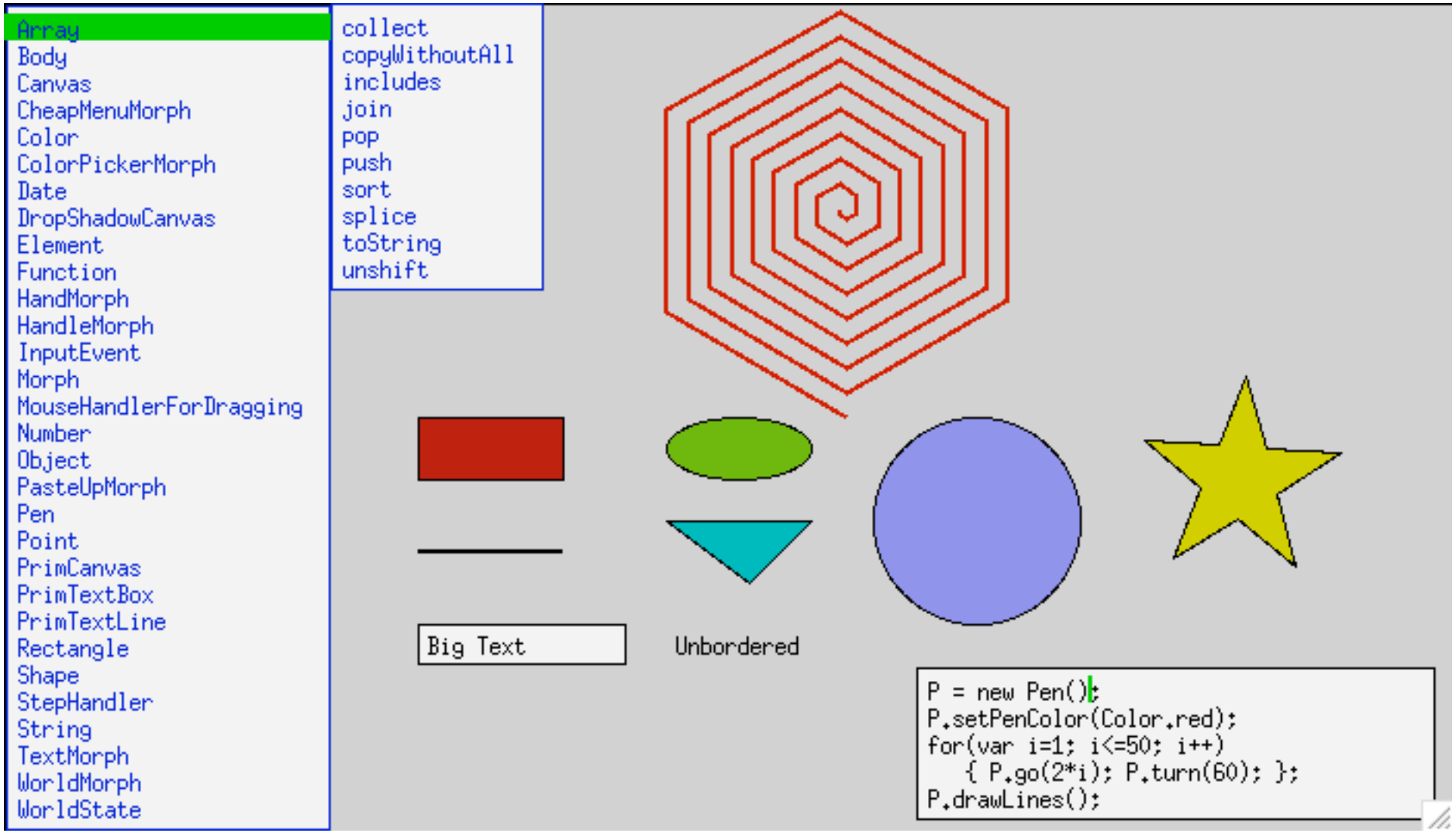

an OO language
for pattern
matching
OMeta


ometa®
inside

JavaScript (OMeta/Squeak)



Sun's Lively Kernel (OMeta/COLA)



Toylog

- Front-end to Prolog for children
- Runs on Squeak
- ~70 LOC

```
Homer is Bart's father.
Marge is Bart's mother.
x is y's parent if x is y's father or
                   or x is y's mother.
x is Bart's parent?
```

Toylog Demo

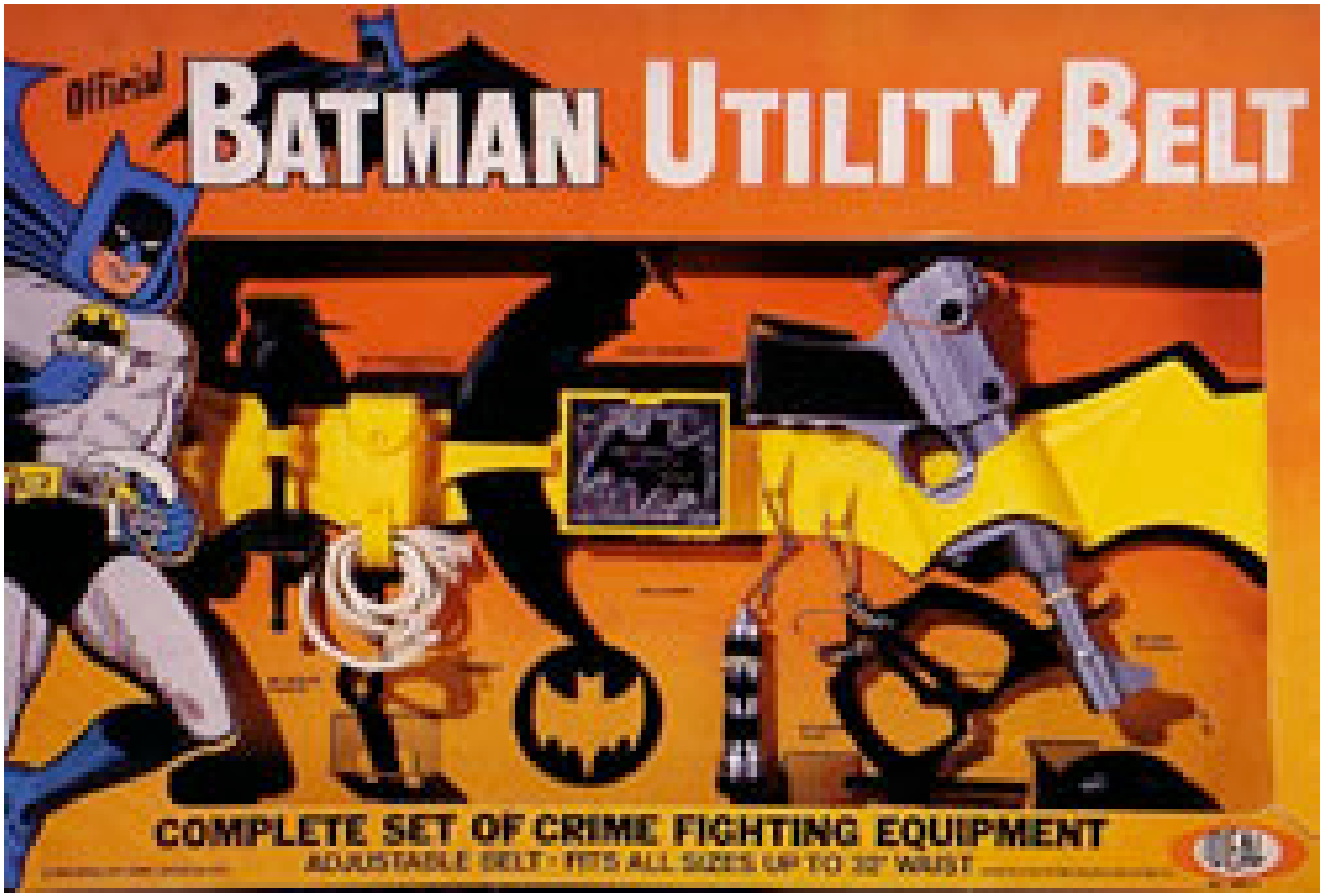
What can OMeta do for you?

- Make your life easier with DSLs
  - good DSLs come from people who need them
  - not PL people
- Make your apps scriptable by end-users
  - ... without making them learn Smalltalk

Roadmap

- A brief overview of OMeta
  - pattern matching
  - OO features
  - ...
- OMeta/JS

Traditional PL Implementation



Traditional PL Implementation

visitors,
for
AST transformations
and code generation

lex,
for lexical analysis

yacc,
for parsing

Pattern Matching: A Unifying Idea!

- **lexical analysis:** characters → tokens
- **parsing:** tokens → parse trees
- **constant folding and other optimizations:** parse trees → parse trees
- **code generation:** parse trees → code

Why use PM for everything?

- Simplicity
- Less stuff to learn (lowers learning curve)
- Great for extensibility
  - trad. impls hard to extend
  - OMeta: every part of PL impl. (e.g., parsing, tree traversals, codegen) can be extended using same mechanism

Pattern Matching

- Other langs have PM, do we need OMeta?
- ML-style pattern matching
  - great for tree transformations
  - not good for lexing/parsing
  - “That’s what ML-lex and ML-yacc are for!”
- OMeta is based on PEGs

Parsing Expression Grammars (PEGs) [Ford, '04]

- Recognition-based foundation for describing syntax
- Only **prioritized choice**
  - no ambiguities
  - easy to understand
- Backtracking, unlimited lookahead
- Semantic predicates, e.g., `?[x == y]`

PEGs, OMeta style

```
dig  ::=  $0 | ... | $9

num  ::= <num>   <dig>
       | <dig>

expr ::= <expr>   $+ <num>
       | <num>
```

PEGs, OMeta style

```
dig  ::= ($0 | ... | $9):d

num  ::= <num>:n <dig>:d
       | <dig>

expr ::= <expr>:e $+ <num>:n
       | <num>
```

PEGs, OMeta style

```
dig  ::= ($0 | ... | $9):d    => [d digitValue]

num  ::= <num>:n <dig>:d      => [n * 10 + d]
       | <dig>

expr ::= <expr>:e $+ <num>:n => [{#plus. e. n}]
       | <num>
```

Increasing Generality

- PEGs operate on streams of characters
- OMeta operates on streams of *objects*
  - `anything` matches any one object
  - strings, e.g., `'hello'`
  - symbols, e.g,. `#ans`
  - numbers, e.g., `42`
  - "listy" objects, e.g., `{'hello' #ans 42 {}}`

Example: evaluating parse trees

```
eval ::= {#plus <eval>:x <eval>:y}  => [x + y]
       | <anything>:n ?[n isNumber] => [n]
```

{#plus. {#plus. 1. 2}. 3} → 6

```
dig  ::= ($0 | ... | $9):d    => [d digitValue]

num  ::= <num>:n <dig>:d       => [n * 10 + d]
       | <dig>

expr ::= <expr>:e $+ <num>:n => [{#plus. e. n}]
       | <num>
```

OMeta is Object-Oriented

```
dig  ::= ($0 | ... | $9):d   => [d digitValue]

num  ::= <num>:n <dig>:d     => [n * 10 + d]
       | <dig>

expr ::= <expr>:e $+ <num>:n => [{#plus. e. n}]
       | <num>
```

OMeta is Object-Oriented

MyLang

```
dig  ::= ($0 | ... | $9):d   => [d digitValue]

num  ::= <num>:n <dig>:d     => [n * 10 + d]
       | <dig>

expr ::= <expr>:e $+ <num>:n => [{#plus. e. n}]
       | <num>
```

OMeta is Object-Oriented

OMeta

```
anything ::= ...

...
```

MyLang

```
dig  ::= ($0 | ... | $9):d   => [d digitValue]

num  ::= <num>:n <dig>:d     => [n * 10 + d]
       | <dig>

expr ::= <expr>:e $+ <num>:n => [{#plus. e. n}]
       | <num>
```

OMeta is Object-Oriented

OMeta

```
anything ::= ...

...
```

MyLang

```
dig  ::= ($0 | ... | $9):d    => [d digitValue]

num  ::= <num>:n <dig>:d      => [n * 10 + d]
       | <dig>

expr ::= <expr>:e $+ <num>:n => [{#plus. e. n}]
       | <num>
```

MyLang++

```
expr ::= <expr>:e $- <num>:n => [{#minus. e. n}]
       | <super #expr>
```

Parameterized rules

```
digit ::= $0 | $1 | $2 | $3 | $4
        | $5 | $6 | $7 | $8 | $9
```

Parameterized rules

```
digit ::= $0 | $1 | $2 | $3 | $4
        | $5 | $6 | $7 | $8 | $9
```

```
range :a :b ::= <anything>:x ?[x >= a]
                             ?[x <= b] => [x]
```

Parameterized rules

```
digit ::= $0 | $1 | $2 | $3 | $4
        | $5 | $6 | $7 | $8 | $9
```

```
range :a :b ::= <anything>:x ?[x >= a]
                             ?[x <= b] => [x]

digit ::= <range $0 $9>
```

Higher-order rules

```
formals ::= <name> ($, <name>)*
args    ::= <expr> ($, <expr>)*
```

Higher-order rules

```
formals ::= <name> ($, <name>)*
args    ::= <expr> ($, <expr>)*
```

```
listOf :p ::= <apply p> ($, <apply p>)*
```

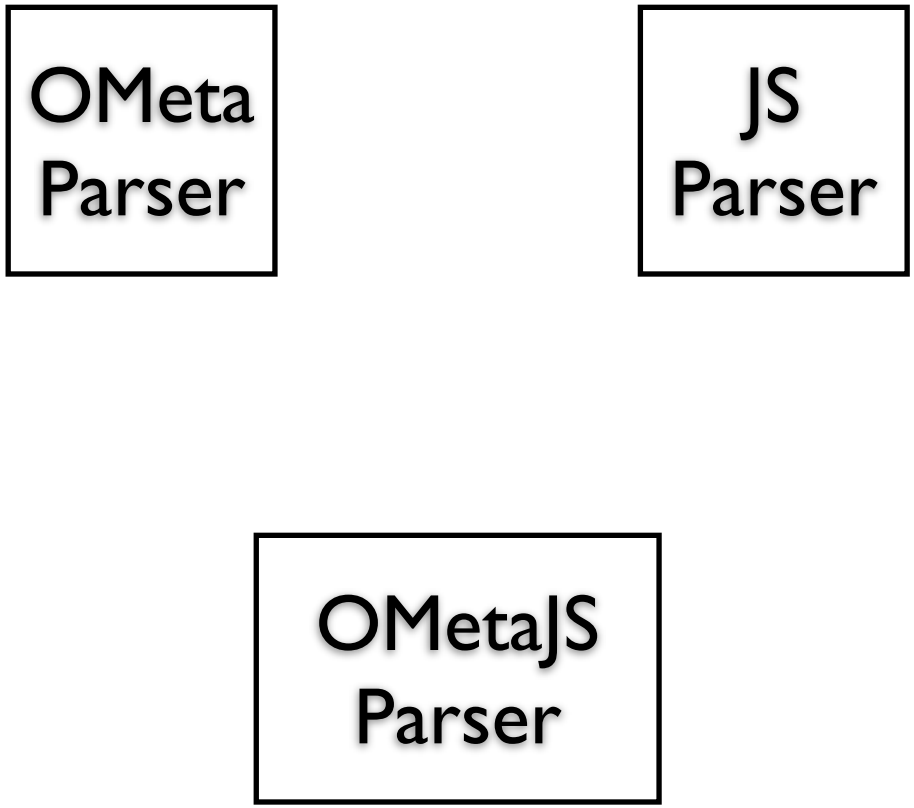
Higher-order rules

```
formals ::= <name> ($, <name>)*
args    ::= <expr> ($, <expr>)*
```

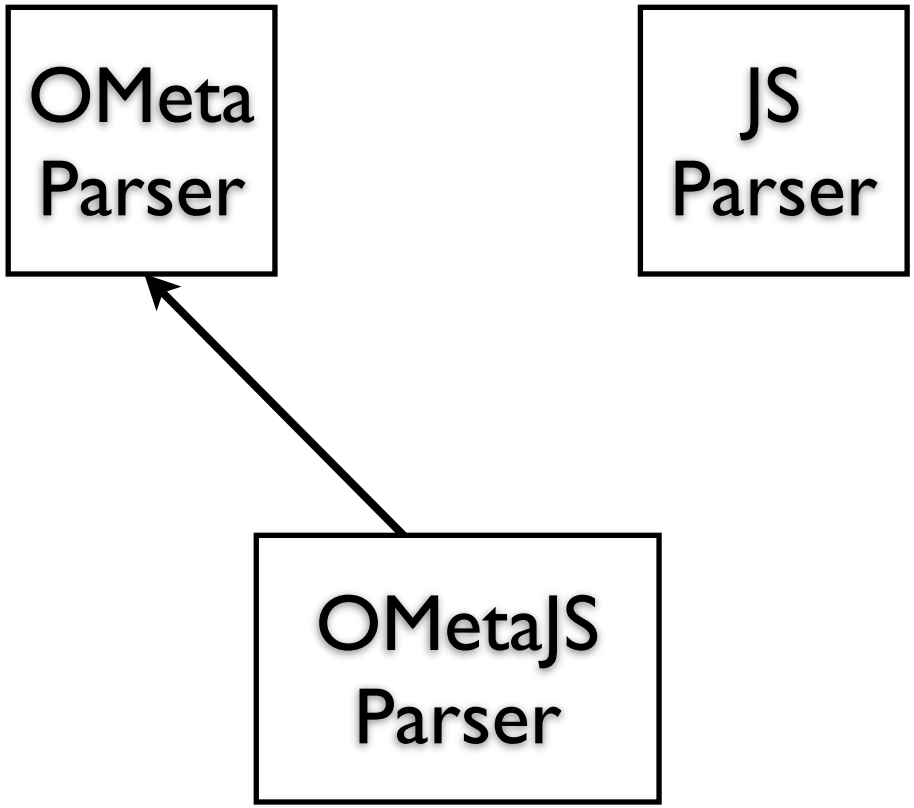
```
listOf :p ::= <apply p> ($, <apply p>)*

formals ::= <listOf #name>
args    ::= <listOf #expr>
```

OMetaJS = OMeta + JavaScript

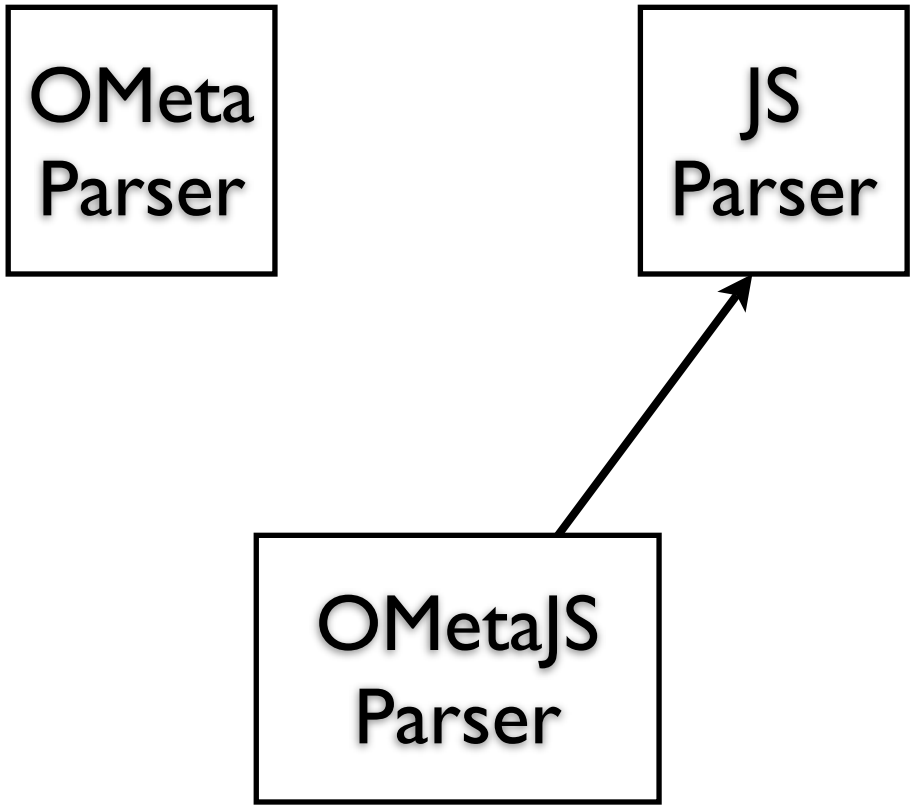
OMetaJS = OMeta + JavaScript



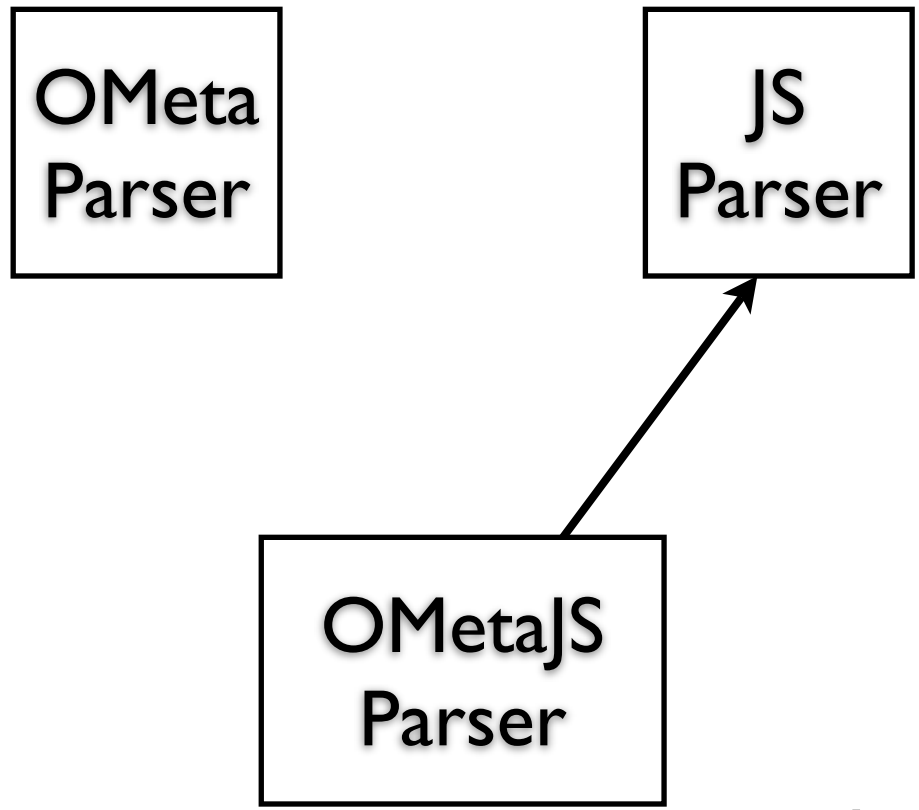
OMetaJS = OMeta + JavaScript



OMetaJS = OMeta + JavaScript



OMetaJS = OMeta + JavaScript



- duplicated effort
- versioning problem

Foreign rule invocation

- Lend input stream to another grammar

```
ometa OMetaJSParser {
  ometajs ::= <foreign OMetaParser #grammar>
            | <foreign JSParser    #stmt>
}
```

- Compose multiple grammars w/o worrying about name clashes

This OMeta, That OMeta

- Several versions of OMeta:
  - OMeta/Squeak
  - OMeta/COLA
  - OMeta/JS
  - ...
- Slightly different syntaxes
- Use different languages for semantic actions and predicates

OMeta/JS

JavaScript Workspace

- Takashi Yamamiya's handy work
- Inspiration for OMeta/JS
- Workspace-style interface for JavaScript
- Runs inside the web browser,
  works like a Wiki

JavaScript

- Dynamic language
- First-class functions (closures)
- Late-bound
- `eval()`
- Huge performance improvements lately
  - new webkit runs at ".5 Squeaks"

Plus…

IT'S EVERYWHERE!

ASSEMBLY LANGUAGE

Unearthing the excellence in JavaScript

JavaScript: The Good Parts

O'REILLY® | YAHOO! PRESS

Douglas Crockford

Switch to web browser


Wii
Wii


Forget Guitar Hero...
I could be the next Dan Ingalls!
OMeta/JS

For more info...

- **DLS'07 Paper**
- **OMeta Mailing List**
  http://vpri.org/mailman/listinfo/ometa
- **OMeta/JS wiki**
  http://jarrett.cs.ucla.edu/ometa-js
- **Ask questions now**

THE END

Selected Related Work

- Parsing Expression Grammars [Ford, '04]
- LISP70 Pattern Matcher [Tesler et al., '73]
- Parser combinator libraries [Hutton, '92], [Bracha'07]
- "Modular Syntax" [Grimm, '06]