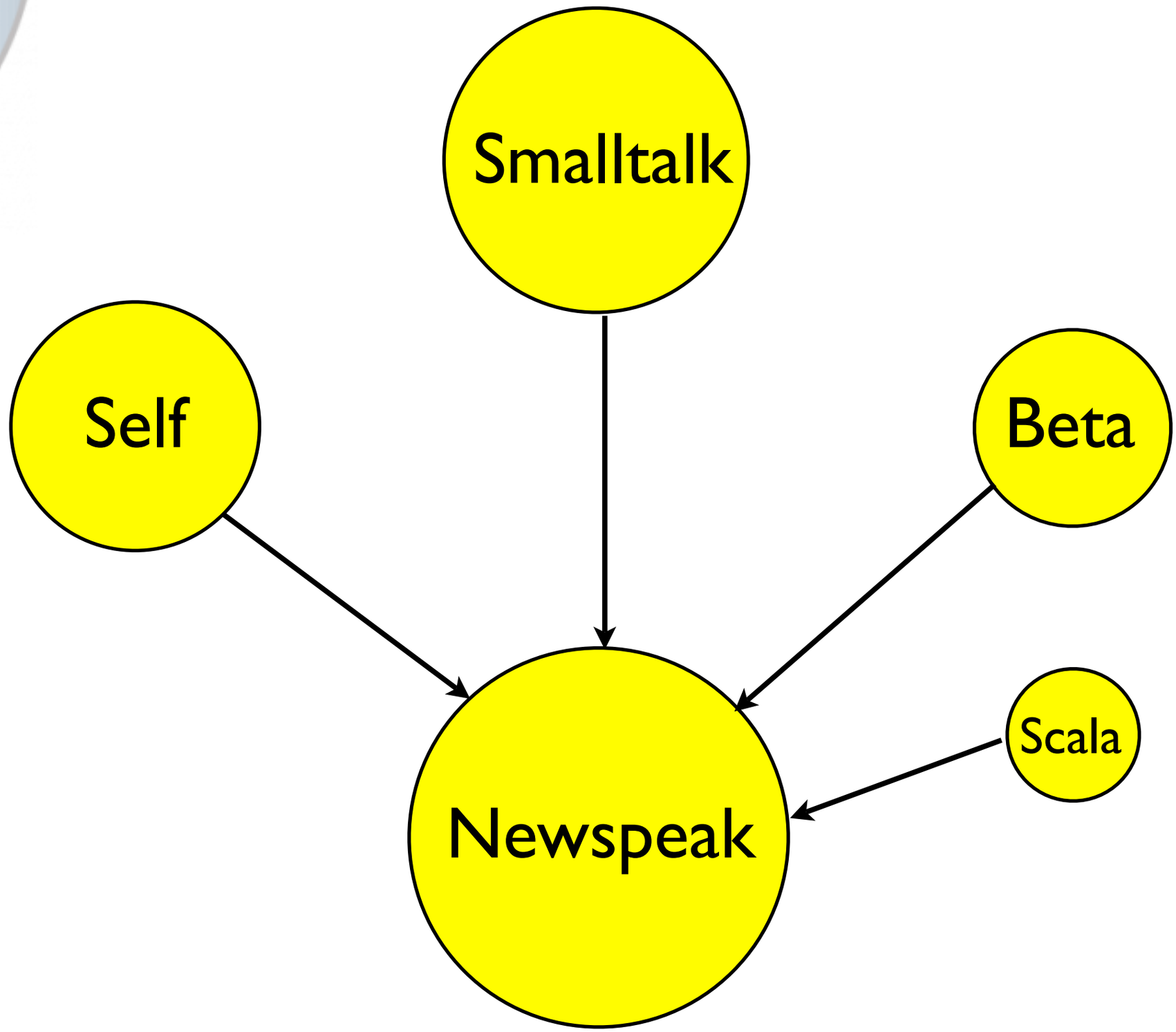
Newspeak:
Evolving Smalltalk for the Age of the Net

Gilad Bracha
Distinguished Engineer
Cadence Design Systems

Genealogy of Newspeak



Goals and Principles

- Goals
  - Modularity
  - Security
  - Interoperability
- Design Principle: Message based Programming

Message-based Programming

Smalltalk should have been message-oriented
- Alan Kay

Message-based Programming

Every run time operation is a message send.

In Smalltalk

```
t := Array new: n.
^t
```

In Smalltalk

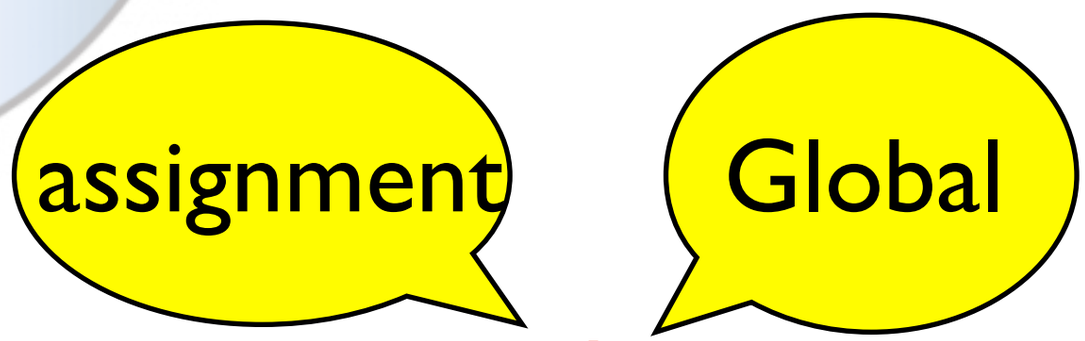
```
t := Array new: n.
^t
```

In Smalltalk


```
t := Array new: n.

^t
```

In Smalltalk




```
t := Array new: n.

^t
```

In Smalltalk

assignment

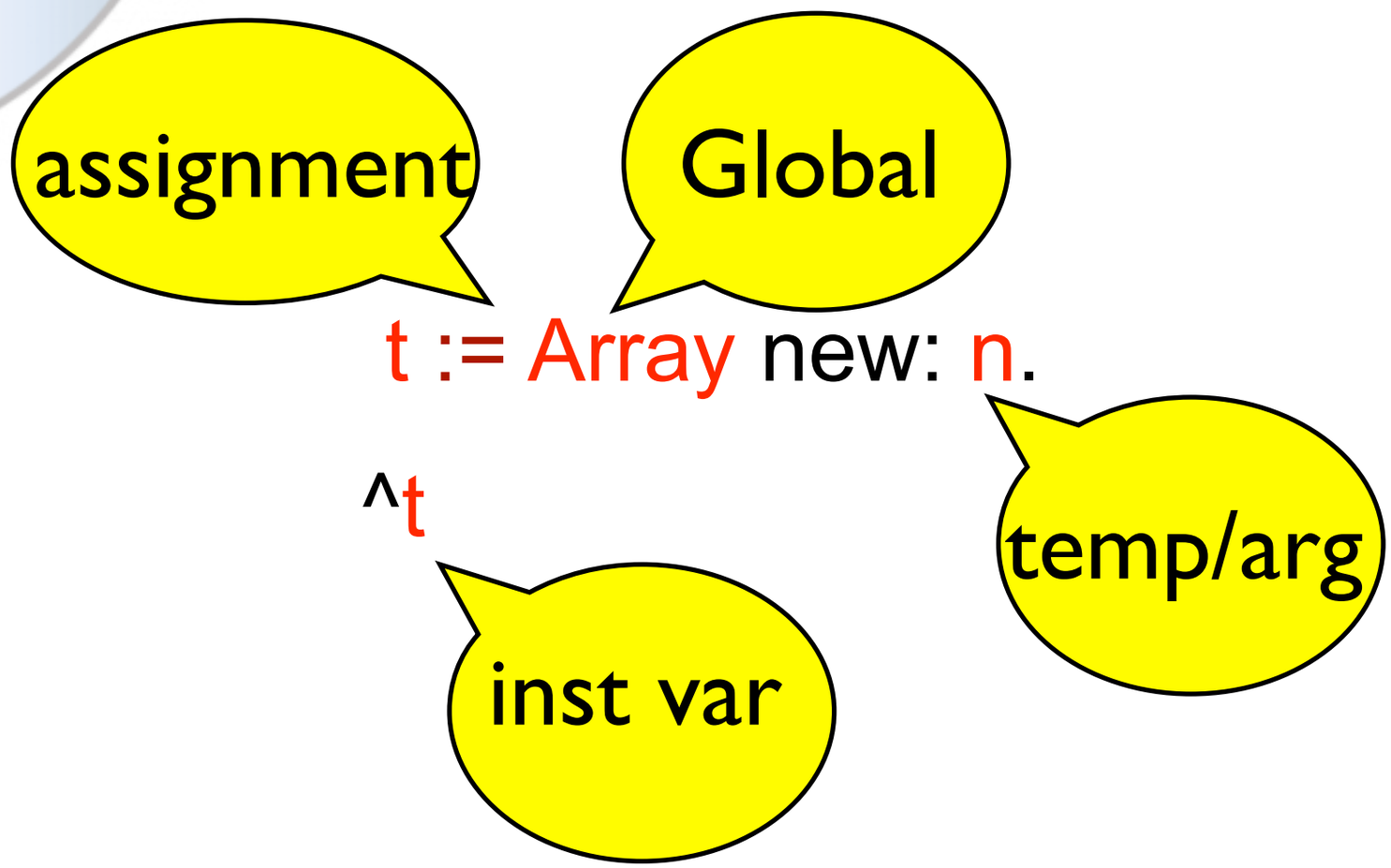
Global

```
t := Array new: n.

^t
```

temp/arg

In Smalltalk



In Smalltalk, using messages

```
self t: (self Array new: (self n)).

^self t
```

In Smalltalk, using messages

```
self t: (self Array new: (self n)).
^self t
```

Implicit receivers

```
t: (Array new: n).
^t
```

In Newspeak

```
t:: Array new: n.

^t
```

In Smalltalk, using messages

```
self t: (self Array new: (self n)).

^self t
```

In Smalltalk

```
t := Array new: n.

^t
```

Representation Independence

- No code depends on our choice of storage representation
- Clients never depended on it in Smalltalk
- Now subclasses don't depend on it either
- Even the class itself doesn't

In Smalltalk


```
t := Array new: n.

^t
```

No Static

No globals, no class variables, no pool variables of any kind; no class instance variables

No Static

- Good for
- Distribution
- Re-entrancy
- Testing
- Startup
- Memory management
- Security

Goals

- Modularity
- Security
- Interoperability

Objects as Capabilities

- Object-capability model (Miller06)
  - Object reachability defines authority
  - No static state
    - No Ambient Authority

Objects as Capabilities

- Object-capability model (Miller06)
  - Object reachability defines authority
  - No static state
    - No Ambient Authority
- Access control

Access Control

- In Smalltalk, the only means of data hiding are instance variables & blocks
- Messages are always public
- If everything is a message send, everything is public
- Public, protected and private messages

Mirror based Reflection

- Mirrors act as capabilities for reflection
- Availability of reflection can be controlled
  - For security
  - For deployment

No Static State

Where does the state go?

No Static State

How do different objects conveniently share state?

No Static State

How do different objects conveniently share state?

Via shared lexical scope

Nested Classes

- Nested as in Beta, not as in Java
  - Great for Modeling
  - Natural Modularity Solution

Goals

- Modularity
- Security
- Interoperability

No References to Classes

- Always use accessors
- Classes are first class objects
- Classes are always virtual
- Classes are always mixins
- Class hierarchy inheritance

External Dependencies are Explicit

- Module definition = Class not nested within another class
- No access to surrounding namespace
- All names locally declared or inherited

Modules are Sandboxes

- Factory method parameters are objects/ capabilities that determine per-module sandbox

Side by Side

- Module definitions are instantiated into stateful objects known as ***modules***
- Easy to create multiple instances, with different parameters

Modules are Re-entrant

- Module definitions are deeply immutable
- Modules cannot step on each other's state

Multiple Implementations

- Modules are objects, accessed via their protocol
- Different implementations can co-exist

Deployment

- Module definitions are objects
- Instantiate module in deeply immutable object literal's main: method
- Deployment amounts to object serialization
- Start up app by deserializing and invoking main:

Goals

- Modularity
- Security
- Interoperability

Smalltalk has Primitive Features

```
primFree: address

    <primitive: 'primFree'

    error: errorCode

    module: 'IA32ABI'>

    ^self primitiveFailed
```

Alberto Salguero

Smalltalk has Primitive Features

```
primFree: address

    <primitive: 'primFree'

    error: errorCode

    module: 'IA32ABI'>

    ^self primitiveFailed
```

Smalltalk has Primitive Features

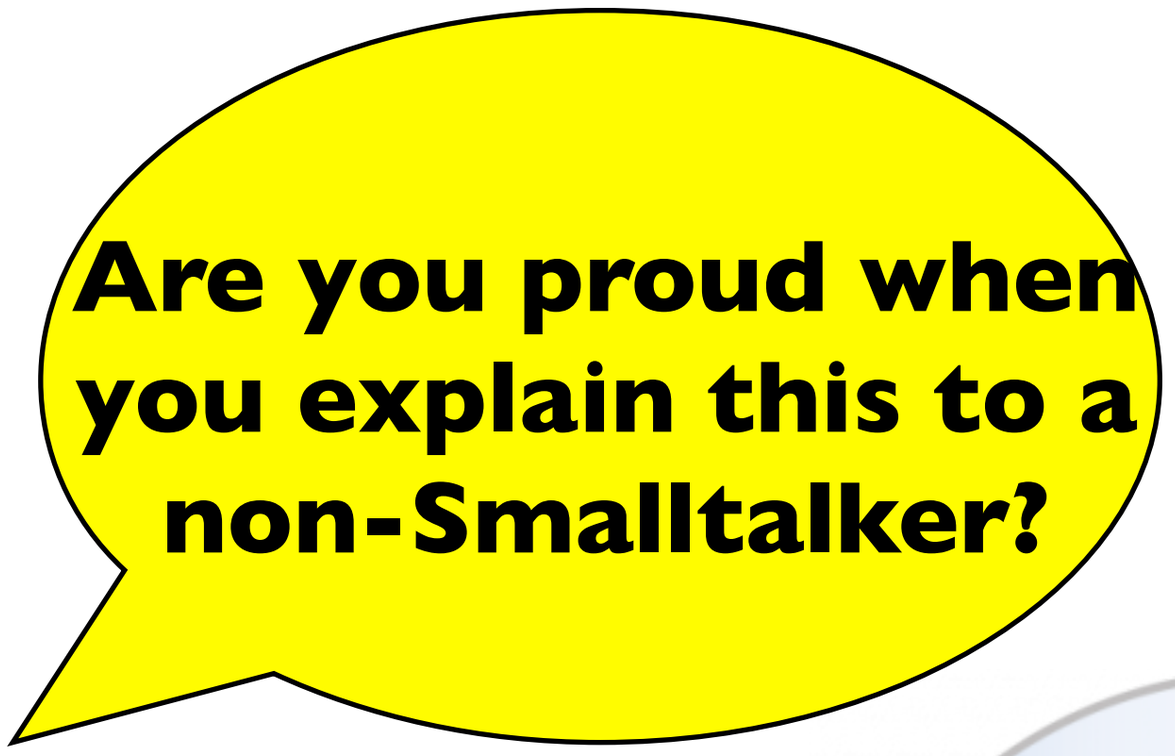
primFree: address

<primitive: 'primFree'

error: errorCode

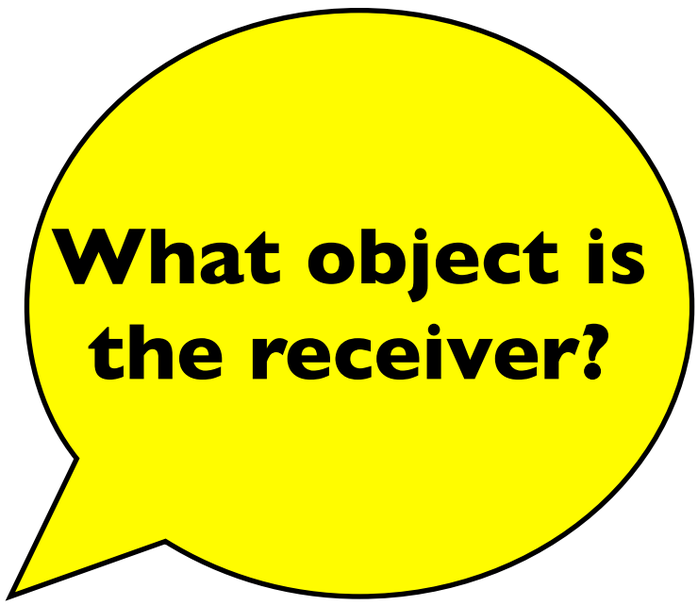
module: 'IA32ABI'>

^self primitiveFailed



Smalltalk has Primitive Features

```
primFree: address
    <primitive: 'primFree'
    error: errorCode
    module: 'IA32ABI'>
        ^self primitiveFailed
```



Nothing Primitive about Newspeak

- There is no "primitive" construct in Newspeak - contradicts message-based philosophy
- Instead, you send a message to the VM, as reified via a VM mirror
- VM mirror behavior should be standardized

Primitives & Foreign calls

- In many languages, primitives are just foreign calls
- These notions are distinct
  - Primitives do not need to worry about marshaling
  - Assumption that VM is in another language is inappropriate

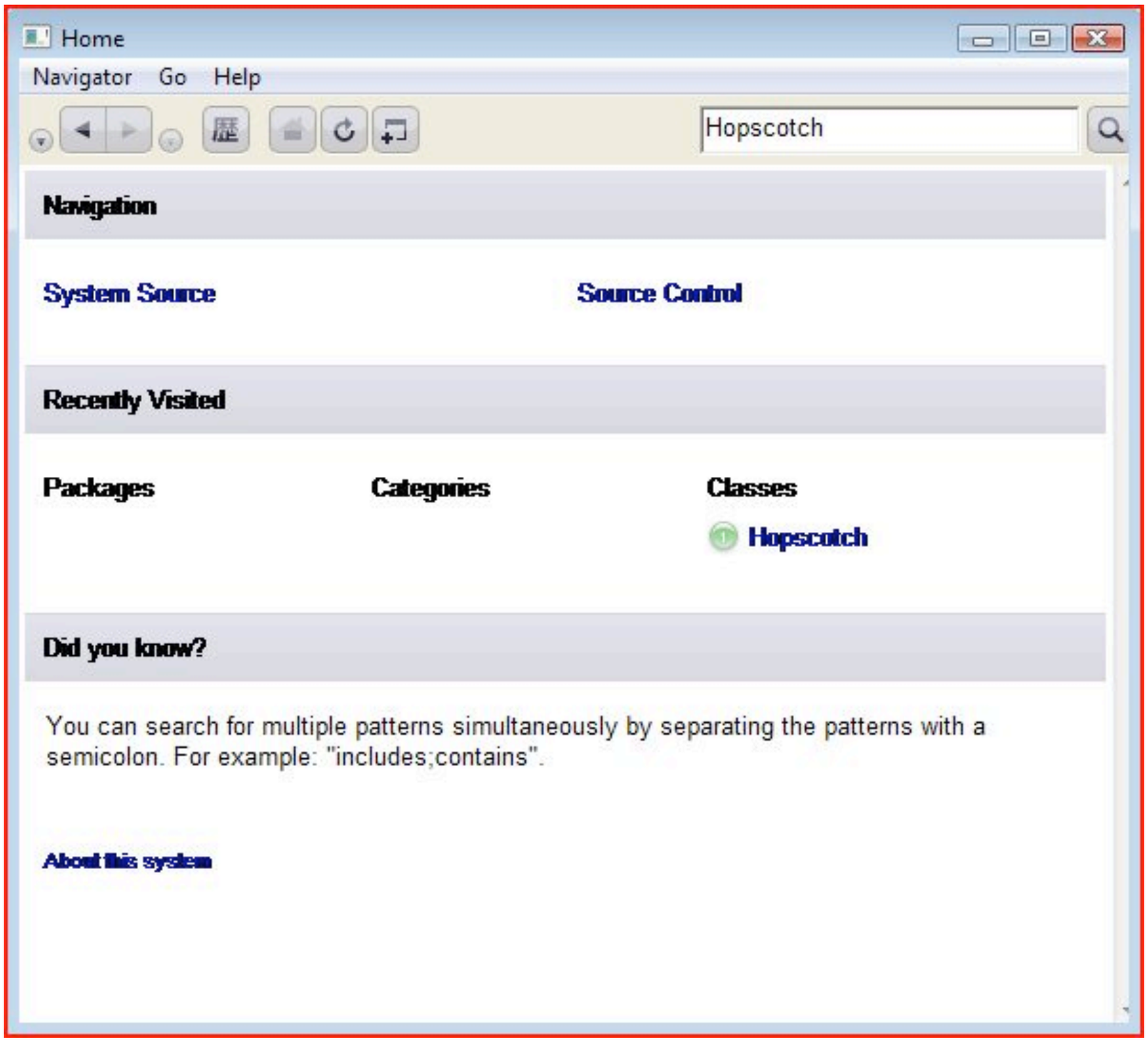
Primitives & Foreign calls

- In most Smalltalks, foreign calls are variations of primitives
- They therefore inherit
  - Ugliness
  - Lack of portability
- Often an afterthought

Aliens

- In Newspeak, calling out is achieved by sending messages to ***alien*** objects
- You can send blocks as arguments to implement call backs
- Different alien libraries can support different languages

Portable Native GUI



The Age of the Net

Cloud Computing

The Age of the Net

Cloud Computing

The Age of the Net

- Cloud Computing
  - Software delivery and maintenance over the net
  - Javascript is the assembly language of the internet, browser is the OS

The Age of the Net

Back to 1970s timesharing or 1990 X-terminals?

Web Apps have Downsides

- System software has to be local
- UI issues
- Depend on Reliable, Fast, Cheap Network
- Make you "reboot" - it's called: *session expired*

Network Serviced Applications

- Combine advantages of web services and traditional client applications
  - Always Available (even w/o network)
  - Always Up to date
- Run locally, think globally
- Restart-free

Network Software Service

- Maintains software and data on server
- Provides backup, audit trail, software distribution & maintenance
- Software distributed and updated when client syncs with server
- Client can run offline using locally cached software and data
- Sync does not imply application restart

Platform Support for Network Serviced Applications

- Hotswapping: Running applications can be updated on-the-fly over the network
- Modules: well defined units of deployment and update
- Security: Object capability model
- Orthogonal Synchronization: Natural update of program and data*

Status

- Work in Progress
  - Expect some tweaks to syntax and semantics
  - Implementation still incomplete - especially libraries
- Working toward public release
  - to be open sourced under Apache 2.0 license

Future Work

- Libraries
- System issues: Linux & Mac Native GUI, FFI refinements
- Concurrency: value types, actors
- Orthogonal synchronization
- Object literals, Metadata, Access control, Pluggable Types

Synergy

- Message-based programming
- Component style modularity
- Virtual classes, mixins, class hierarchy inheritance
- Object capability model and security

Synergy

- Mirror based reflection
- Actor style concurrency
- Pluggable types

Credits

- Peter Ahe
- Vassili Bykov
- Yaron Kashai
- Eliot Miranda (emeritus)