Dynamic Scope and Context-oriented Programming

Christian Neukirchen
Editor in Chief of Anarchaia

Euruko 2005

Overview

- Dynamic scope
- Context-oriented Programming
- Implementing ContextR
- “Surprise”

Chapter 1
Dynamic Scope

Review: Lexical Scope

```
def adder(n)
  v = 0                # lexical scope
  lambda { v += n } # closure
end

add_one = adder 1
p add_one.call         # => 1
p add_one.call         # => 2
p add_one.call         # => 3
```

Comparision

Lexical Scope:

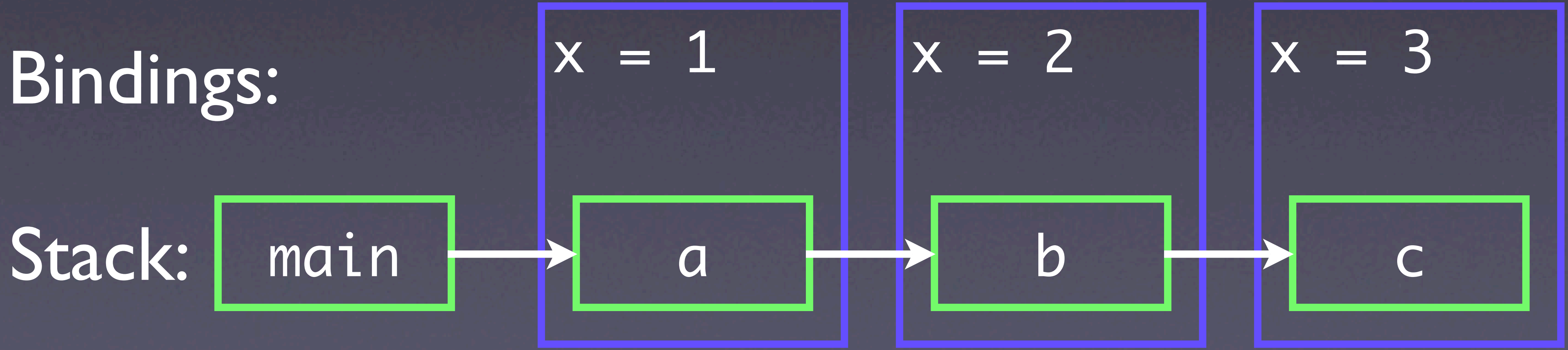
Variables are looked up in the binding they **were defined**.

Dynamic Scope:

Variables are looked up **dynamically**, in the **current** binding (not the defining one).

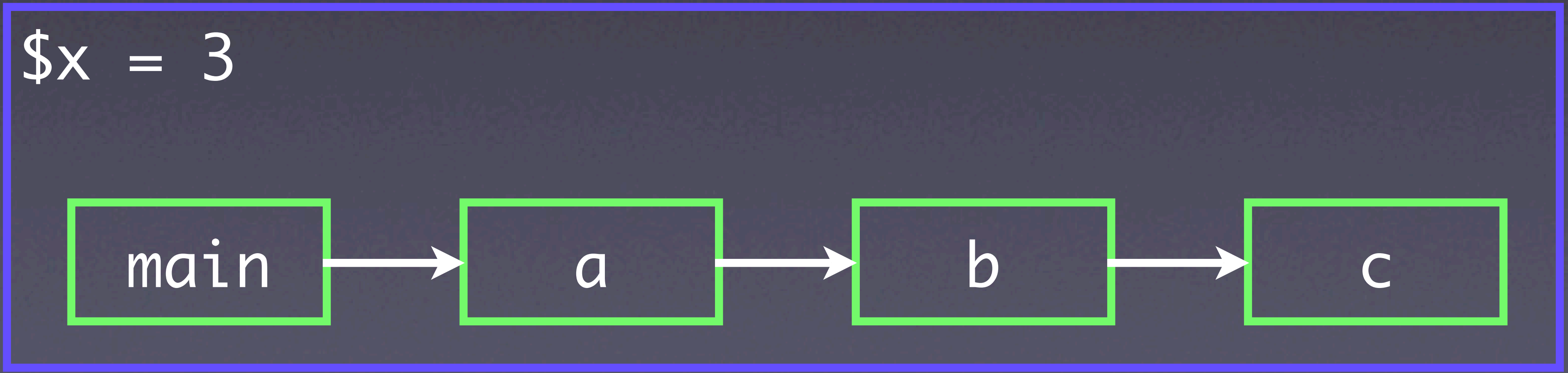
Lexical lookup

```
def a; x = 1; b; print x; end
def b; x = 2; c; print x; end
def c; x = 3;    print x; end
                       # 3 2 1
```



Global lookup

```
def a; $x = 1; b; print $x; end
def b; $x = 2; c; print $x; end
def c; $x = 3;    print $x; end
                           # 3 3 3
```



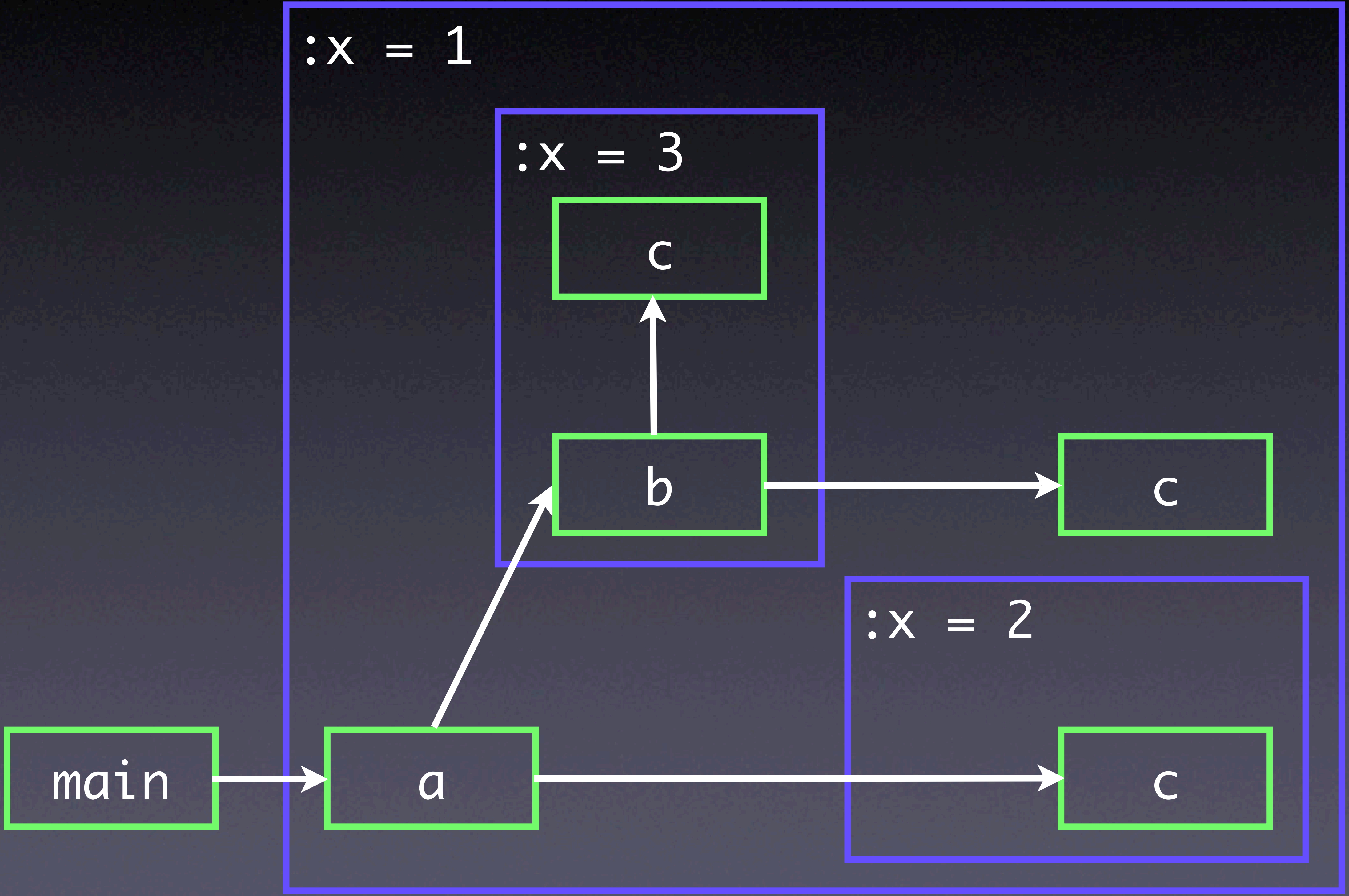
Dynamic lookup

```
Dynamic.variable :x
def a
  Dynamic.let :x => 1 do
    b
    Dynamic.let :x => 2 do  c  end   # 3 1 2
  end
end

def b
  Dynamic.let :x => 3 do  c  end
  c
end

def c
  print Dynamic[:x]
end
```

Scope diagram



Using Dynamic Scope

- Parametrization:
  - STDIN, STDOUT (think ERb + puts)
- Passing objects around without explicit mention
  - Dissident, my DI container, stores the currently active containers in a dynamic variable

Using dynamic.rb

```
require 'dynamic'

Dynamic.variable :eur2usd_factor => 1.3068

def eur2usd(euro)
  euro * Dynamic.eur2usd_factor
end

p eur2usd(10)                # => 13.068
p eur2usd(0.77)              # => 1.006236

Dynamic.let :eur2usd_factor => 0.9267 do
  p eur2usd(10)              # => 9.267
  p eur2usd(0.77)            # => 0.713559
end

p eur2usd(10)                # => 13.068
```

Implementation

- Dynamic variables are stored globally accessible.
- Dynamic.let is roughly:

```
old = Dynamic[variable]
Dynamic[variable] = new
  yield
Dynamic[variable] = old
```

- Using the Ruby stack to keep track of previous definitions

Implementation

- Dynamic variables, not "real" dynamic scope.
- Dynamic scope is easy to implement in C
    - Local variable infrastructure can be reused.

‣ Would make a good addition to future Ruby versions. (Now, fight about a sigil!)

Chapter II
Context-oriented Programming

The idea

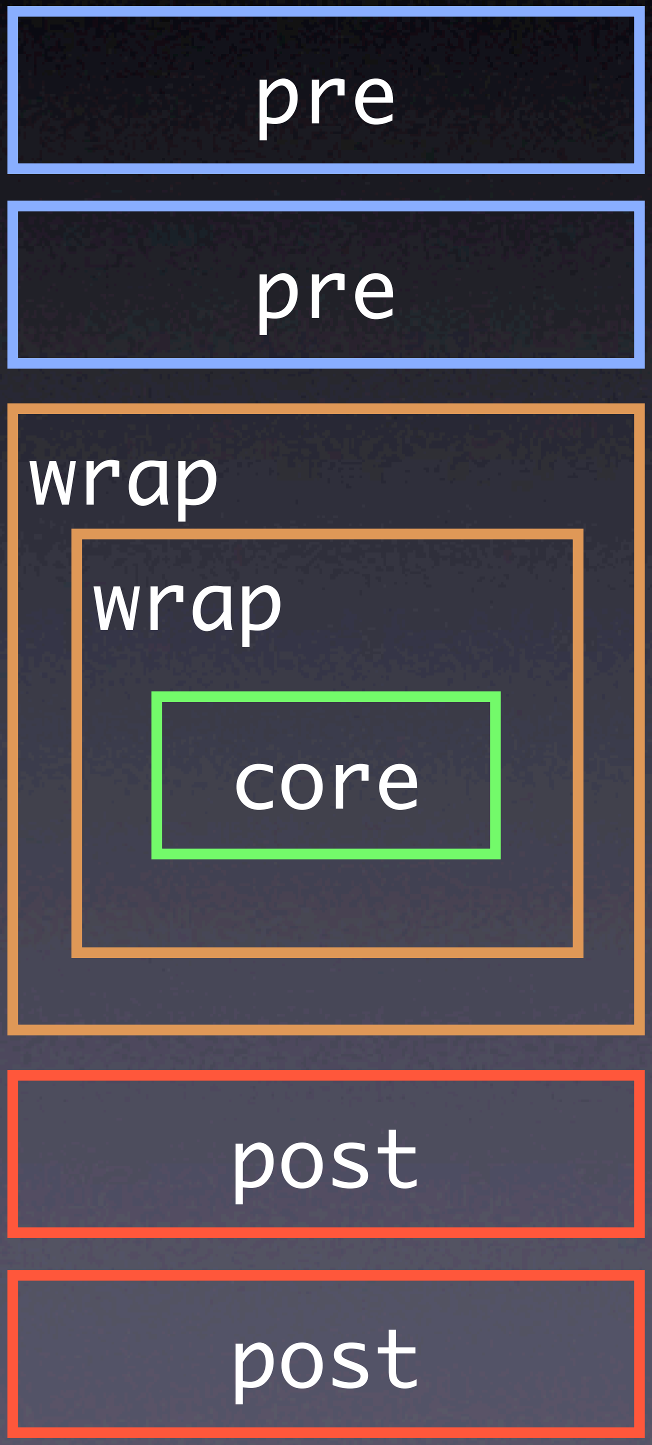
- Imagine you can’t only dynamically scope variables, but also **methods**.

Layering Methods

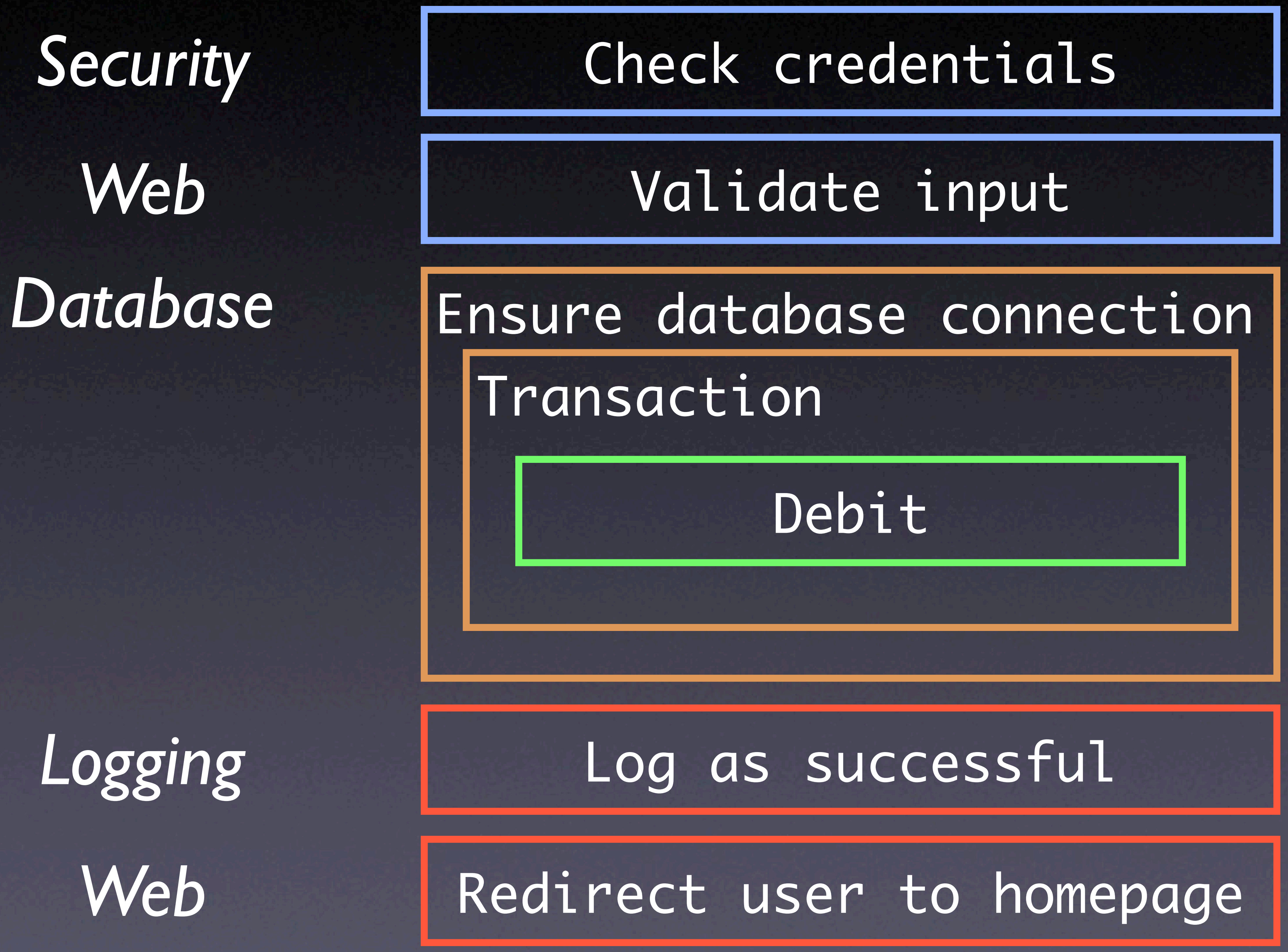
- Methods sometimes need to fulfill several concerns:
  - Logging
  - Data validation
  - Database handling (connecting, transactions)
  - …

Defining layered methods

- Methods have several “slices”:
  - a “core”
  - hooks to run before
  - hooks to run after
  - hooks to wrap the core
- Comparable to AOP
- More are imaginable, but not implemented yet



Website Example



In ContextR:

```
class Website
  layer :security
  layer :web
  layer :database
  layer :logging

  def debit; ...; end
end
```

In ContextR:

```
class Website
  security.pre :debit do
    check_credentials
  end
  web.pre :debit do
    validate_input
  end
end
```

In ContextR:

```
class Website
  database.wrap :debit do |n|
    connect_to_database
    n.call_next
  ensure
    close_database
  end
  database.wrap :debit do |n|
    transaction { n.call_next }
  end
end
```

In ContextR:

```
class Website
  logging.post :debit do |n|
    log "Debit successful: " <<
      n.return_value
  end
  web.post :debit do
    redirect_back_home
  end
end
```

Configuring the Application

```
# Development
ContextR.with_layers :web,
          :database, :logging do
  Website.new
end
```

Configuring the Application

```
# Production
ContextR.with_layers :web, :database,
                     :security do...

# Unit testing
ContextR.with_layers :mock_db do...
```

Comparision to AOP

- Some may know these ideas from “Aspect-oriented Programming”…
- ….but Context-oriented Programming is more:
  - The program can be reconfigured completely **at runtime.**

Reconfiguration for testing:

```
def test_logging
  ContextR.with_layers :logging do
    assert_logged ...
  end
end

ContextR.with_layers :mock_db do
  run_tests
end
```

More usages

- Layers also can be defined in Modules:
  - Mix-in and ducktyping allow for boundless extensibility
  - Generic User Interfaces (Naked Objects on steroids)
  - …

Chapter III
Implementing
ContextR

Implementation

- ContextR was written in about four hours this week.
- API inspired by ContextL, written by Pascal Constanza (see references).
- 281 LoC + 171 LoC for dynamic variables.
- Proof-of-Concept, but not ugly.

Implementing compound methods

- Compund methods are implemented using "salami tactics"
  - Each method gets split up into lots of smaller methods
  - A driver method figures which to call…
  - …and what to do with the results.

Rough translation of the Website example

```
def debit
  _debit_pre_00001_;_debit_pre_00002_
  _debit_wrap_00003_ {
    _debit_wrap_00004_ {
      r = _debit_core_00005_
    }
  }
  _debit_post_00006_;debit_post_00007_
  r
end
```

Implementation

- In reality, it does more:
  - Check for active layers
  - Keep track of arguments and return values
  - Allow for premature exits
- Fully dynamic, for now

Limitations

- Most severe limitation in Ruby <1.9
  - Blocks can’t take blocks as arguments
  - Blocks are used heavily in ContextR
  - ContextR can’t pass blocks to slices
  - No problem to do in Ruby >=1.9

Performance of ContextR

- In one word: **horrible**.
  - Method calls are up to 200x slower.
- You can stop laughing now.
  - Optimization is possible…

Ideas for optimizing ContextR

- "Compilation" of methods by generating a string that calls the method slices
- Caching generated methods by active contexts
- "Deoptimization"
    - Redefining all affected methods on context changes (heavily depends on the way ContextR is used).

Ideas for optimizing ContextR

- Hoping that YARV will be more efficient to enable above techniques in an useful way.
- “It’s just method calls.”

Chapter IV
“Surprise, surprise”

Using ContextR to implement…

Namespace Selectors

I live “behind the moon”, what are they?

- First introduced by Matz at RubyConf 2004
- To appear in Ruby 2.0
- Solving an “old” problem of Ruby
  - “How can I change Ruby’s core methods without breaking other code?”

ContextR Namespaces: Declaration

```
class Array
  namespace :foo do
    def mungle
      zip(reverse).flatten
    end
  end
end
```

ContextR Namespaces: Usage

```
class Foo
  namespace :foo

  def initialize
    p [1,2,3].mungle
  end
end
```

ContextR Namespaces: Trying...

```
Foo.new      # [1, 3, 2, 2, 3, 1]

[1,2,3].mungle  rescue p $!
# ~> #<NoMethodError: undefined
method `mungle' for 123:Array
(only in :namespace_foo)>
```

Implementation of ContextR Namespaces

- Each namespace gets a layer
- `namespace(symbol)` makes the `default` layer wrap all methods with appropriate `with_layers` calls
  - using `method_added`

Implementation of ContextR Namespaces

- `namespace(symbol, &block)` defines a layer on `method_added`, activates it, and `class_evals` the block to automatically claim all the methods defined in the block.
- This probably qualifies as hack. `:^)`

Summary:

- ~680 LoC written in about six hours total
- Possible to implement ~97% (estimated) of CLOS in pure Ruby
  - Lacking `const_defined`, e.g.
- Not a single use of `eval(string)`
- Loads of fun

```
Question::Time ===
Time.now
```

References

- “Language Constructs for Context-oriented Programming–An Overview of ContextL” by Pascal Costanza and Robert Hirschfeld

  http://p-cos.net/documents/contextl-overview.pdf

- “Dynamically Scoped Functions as the Essence of AOP” by Pascal Costanza.

  http://p-cos.net/documents/dynfun.pdf

- http://chneukirchen.org/blog/archive/2005/04/dynamic-variables-in-ruby.html

Thanks to…

- Mauricio Fernández for telling me I already was half-way done implementing namespaces and helping me polishing the slides.
- `#ruby-lang` on freenode for help in deepest metaprogramming dungeons.
- **You,** following this talk until the end.

On the web:

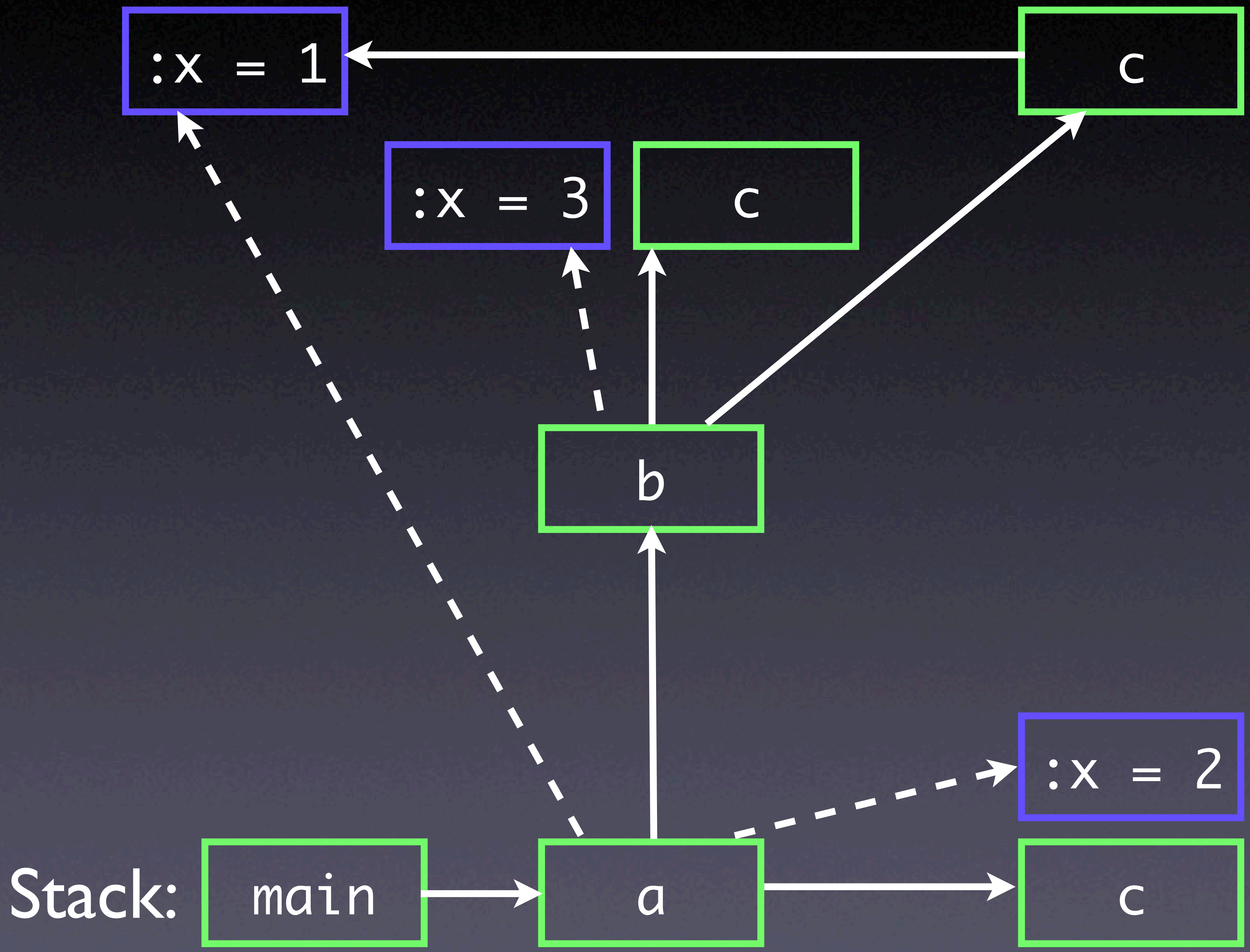
http://chneukirchen.org/talks/euruko-2005

Outtakes:

History of Dynamic Scope

- Used by default in old Lisps
  - Lisp 1.5
  - MacLisp
  - Emacs Lisp
- Still provided and used by modern Lisps
  - "special variables" (defvar)

Analysis


:x = 1
c
:x = 3
c
b
:x = 2
Stack:
main
a
c