Combined Object-Lambda Architectures

John Quigley
www.jquigley.com
jquigley#jquigley.com

Chicago Lisp
April 2008

Research Goals

Question: How to make COLAs with Pepsi and Coke?

The Goal: A new way to construct programming languges and software systems.

Priorities emphasize simplicity, openness, evolution, user-centered.

Users should be able to understand and modify anything; system organization encourages such tinkering.

Research Goals

Question: How to make COLAs with Pepsi and Coke?

The Goal: A new way to construct programming languges and software systems.

Priorities emphasize simplicity, openness, evolution, user-centered.

Users should be able to understand and modify anything; system organization encourages such tinkering.

Research Goals

Question: How to make COLAs with Pepsi and Coke?

The Goal: A new way to construct programming languges and software systems.

Priorities emphasize simplicity, openness, evolution, user-centered.

Users should be able to understand and modify anything; system organization encourages such tinkering.

Research Goals

Question: How to make COLAs with Pepsi and Coke?

The Goal: A new way to construct programming languges and software systems.

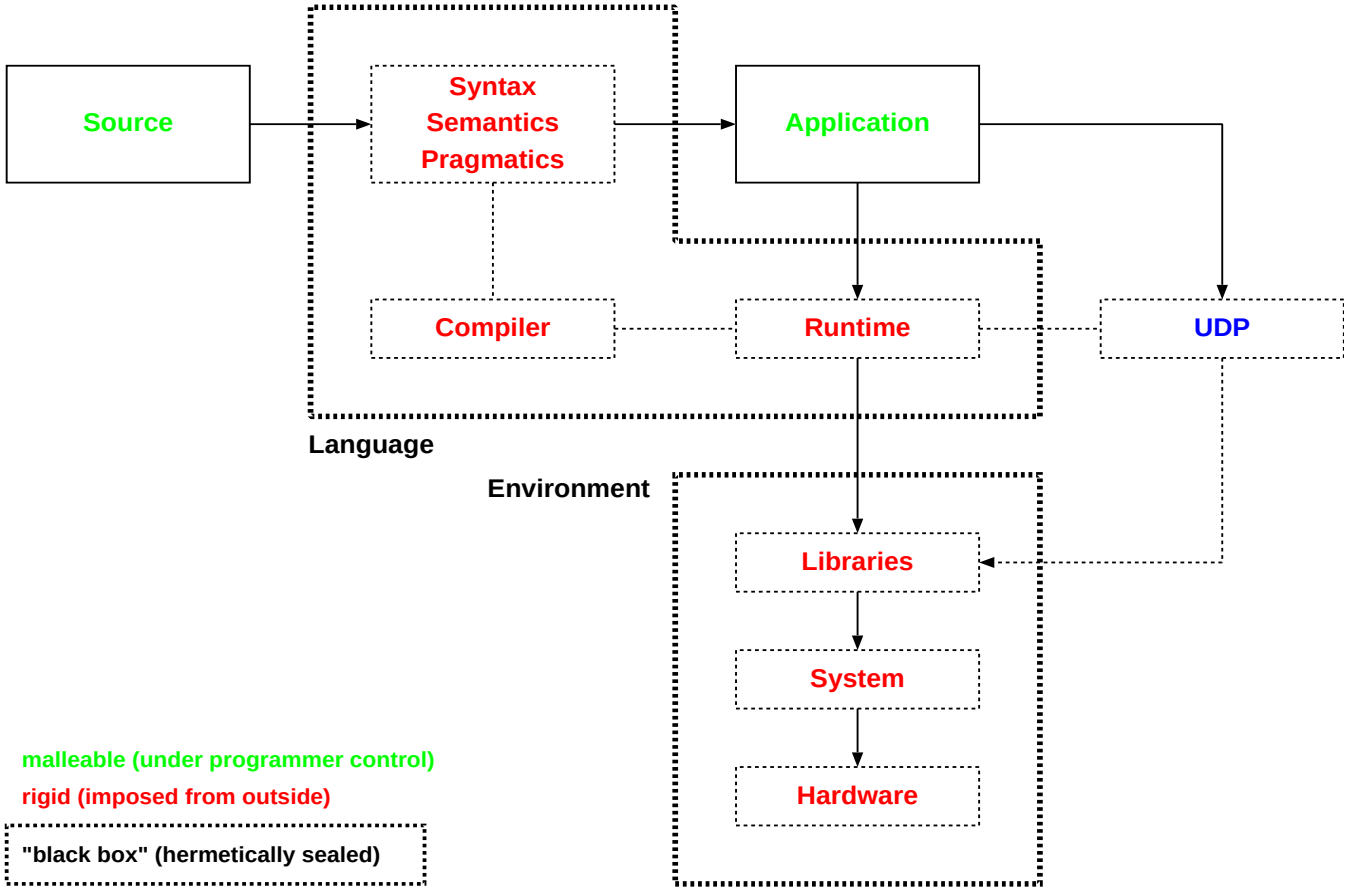
Priorities emphasize simplicity, openness, evolution, user-centered.

Users should be able to understand and modify anything; system organization encourages such tinkering.

System Goals

1. **Simplicity**: simple to understand and modify; pervasive self-similarity, system entirely describes itself within homogenous object-oriented paradigm.
2. **Openness**: all stages visible, accessible to, and modifiable by, user.
3. **Evolutionary Programming**: supports fluid design, implementation, maintenance phases.
4. **User-centered**: system strives to serve the programmer.

Conventional Systems



Conventional Systems

Programmer dealing with two hermetically-sealed black boxes: the language and the environment.

Language is a combination of:

1. **Syntax**: restricts legal content of source code
2. **Semantics**: predefined meaning of syntactic content
3. **Pragmatics**: range of externally-visible effects

All are rigid (designed by committee) and inaccessible to programmer.

Conventional Systems

Programmer dealing with two hermetically-sealed black boxes: the language and the environment.

Language is a combination of:

1. **Syntax**: restricts legal content of source code
2. **Semantics**: predefined meaning of syntactic content
3. **Pragmatics**: range of externally-visible effects

All are rigid (designed by committee) and inaccessible to programmer.

Conventional Systems

Environment (libraries, OS) accessible only through runtime facilities.

Runtime equally rigid and inaccessible, designed by same committee responsible for language.

Accessing nonstandard facilities generally:

1. Impossible
2. Inefficient
3. Profoundly disruptive to creative process

Conventional Systems

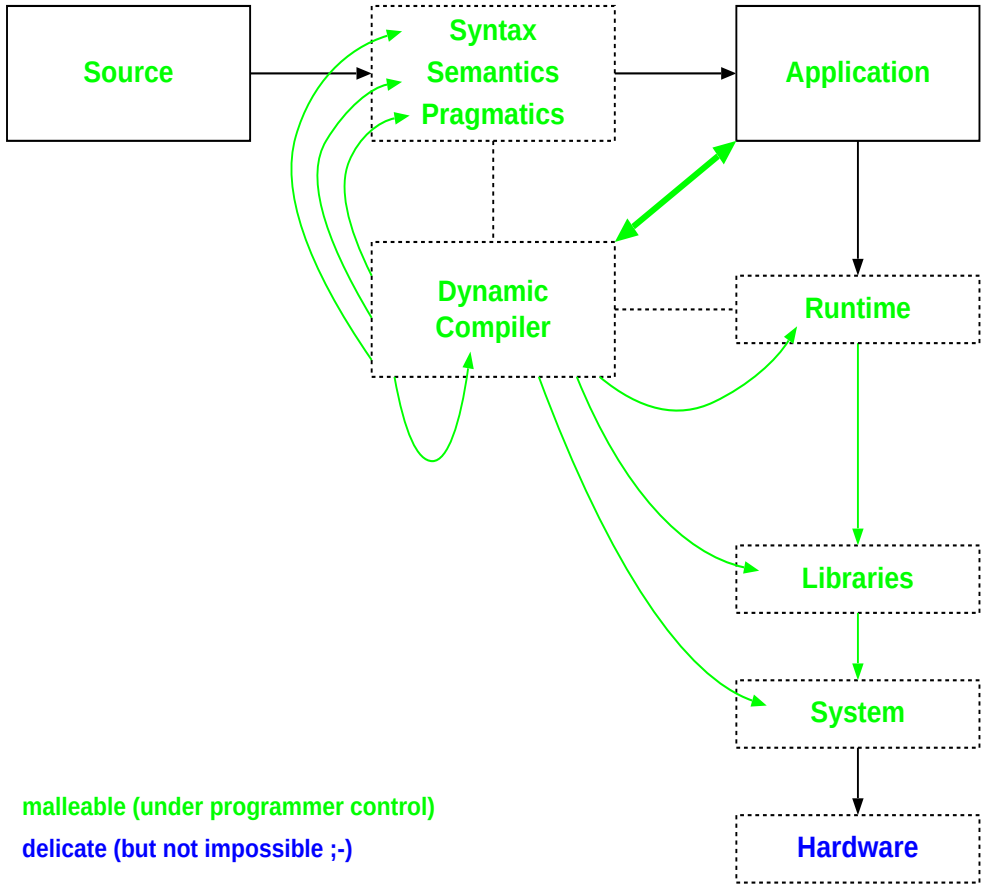
Access to non-standard facilities usually in the guise of a FFI or UDP.

FFIs are expensive!

UDPs demand specialized knowledge from programmer (shift in abstraction levels and representation).

Modification of language itself demands knowledge even more specialized.

Unconventional Systems



Unconventional Systems

A COLA gives users control over all aspects of system implementation and execution.

This environment must provide:

1. **Unified representation**: from source to executable transformation, runtime implementation and application code
2. **Pervasive dynamism**: nothing is static, nor any aspect early-bound or rigidly defined
3. **First-class everything**: system implementation and runtime are first-class components of running application

Unconventional Systems

A COLA gives users control over all aspects of system implementation and execution.

This environment must provide:

1. **Unified representation**: from source to executable transformation, runtime implementation and application code
2. **Pervasive dynamism**: nothing is static, nor any aspect early-bound or rigidly defined
3. **First-class everything**: system implementation and runtime are first-class components of running application

Unconventional Systems

An entire end-user application is just an extension of its own implementation mechanism!

Programming environment is made homogenous: no artificial distinctions between language and implementation.

No a priori fixed points of reference, everything is flexible.

Unconventional Systems

How is any of this even possible?

Let's take the deep dive!

Unconventional Systems

How is any of this even possible?
Let's take the deep dive!

COLAs

Question: What exactly is a COLA?

It's a pair of mutually-sustaining abstractions:

One provides representation and the other behavioral meaning.

Minimal requirement: simplest structures and abstractions that can produce fully self-describing system.

COLAs

Question: What exactly is a COLA?

It's a pair of mutually-sustaining abstractions:

One provides representation and the other behavioral meaning.

Minimal requirement: simplest structures and abstractions that can produce fully self-describing system.

COLAs

Question: What exactly is a COLA?

It's a pair of mutually-sustaining abstractions:

One provides representation and the other behavioral meaning.

Minimal requirement: simplest structures and abstractions that can produce fully self-describing system.

COLAs

Question: What exactly is a COLA?

It's a pair of mutually-sustaining abstractions:

One provides representation and the other behavioral meaning.

Minimal requirement: simplest structures and abstractions that can produce fully self-describing system.

COLAs

Representation provided by prototype-like objects exchanging messages, organized into clone families.

"Messaging" is defined (recursively) as sending messages to objects.

One provides representation and the other behavioral meaning.

Minimal requirement: simplest structures and abstractions that can produce fully self-describing system.

COLAs

Representation provided by prototype-like objects exchanging messages, organized into clone families.

"Messaging" is defined (recursively) as sending messages to objects.

One provides representation and the other behavioral meaning.

Minimal requirement: simplest structures and abstractions that can produce fully self-describing system.

COLAs

Meaning imposed on representation by transforms that convert structures into executable forms.

These structures are analogous to symbolic expressions in the lambda calculus.

Semantics of "structures" defined (recursively) by representing transforms as structures.

These transforms structures are indistinguishable from the structures they operate upon!

COLAs

Meaning imposed on representation by transforms that convert structures into executable forms.

These structures are analogous to symbolic expressions in the lambda calculus.

Semantics of "structures" defined (recursively) by representing transforms as structures.

These transforms structures are indistinguishable from the structures they operate upon!

COLAs

One such executable form provides the implementation of methods installed in objects of representation.

What all this means is that the overall implementation is very much a circular one.

Put another way, the implementation language and abstractions of the system are precisely those that the system implements.

Lispniks will find this a familiar and provacative idea ... think metacircular!

COLAs

One such executable form provides the implementation of methods installed in objects of representation.

What all this means is that the overall implementation is very much a circular one.

Put another way, the implementation language and abstractions of the system are precisely those that the system implements.

Lispniks will find this a familiar and provacative idea ... think metacircular!

COLAs

One such executable form provides the implementation of methods installed in objects of representation.

What all this means is that the overall implementation is very much a circular one.

Put another way, the implementation language and abstractions of the system are precisely those that the system implements.

Lispniks will find this a familiar and provacative idea ... think metacircular!

COLAs

And now a momentary break to explain nomenclature.

The representation layer provides language similar to desirable end-user language.

It's not, however, an ideal (pervasively late-bound) implementation of that language.

It is code-named 'Pepsi.'

COLAs

And now a momentary break to explain nomenclature.

The representation layer provides language similar to desirable end-user language.

It's not, however, an ideal (pervasively late-bound) implementation of that language.

It is code-named 'Pepsi.'

COLAs

The meaning layer provides everything required for pervasively late-bound implementation of Pepsi.

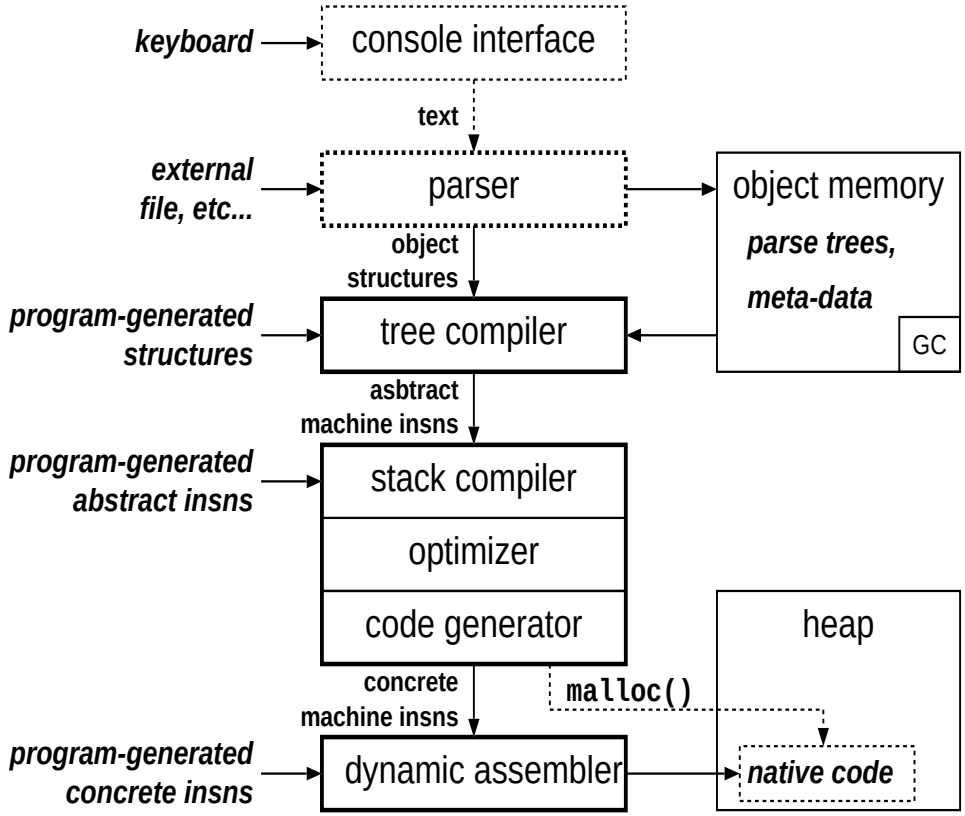
Since this is *the real thing*, it is code-named 'Coke.'

High-level abstractions

A COLA consists of a heirarchy of stages.

1. **Front end**: acquires text from input device
2. **Parser**: converts text into structured form (abstract syntax tree)
3. **Tree compiler**: walks ASTs, applying transformations
4. **Virtual processor**: translates abstract instructions into native instructions
5. **Dynamic assembler**: converts native instructions into binary for execution

High-level abstractions



Communication Algebra

Let's discuss communication between objects.

Dynamic binding is not a primitive operation.

During binding a message is sent to a (real) object to perform (define semantics of) method lookup.

Additional mechanisms (e.g., delegation, inheritance) implemented by overriding default methods that implement dynamic binding.

Communication Algebra

Let's discuss communication between objects.

Dynamic binding is not a primitive operation.

During binding a message is sent to a (real) object to perform (define semantics of) method lookup.

Additional mechanisms (e.g., delegation, inheritance) implemented by overriding default methods that implement dynamic binding.

Communication Algebra

All manipulation of objects accomplished by message passing.

All runtime structures (selectors, vtables) are real objects.

Contents of an objects are defined functionally, by methods that access its state.

Behavioral Algebra

Now on to a theory of meaning (behavior) to describe internal implementation of methods.

Structures are build from objects, and formed into a forest of ASTs.

Each successive AST in the forest is evaluated, by compiling and executing resulting form.

The meaning of each AST node is given by the dynamic binding of a compilation closure.

Behavioral Algebra

Now on to a theory of meaning (behavior) to describe internal implementation of methods.

Structures are build from objects, and formed into a forest of ASTs.

Each successive AST in the forest is evaluated, by compiling and executing resulting form.

The meaning of each AST node is given by the dynamic binding of a compilation closure.

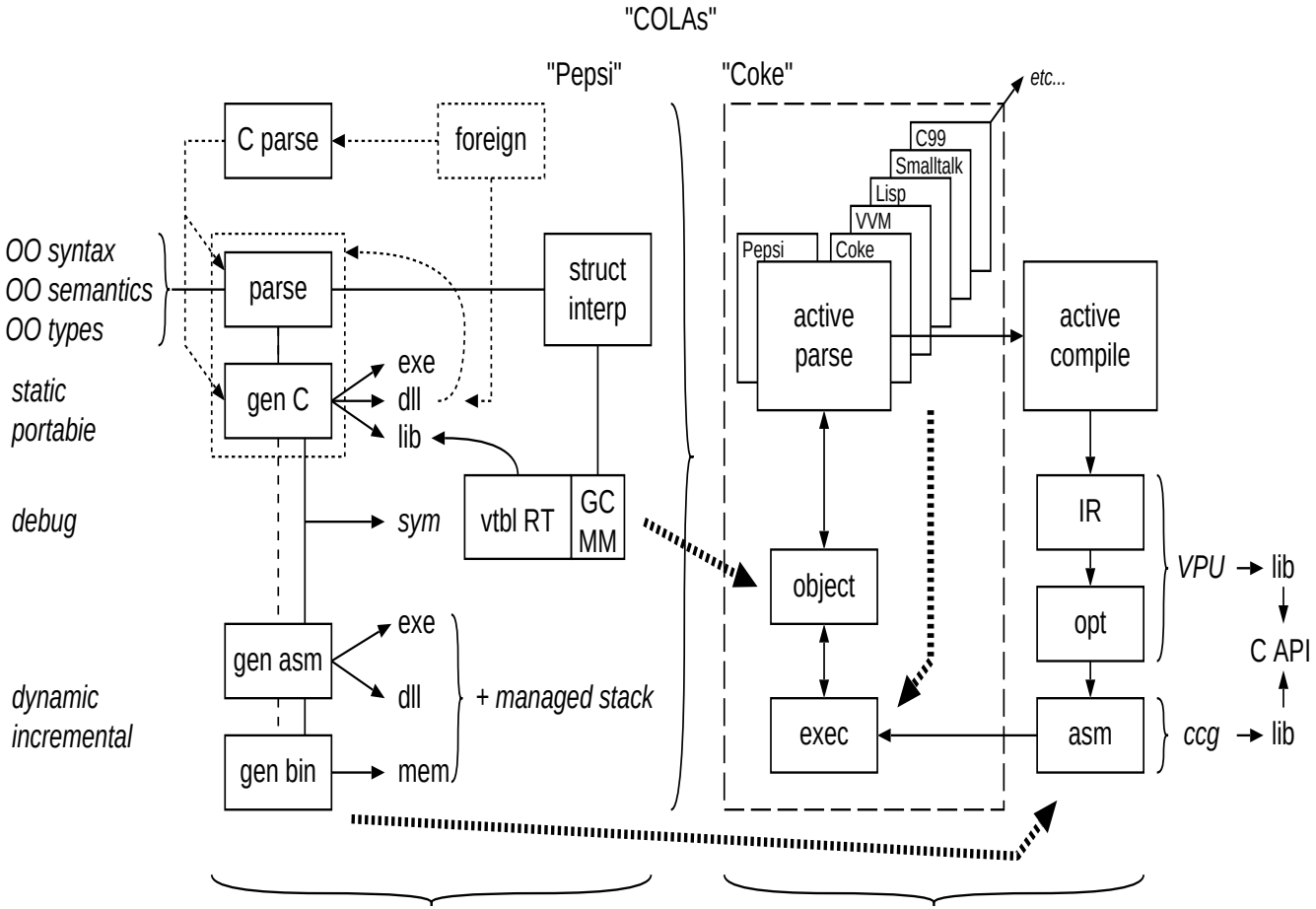
Behavioral Algebra

A syntactic closure produces a rewritten AST (similar to Lisp macros).

A semantic closure produces full or partial implementations of their AST.

Either can have arbitrary side effects, and neither need produce a runtime effect.

Behavioral Algebra



Object Model

Question: What makes a COLA object model special?

Intrinsic object model is simplest possible that can support messaging.

Method lookup (the operation that defines messaging semantics) not defined primitively.

All structures involved in implementation of messaging are full objects that respond to messages.

Object Model

Question: What makes a COLA object model special?

Intrinsic object model is simplest possible that can support messaging.

Method lookup (the operation that defines messaging semantics) not defined primitively.

All structures involved in implementation of messaging are full objects that respond to messages.

Object Model

To preserve generality, flexibility, orthogonality, object model is too simple to be of practical use on its own

It is transformed into a usable model (supporting reuse, composition, etc.) by extending it in terms of itself.

Object Model

To preserve generality, flexibility, orthogonality, object model is too simple to be of practical use on its own

It is transformed into a usable model (supporting reuse, composition, etc.) by extending it in terms of itself.

Messaging Model

Intrinsic dymanic (late-bound) behavior associated with object through virtual table.

The association of an object with its virtual table is unspecified (can be explicit or implicit).

Sending message to object consists of finding an implementation (at message send time) with receiver's vtable.

That implementation corresponds to selector of message being sent.

Messaging Model

Intrinsic model provides objects and messaging, but no way to add behavioral composition or reuse.

Consider instead the object just described as a binding object.

Its primary responsibility is implementing a "lookup" method.

Extending the model gives an object whose dynamic behavior (response to message send) is implemented by user-accessible object.

Messaging Model

To complete the model:

Consider initial vtable (providing behavior for binding objects) is made real object by associating it with vtable.

Simplest solution is for it to be its own vtable.

Messaging Model

To complete the model:

Consider initial vtable (providing behavior for binding objects) is made real object by associating it with vtable.

Simplest solution is for it to be its own vtable.

Behavioral Model

Question: How do we give COLA objects meaning?

Objects are composed into syntactic structures representing meaning (behavior).

These structures translated into an executable form by successive applications of transforms.

These transforms give semantic meaning to syntactic structures.

Behavioral Model

Question: How do we give COLA objects meaning?

Objects are composed into syntactic structures representing meaning (behavior).

These structures translated into an executable form by successive applications of transforms.

These transforms give semantic meaning to syntactic structures.

Behavioral Model

Since a transform is just behaviour itself, semantic structures are syntactic structures.

They're behavior is simply applied to syntactic structures.

There is no "meta" level!

Tree Compiler

Objects are formed into structures representing symbolic syntactic expressions.

Their meaning (semantics, behavior, implementation) is described by symbolic semantic expressions.

Syntactic expressions are transformed ('evaluated') according to semantic expressions by tree compiler.

Tree Compiler

Tree compiler places no intrinsic semantic meaning on structures it compiles.

Evaluation eventually yields:

1. executable representation in memory (or file)
2. side-effects modifying evaluation context itself

What's Left

So many things left to talk about!

1. **OMeta**: object-oriented PEG (Parsing Expressing Grammars) for describing syntax
2. **Bootstrapping**: how three object types and five methods can bootstrap entire model
3. **FONC**: The overarching goals of this entire ambitious effort

What's Left

So many things left to talk about!

1. **OMeta**: object-oriented PEG (Parsing Expressing Grammars) for describing syntax
2. **Bootstrapping**: how three object types and five methods can bootstrap entire model
3. **FONC**: The overarching goals of this entire ambitious effort

Constraints

Issue: Model depends on using C stack and has design goal of complete compatability with native calling method.

Limits how you might implement concurrency, coroutines or threads.

Limits tail call optimization to whatever the C compiler might provide.

Constraints

Issue: Model depends on using C stack and has design goal of complete compatability with native calling method.

Limits how you might implement concurrency, coroutines or threads.

Limits tail call optimization to whatever the C compiler might provide.

Constraints

Issue: Object model depends on method caching scheme that uses hash based on message selector and vtable of first argument only.

Restricts what algorithms that could be used to implement multiple argument dispatch (since hash will not depend on other arguments).

Constraints

Issue: Object model depends on method caching scheme that uses hash based on message selector and vtable of first argument only.

Restricts what algorithms that could be used to implement multiple argument dispatch (since hash will not depend on other arguments).

More Information

Material for this presentation taken from
"Making COLAs with Pepsi and Coke"
Written by Ian Piumarta

http://piumarta.com/software/cola/colas-whitepaper.pdf

More Information

Other sources of interest.

1. Viewpoints Research Institute: www.vpri.org
2. Fundamentals of New Computing: http://www.vpri.org/html/work/ifnct.htm
3. COLA project page: http://piumarta.com/software/cola/

The End

That's all I've got for now.
Can I take any questions?