# Dancing with Symmetry to Harness the Power of Complexity: Subjective Programming in Context

David Ungar
Harold Ossher
IBM Research



© 2013, 2014 IBM Corporation

# Korz

Replace the single implicit receiver with many implicit objects, each playing a different role.



Use multi-dimensional context to both organize and dispatch state and behavior.





© 2013, 2014 IBM Corporation

# Contents

- What's the big idea?
- What's it good for?
- How real is it?
- How does it work?
- Are you guys through yet?

# Contents

- What's the big idea?
- What's it good for?
- How real is it?
- How does it work?
- Are you guys through yet?



© 2013, 2014 IBM Corporation

# Context?!
# The Hidden Variable

- Tacit, background
- Affects how you program
- Affects the computation
- Enables reuse and incremental variation




Requester omits details, e.g. how.
Fetcher doesn't care what's next.
*Context* gives language power.



© 2013, 2014 IBM Corporation

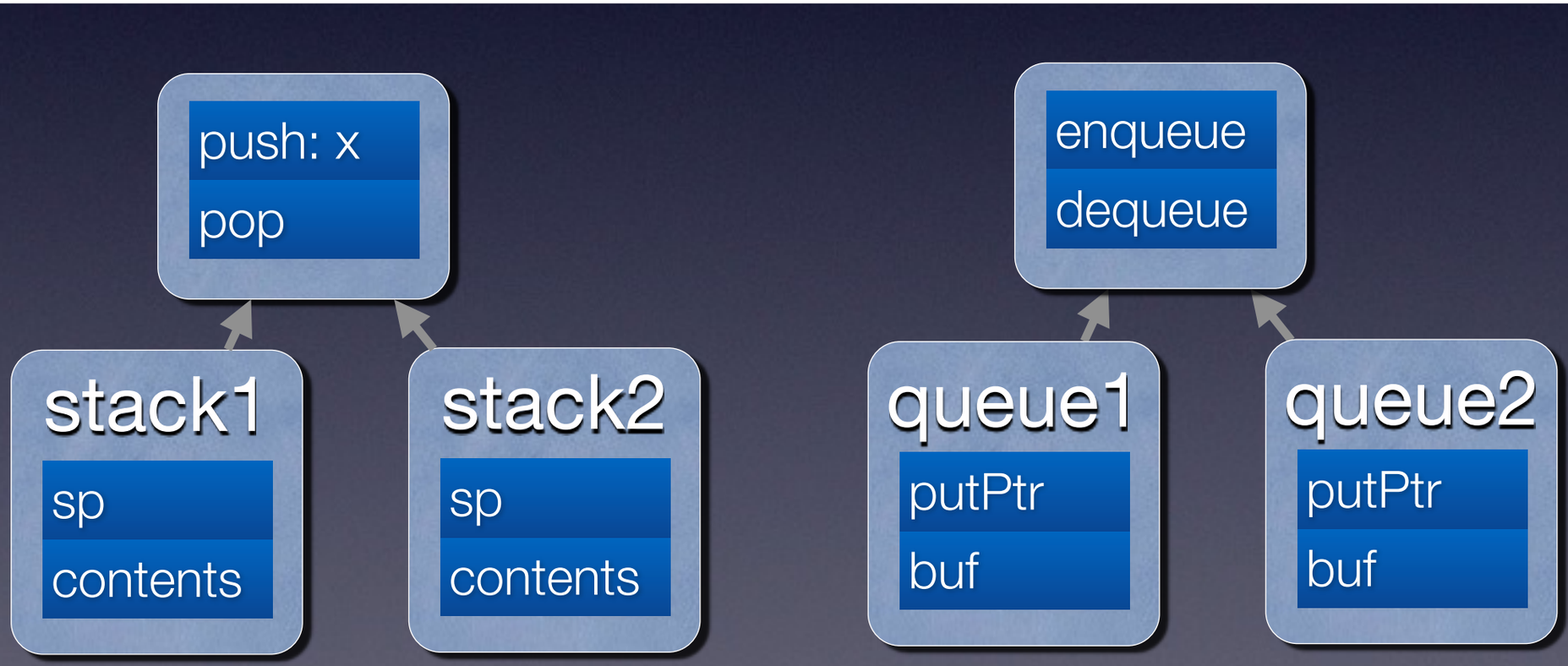
# Example: Cost of object-oriented programming

## Figuring out how to carve up the program

Before




After




## Why do objects help?



© 2013, 2014 IBM Corporation

# Benefits of Objects

Clients:

details of object fade into background

- how to do it
- where to store variables

Objects:

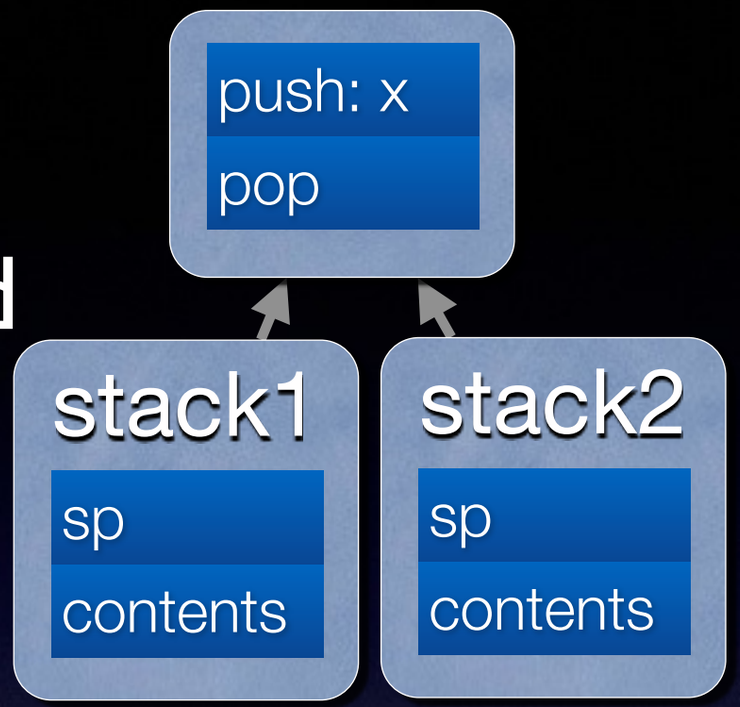
details of client fade into background

- order of sends
- exact requirements
- storage management

Subclasses:

details of superclass fade into background

Fade into background ≡ *context*






© 2013, 2014 IBM Corporation

# Objects add *context*

storage management recedes into background

‘self’ / ‘this’ recedes into background

```
this.pop(){ return this.contents[ --this.sp]; }
```

# aids reuse

Object becomes a unit of thought & reuse

# aids evolution

Can subclass/refine for incremental variation

Can interpose another object



* Smalltalk excepted

© 2013, 2014 IBM Corporation

# Object-oriented context: *one* object

Most messages in receiver's context

```
push(x) {
    expandIfNeeded();
    contents[sp] = x;
    sp = sp + 1;
}
```

A few not in receiver's context

# Korz context: *many* objects



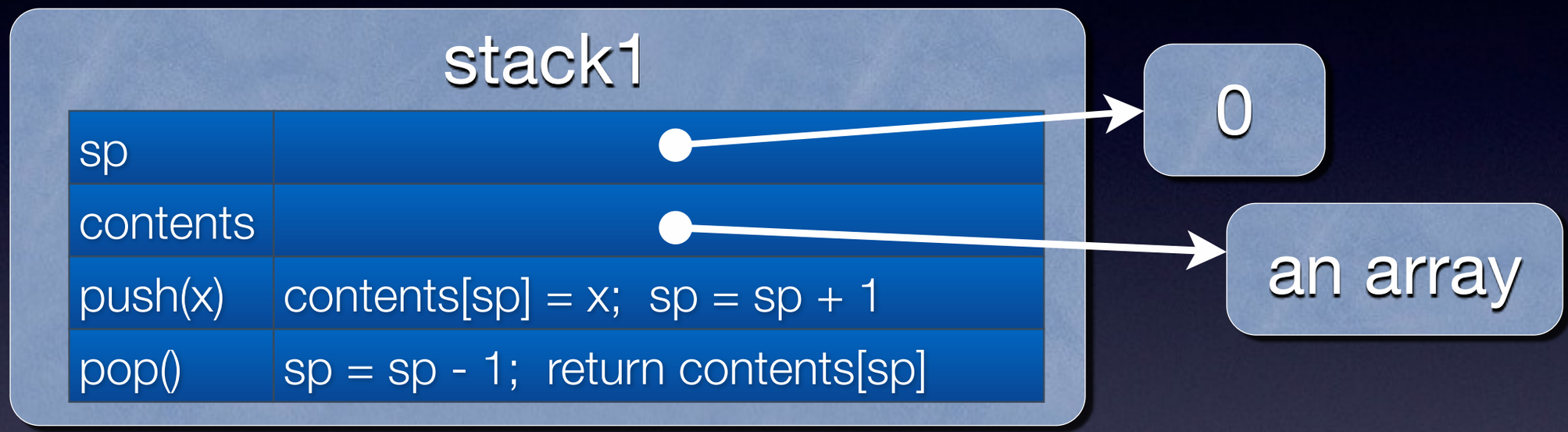
© 2013, 2014 IBM Corporation

# Preliminary details

- Prototype implementation is in Self
- Syntax is Self-like; will change later
  - JS-syntax used below for clarity
- Prototype-based, could be class-based
  - Can add individual slots
- Unified state and behavior
  - Parenthesis optional on call with no arguments
    - pop() ≡ pop
    - sp ≡ sp()
  - Find slot & run it
    - running data returns data


© 2013, 2014 IBM Corporation

# A Self-style stack




```
stack1.push(2)
stack1.push(3)
stack1.pop() + stack1.pop()
```



© 2013, 2014 IBM Corporation

# Self-style terms

References

Data Slots

Method Slots

stack1

| | |
|---|---|
| sp | |
| contents | |
| push(x) | contents[sp] = x;  sp = sp + 1 |
| pop() | sp = sp - 1;  return contents[sp] |

0

an array

Objects

```
stack1.push(2)
stack1.push(3)
stack1.pop() + stack1.pop()
```



© 2013, 2014 IBM Corporation

# Simplifying diagram

Data Slots

Method Slots

**stack1**

| | |
|---|---|
| sp | 0 |
| contents | an array |
| push(x) | contents[sp] = x;  sp = sp + 1 |
| pop() | sp = sp - 1;  return contents[sp] |

Objects

```
stack1.push(2)
stack1.push(3)
stack1.pop() + stack1.pop()
```



© 2013, 2014 IBM Corporation

Object

Data Slots

Method Slots

| stack1 | |
|---|---|
| sp | 0 |
| contents | an array |
| push(x) | contents[sp] = x;  sp = sp + 1 |
| pop() | sp = sp - 1;  return contents[sp] |

# Two stacks

Object

Data Slots

Method Slots

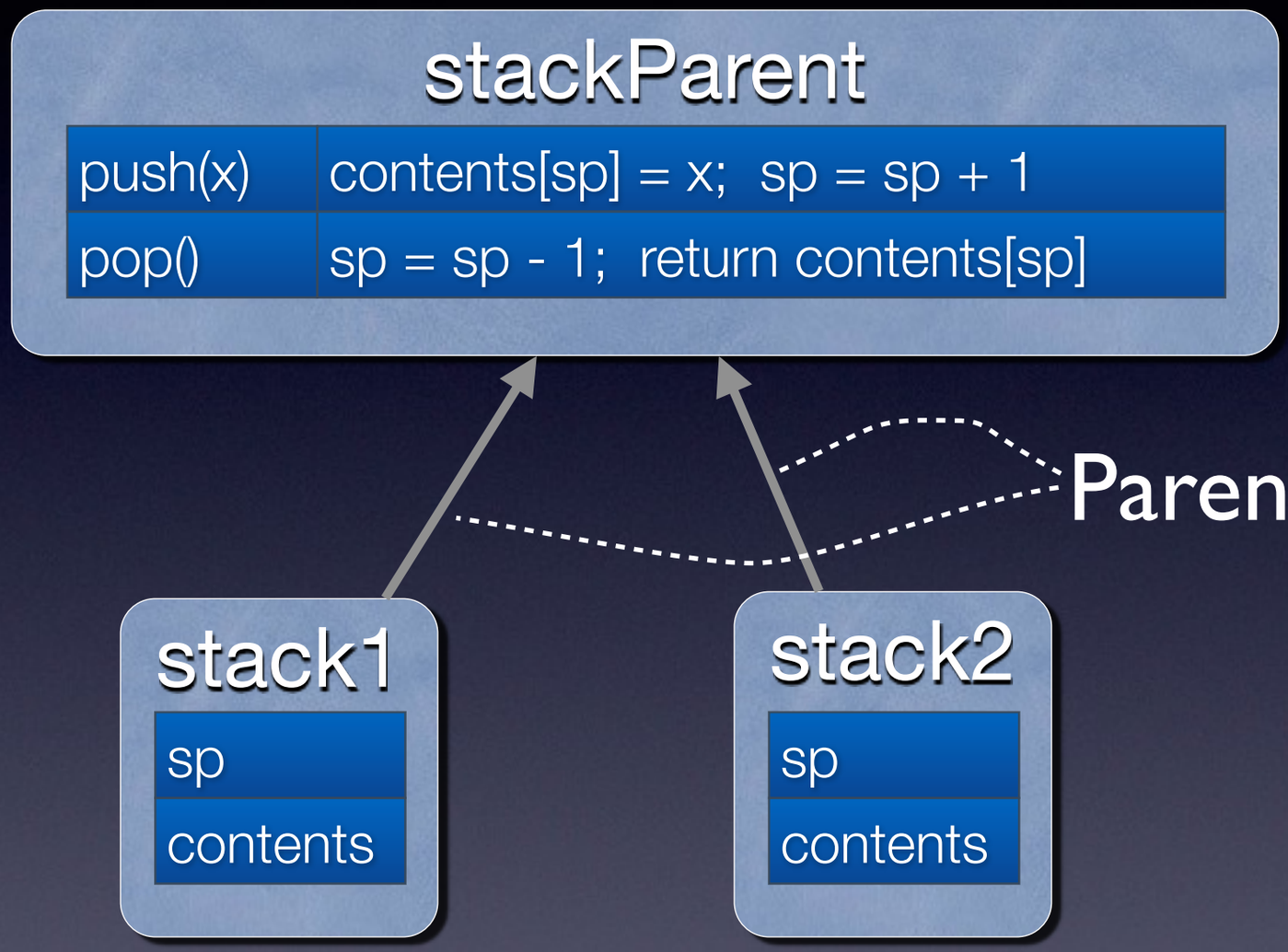
| stack2 | |
|---|---|
| sp | 0 |
| contents | an array |
| push(x) | contents[sp] = x;  sp = sp + 1 |
| pop() | sp = sp - 1;  return contents[sp] |

```
stack1.push(2)
stack2.push(3)
stack1.pop()  +  stack2.pop()
```



© 2013, 2014 IBM Corporation

# Inheritance / delegation: Lookup finds most specific match

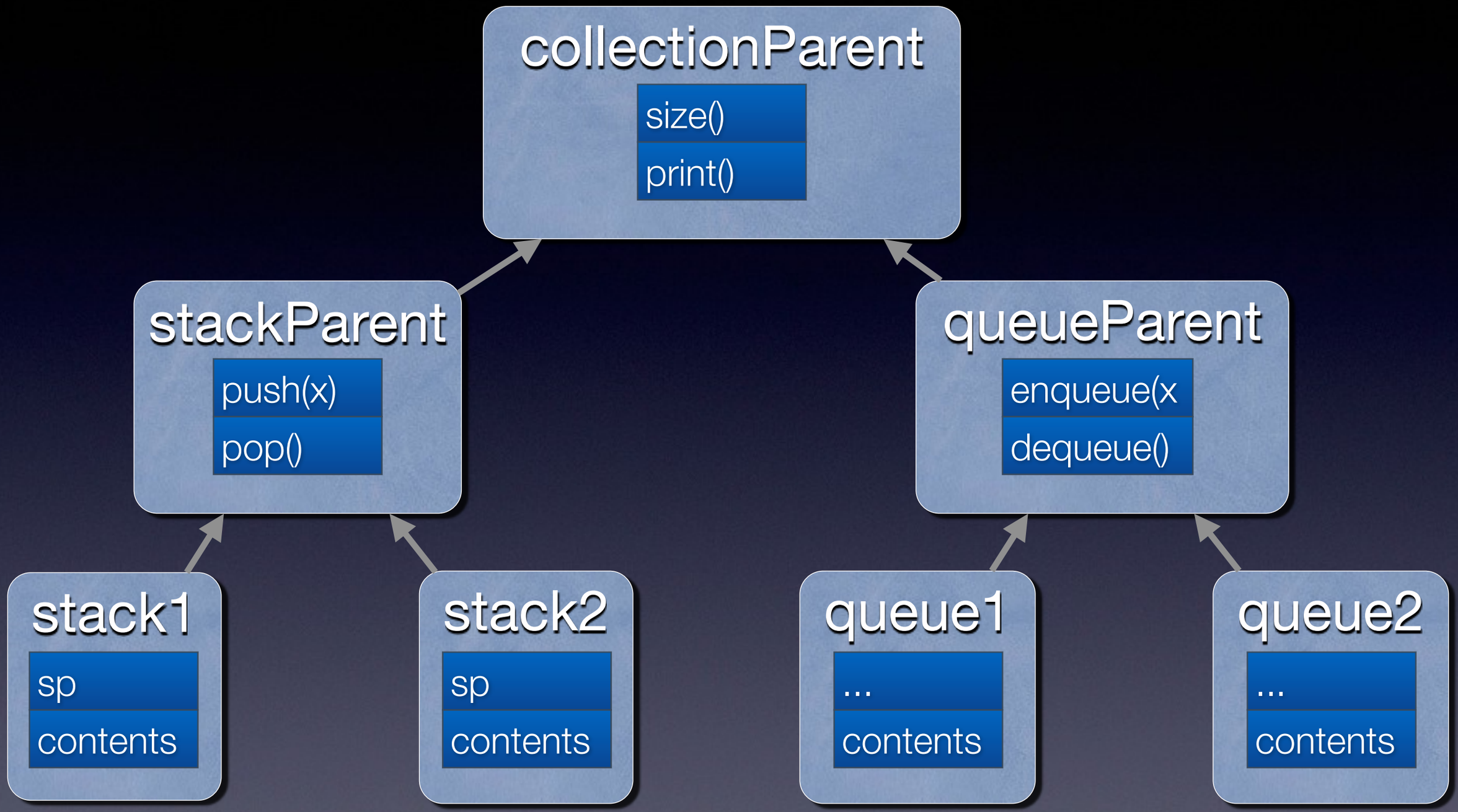



```
stack1.push(2)
stack2.push(3)
stack1.pop()  +  stack2.pop()
```



© 2013, 2014 IBM Corporation

# More inheritance / delegation




```
stack1.push(2)
stack2.push(3)
stack1.pop()  +  stack2.pop()
```

```
queue1.enqueue(2)
queue2.enqueue(3)
queue1.dequeue()  +  queue2.dequeue()
```



© 2013, 2014 IBM Corporation

Can get more *context,*
a richer unstated background,

with

# Many implicit arguments influencing computation by controlling dispatch



© 2013, 2014 IBM Corporation

# Preventing a mob scene

## Assign named roles to each implicit argument

Examples of roles

| Role name | Role use |
|---|---|
| rcvr | the “receiver” object |
| assertions | how much assertion checking |
| logging | how much logging |
| codeVersion | which revision |
| user | on whose behalf |



© 2013, 2014 IBM Corporation

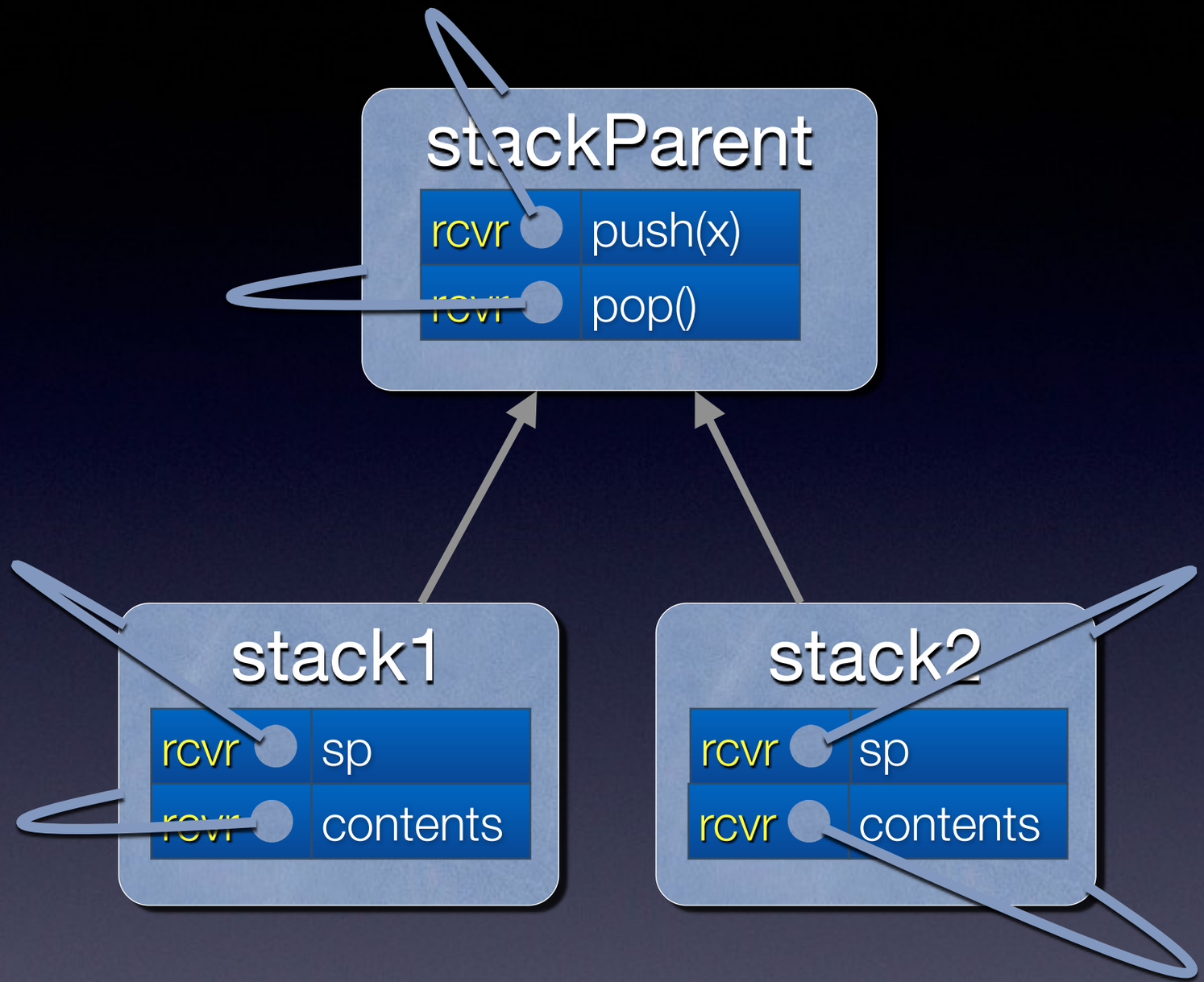
# Preview

- Each message sent to tuple of role names & objects
- Each particle of state or behavior (slot) will say required object for each role that matters to it
- Sea of slots clumping together depending on viewpoint



© 2013, 2014 IBM Corporation

# Slot could have a piece with receiver

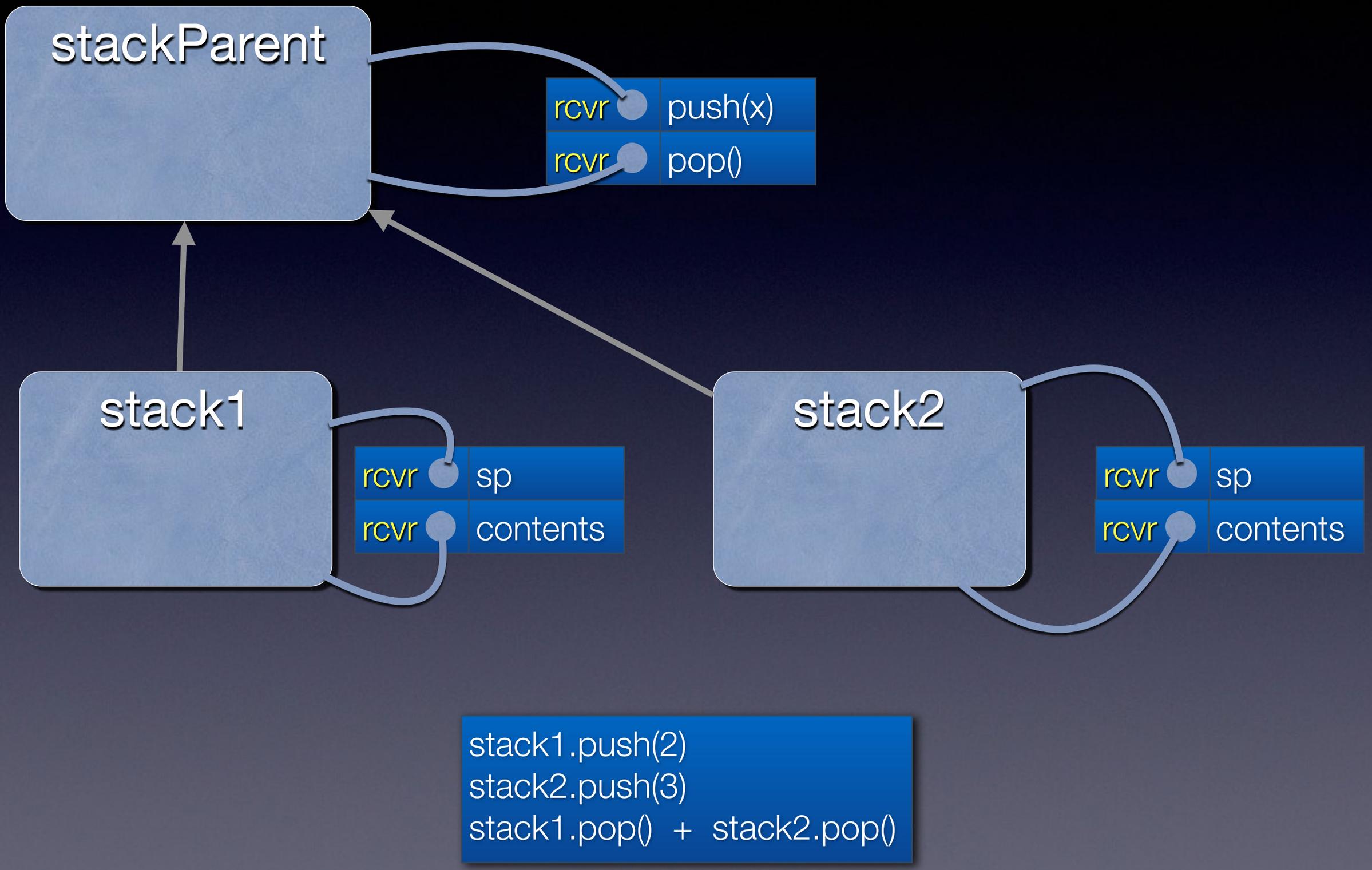



```
stack1.push(2)
stack2.push(3)
stack1.pop()  +  stack2.pop()
```



© 2013, 2014 IBM Corporation

# Slots linked instead of contained

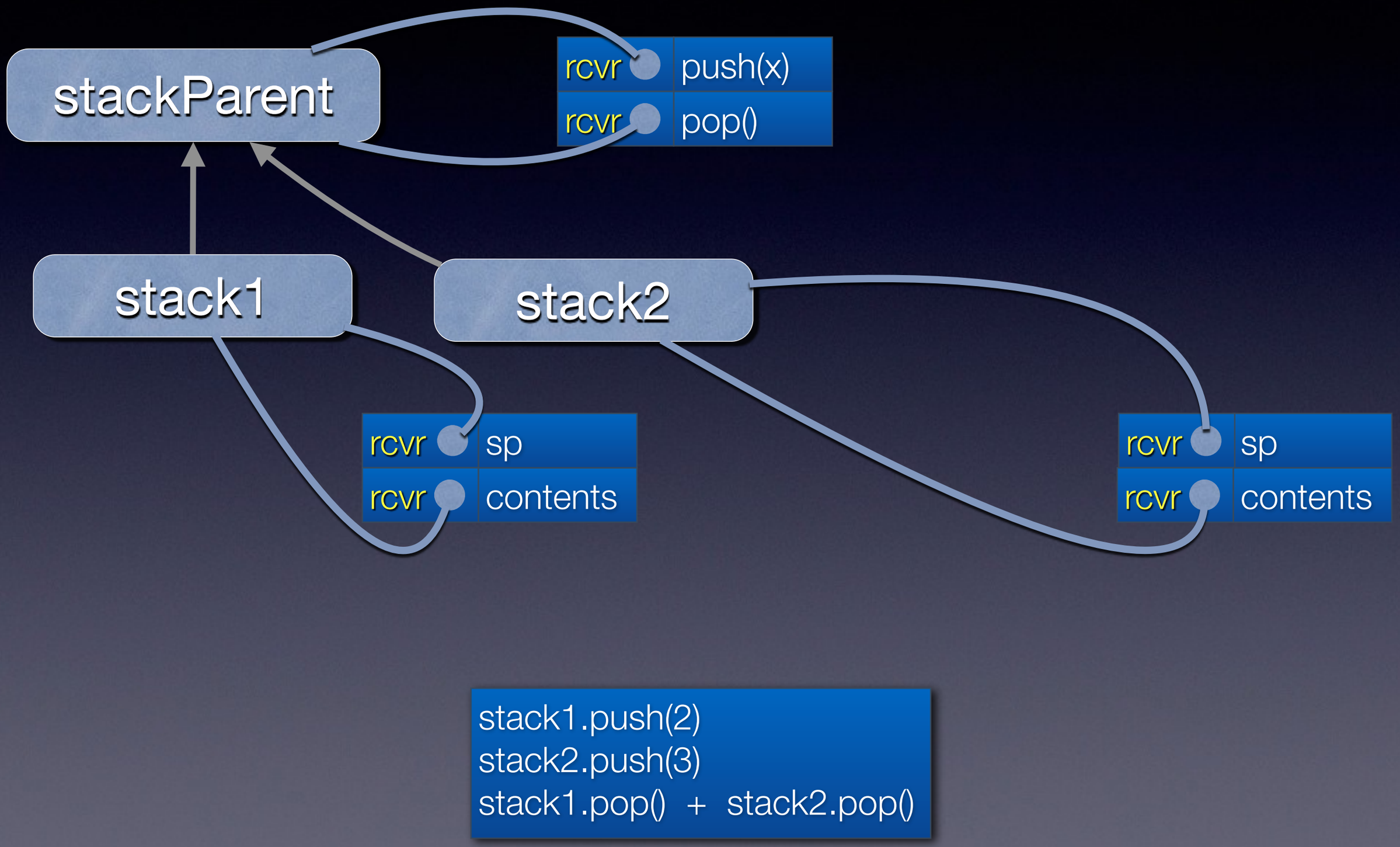



```
stack1.push(2)
stack2.push(3)
stack1.pop()  +  stack2.pop()
```



© 2013, 2014 IBM Corporation

# Objects are just unique coordinates, with identity

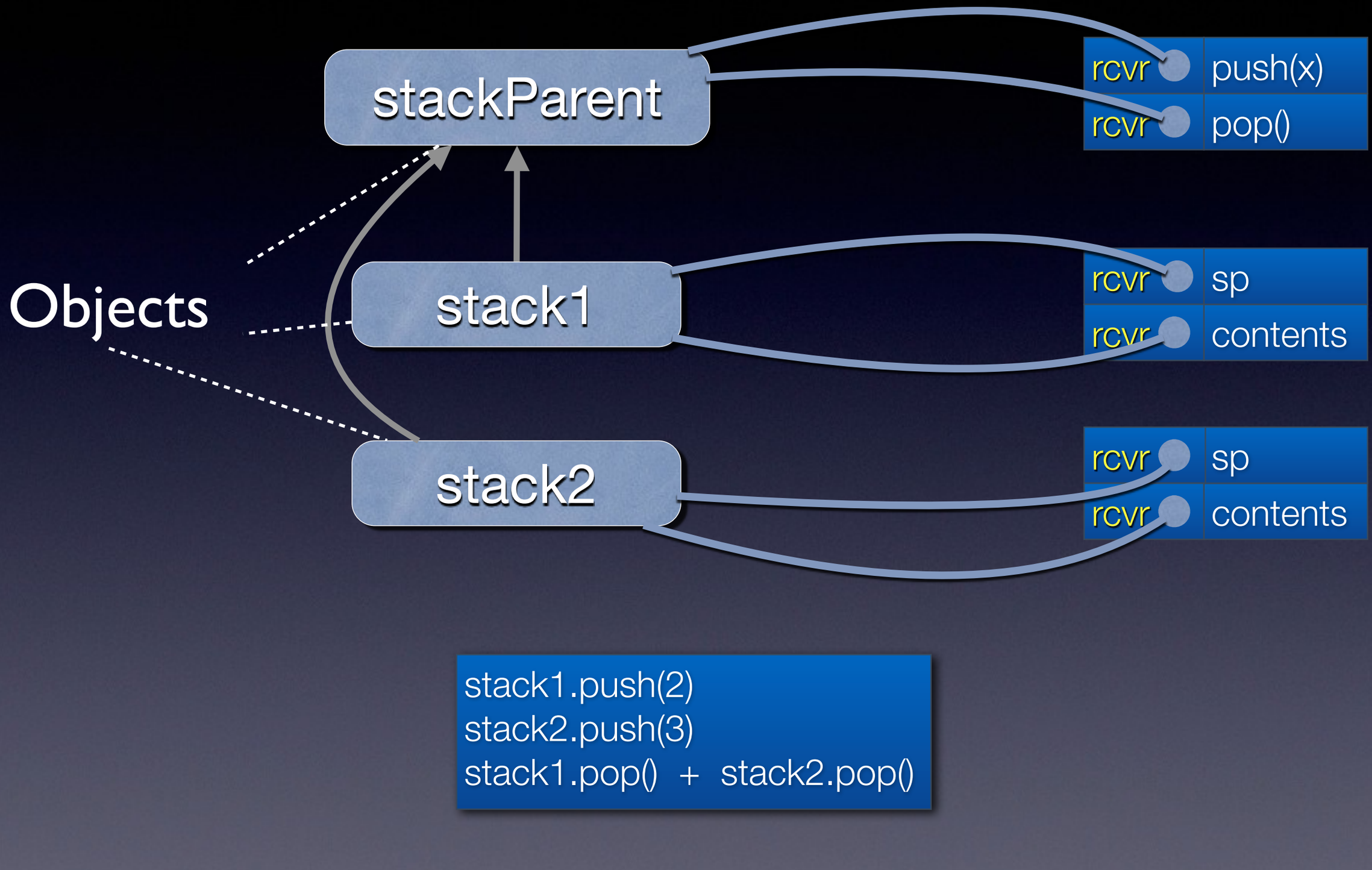



```
stack1.push(2)
stack2.push(3)
stack1.pop()  +  stack2.pop()
```



© 2013, 2014 IBM Corporation

# Objects are just unique coordinates, with identity

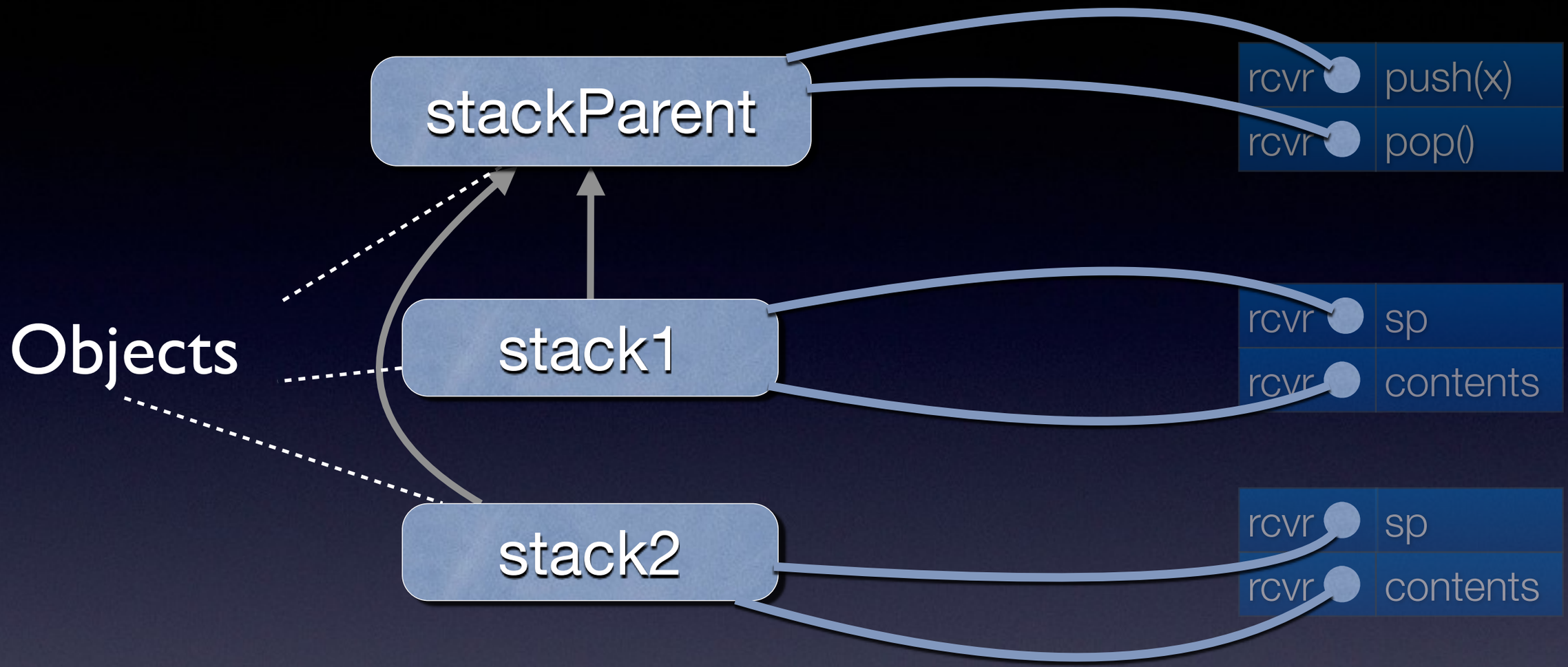



```
stack1.push(2)
stack2.push(3)
stack1.pop()  +  stack2.pop()
```



© 2013, 2014 IBM Corporation

# Objects are just unique coordinates, with identity

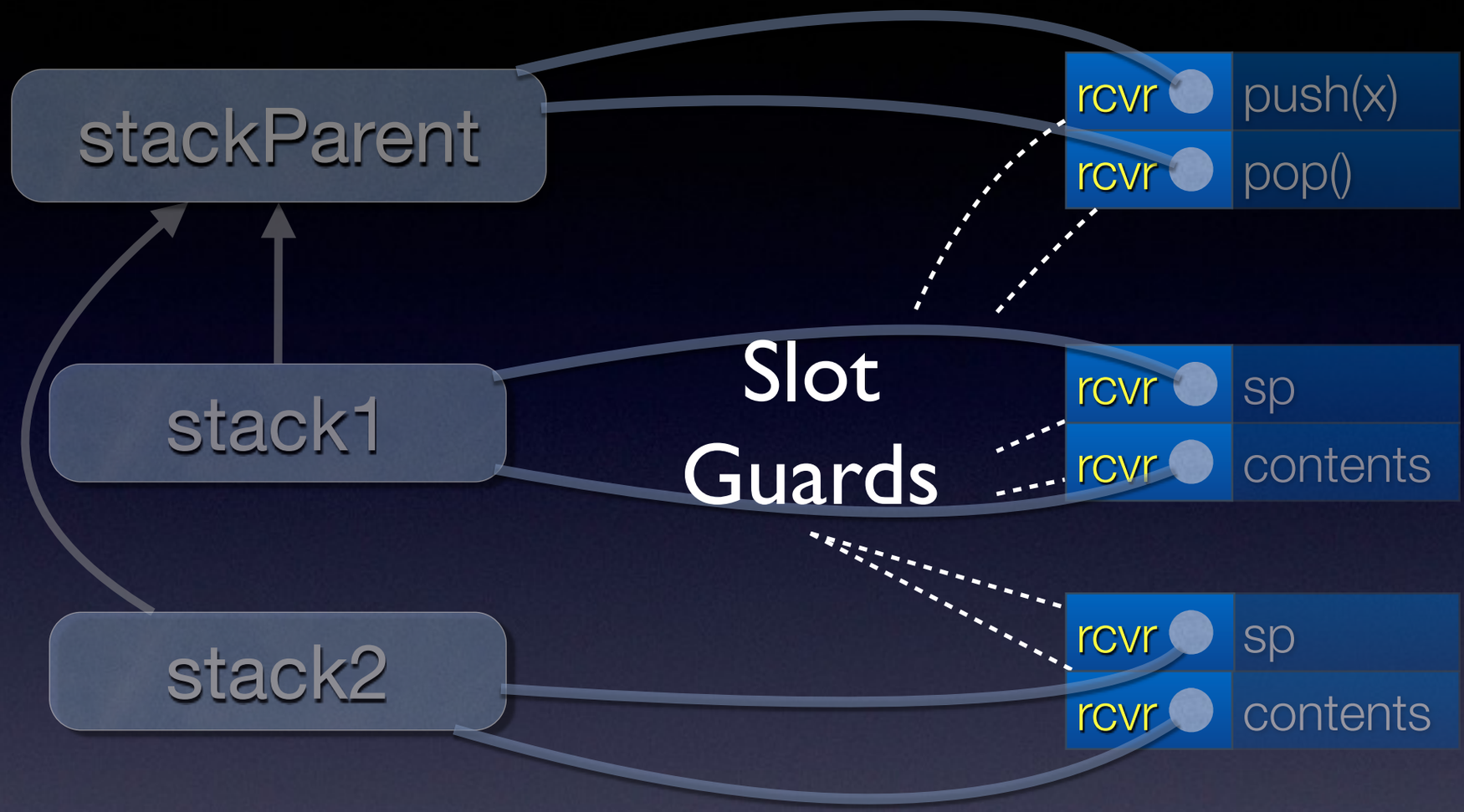



```
stack1.push(2)
stack2.push(3)
stack1.pop()  +  stack2.pop()
```



© 2013, 2014 IBM Corporation

# Slot guards control dispatch

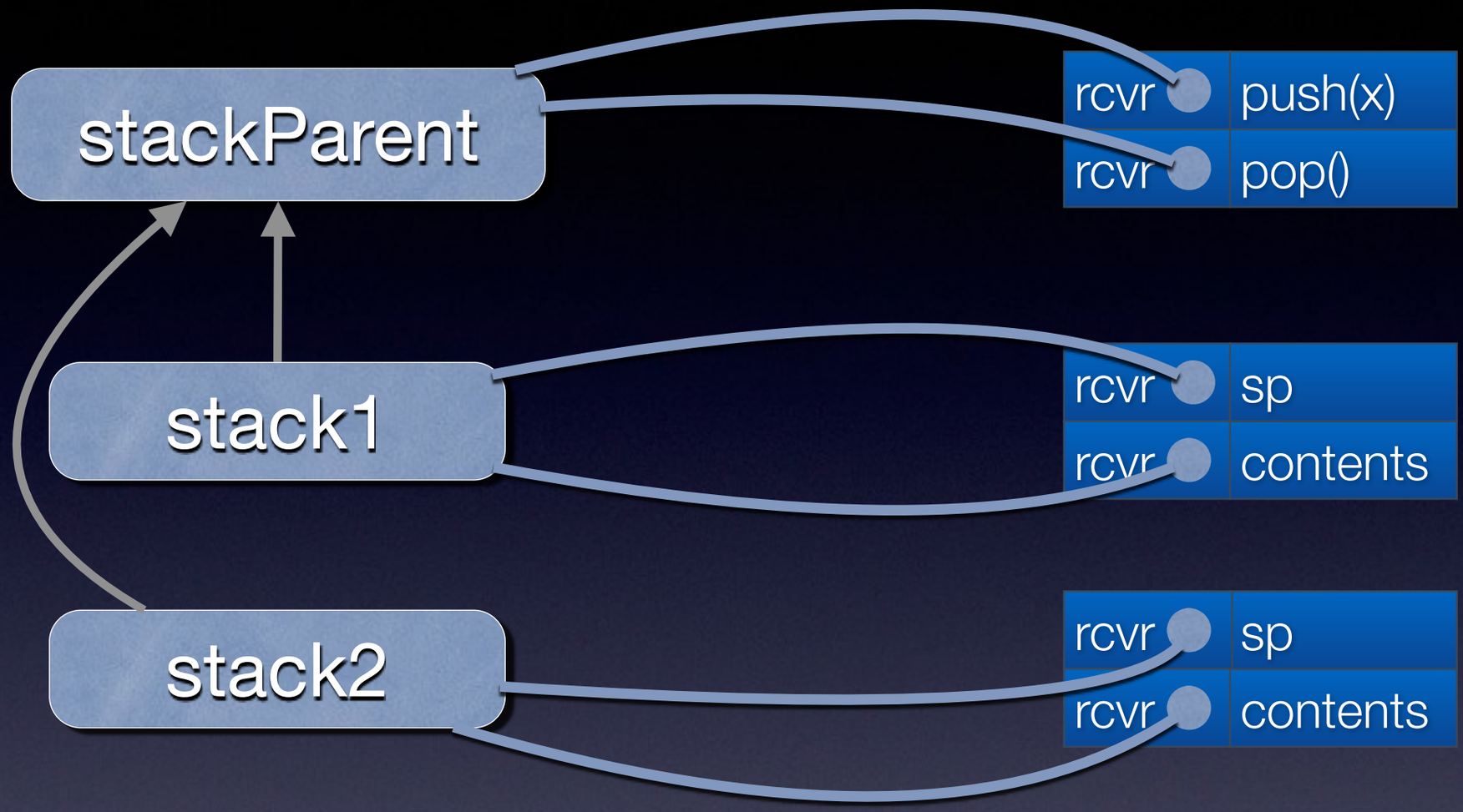



```
stack1.push(2)
stack2.push(3)
stack1.pop()  +  stack2.pop()
```



© 2013, 2014 IBM Corporation

# Slot guards control dispatch



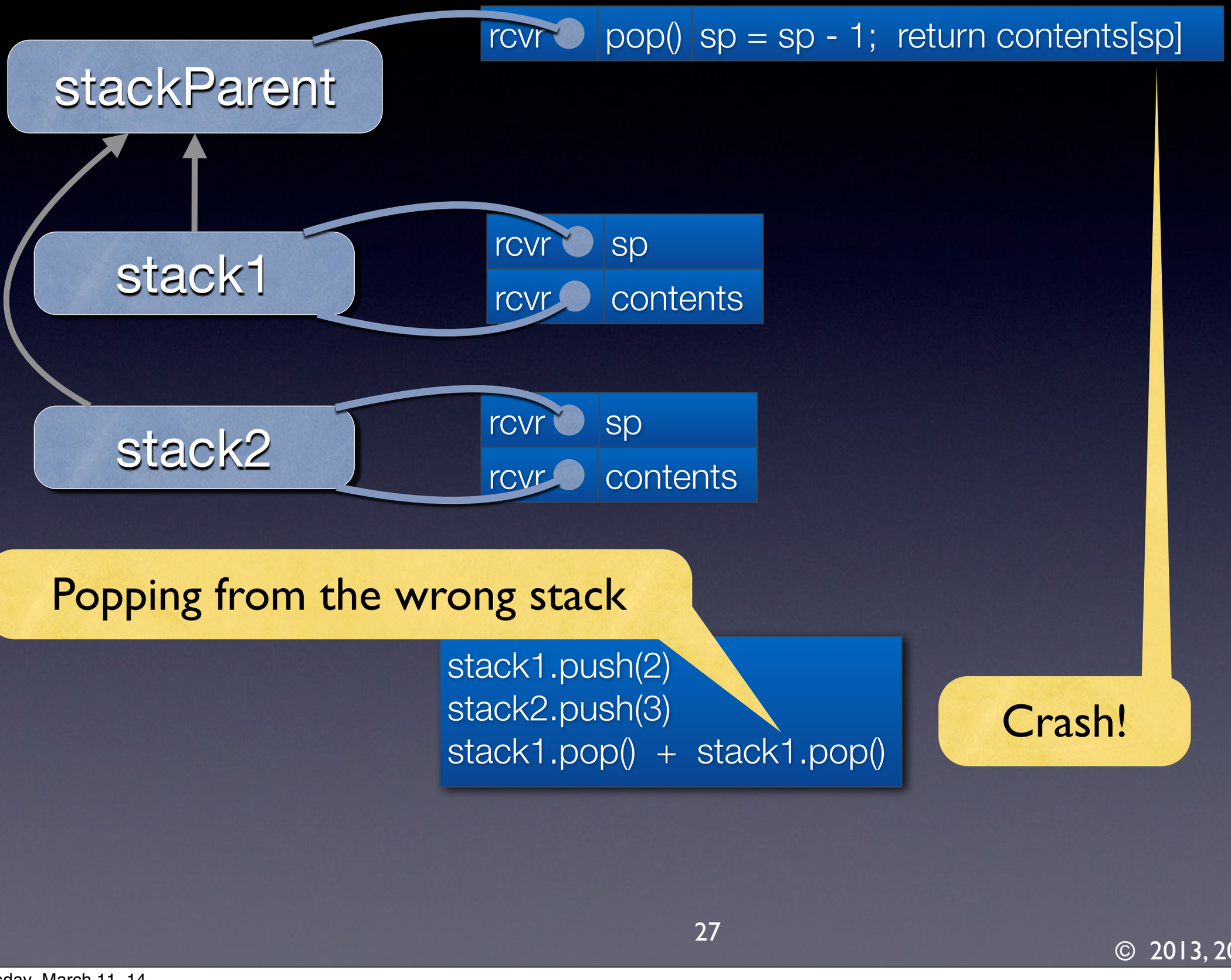

```
stack1.push(2)
stack2.push(3)
stack1.pop()  +  stack2.pop()
```

Nothing really new so far



© 2013, 2014 IBM Corporation

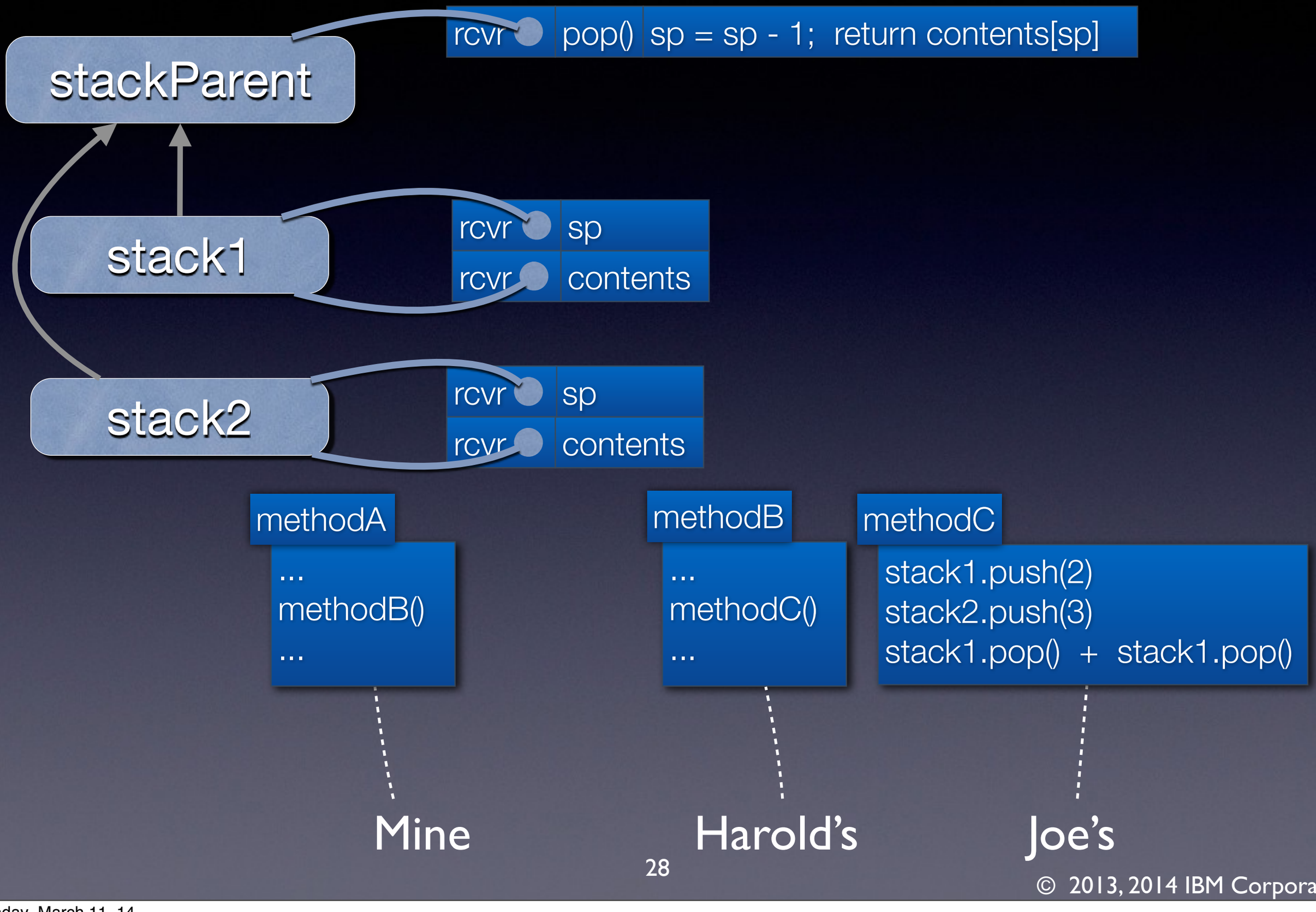
# Suppose there's a bug






© 2013, 2014 IBM Corporation

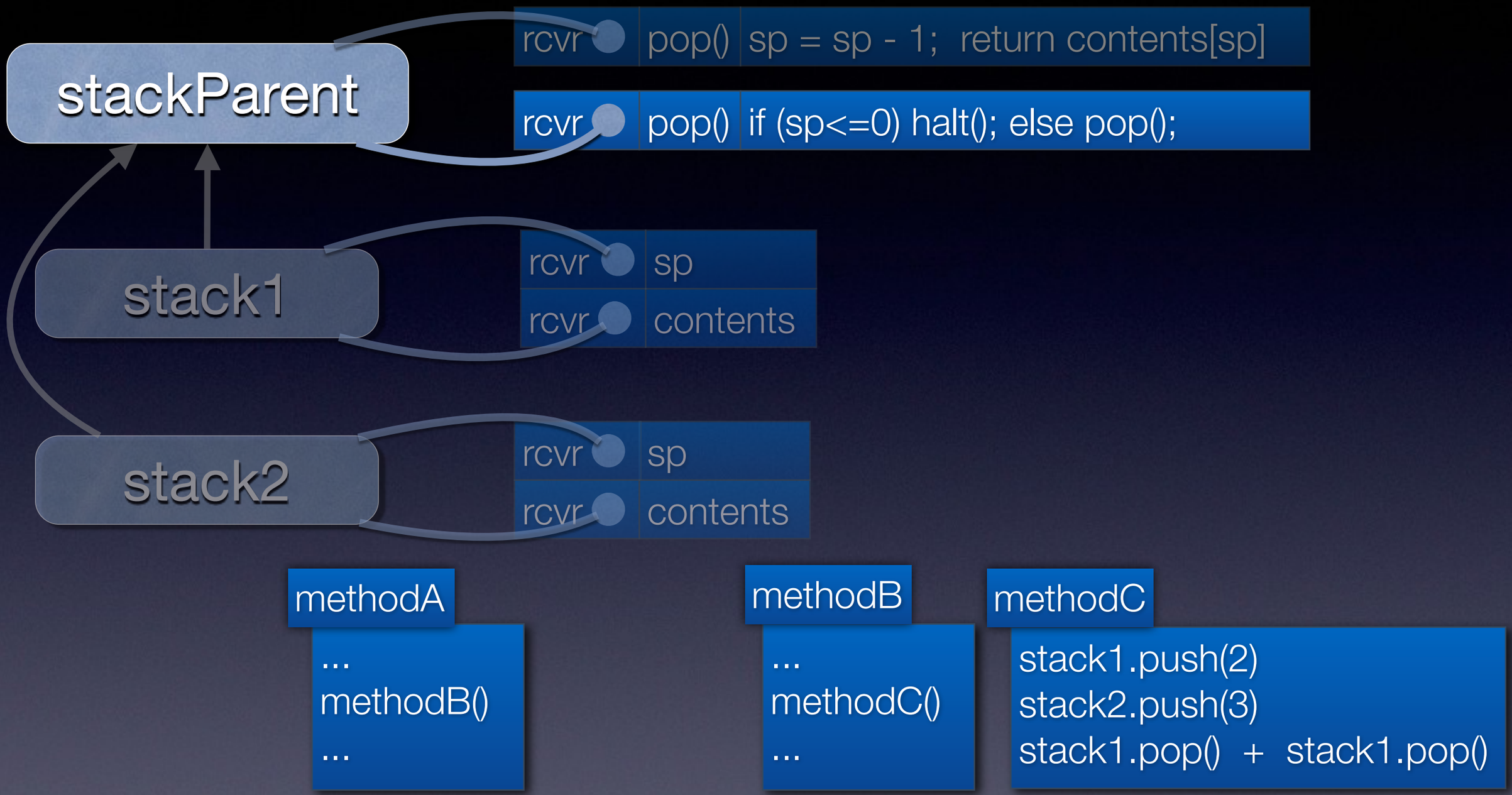
# Not in *my* code






© 2013, 2014 IBM Corporation

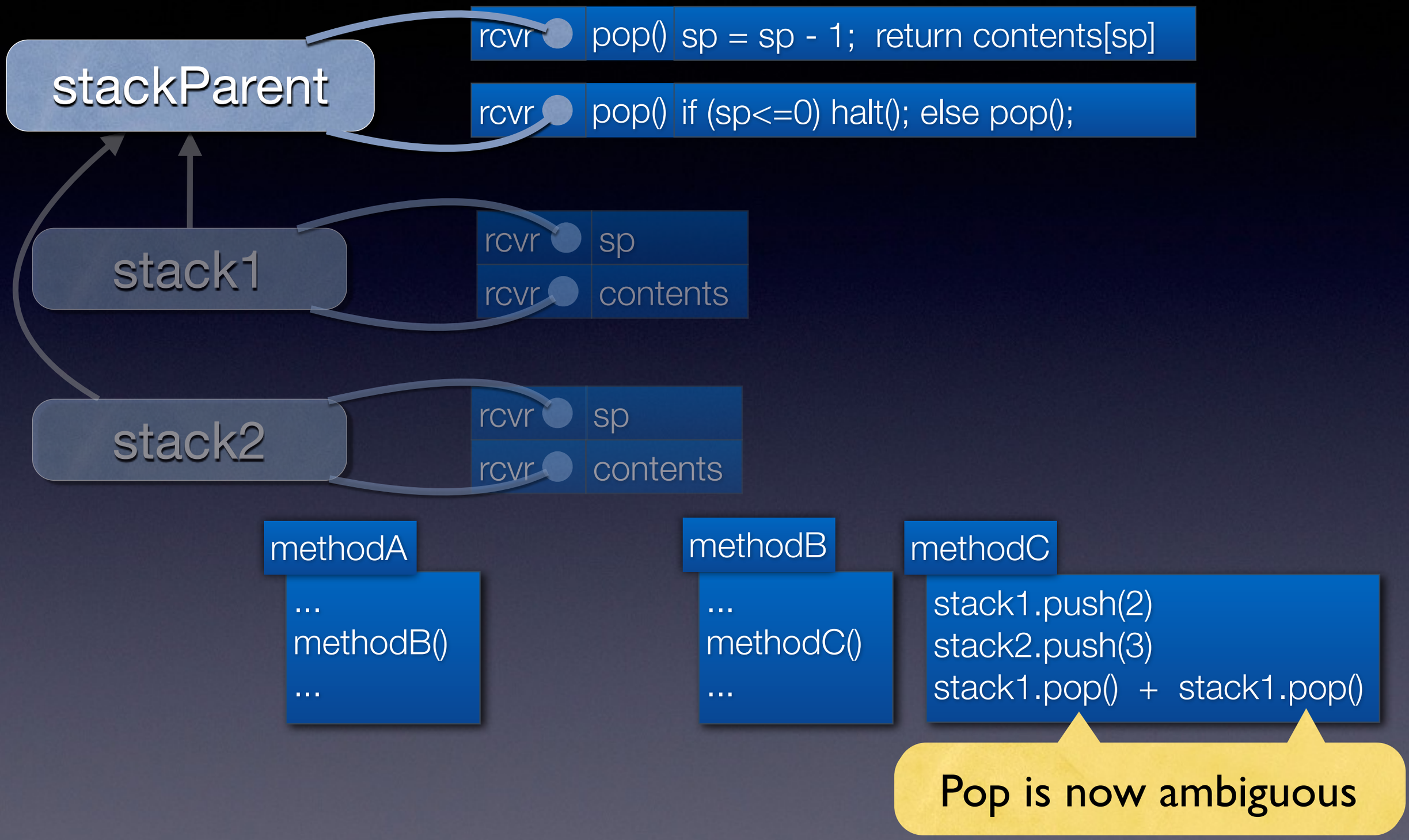
# Need a pop with a check






© 2013, 2014 IBM Corporation

# Need a pop with a check






© 2013, 2014 IBM Corporation

# Add to the guards






© 2013, 2014 IBM Corporation

# Set “debug” argument
# Slot runs if all arguments match

false

| rcvr | debug | pop() | sp = sp - 1;  return contents[sp] |
|---|---|---|---|

| rcvr | debug | pop() | if (sp<=0) halt(); {debug=>false}.pop(); |
|---|---|---|---|

true

stackParent

stack1

| rcvr | sp |
|---|---|
| rcvr | contents |

stack2

| rcvr | sp |
|---|---|
| rcvr | contents |

methodA

```
...
{debug=>true}.methodB()
...
```

methodB

```
...
methodC()
...
```

methodC

```
stack1.push(2)
stack2.push(3)
stack1.pop()  +  stack1.pop()
```

Set debug argument

Implicitly passes down debug

# Names & values of implicit arguments ≡ Message Context



© 2013, 2014 IBM Corporation

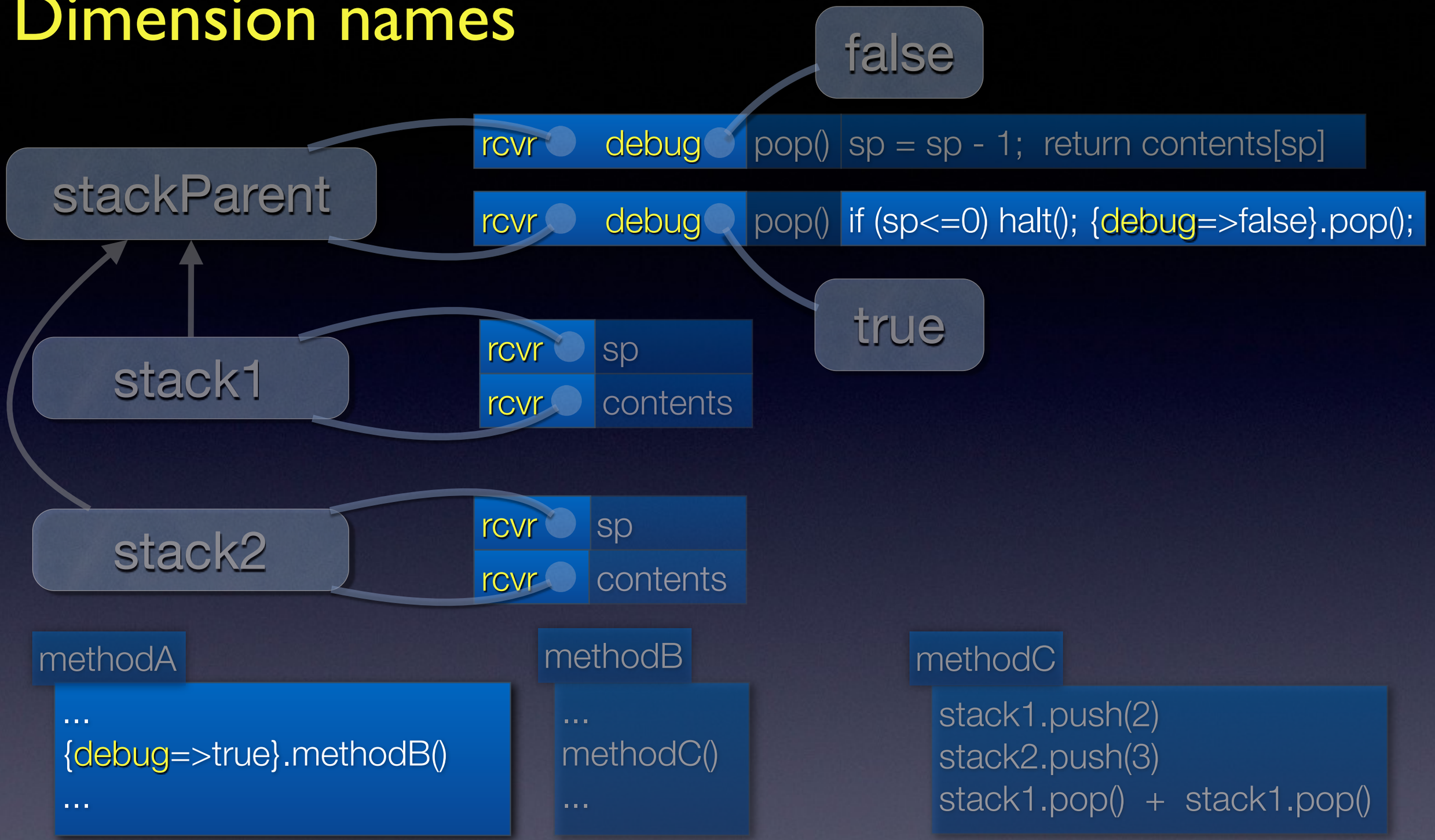
# Message Context

- Message *not* sent (just) to one receiver
- Message sent to zero-or-more name-value pairs
  - Name describes role of its value in that send
    - Role ≡ Dimension
  - Each value is an object & participates in dispatch
    - Value ≡ Coordinate



© 2013, 2014 IBM Corporation

# Dimension names

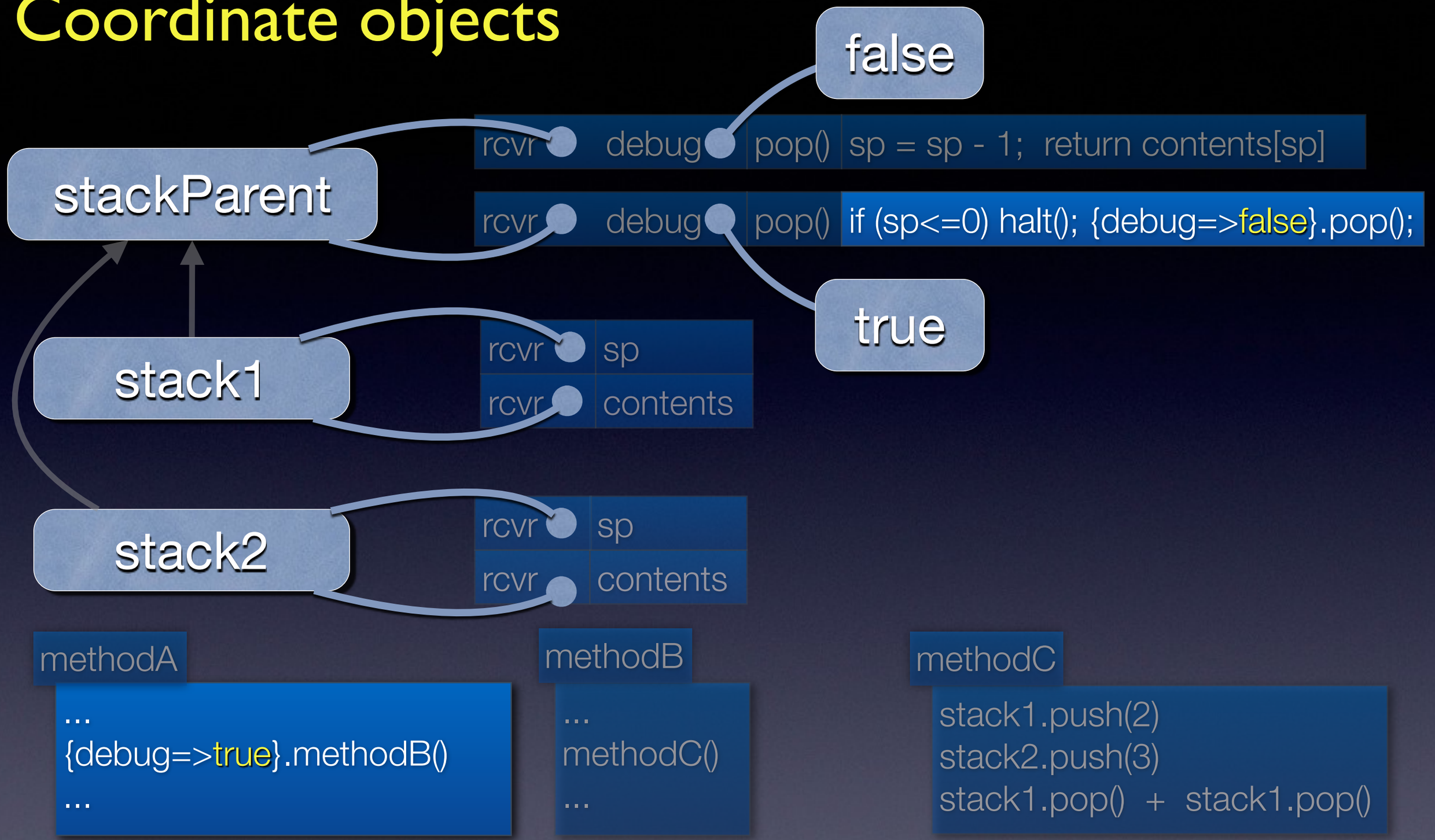



Dimension names ≡ simple strings



© 2013, 2014 IBM Corporation

# Coordinate objects

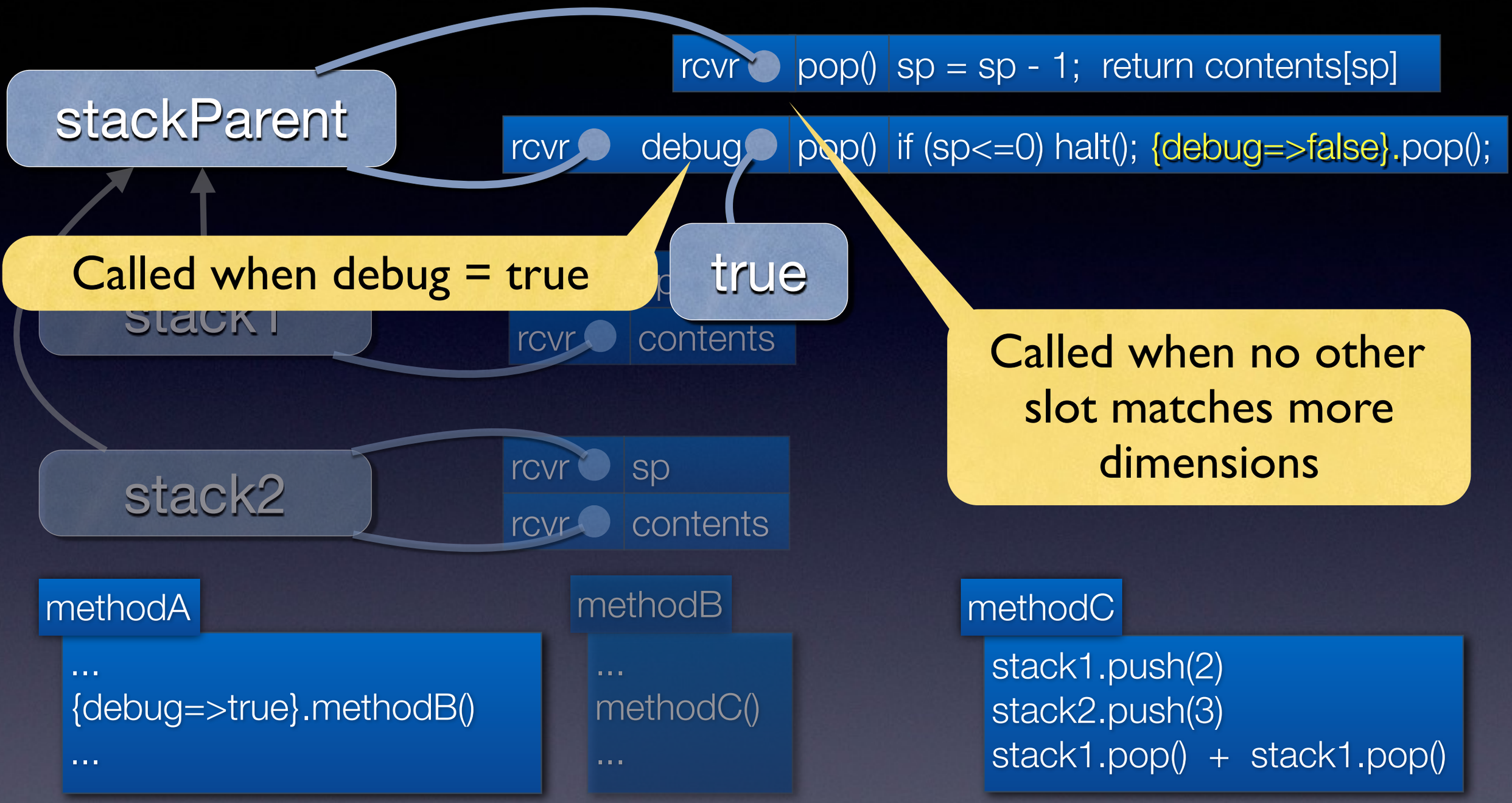



## Coordinates ≡ (references to) objects



© 2013, 2014 IBM Corporation

# Slot guard can ignore dimensions



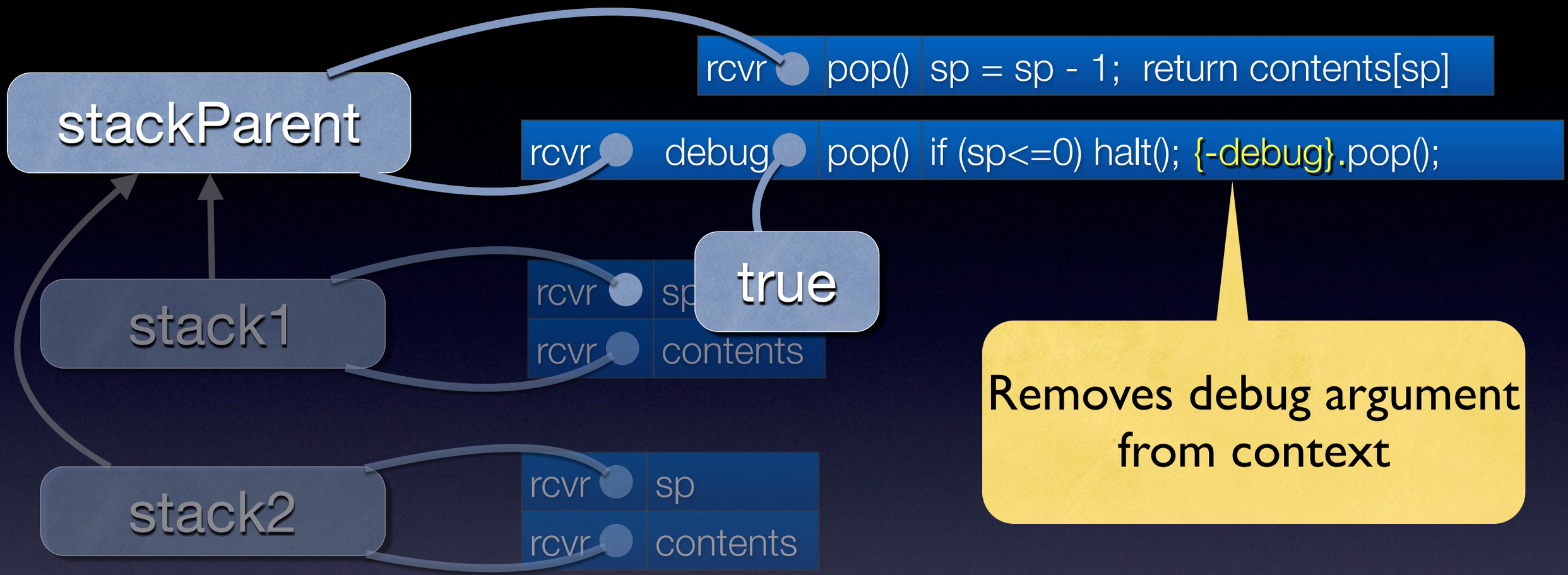

Ignored dimension in guard ≡ Don't care (less specific)

Easy to add dimension: need not add to every slot guard



© 2013, 2014 IBM Corporation

# Message send can remove a dimension




**methodA**

```
...
{debug=>true}.methodB()
...
```

**methodB**

```
...
methodC()
...
```

**methodC**

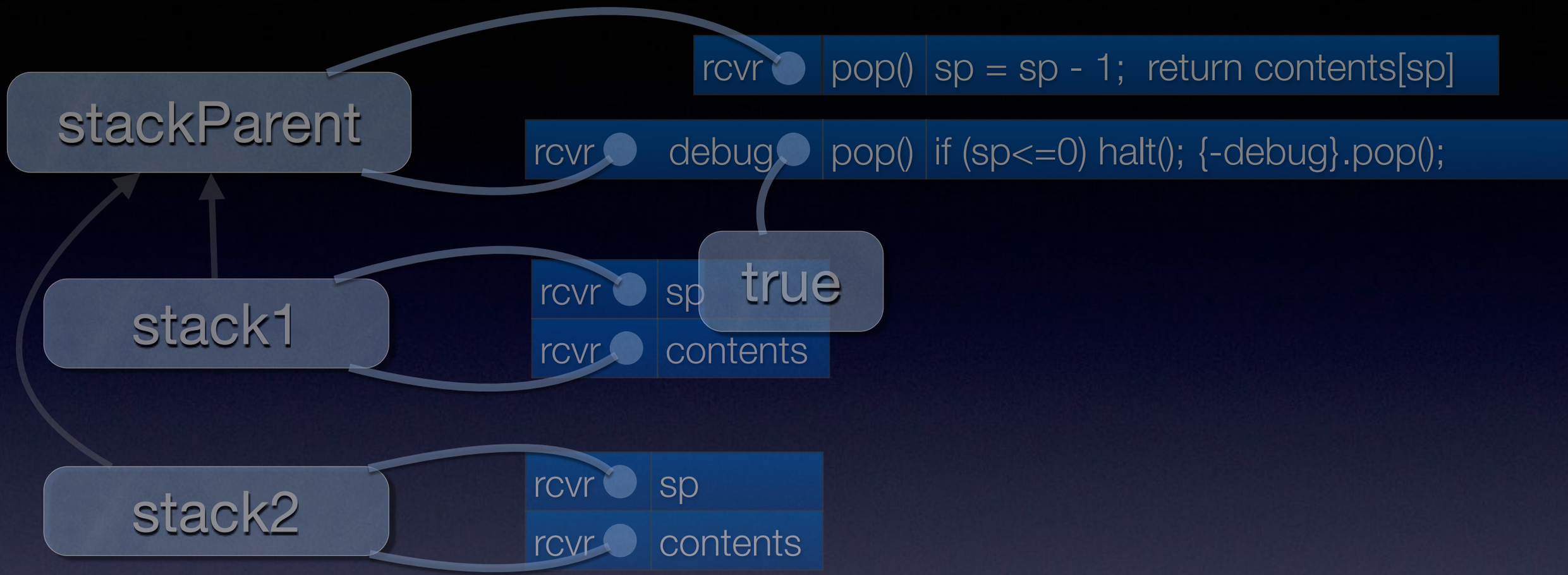
```
stack1.push(2)
stack2.push(3)
stack1.pop()  +  stack1.pop()
```

## {-dimensionName} = Remove dimension from context



© 2013, 2014 IBM Corporation

# stack1.pop() is just sugar for {rcvr=>stack1}.pop()




methodA

```
...
{debug=>true}.methodB()
...
```

methodB

```
...
methodC()
...
```

methodC

```
stack1.push(2)
stack2.push(3)
{rcvr=>stack1}.pop()  +  stack1.pop()
```

## “normal” syntax privileges receiver



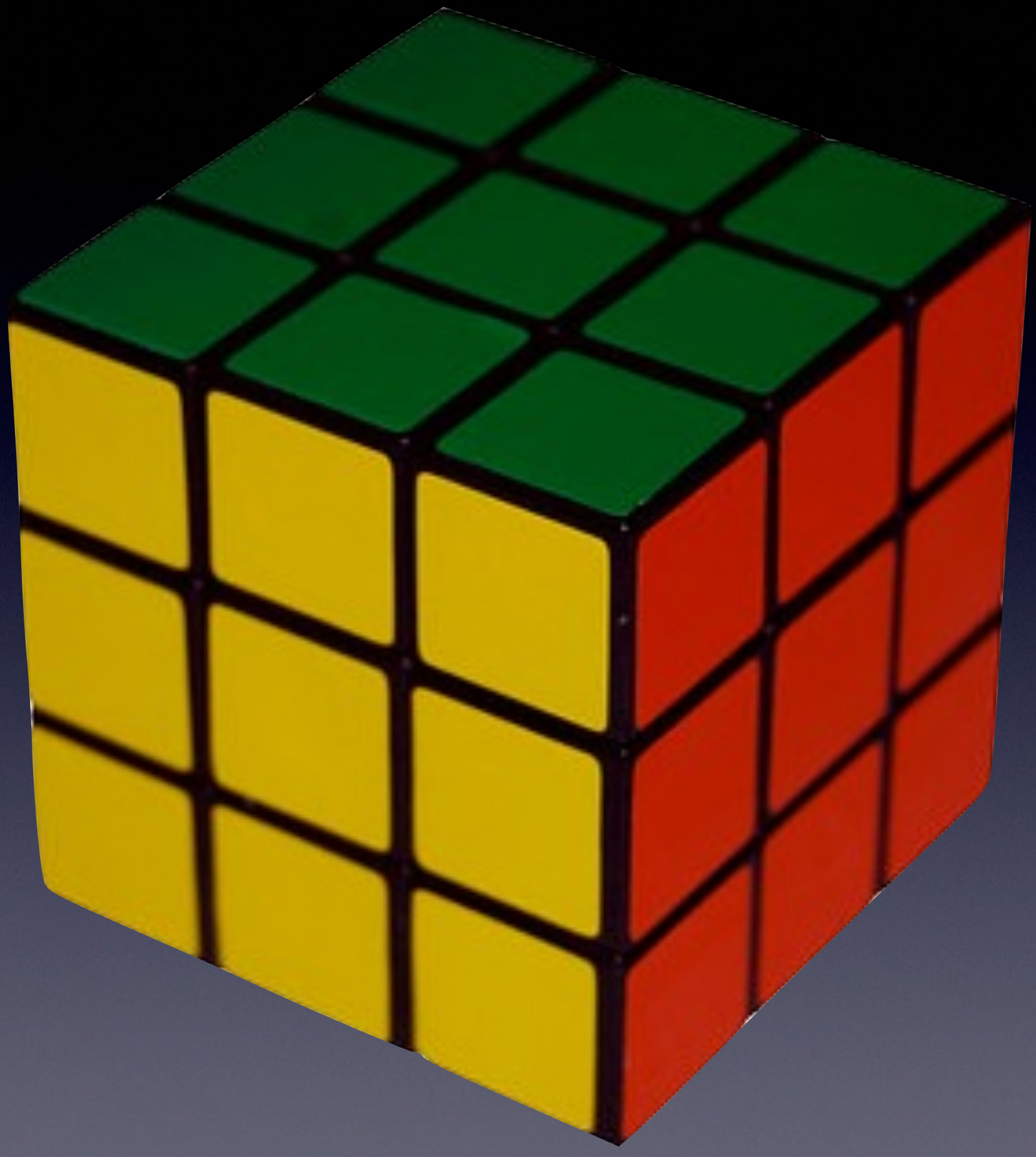
© 2013, 2014 IBM Corporation

# Summary

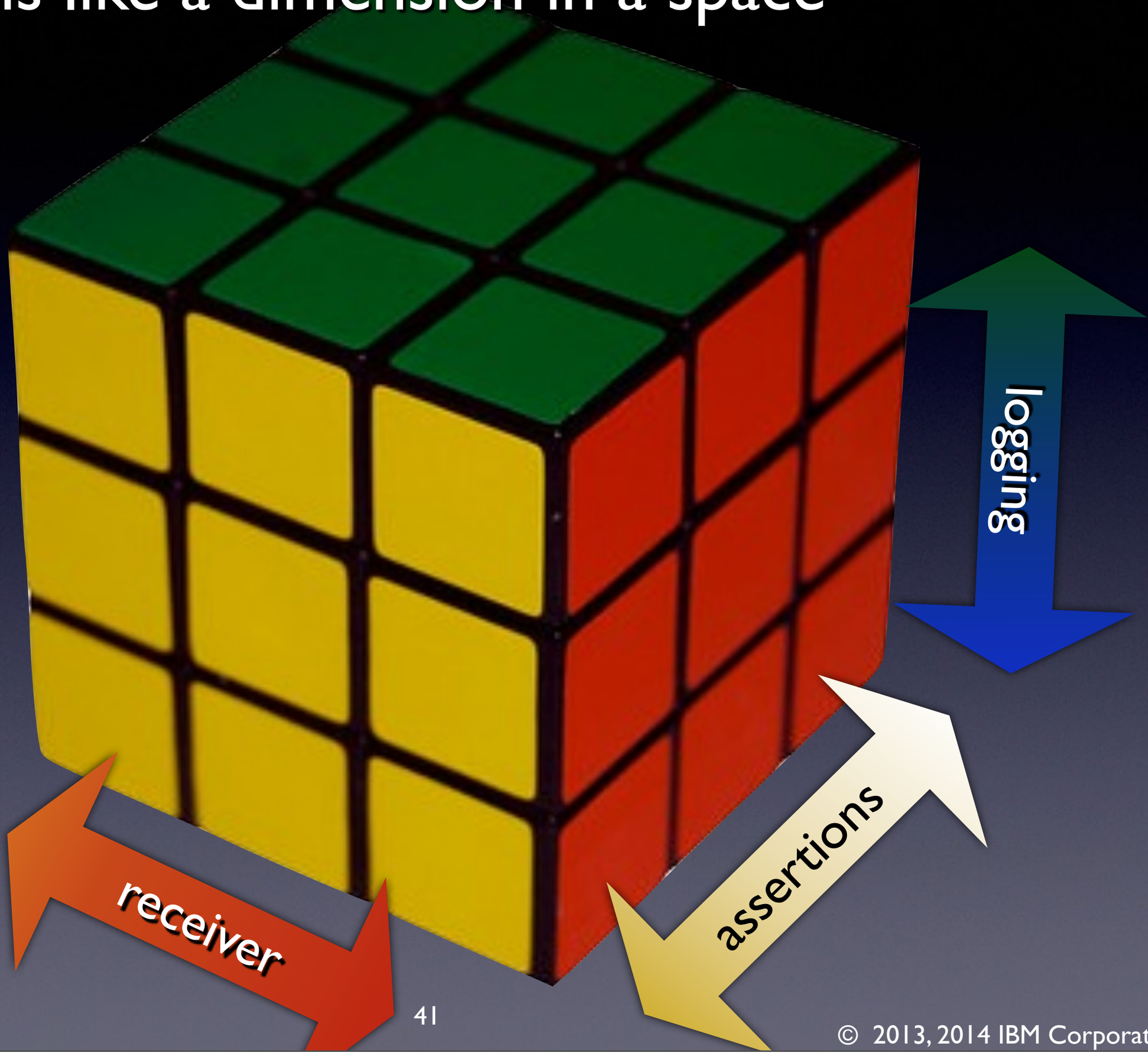
- Self-ish in this incarnation
  - Prototype inheritance/delegation
  - Slots for state and behavior
- Sea of slots
  - Linked instead of contained
  - Linked to any number of objects
  - Dimensions & coordinates
- Message sent to context of named implicit arguments
  - Runs most specific match (will explain)



© 2013, 2014 IBM Corporation



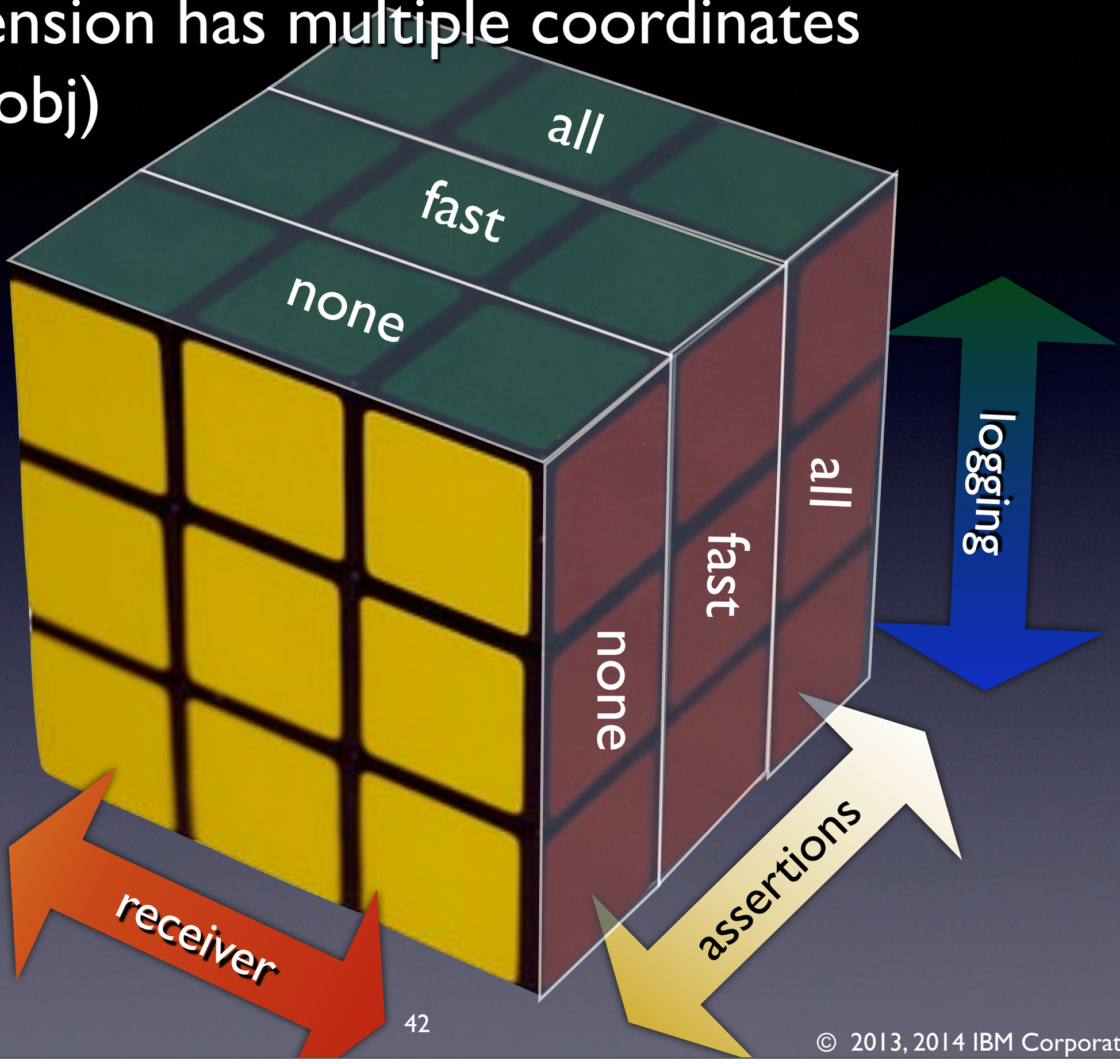
# Each role is like a dimension in a space






© 2013, 2014 IBM Corporation

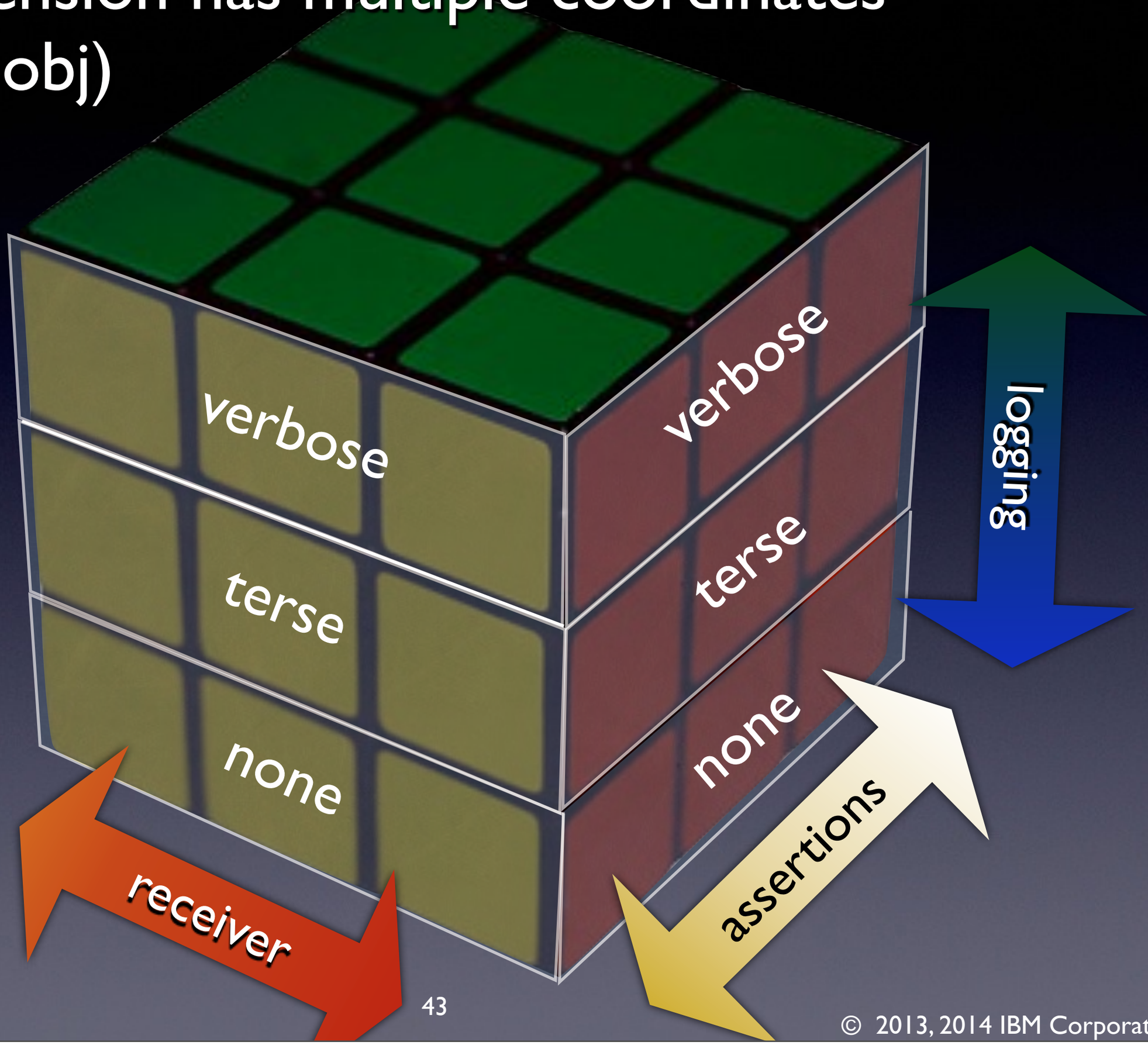
# Each dimension has multiple coordinates (coord = obj)






© 2013, 2014 IBM Corporation

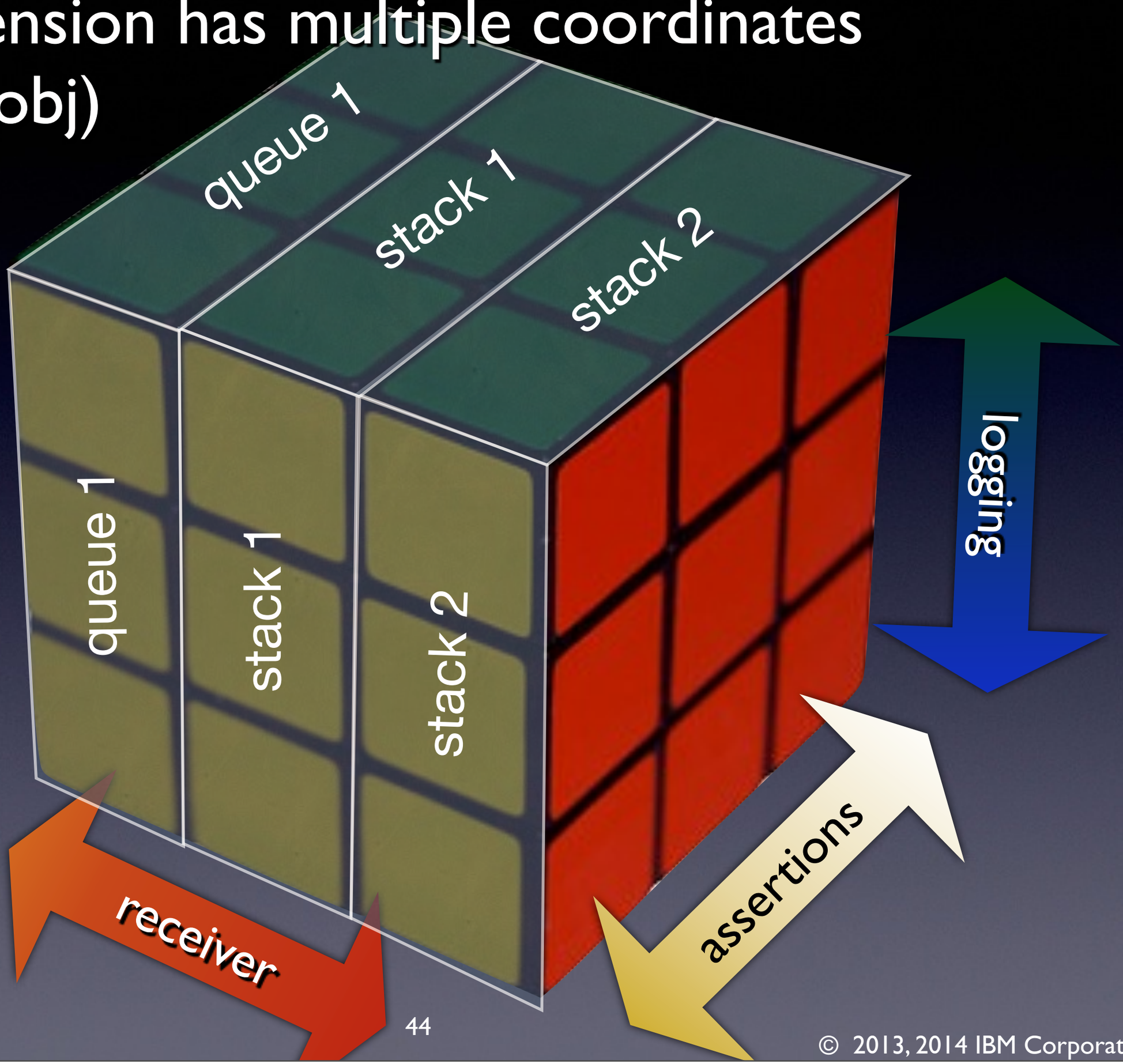
# Each dimension has multiple coordinates (coord = obj)






© 2013, 2014 IBM Corporation

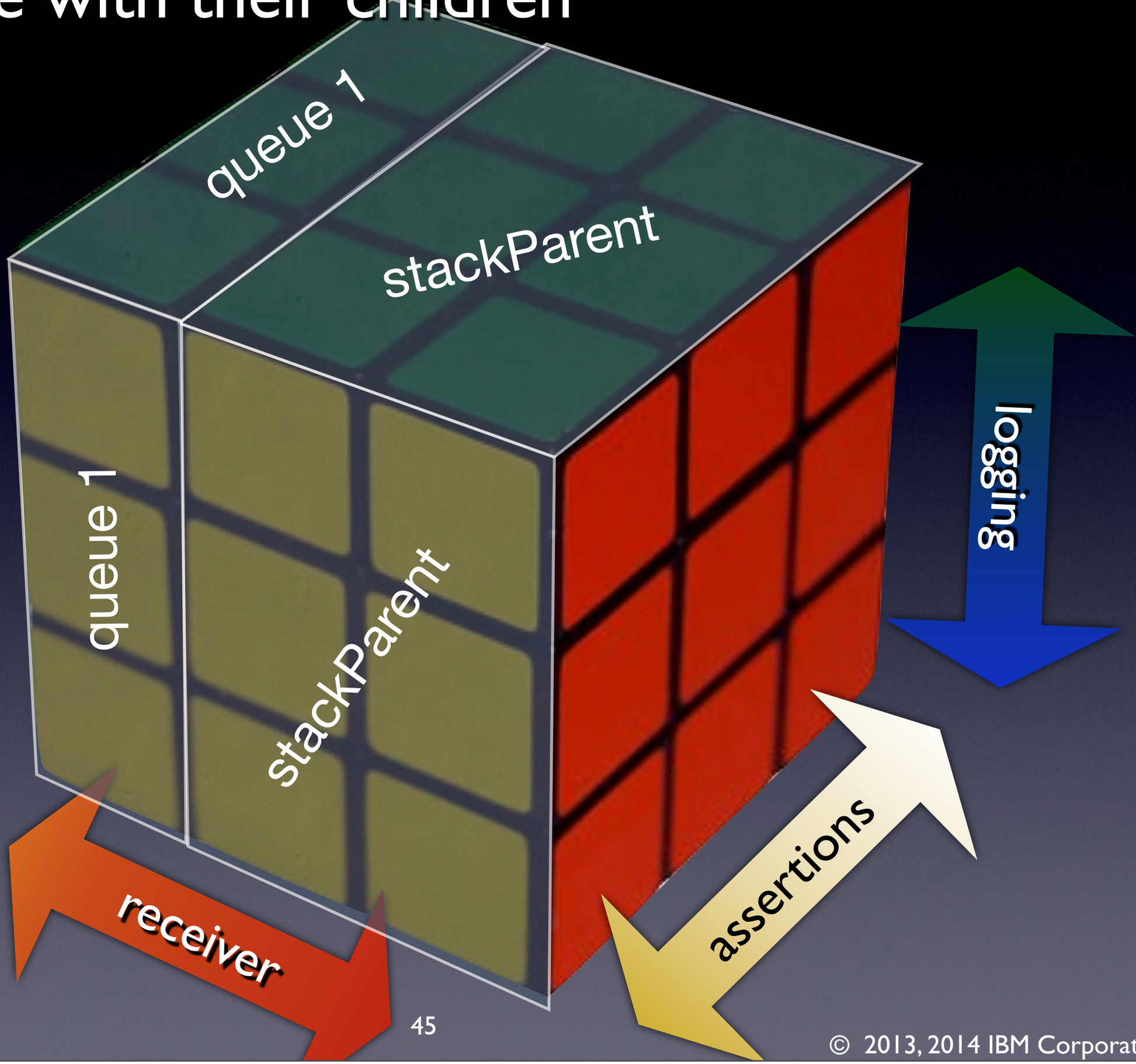
# Each dimension has multiple coordinates (coord = obj)






© 2013, 2014 IBM Corporation

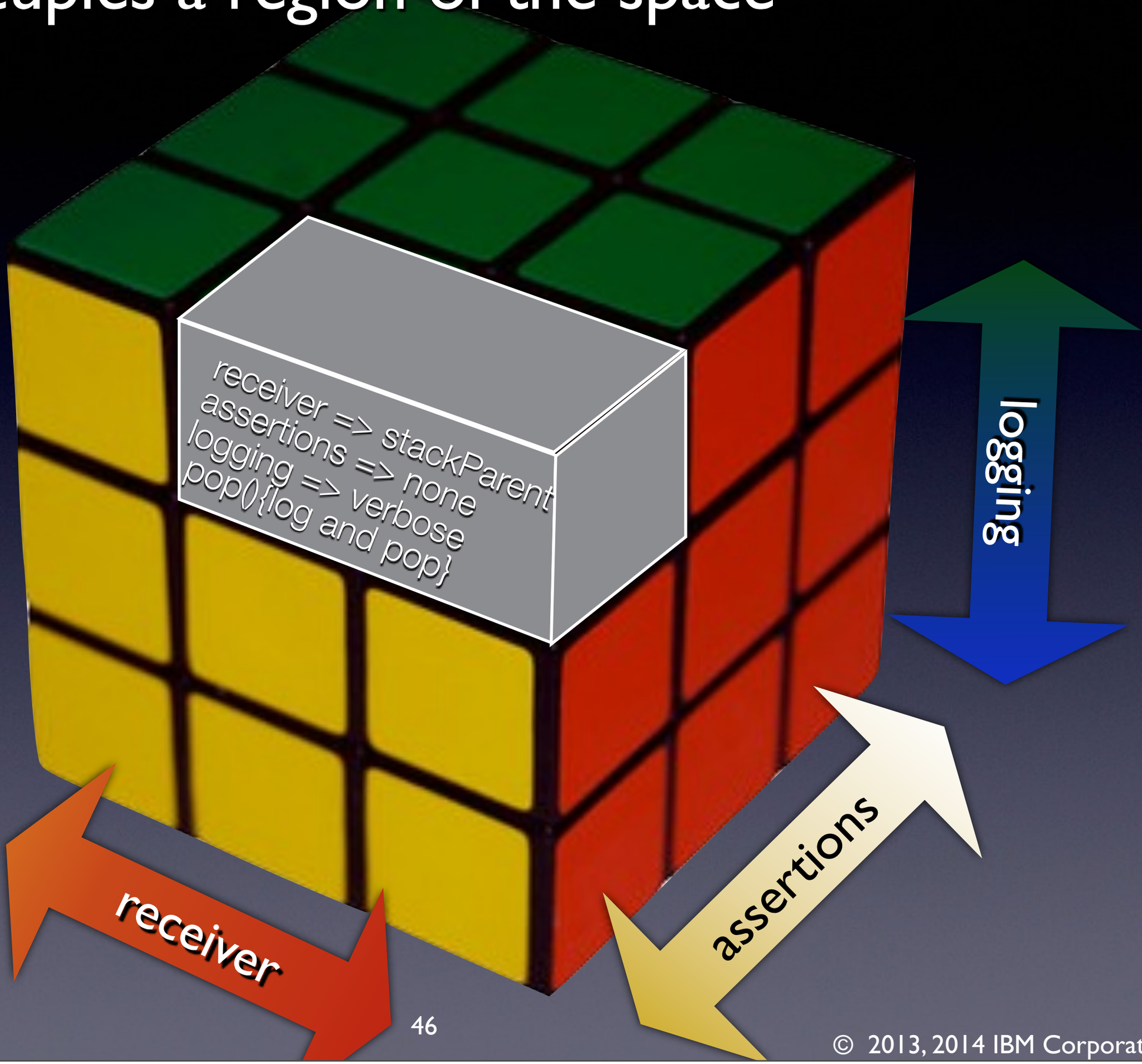
# Parents live with their children






© 2013, 2014 IBM Corporation

# A slot occupies a region of the space

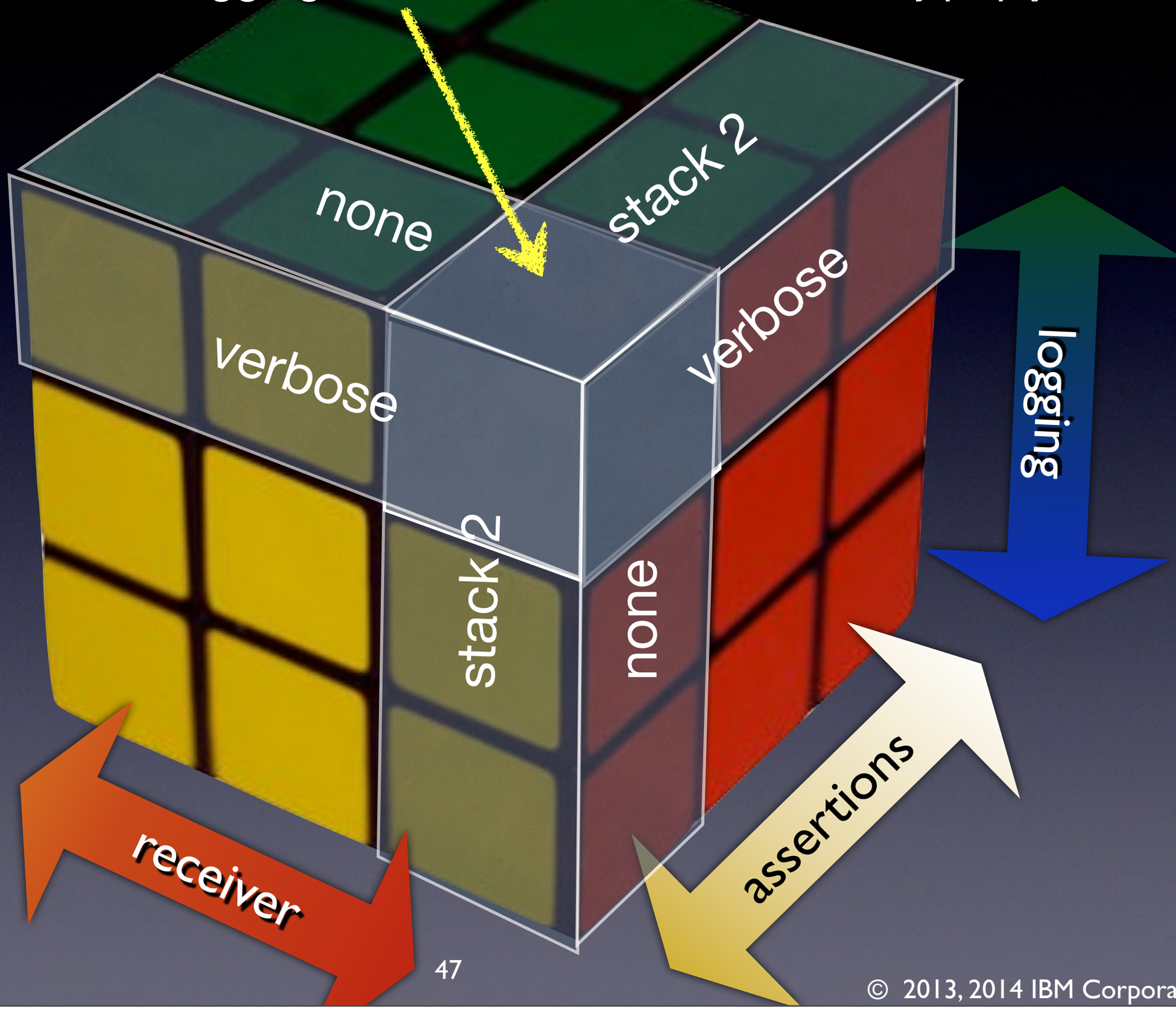





© 2013, 2014 IBM Corporation

# Message is sent to a place in the space: its *context*

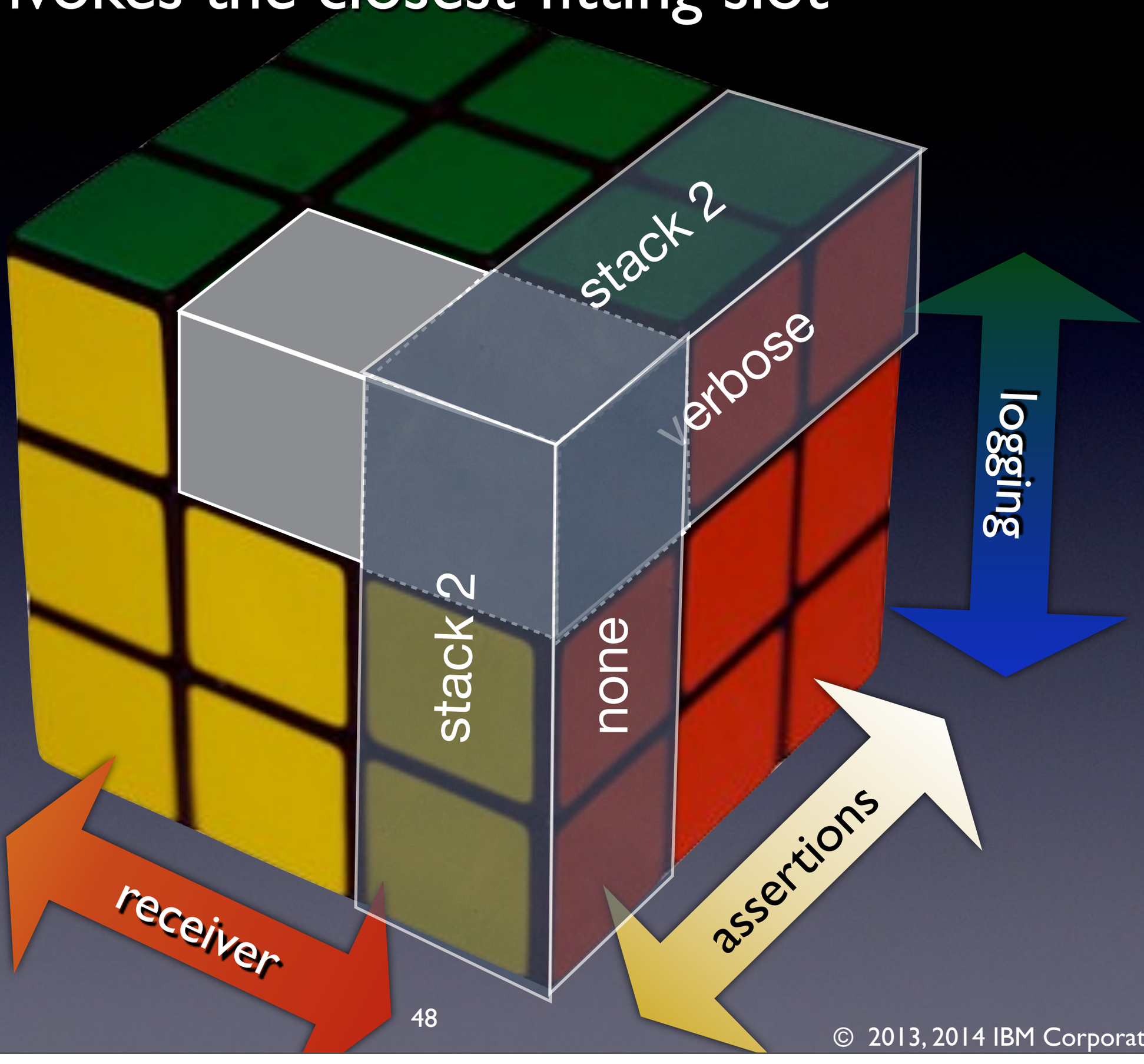
{assertions=>none; logging=>verbose; receiver=>stack2}.pop()






© 2013, 2014 IBM Corporation

# Message invokes the closest-fitting slot






© 2013, 2014 IBM Corporation

# Summary: Multidimensional Spatial Model

- Space (or sea) of slots: state & behavior
- Each slot occupies a volume
- Each message is sent to a location
- Runs the most specific slot there



© 2013, 2014 IBM Corporation

# Contents

- What's the big idea?
- What's it good for?
- How real is it?
- How does it work?
- Are you guys through yet?

# Contents

- What's the big idea?
- What's it good for?
- How real is it?
- How does it work?
- Are you guys through yet?



© 2013, 2014 IBM Corporation

# Examples

- Using multiple dimensions and context-based dispatch for:
  - Hiding slots and protecting invariants
  - Evolution
- Towards a Korz IDE



© 2013, 2014 IBM Corporation

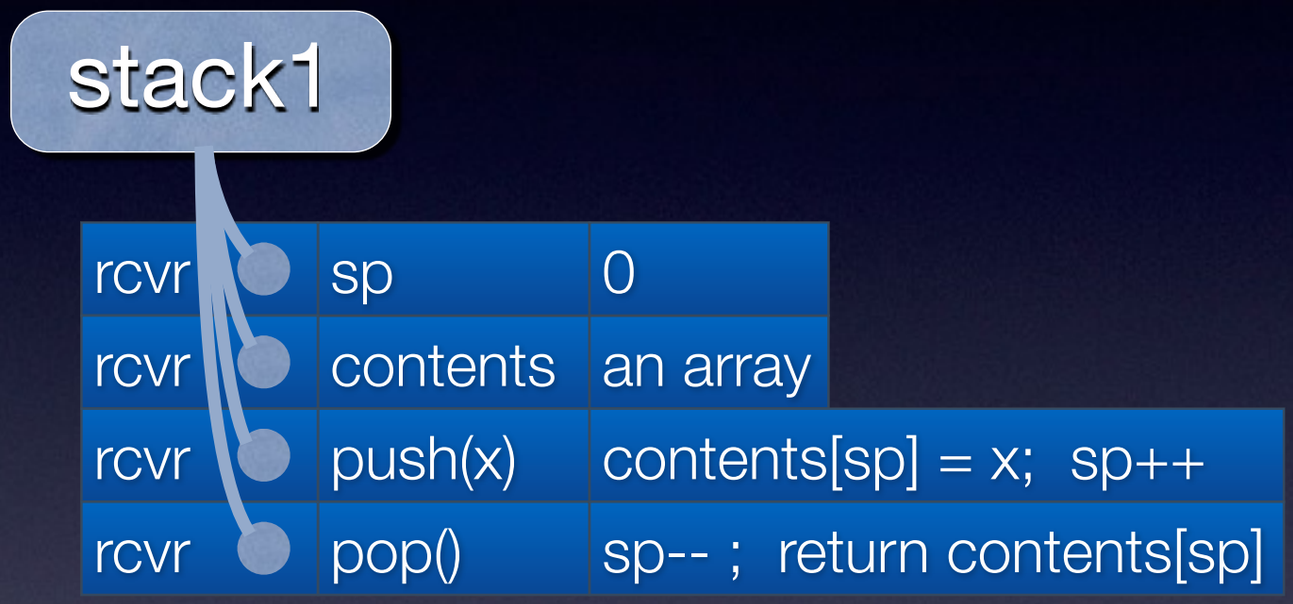
# Examples

- Using multiple dimensions and context-based dispatch for:
  - Hiding slots and protecting invariants
  - Evolution
- Towards a Korz IDE



© 2013, 2014 IBM Corporation

# Simple Korz Stack, No Inheritance

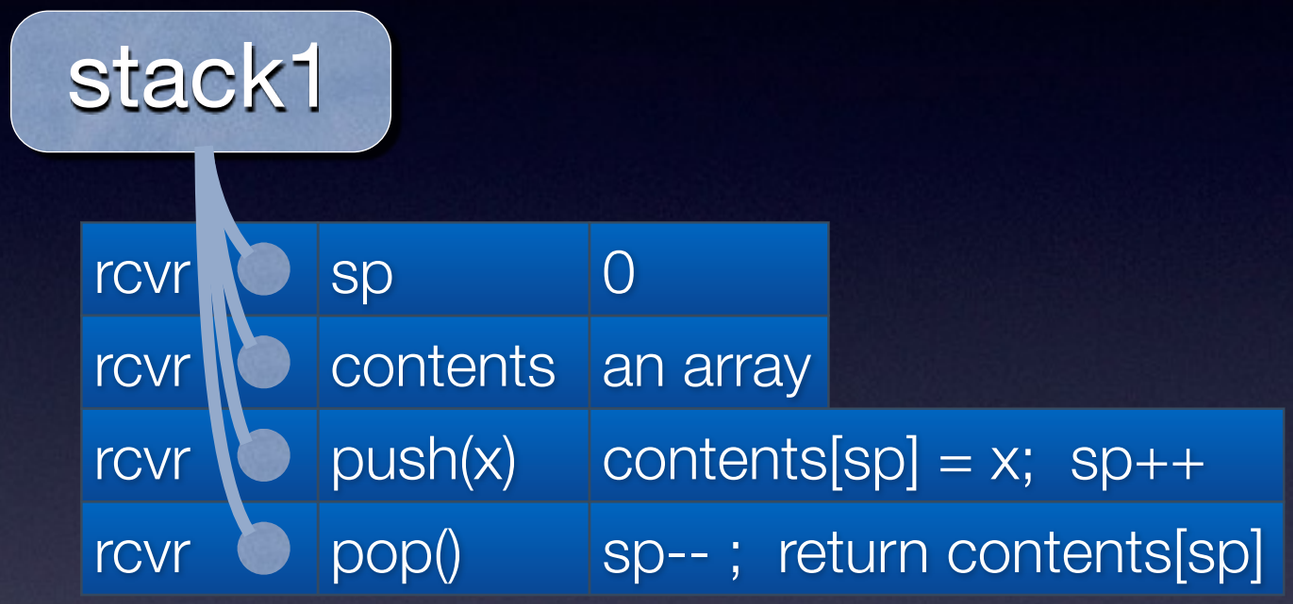



```
stack1.push(2)
stack1.push(3)
stack1.pop() + stack1.pop()
```



© 2013, 2014 IBM Corporation

# Any code can access or change representation slots!




```
stack1.contents[0] = ‘haha’

stack1.sp = 42
```



© 2013, 2014 IBM Corporation

# Protecting the Integrity of the Stack

- Many languages use visibility modifiers
  - Public, private, ...
- Rather than adding them to Korz:
  - What exactly do we want to protect?
  - What is the best way within our paradigm?



© 2013, 2014 IBM Corporation

# The Stack Invariant

size of stack is sp ≥ 0

stack elements are contents[sp-1], ..., contents[0]

stack1

| | | |
|---|---|---|
| rcvr | sp | 0 |
| rcvr | contents | an array |
| rcvr | push(x) | contents[sp] = x;  sp++ |
| rcvr | pop() | sp-- ;  return contents[sp] |



© 2013, 2014 IBM Corporation

# The Stack Invariant

size of stack is sp ≥ 0

stack elements are contents[sp-1], ..., contents[0]




slots that preserve the invariant

slots that can break the invariant



© 2013, 2014 IBM Corporation

# Reify the Stack Invariant

stackInvariant

stack1

| | | |
|---|---|---|
| rcvr | push(x) | contents[sp] = x;  sp++ |
| rcvr | pop() | sp-- ;  return contents[sp] |

slots that preserve the invariant

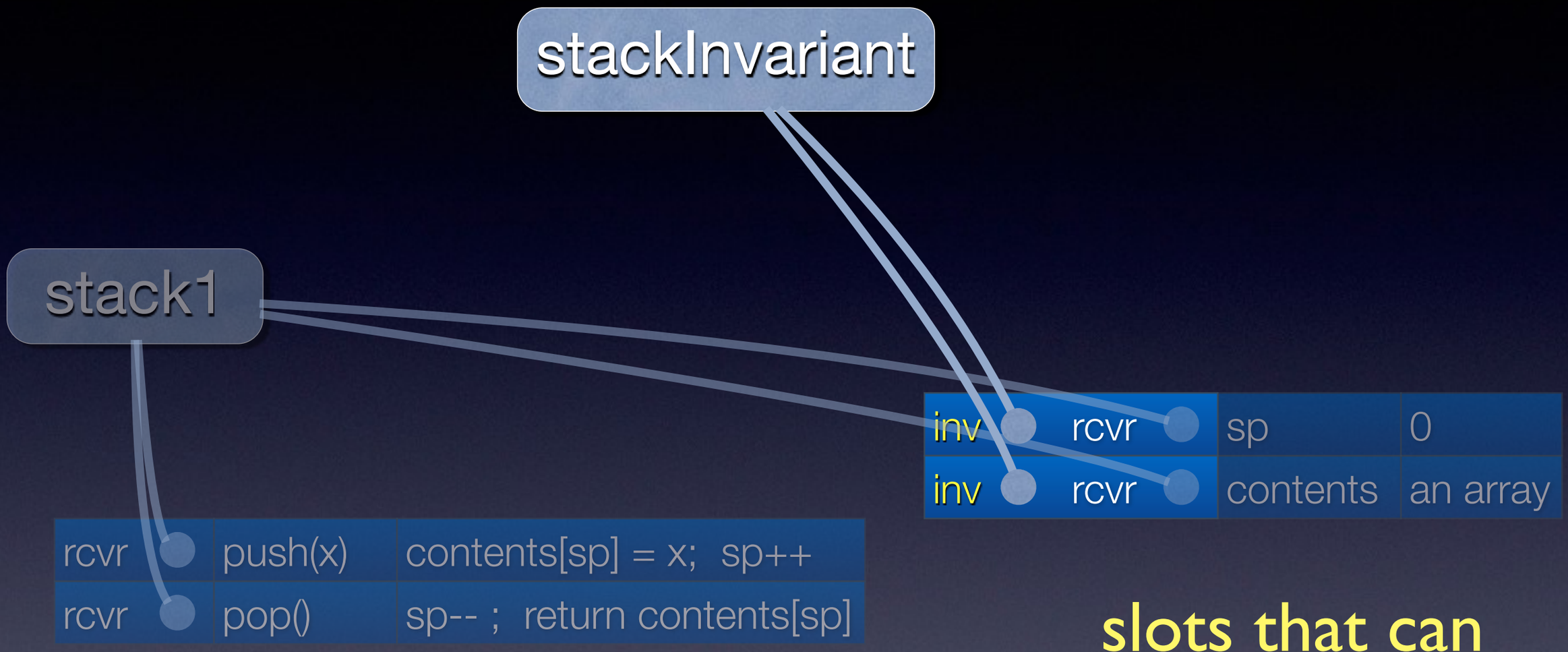
| | | |
|---|---|---|
| rcvr | sp | 0 |
| rcvr | contents | an array |

slots that can break the invariant



© 2013, 2014 IBM Corporation

# Guard ‘Breaking’ Slots



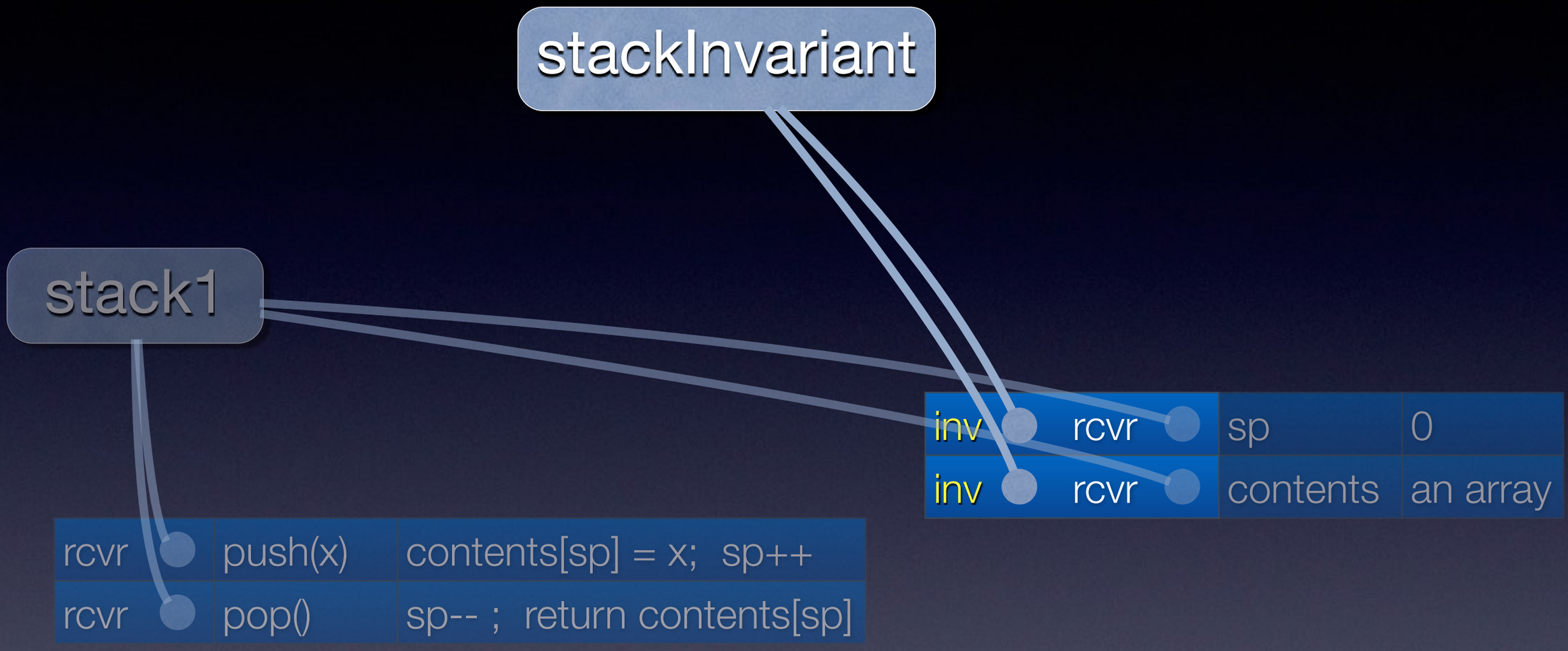

slots that preserve the invariant

slots that can break the invariant



© 2013, 2014 IBM Corporation

# Access is Restricted




To access slots that can break the invariant,

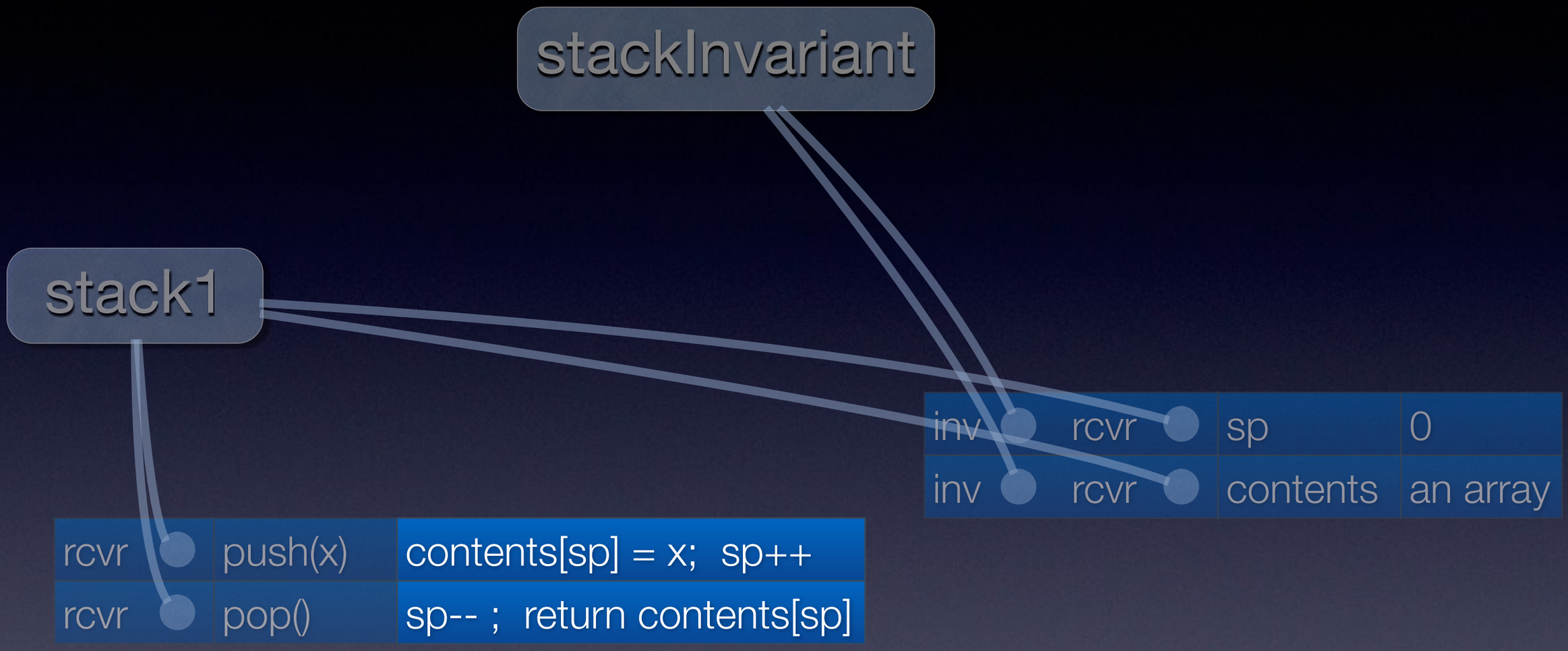
inv => stackInvariant

must be in the message context



© 2013, 2014 IBM Corporation

# Access is Restricted




To access slots that can break the invariant,

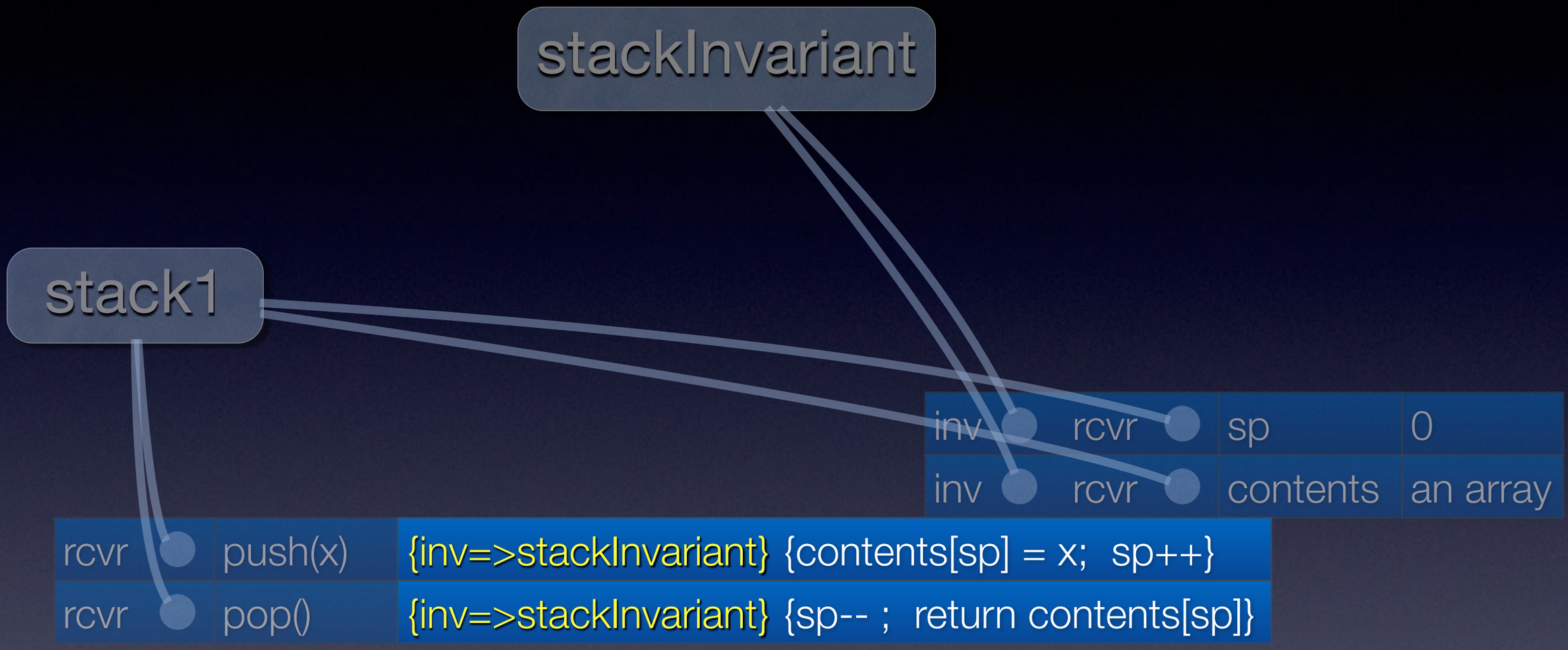
inv => stackInvariant

must be in the message context



© 2013, 2014 IBM Corporation

# Enabling Access




To access slots that can break the invariant,

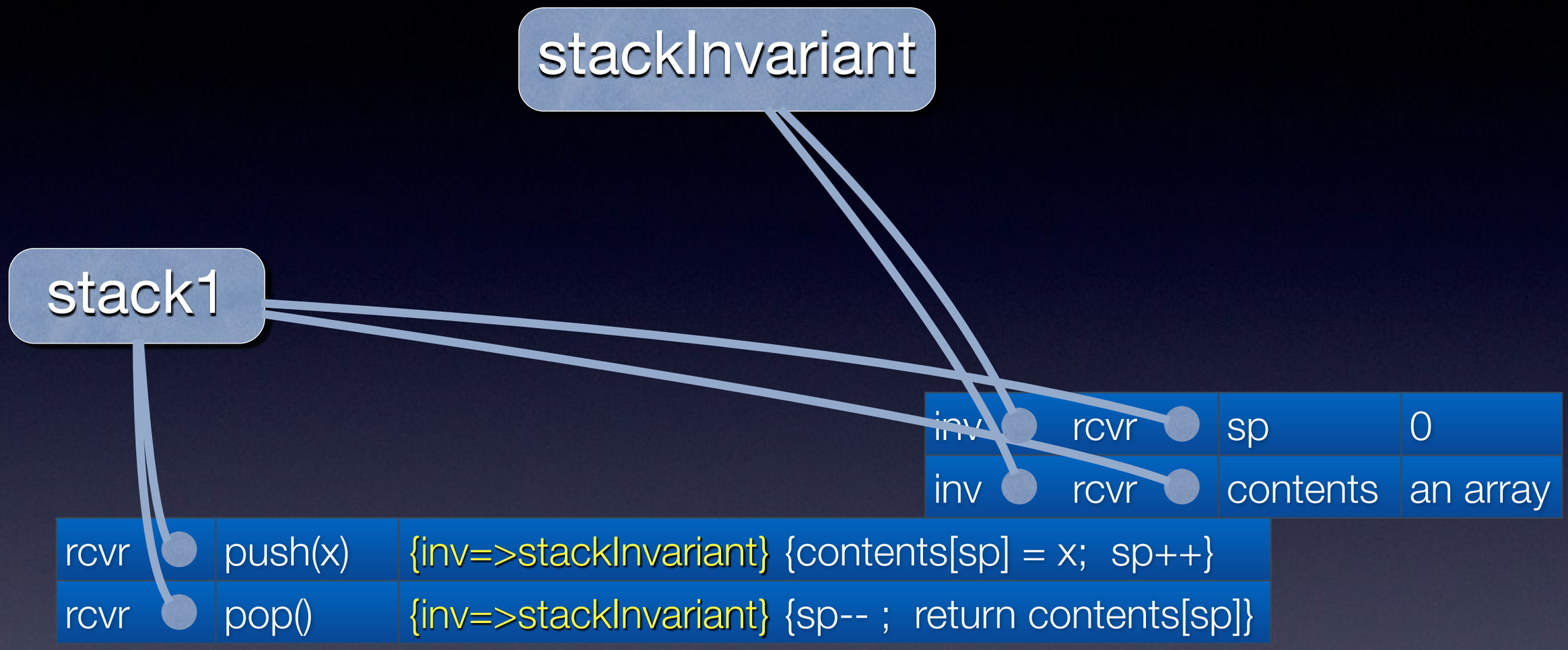
**inv => stackInvariant**

must be in the message context



© 2013, 2014 IBM Corporation

# Examples




```
stack1.push(2)
stack1.push(3)
stack1.pop() + stack1.pop()
```

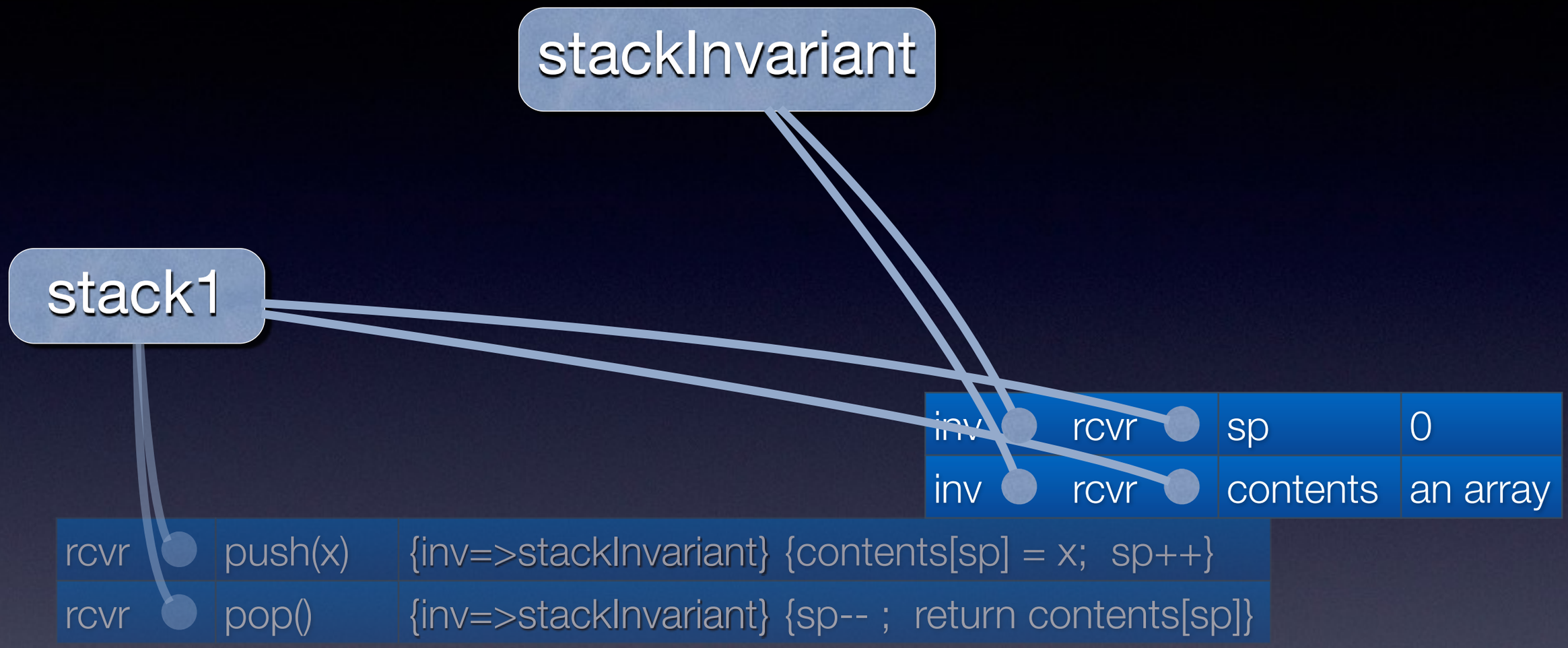
```
stack1.contents[0] = ‘haha’

stack1.sp = 42
```



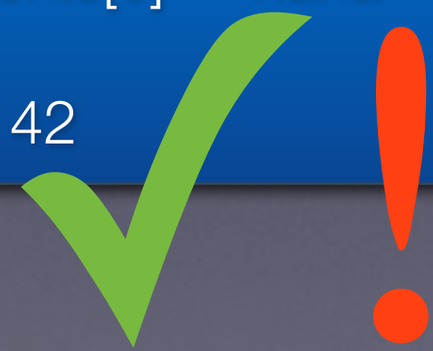
© 2013, 2014 IBM Corporation

# But ...




{inv=>stackInvariant} stack1.contents[0] = ‘haha’

{inv=>stackInvariant} stack1.sp = 42



Must restrict access to stackInvariant



© 2013, 2014 IBM Corporation

# Where are we?

Slots that can break the invariant are accessible
if and only if the message context contains
inv => stackInvariant

Access to these slots can be restricted by
controlling access to the stackInvariant object

*Object capability* as defined by Mark Miller*

Korz multi-dimensional dispatch can
control slot access based on capability access

* Mark S. Miller. *Robust Composition: Towards a Unified Approach to Access Control and Concurrency Control.* PhD thesis, Johns Hopkins University, May 2006.



© 2013, 2014 IBM Corporation

# Hide with Lexical Scoping

```
obj stack1 = {}; // a new object

{




}
```



© 2013, 2014 IBM Corporation

# Hide with Lexical Scoping

Definition of *stackInvariant*

Visibility of *stackInvariant* is restricted to this scope

```
obj stack1 = {}; // a new object

{
    obj stackInvariant = {}; // a new object

    { inv=>stackInvariant, rcvr=>stack1 ?  sp = ... }
    { inv=>stackInvariant, rcvr=>stack1 ? contents = ... }

    { rcvr=>stack1 ? push() = { {inv=>stackInvariant} ... }
    { rcvr=>stack1 ? pop()  = { {inv=>stackInvariant} ... }
}
```



© 2013, 2014 IBM Corporation

# Hide with Lexical Scoping

```
obj stack1 = {}; // a new object

{
    obj stackInvariant = {}; // a new object

    { inv=>stackInvariant, rcvr=>stack1 ?  sp = ... }
    { inv=>stackInvariant, rcvr=>stack1 ? contents = ... }

    { rcvr=>stack1 ? push() = { {inv=>stackInvariant} ... }
    { rcvr=>stack1 ? pop()  = { {inv=>stackInvariant} ... }
}

stack1.contents[1] = 'haha'

{ inv=>stackInvariant } stack1.contents[1] = 'haha'
```

Visibility of *stackInvariant* is restricted to this scope

message not understood



© 2013, 2014 IBM Corporation

# Summary

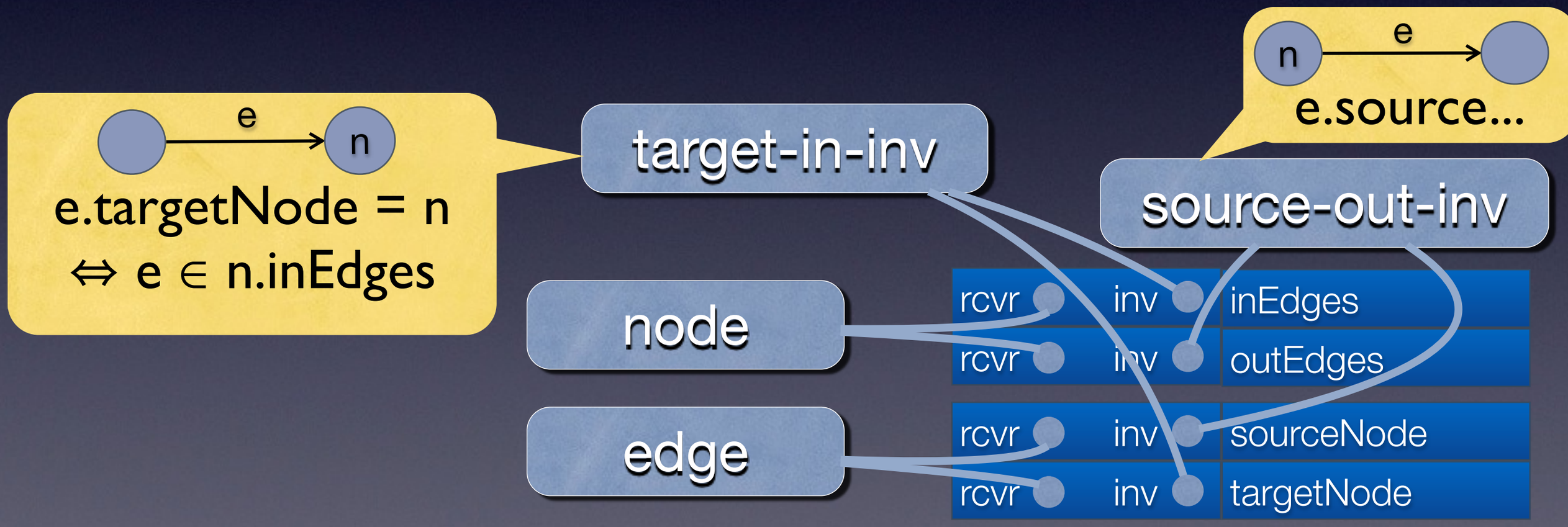
- Capability object reifies invariant
  - Visibility restricted by static nesting
- Breaking slots protected using dimension
  - Capability object required as coordinate in message context
- Different subjective views
  - Access controlled by capability object



© 2013, 2014 IBM Corporation

# Multiple, Crosscutting Invariants

- Multiple invariants need to be protected
- Can cut  across prototypes or classes

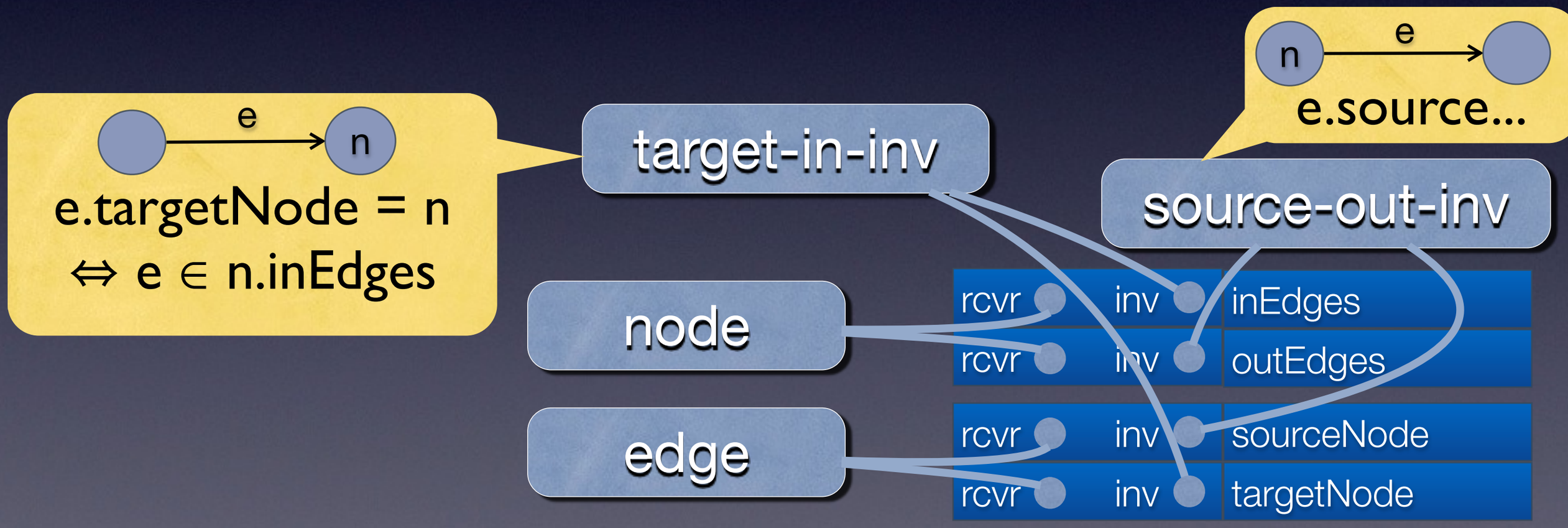





© 2013, 2014 IBM Corporation

# Multiple, Crosscutting Invariants

- Multiple invariants need to be protected
- Can cut  across prototypes or classes




Will lexical scoping run out of gas?



© 2013, 2014 IBM Corporation

# Examples

- Using multiple dimensions and context-based dispatch for:
  - Hiding slots and protecting invariants
  - Evolution
- Towards a Korz IDE



© 2013, 2014 IBM Corporation

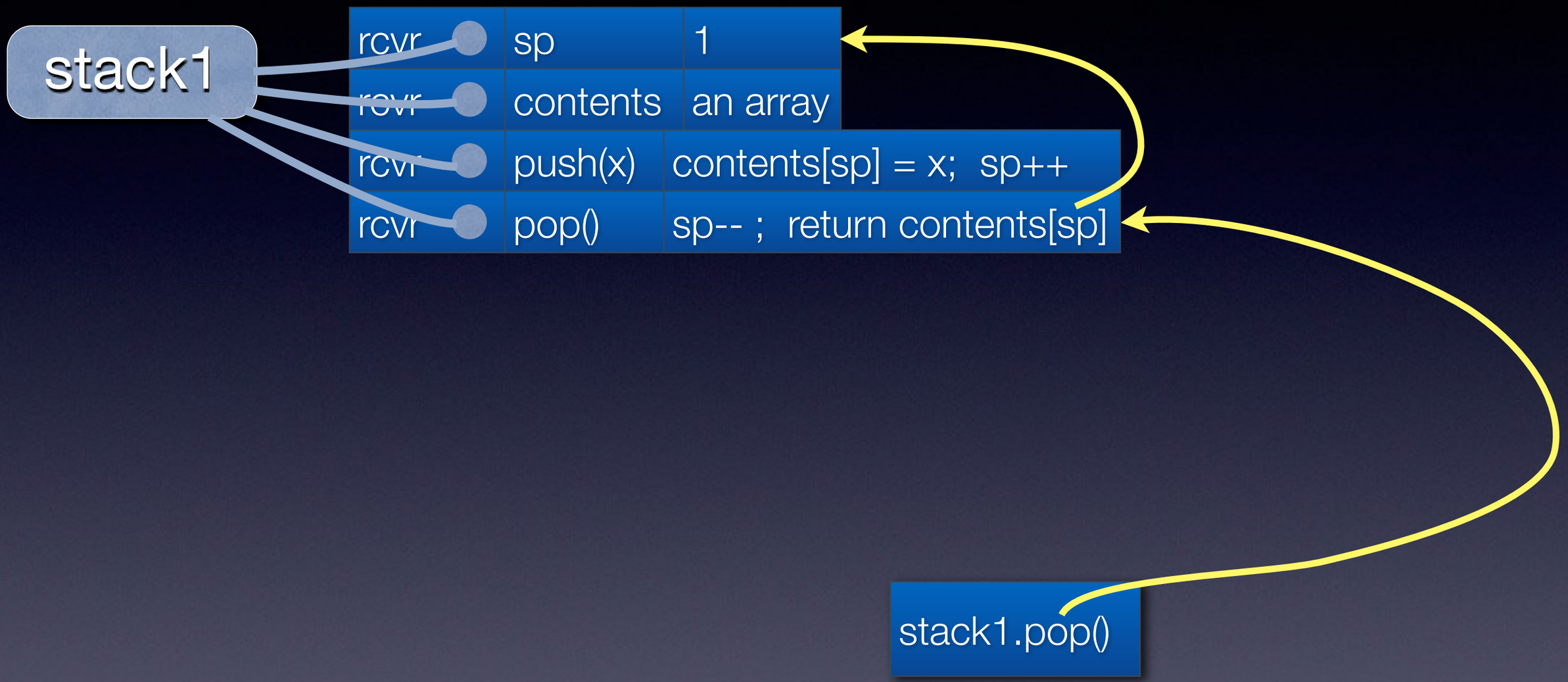
# Examples

- Using multiple dimensions and context-based dispatch for:
  - Hiding slots and protecting invariants
  - Evolution
- Towards a Korz IDE



© 2013, 2014 IBM Corporation

# Simple Korz Stack






© 2013, 2014 IBM Corporation

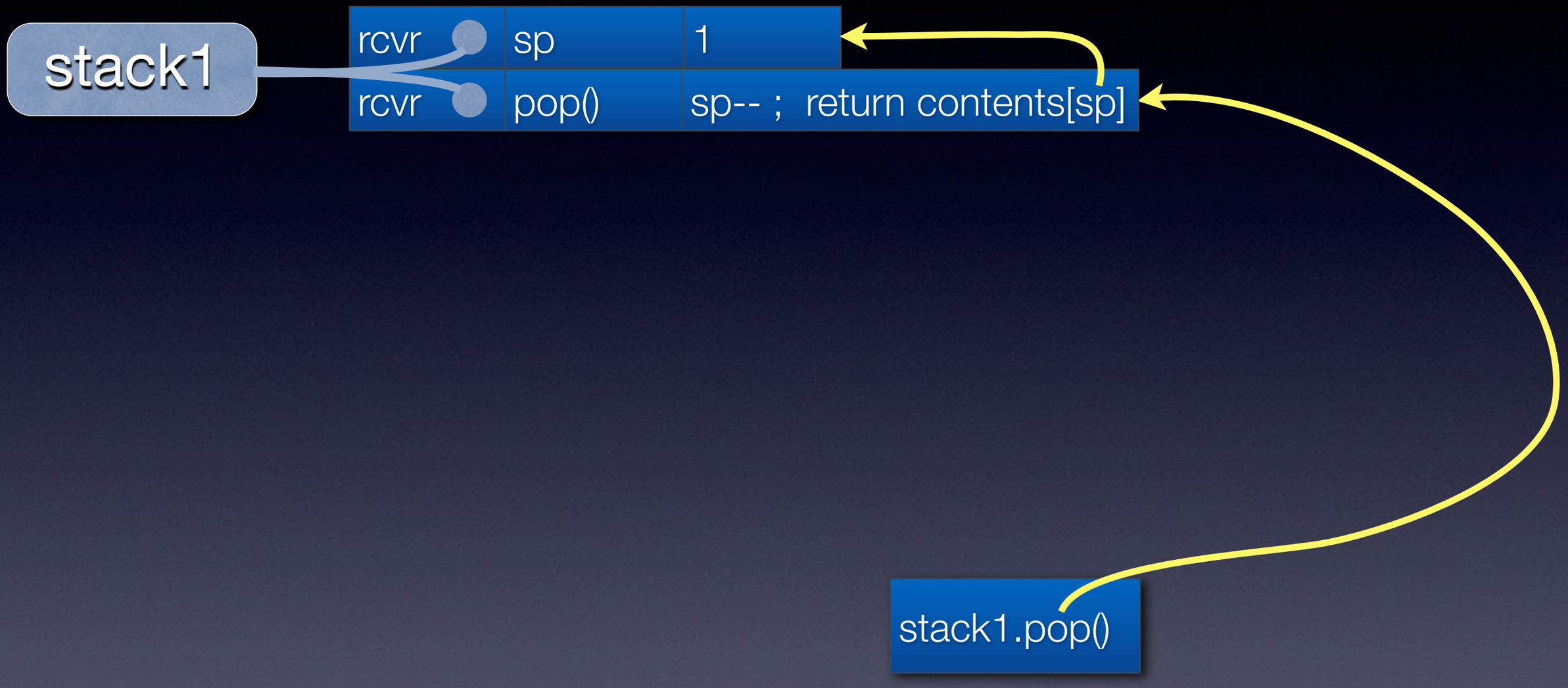



stack1.pop()



© 2013, 2014 IBM Corporation

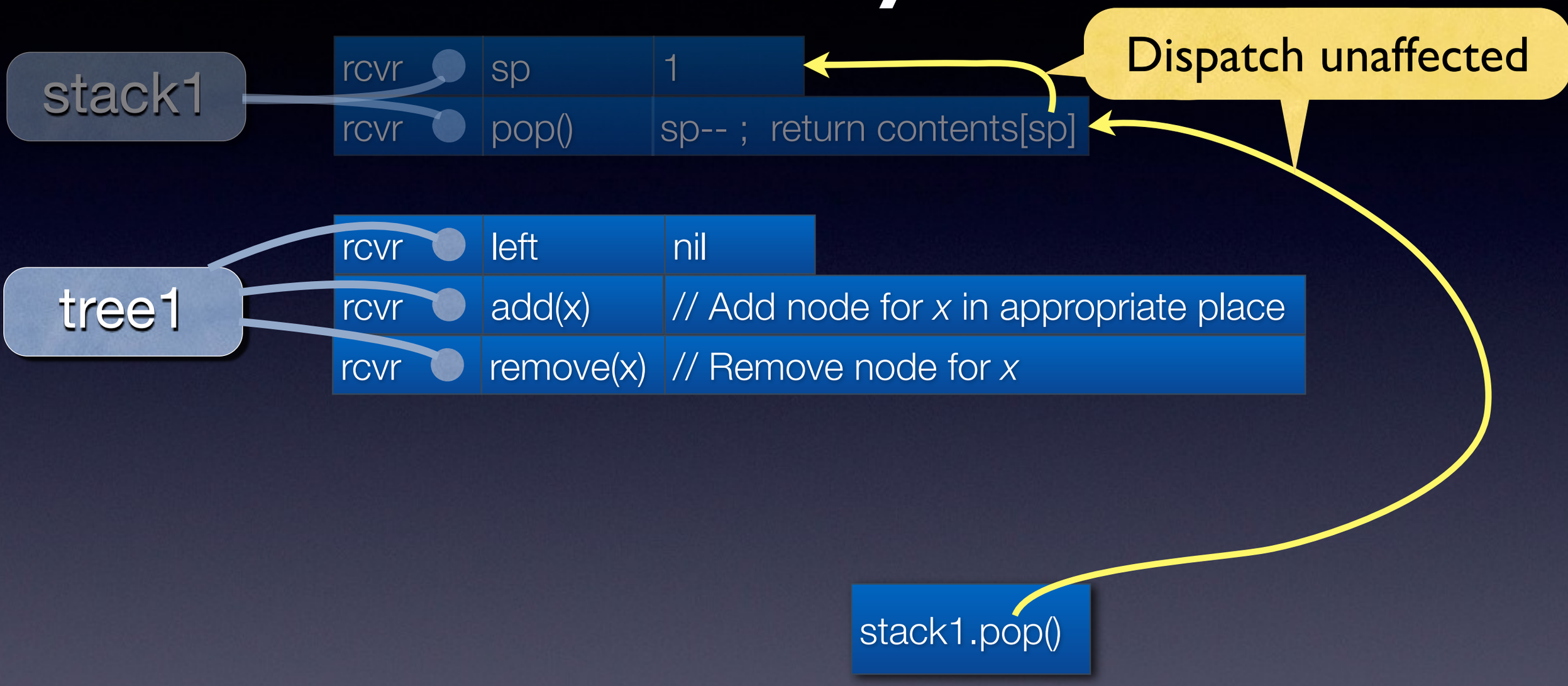
# Subset of Slots






© 2013, 2014 IBM Corporation

# Add Binary Tree

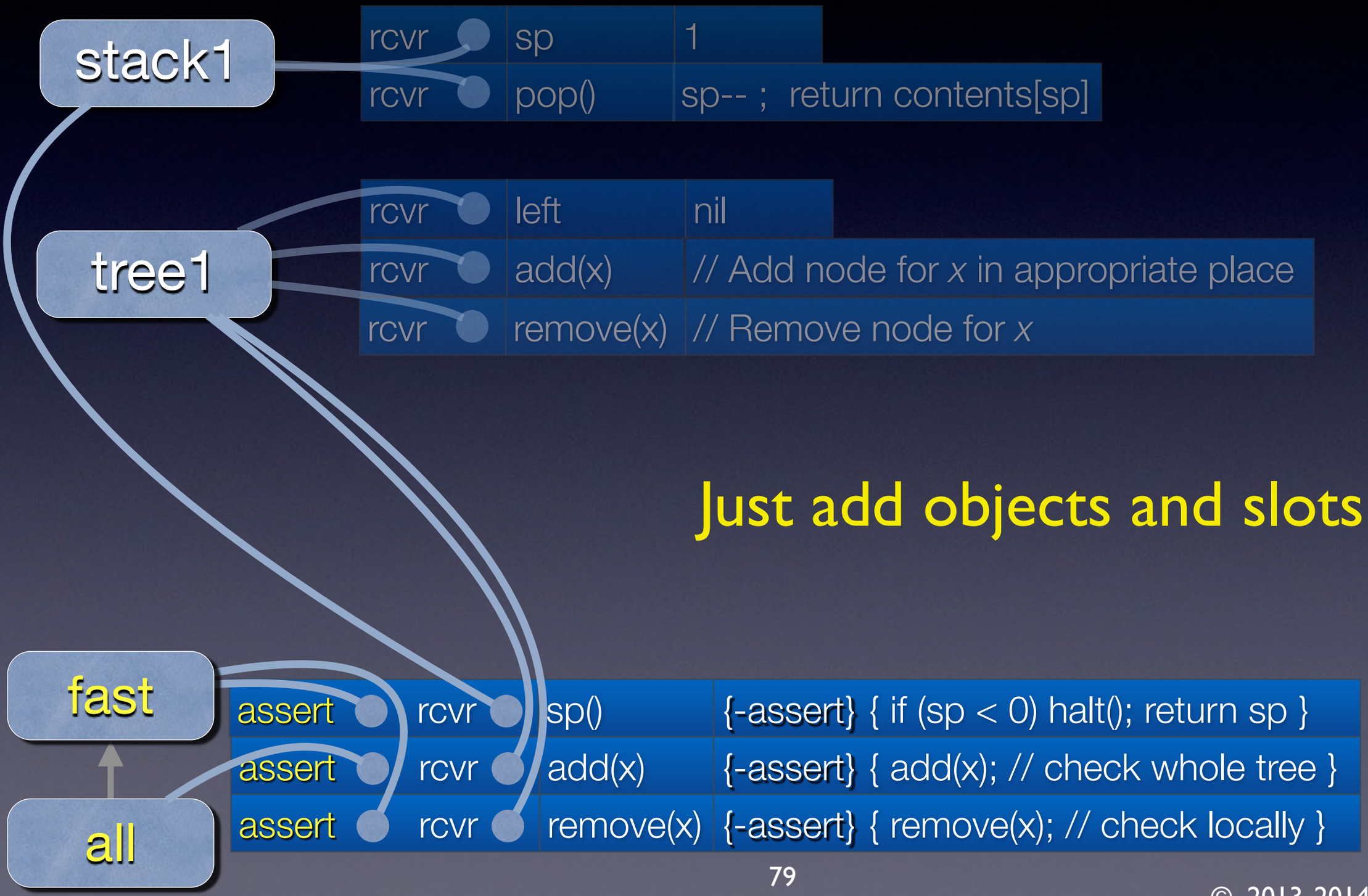



Just add objects and slots!



© 2013, 2014 IBM Corporation

# Add Assertions

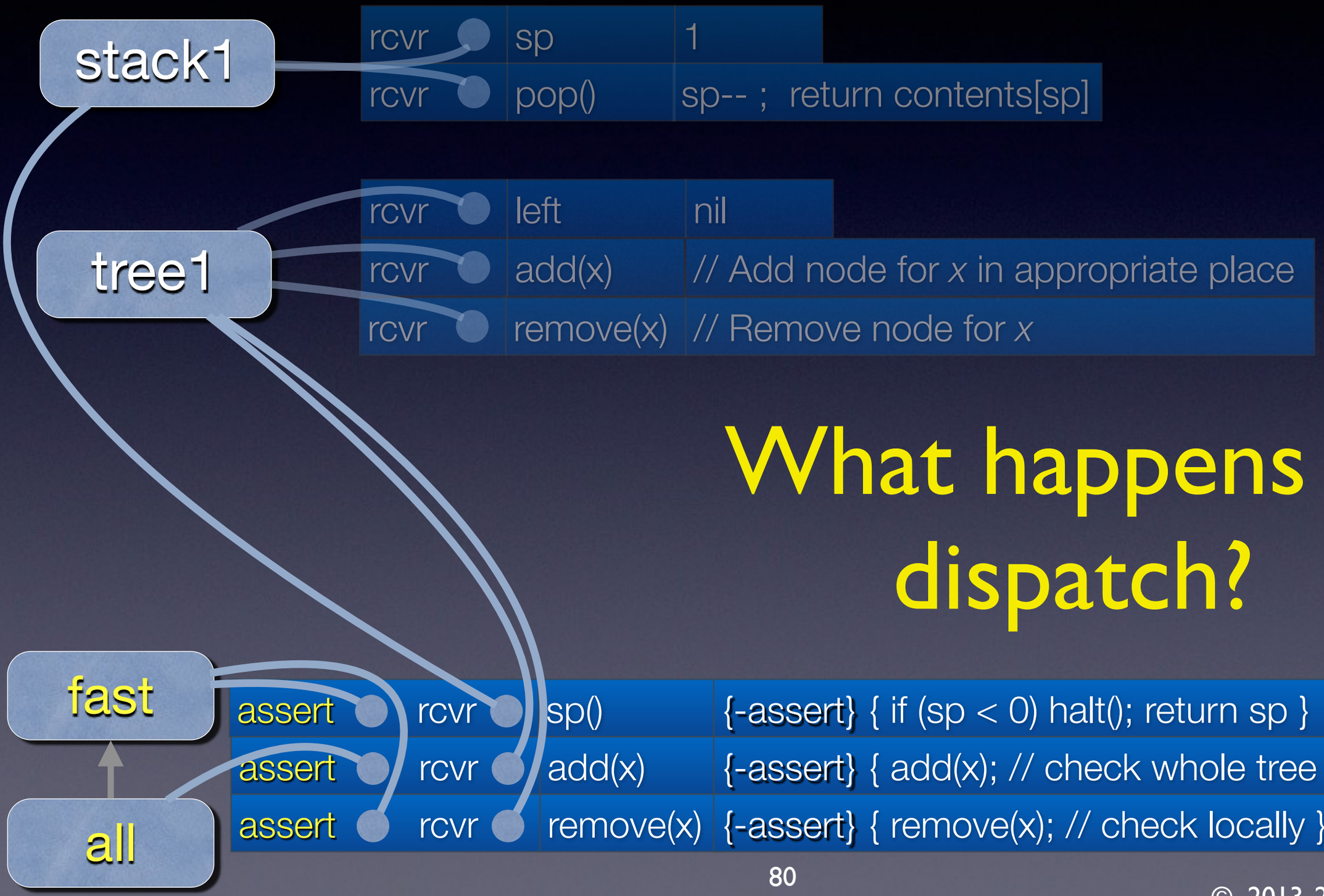



Just add objects and slots!



© 2013, 2014 IBM Corporation

# Add Assertions

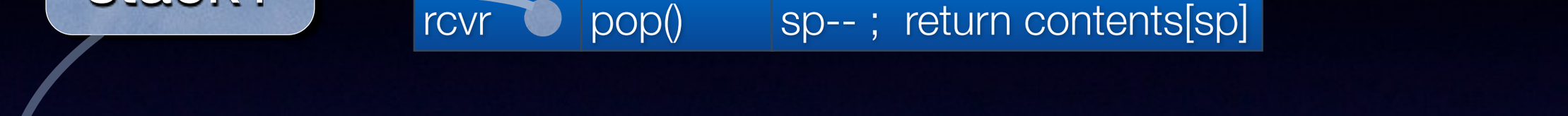



## What happens to dispatch?



© 2013, 2014 IBM Corporation

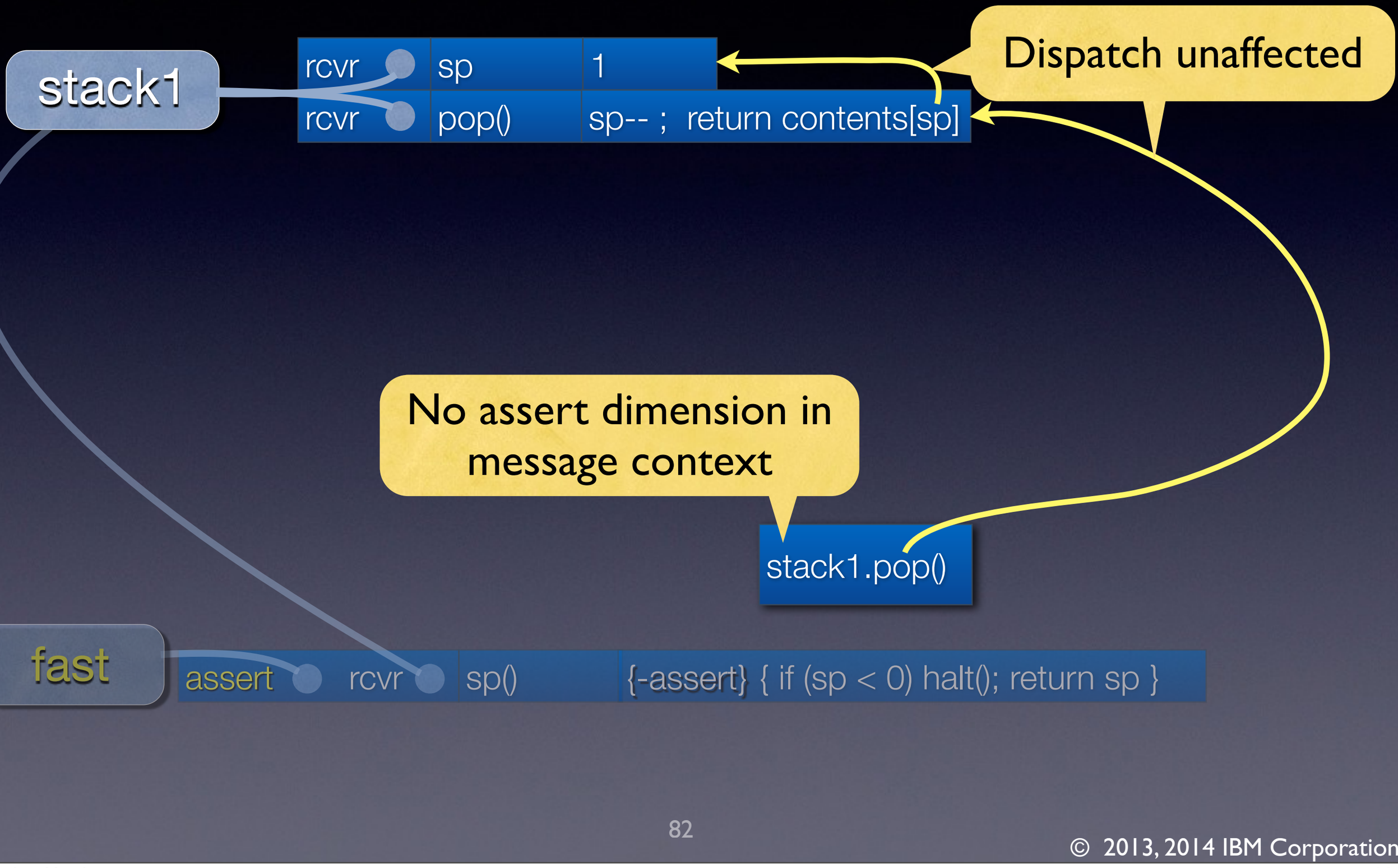





© 2013, 2014 IBM Corporation

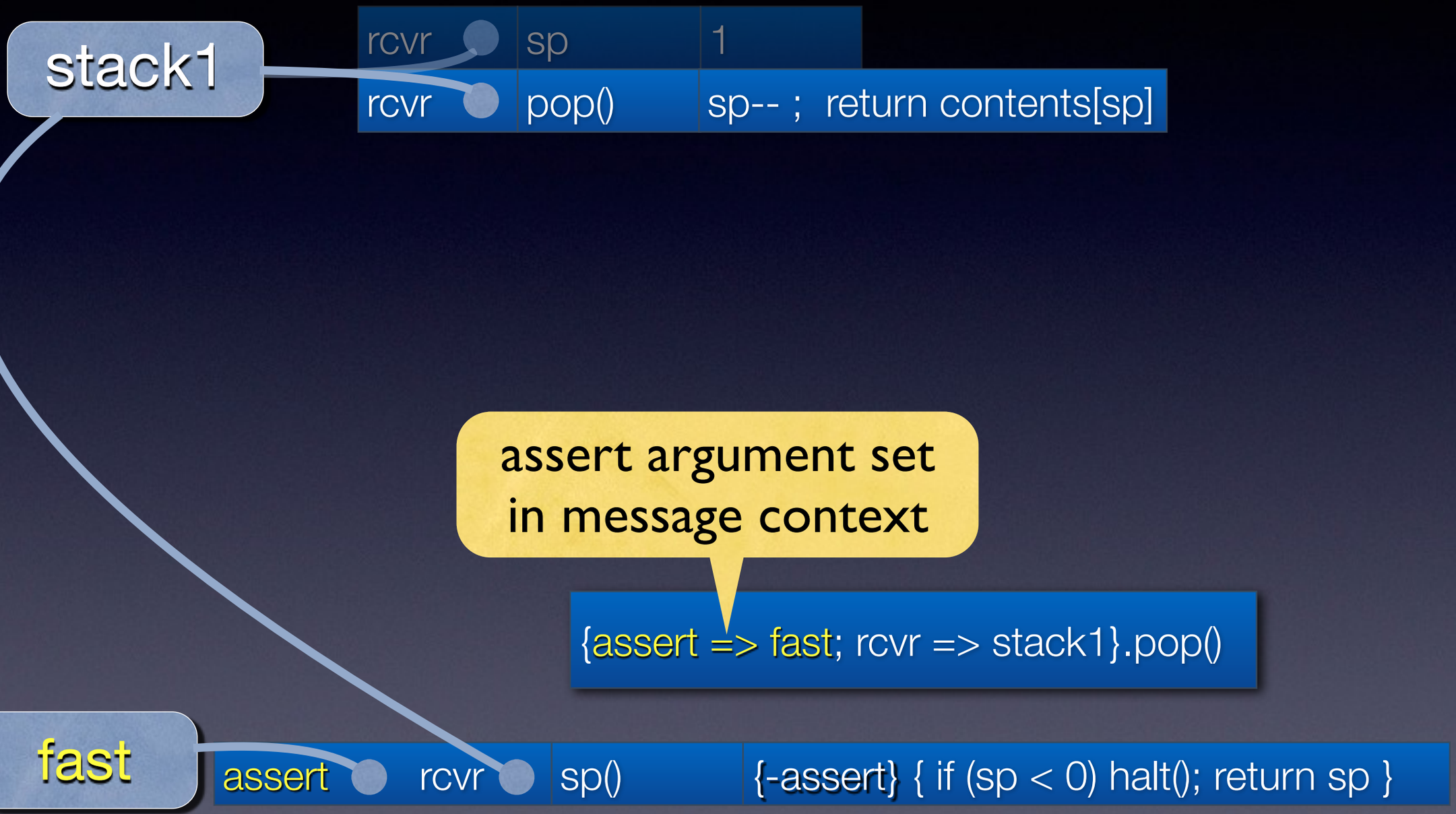
# Prior Code Unaffected






© 2013, 2014 IBM Corporation

# Using Assertions




© 2013, 2014 IBM Corporation

# Prior Slots Still Match

stack1

| rcvr | sp | 1 |
|---|---|---|
| rcvr | pop() | sp-- ; return contents[sp] |

Absence of assert dimension matches any coordinate

assert argument set in message context

pop matches original slot

```
{assert => fast; rcvr => stack1}.pop()
```

fast

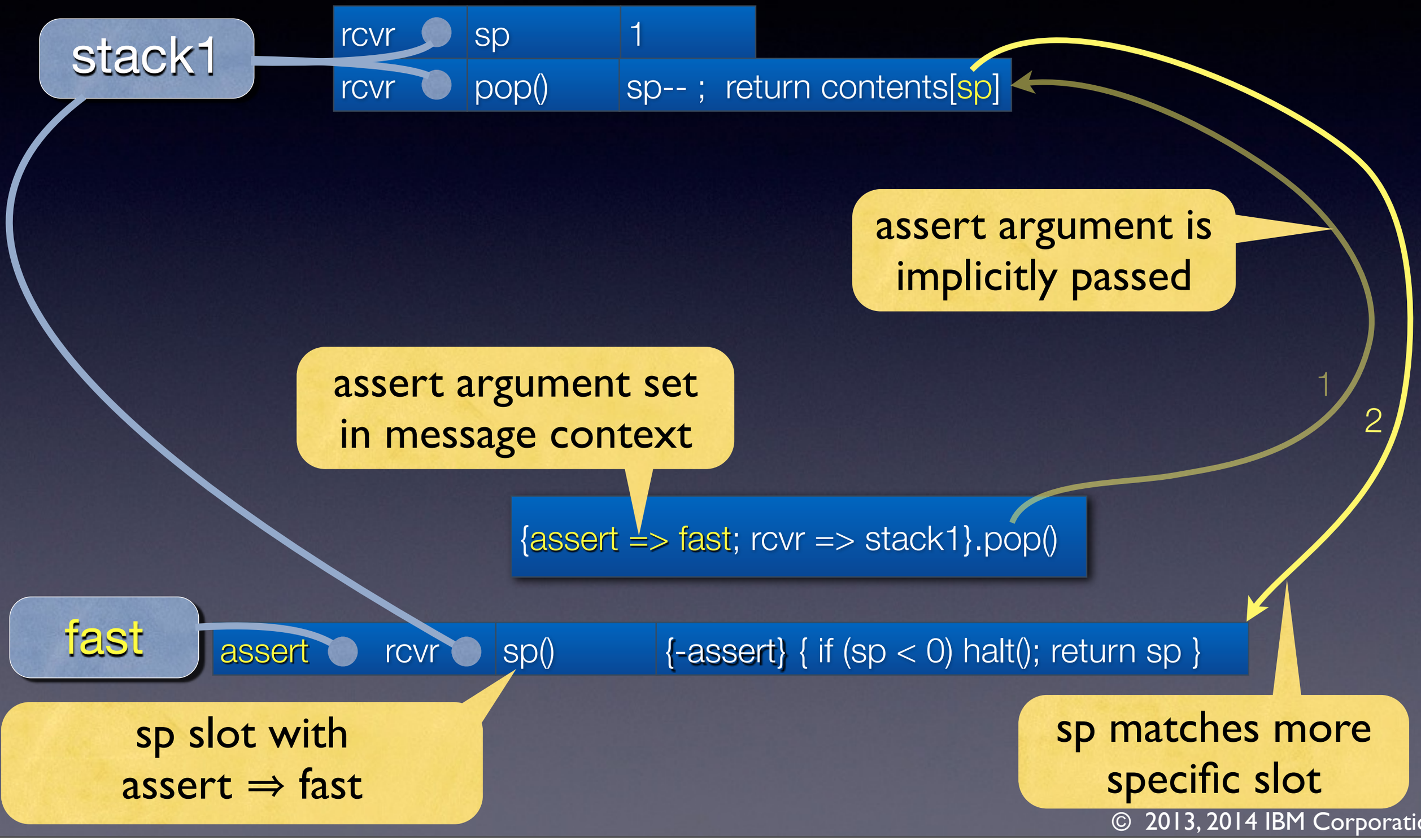
| assert | rcvr | sp() | {-assert} { if (sp < 0) halt(); return sp } |
|---|---|---|---|

No pop slot with assert ⇒ fast



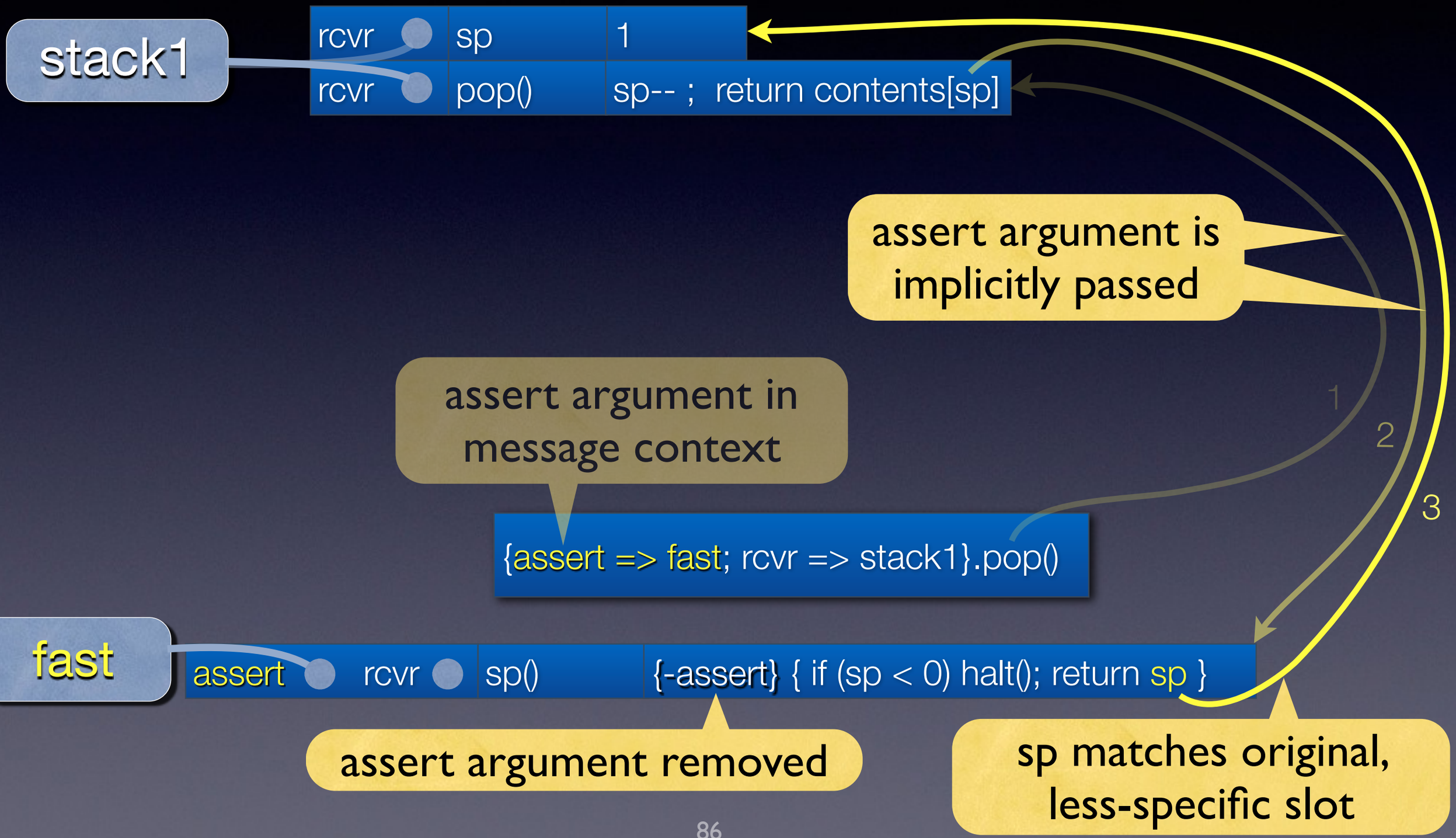
© 2013, 2014 IBM Corporation

# More-Specific Override




© 2013, 2014 IBM Corporation

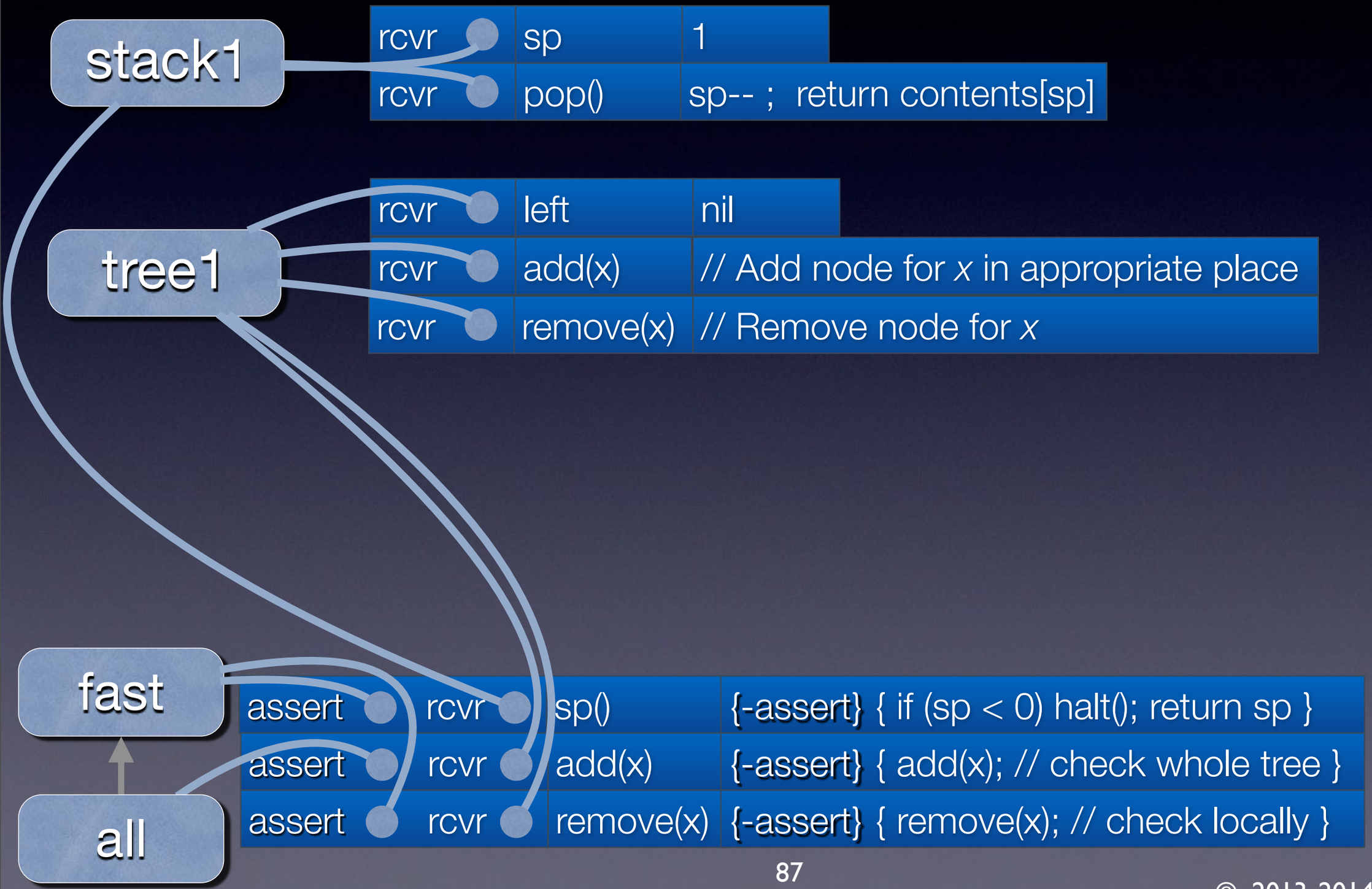
# Access to Less Specific






© 2013, 2014 IBM Corporation

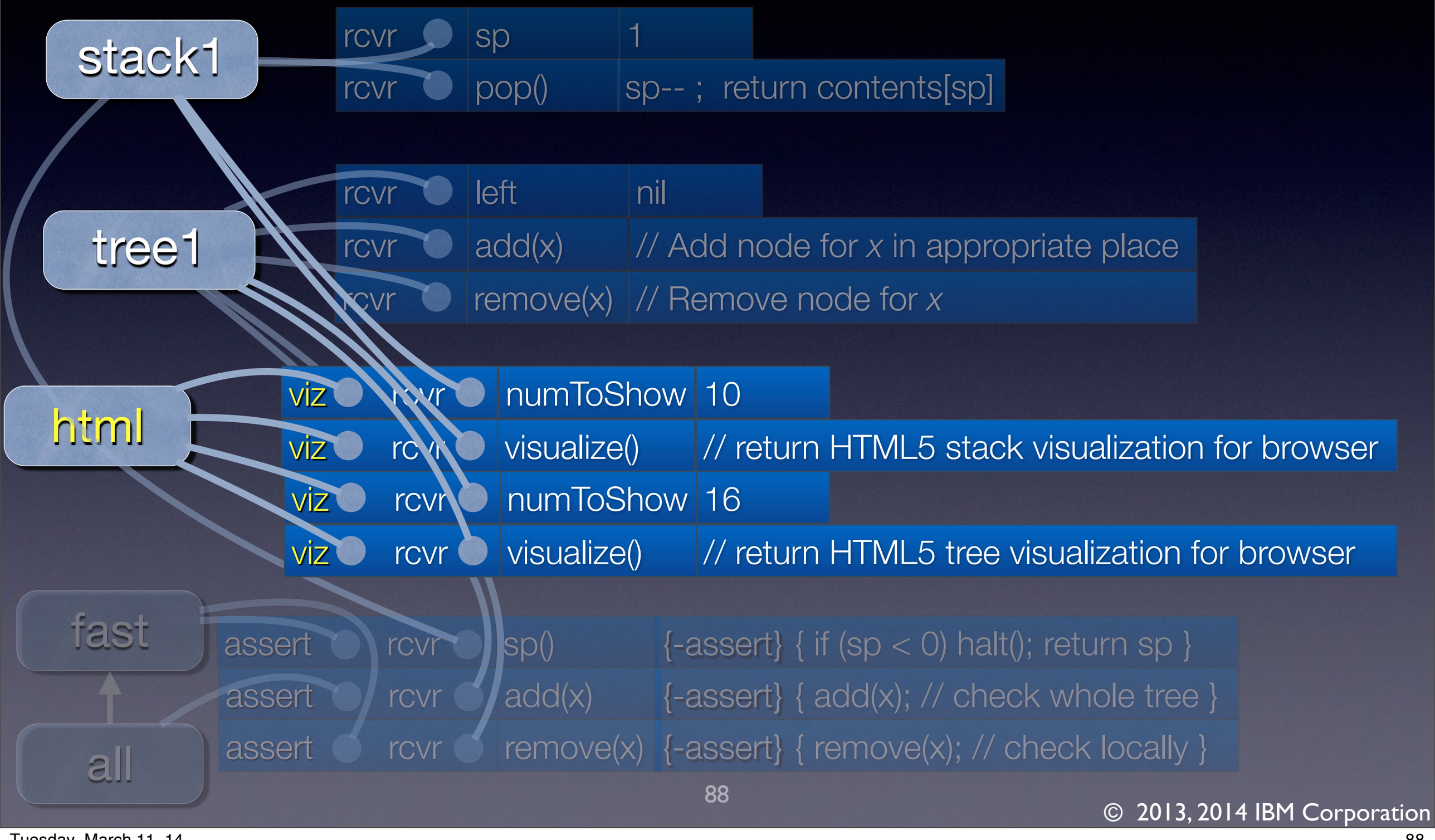
# Stack, Tree, Assertions






© 2013, 2014 IBM Corporation

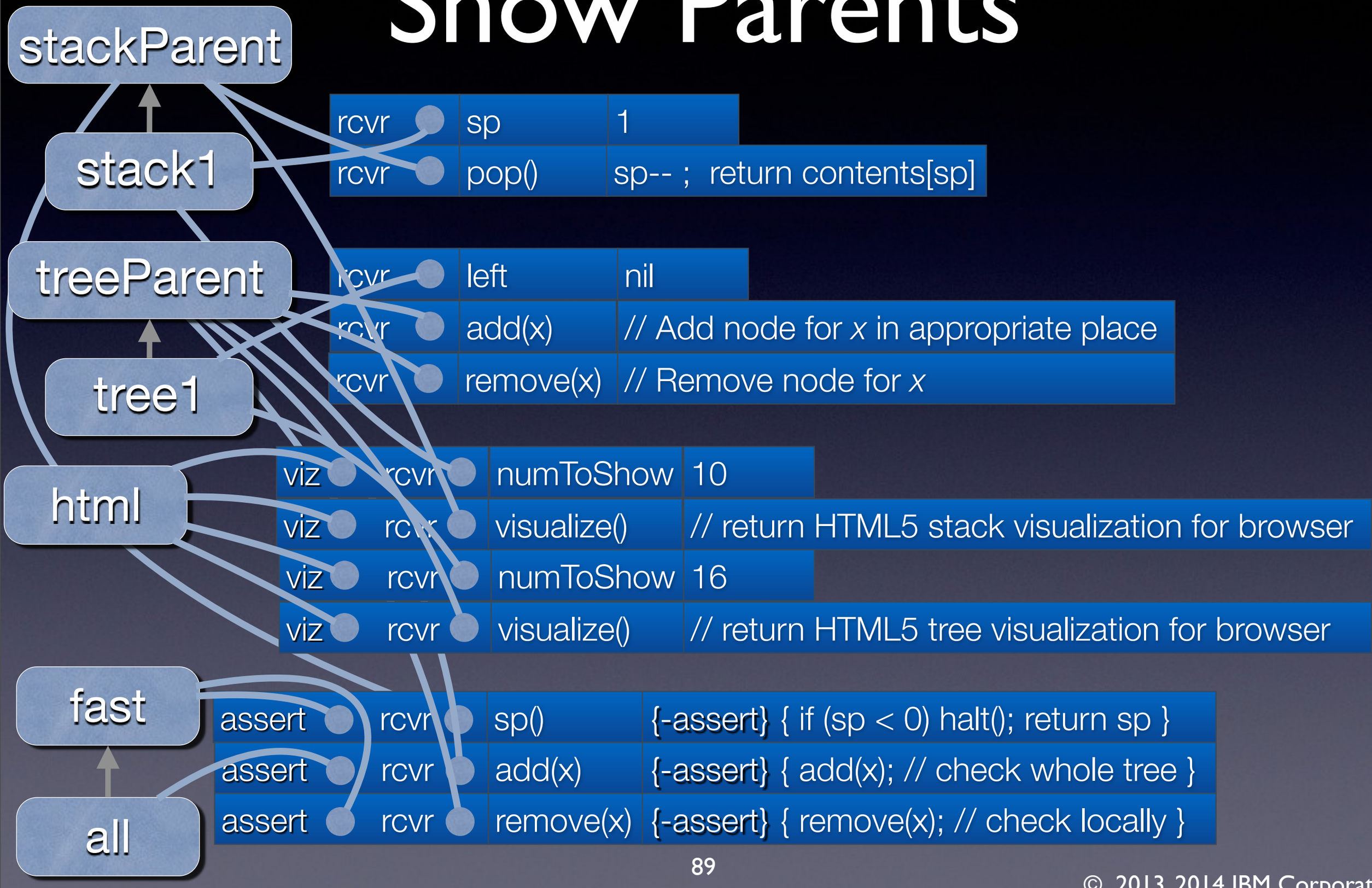
# Add HTML Visualization






© 2013, 2014 IBM Corporation

# Show Parents

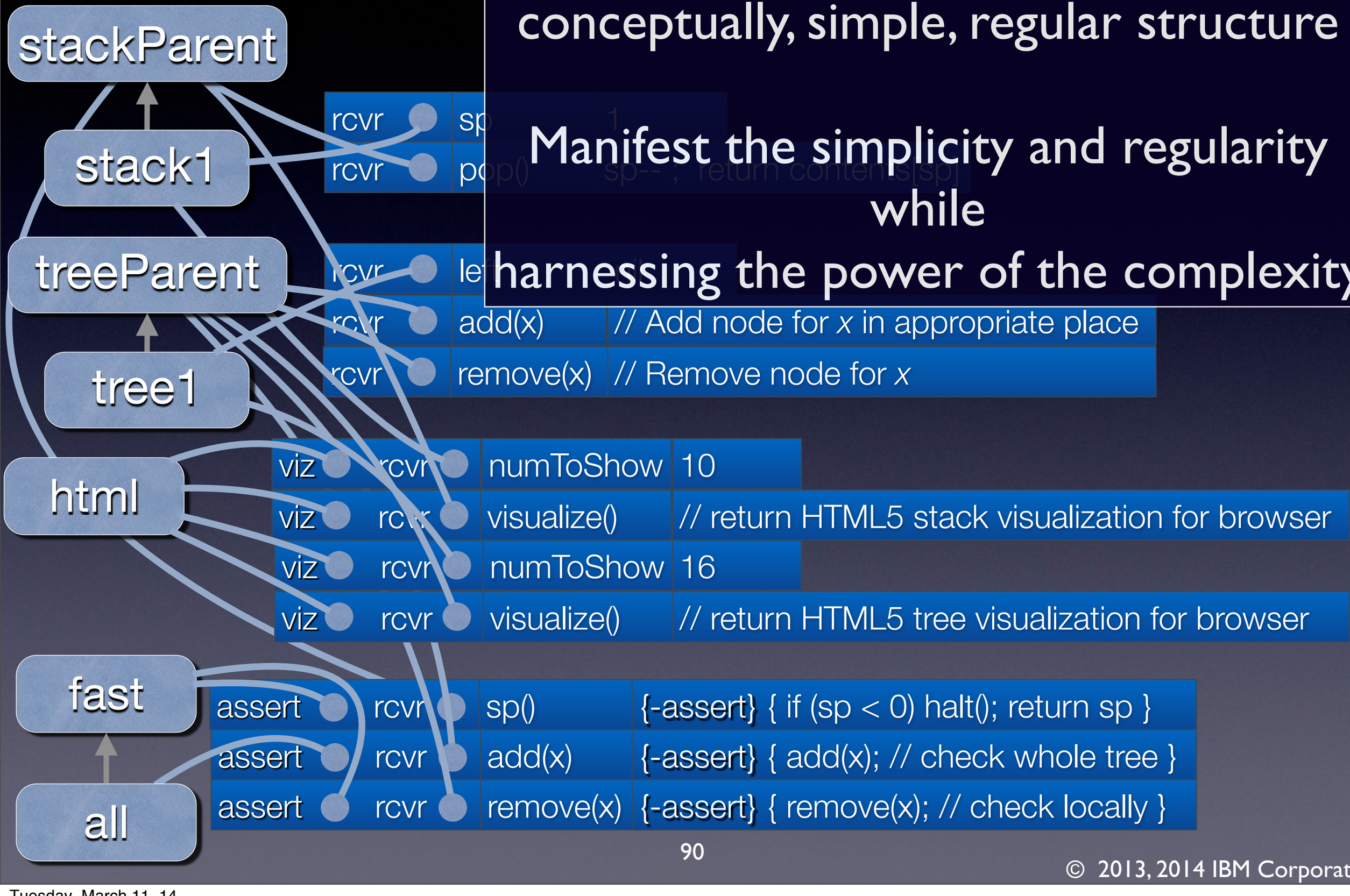





© 2013, 2014 IBM Corporation

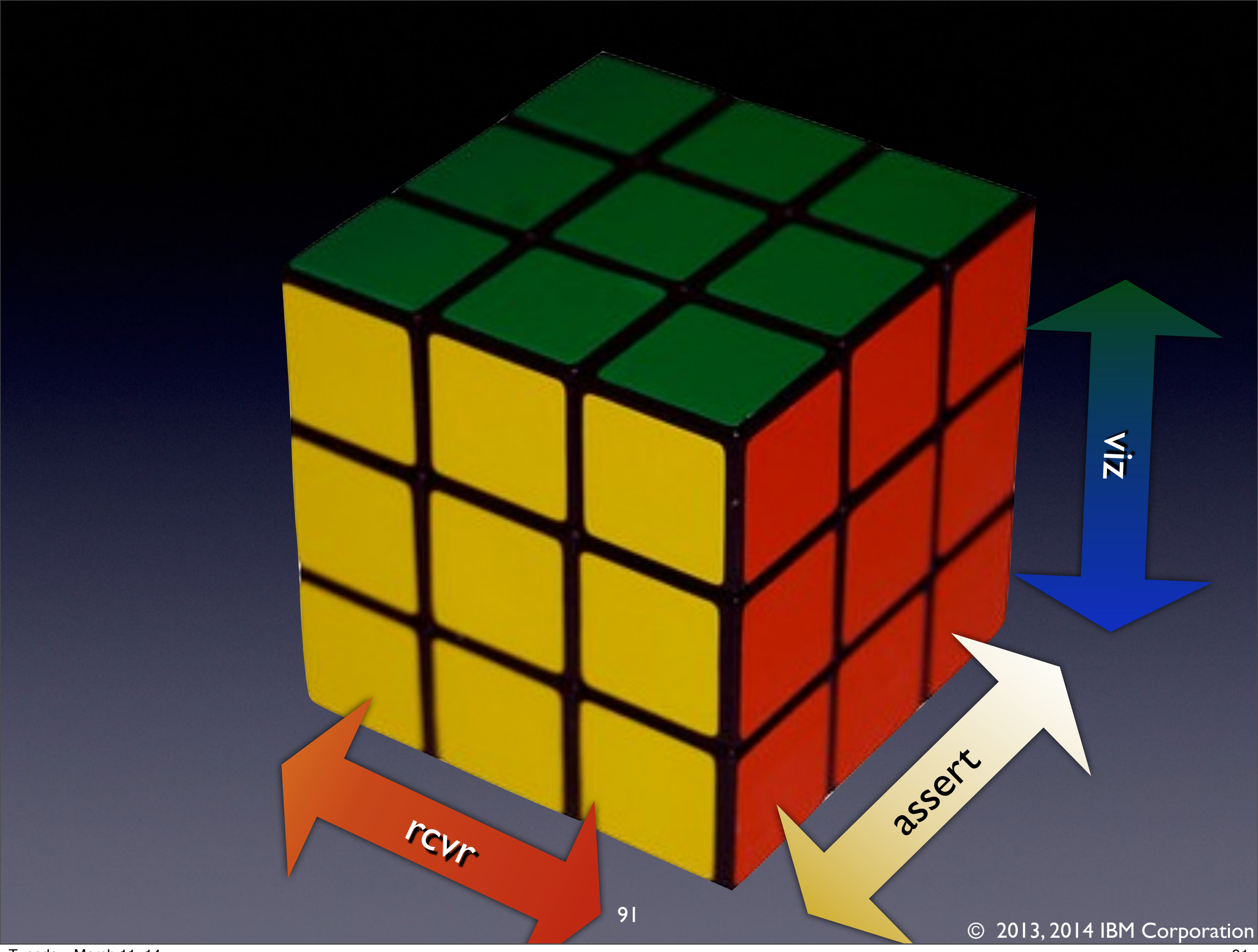
Looks complex and crowded
but
conceptually, simple, regular structure

Manifest the simplicity and regularity
while
harnessing the power of the complexity





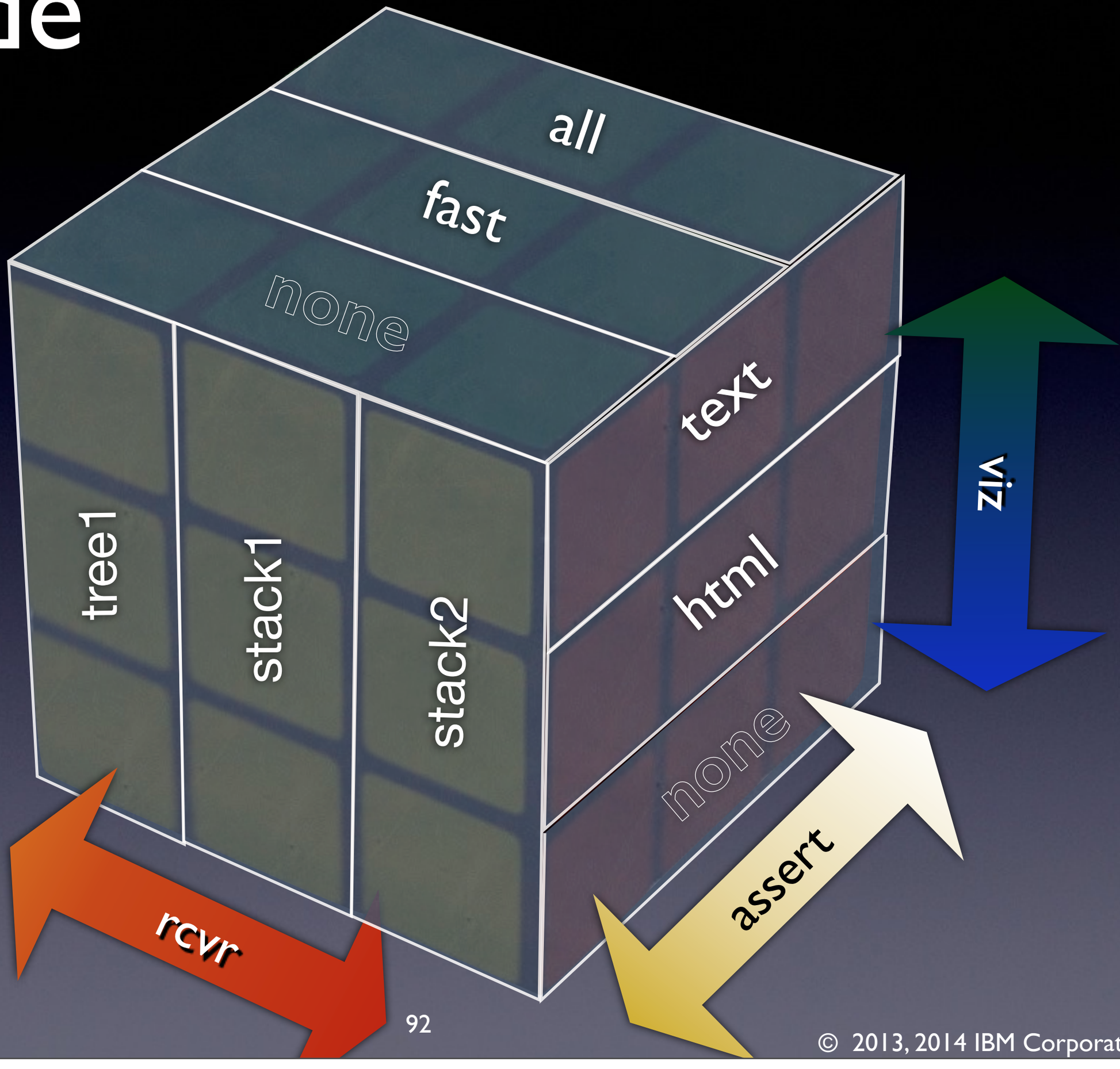

© 2013, 2014 IBM Corporation






© 2013, 2014 IBM Corporation

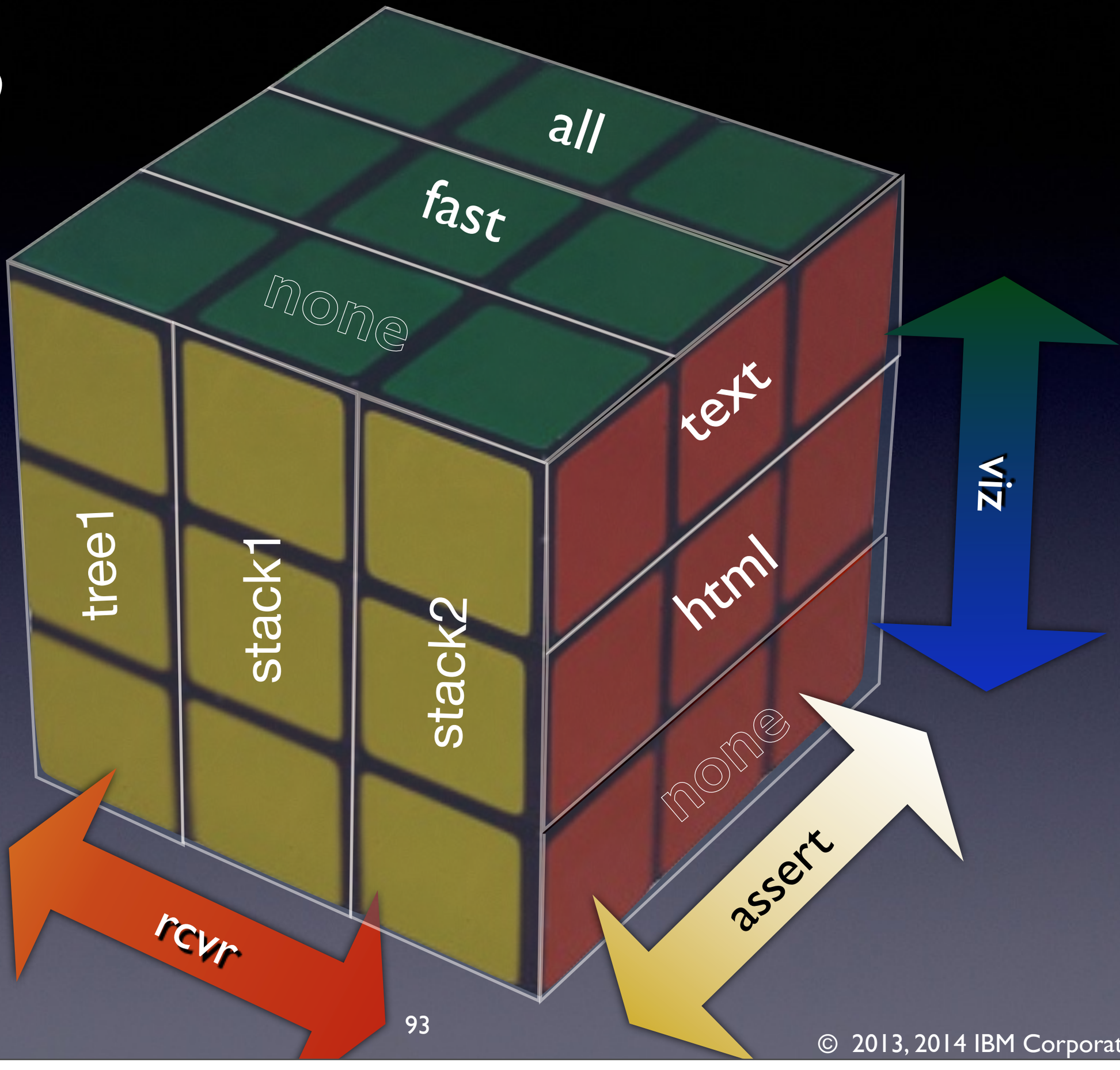
# Outside view






© 2013, 2014 IBM Corporation

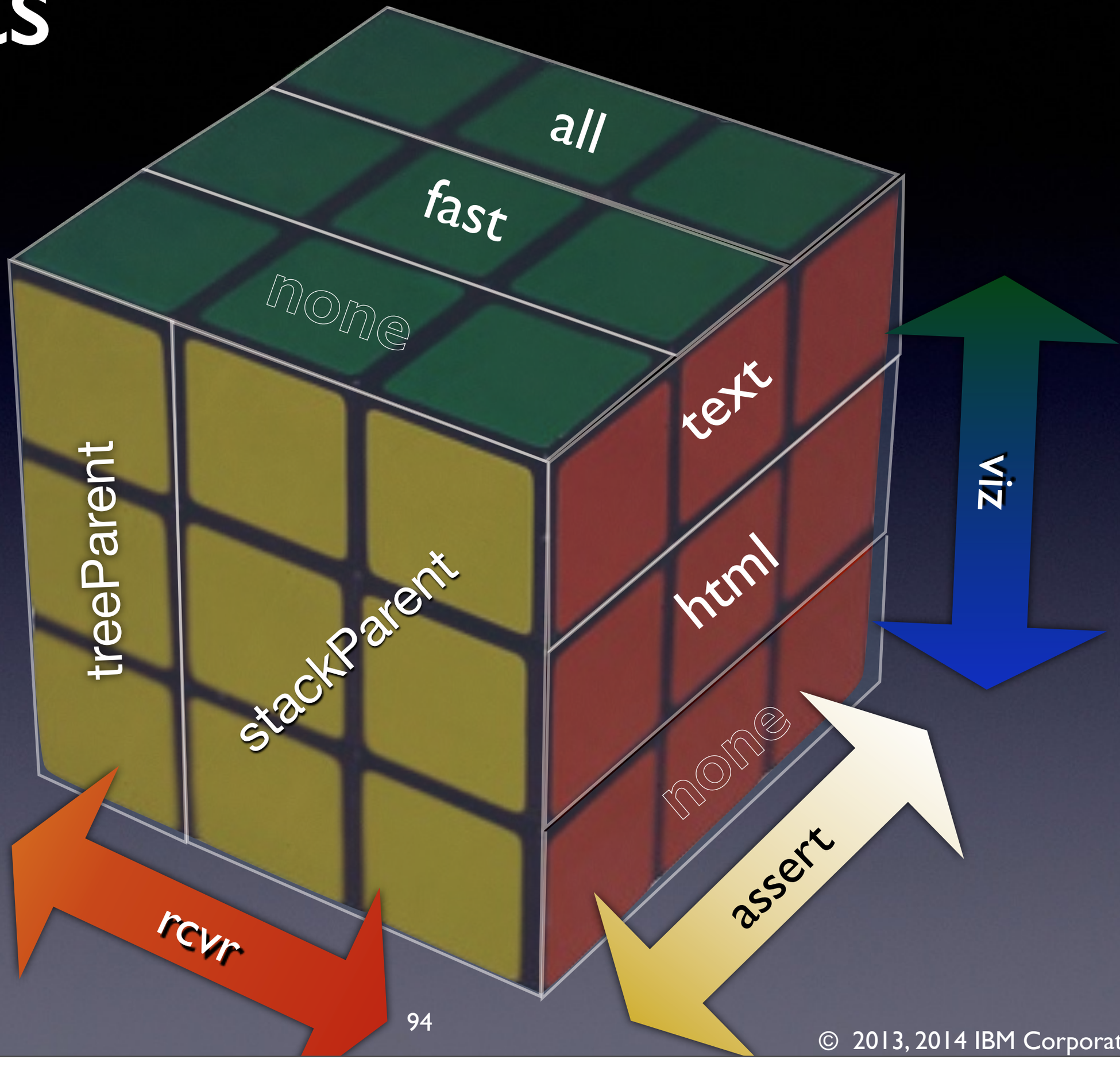
# Inside views?






© 2013, 2014 IBM Corporation

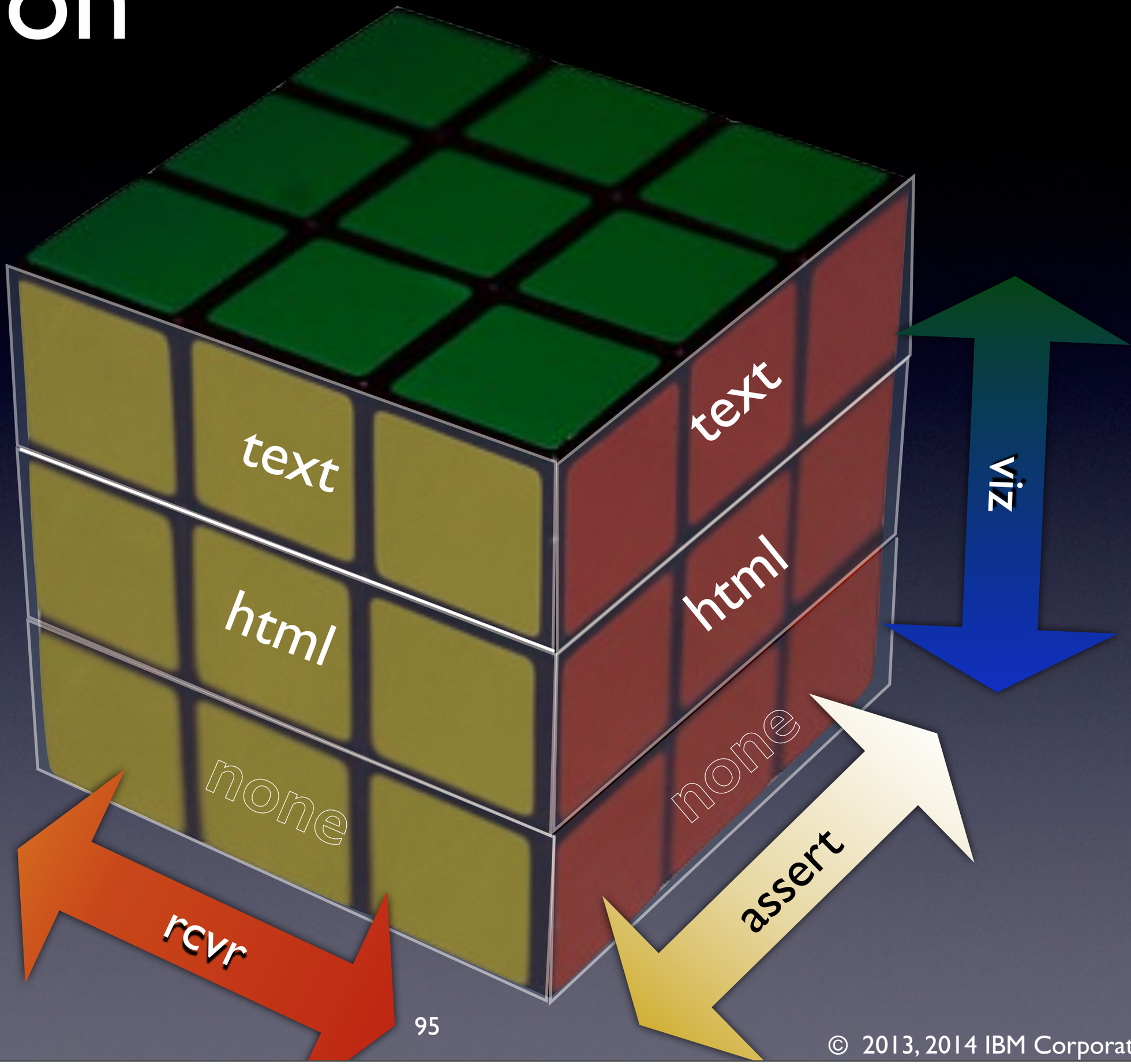
# Parents






© 2013, 2014 IBM Corporation

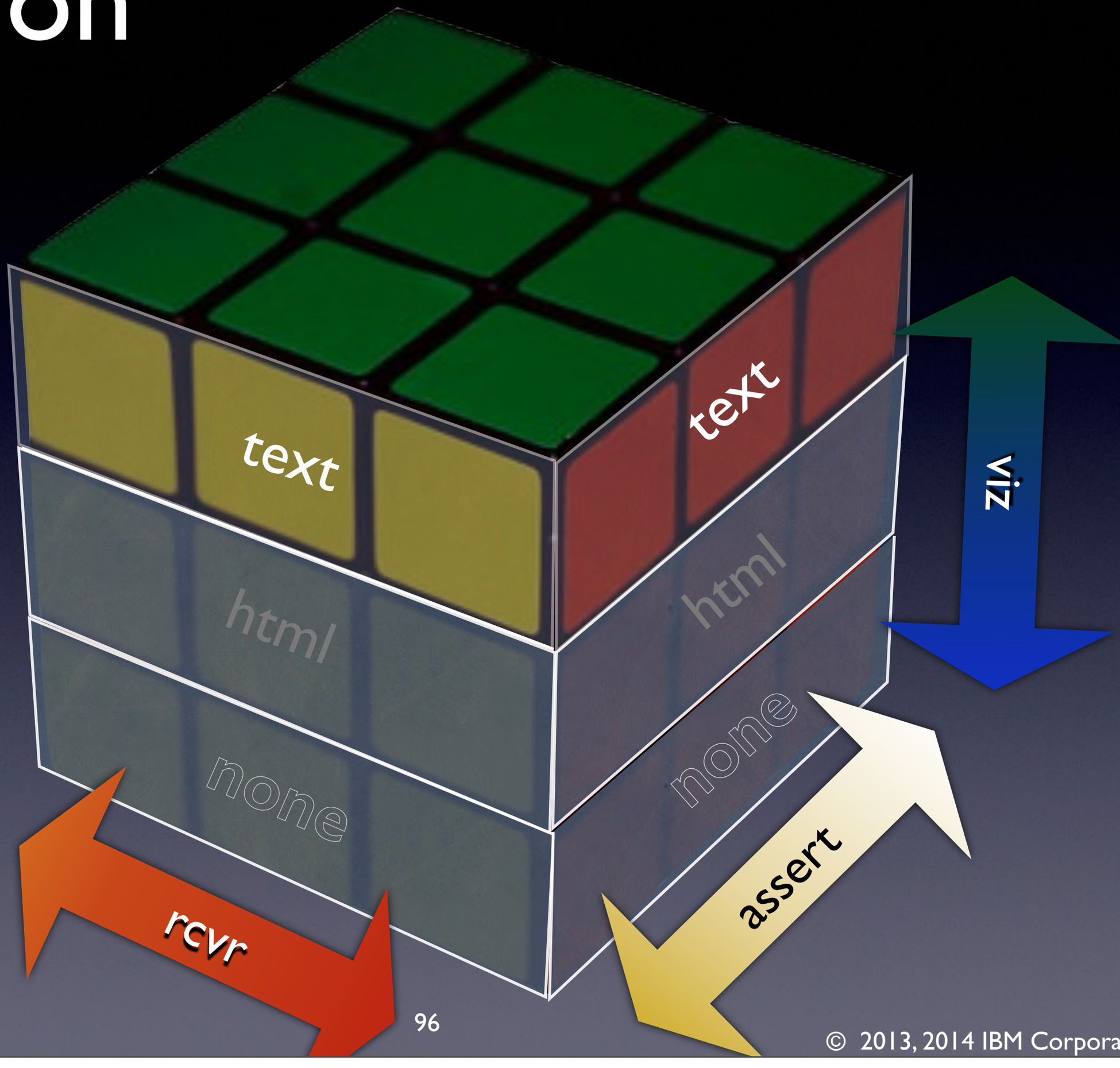
# Focus on viz





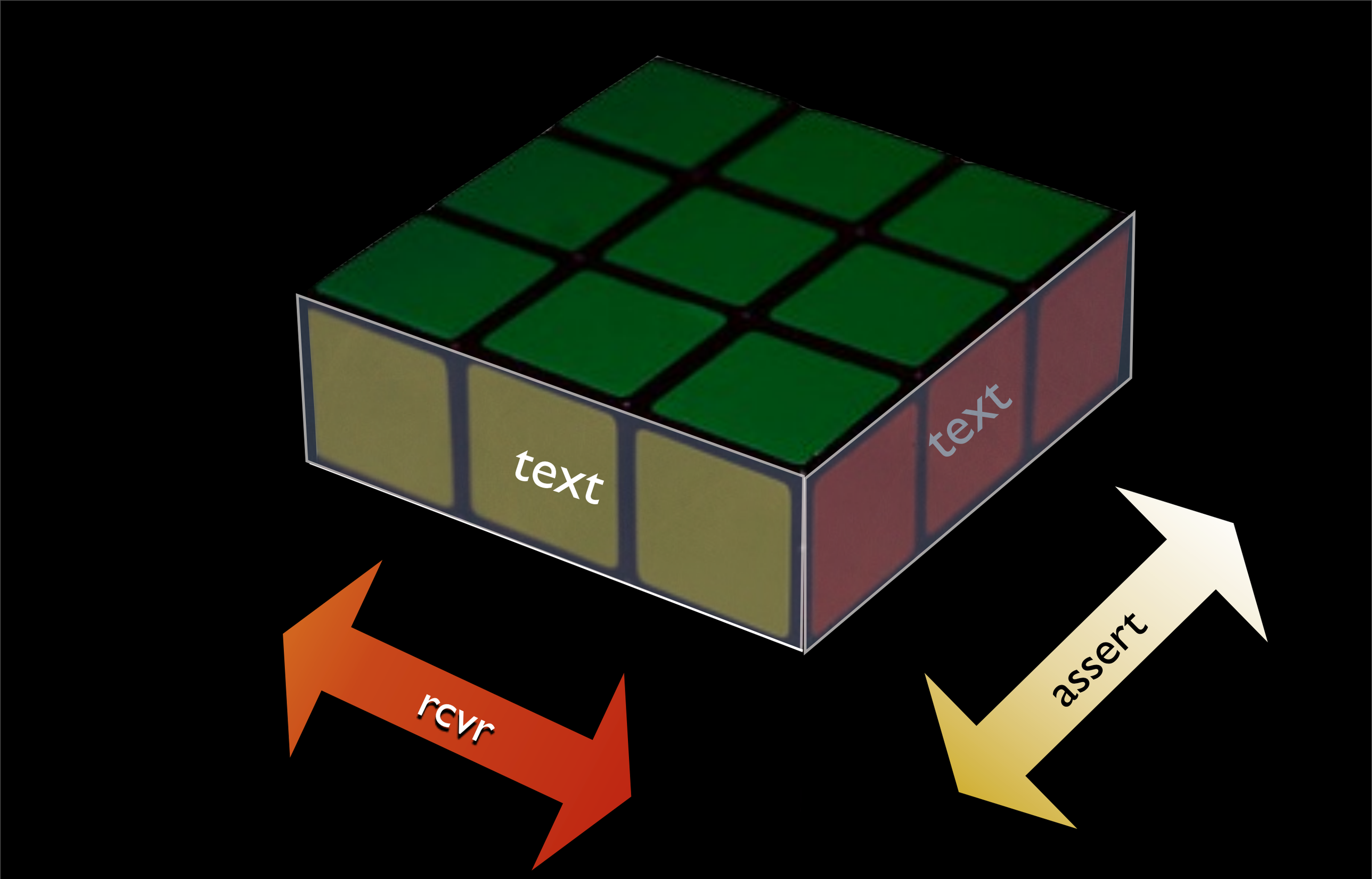

© 2013, 2014 IBM Corporation

# Focus on viz





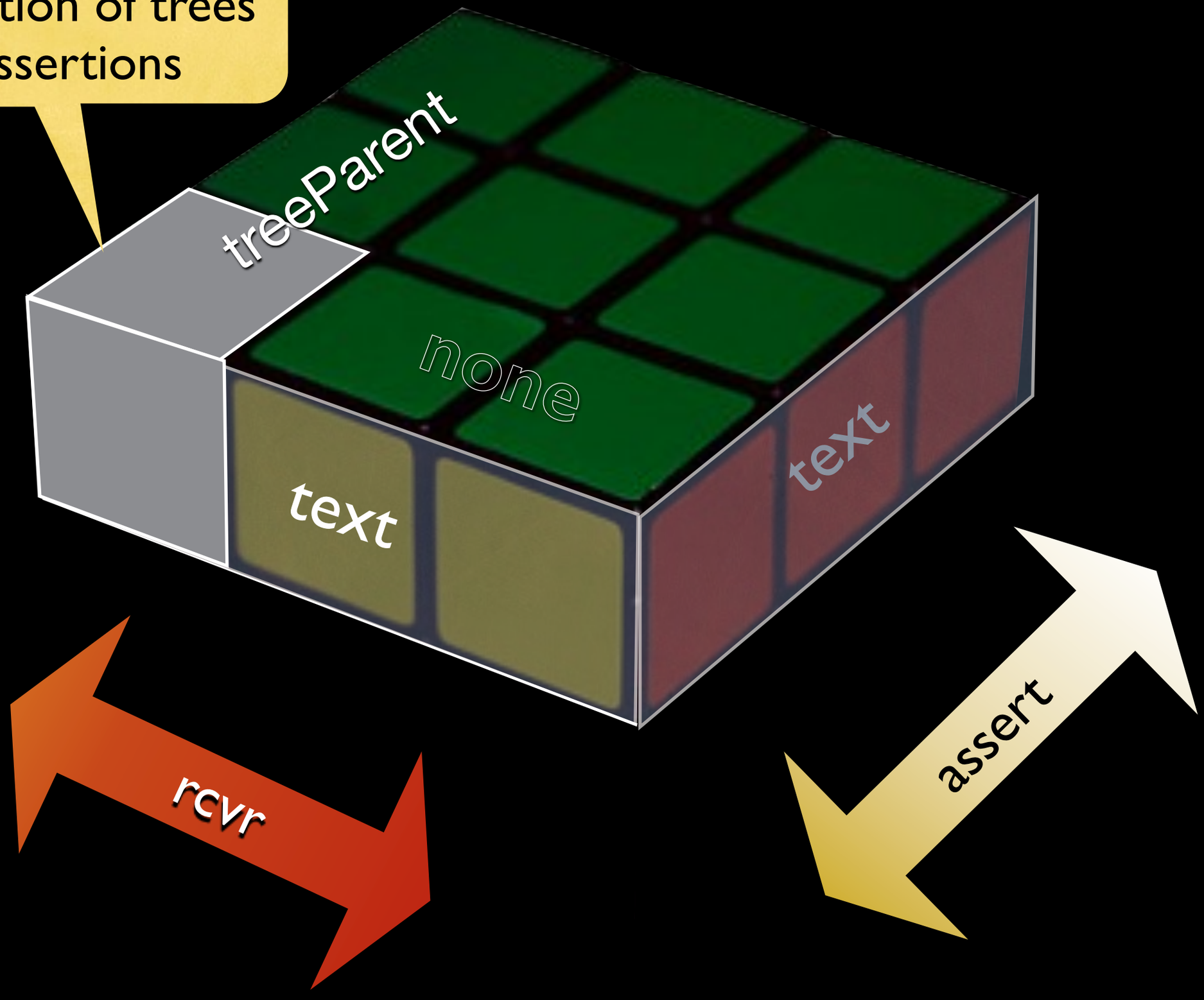

© 2013, 2014 IBM Corporation






© 2013, 2014 IBM Corporation

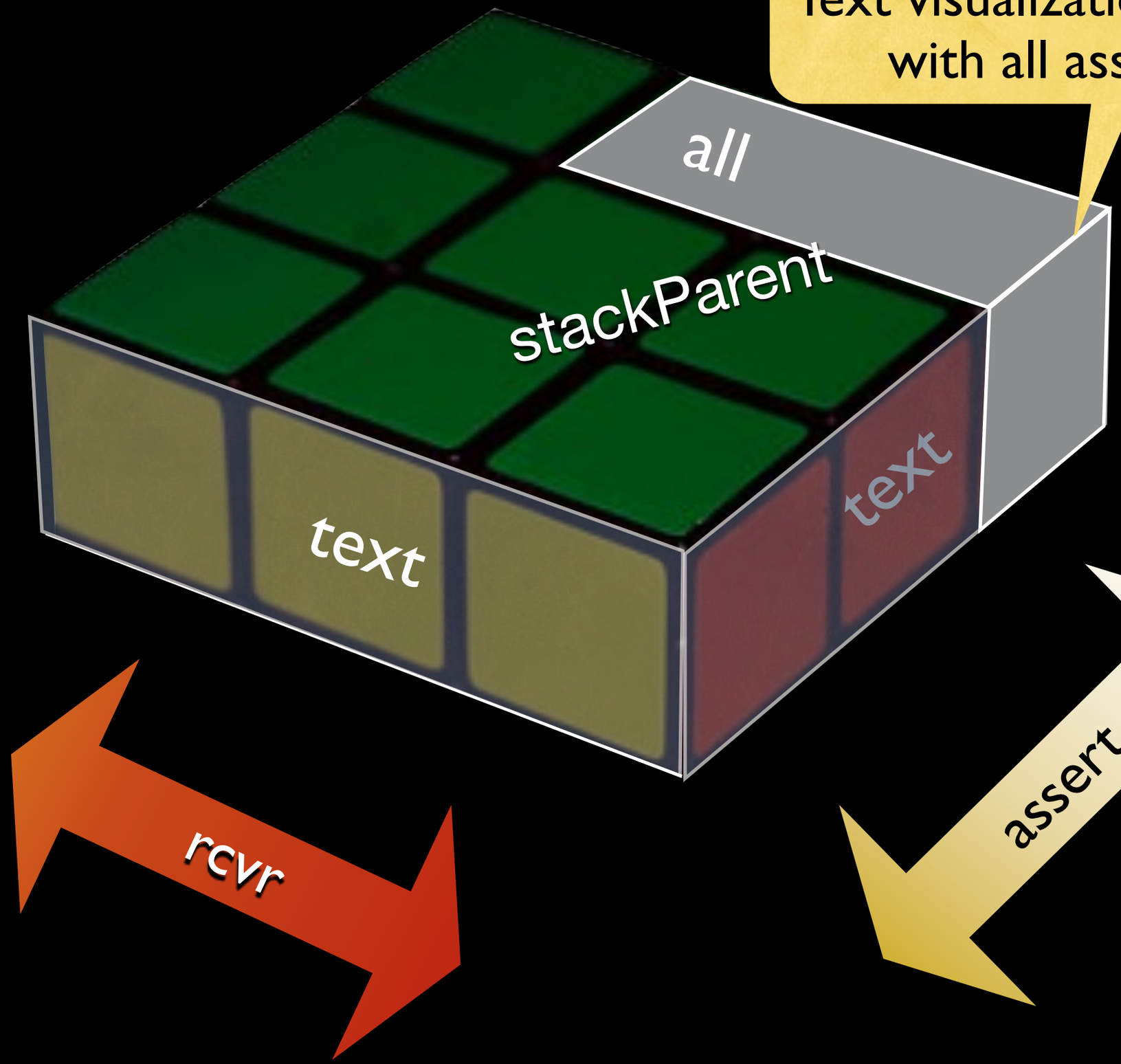
Text visualization of trees with no assertions






© 2013, 2014 IBM Corporation

Text visualization of stacks with all assertions

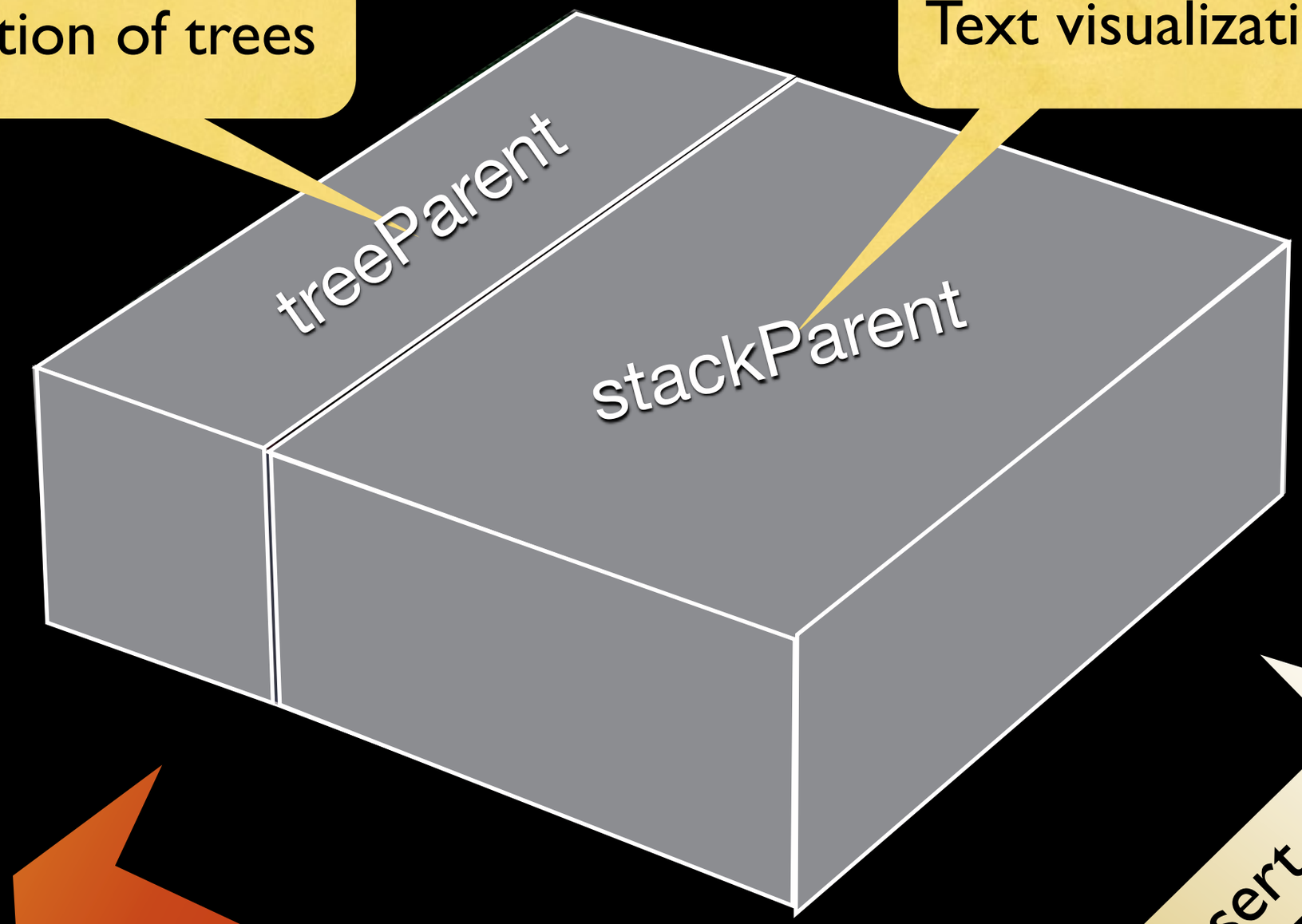





© 2013, 2014 IBM Corporation

Text visualization of trees

Text visualization of stacks












© 2013, 2014 IBM Corporation

Text visualization of trees

Text visualization of stacks

treeParent

stackParent

text

text

html

html

none

none

viz

rcvr

assert

101

© 2013, 2014 IBM Corporation

# Space Structure is Context

- Surfaces structure that is usually hidden
- Shared vocabulary between senders and implementors, and their programmers
  - What cases to implement
  - Options for message context
- Symmetry enables subjectivity
  - No dominant decomposition
  - Different views for different users/purposes


© 2013, 2014 IBM Corporation

# Space Structure is Context

- Surfaces structure that is usually hidden
- Shared vocabulary between senders and implementors, and their programmers
  - What cases to implement
  - Options for message context
- Symmetry enables subjectivity
  - No dominant decomposition
  - Different views for different users/purposes

How to manifest this in the IDE???



© 2013, 2014 IBM Corporation

# Examples

- Using multiple dimensions and context-based dispatch for:
  - Hiding slots and protecting invariants
  - Evolution
- Towards a Korz IDE

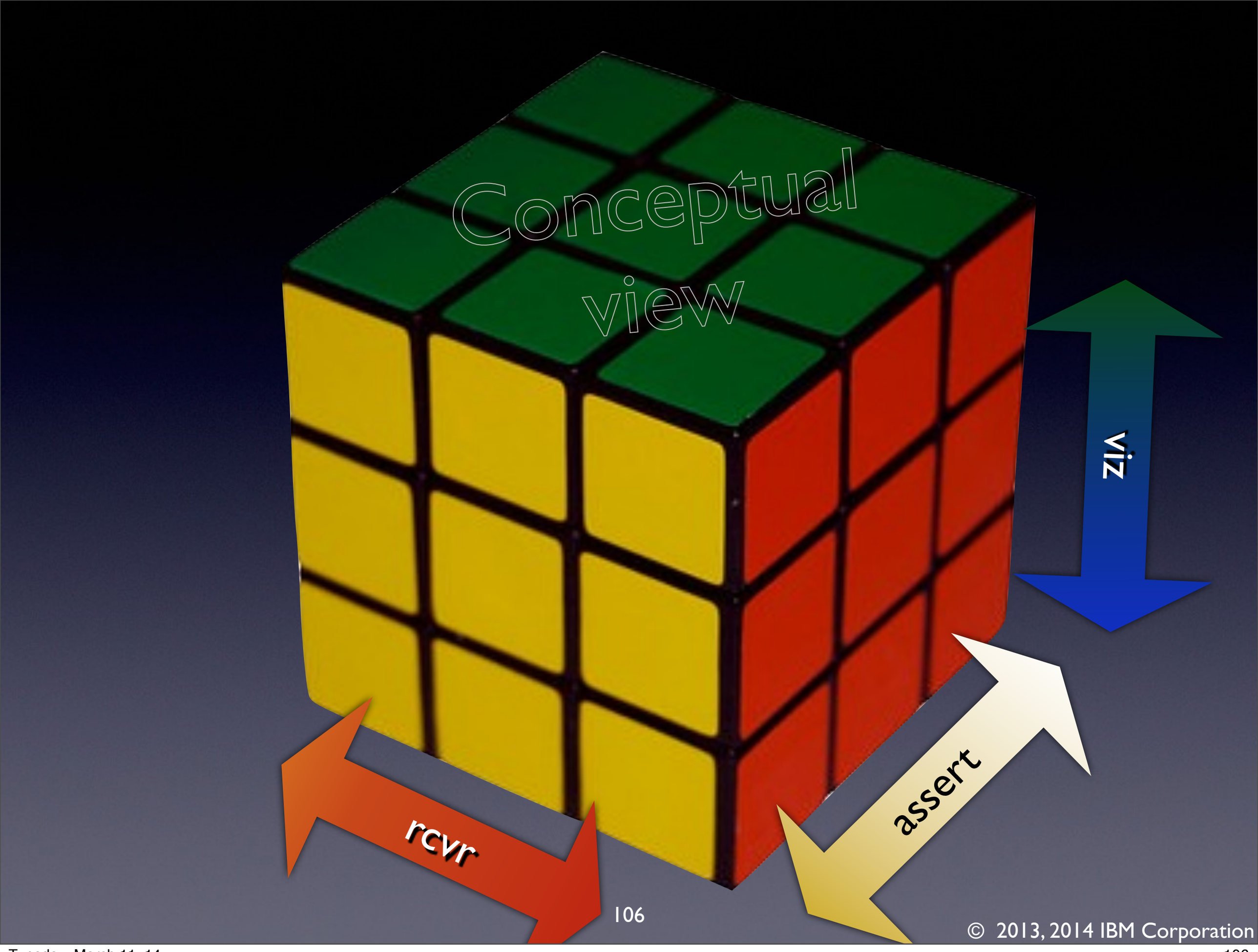

© 2013, 2014 IBM Corporation

# Examples

- Using multiple dimensions and context-based dispatch for:
  - Hiding slots and protecting invariants
  - Evolution
- Towards a Korz IDE

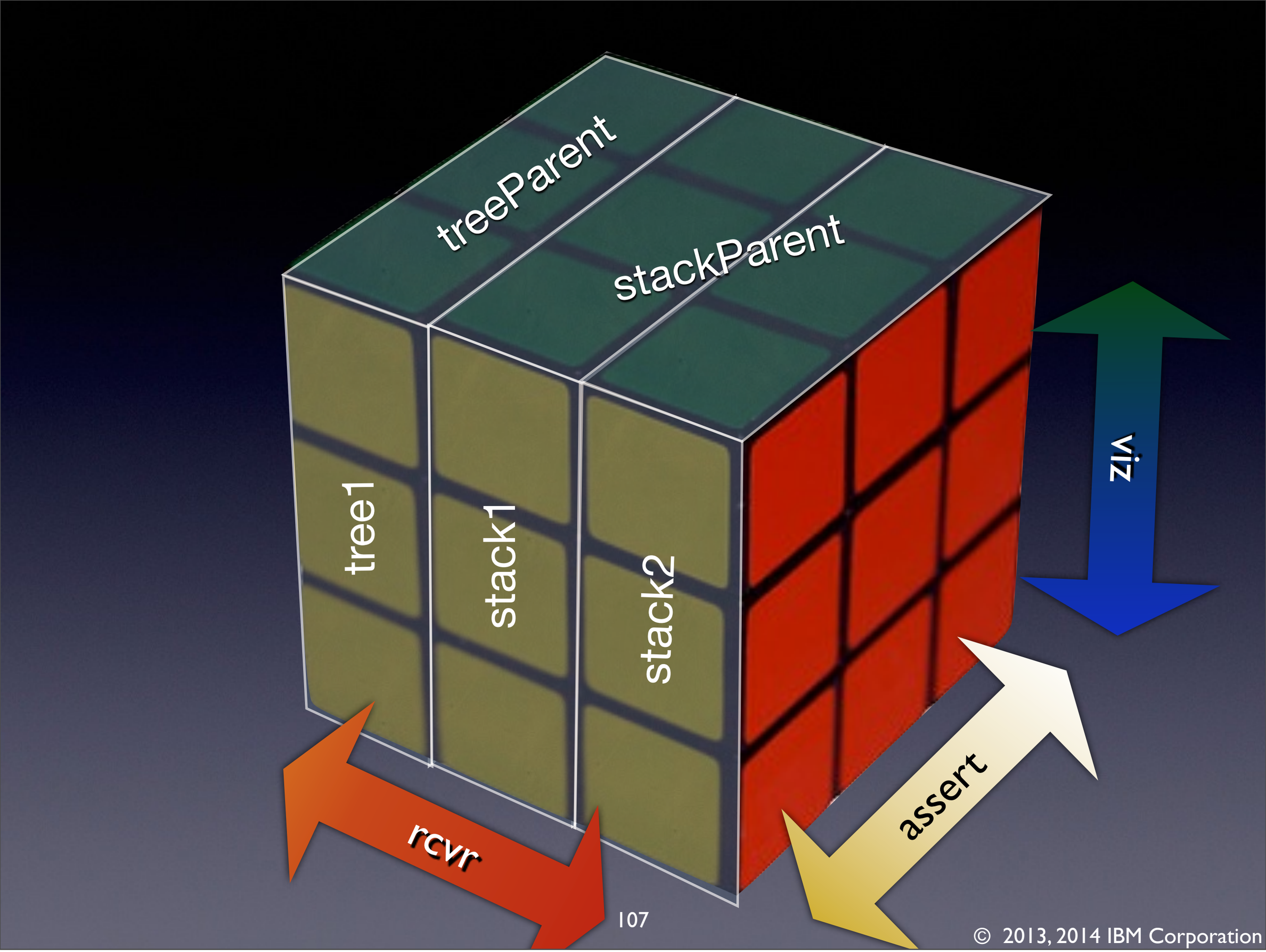


© 2013, 2014 IBM Corporation



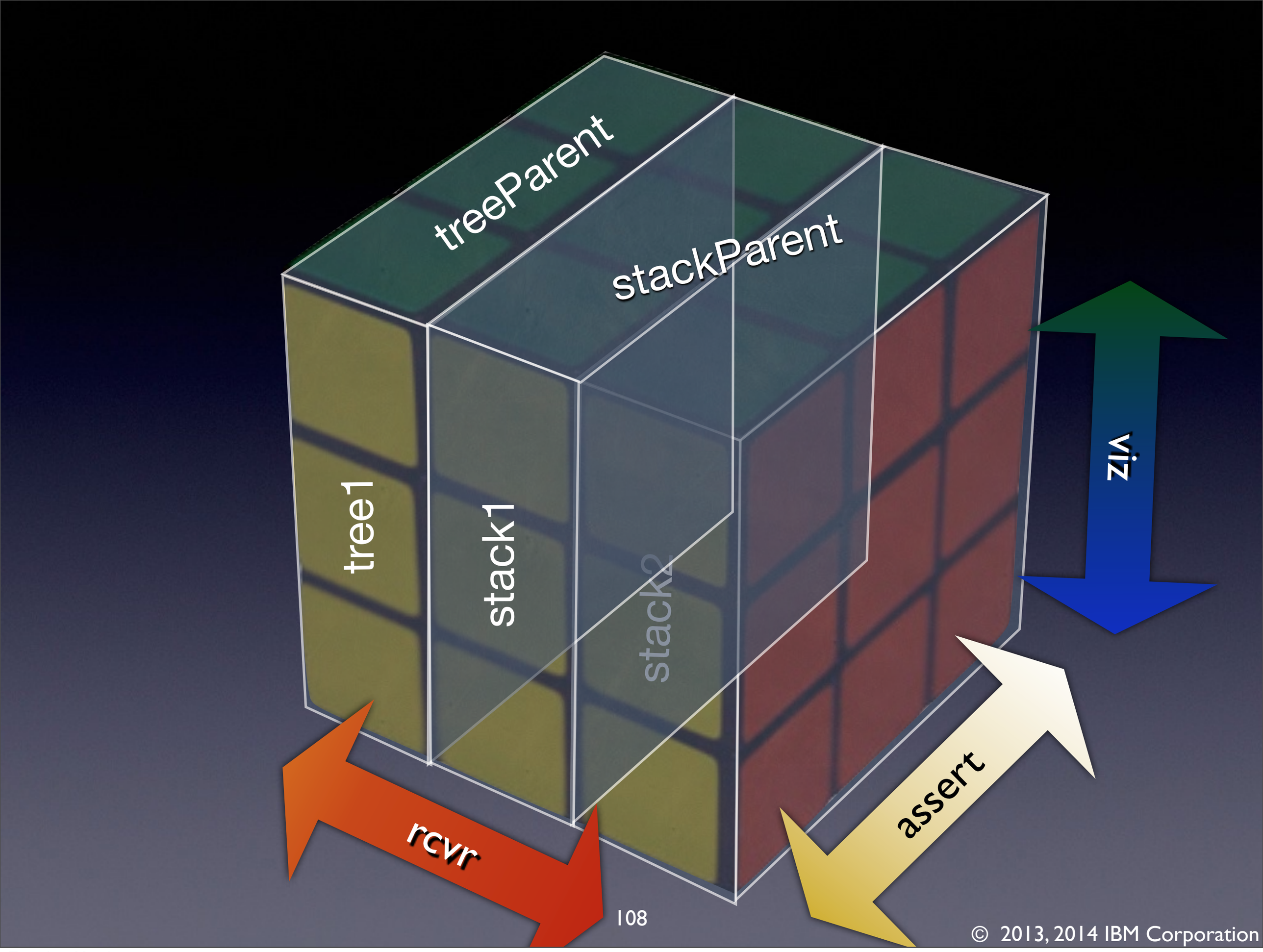



© 2013, 2014 IBM Corporation



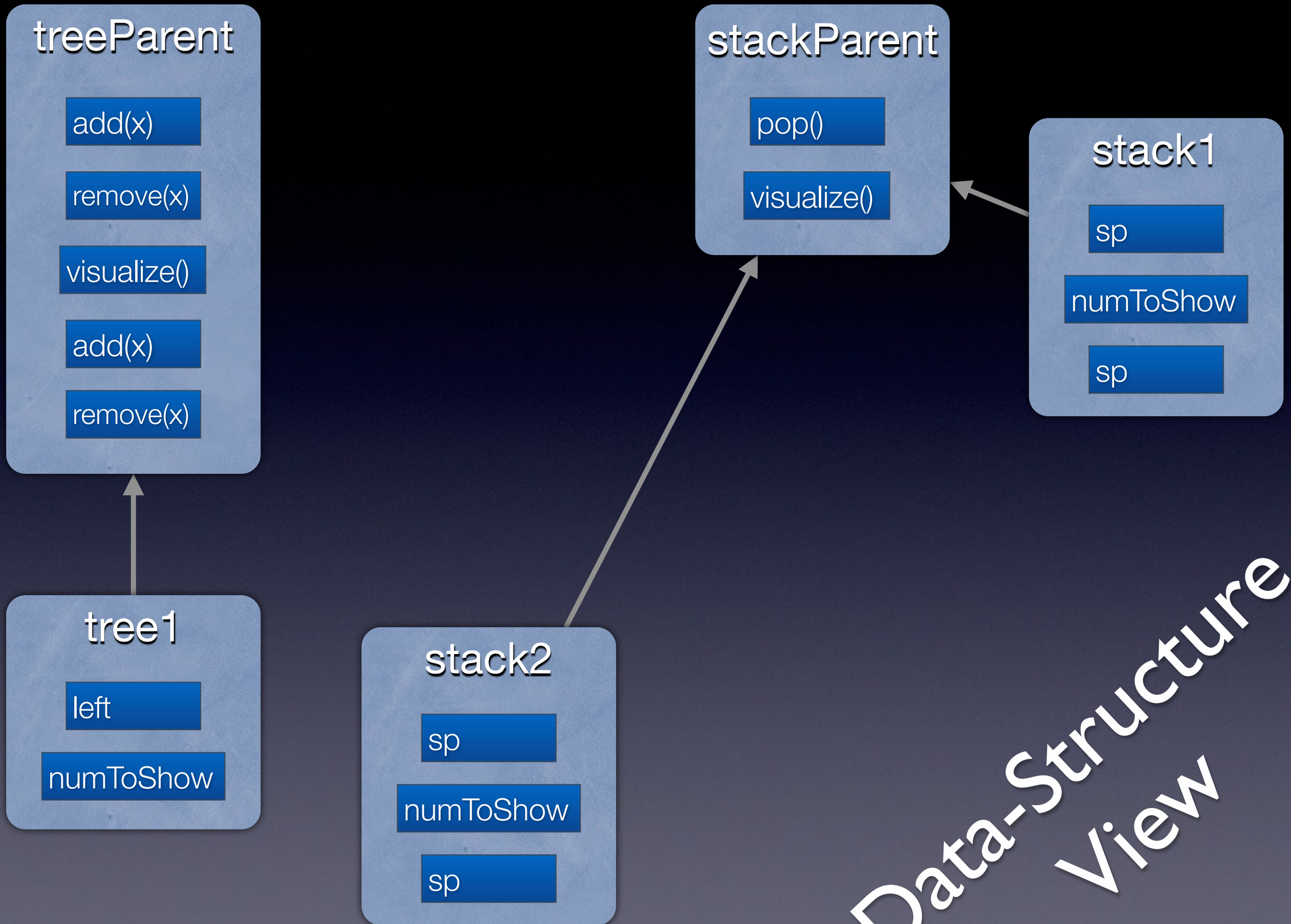



© 2013, 2014 IBM Corporation






© 2013, 2014 IBM Corporation






© 2013, 2014 IBM Corporation

treeParent

add(x)

remove(x)

viz html

visualize()

assert all

add(x)

assert fast

remove(x)

tree1

left

viz html

numToShow

stackParent

pop()

viz html

visualize()

stack1

sp

viz html

numToShow

assert fast

sp

stack2

sp

viz html

numToShow

assert fast

sp

Data-Structure View



© 2013, 2014 IBM Corporation

# Data-Structure View

treeParent

add(x)

remove(x)

viz html

visualize()

assert all

add(x)

assert fast

remove(x)

tree1

left

viz html

numToShow

rcvr => treeParent
assert => all
viz => *none*

stackParent

pop()

viz html

visualize()

stack1

sp

viz html

numToShow

assert fast

sp

rcvr =>stack1
assert => *none*
*viz* => html

stack2

sp

viz html

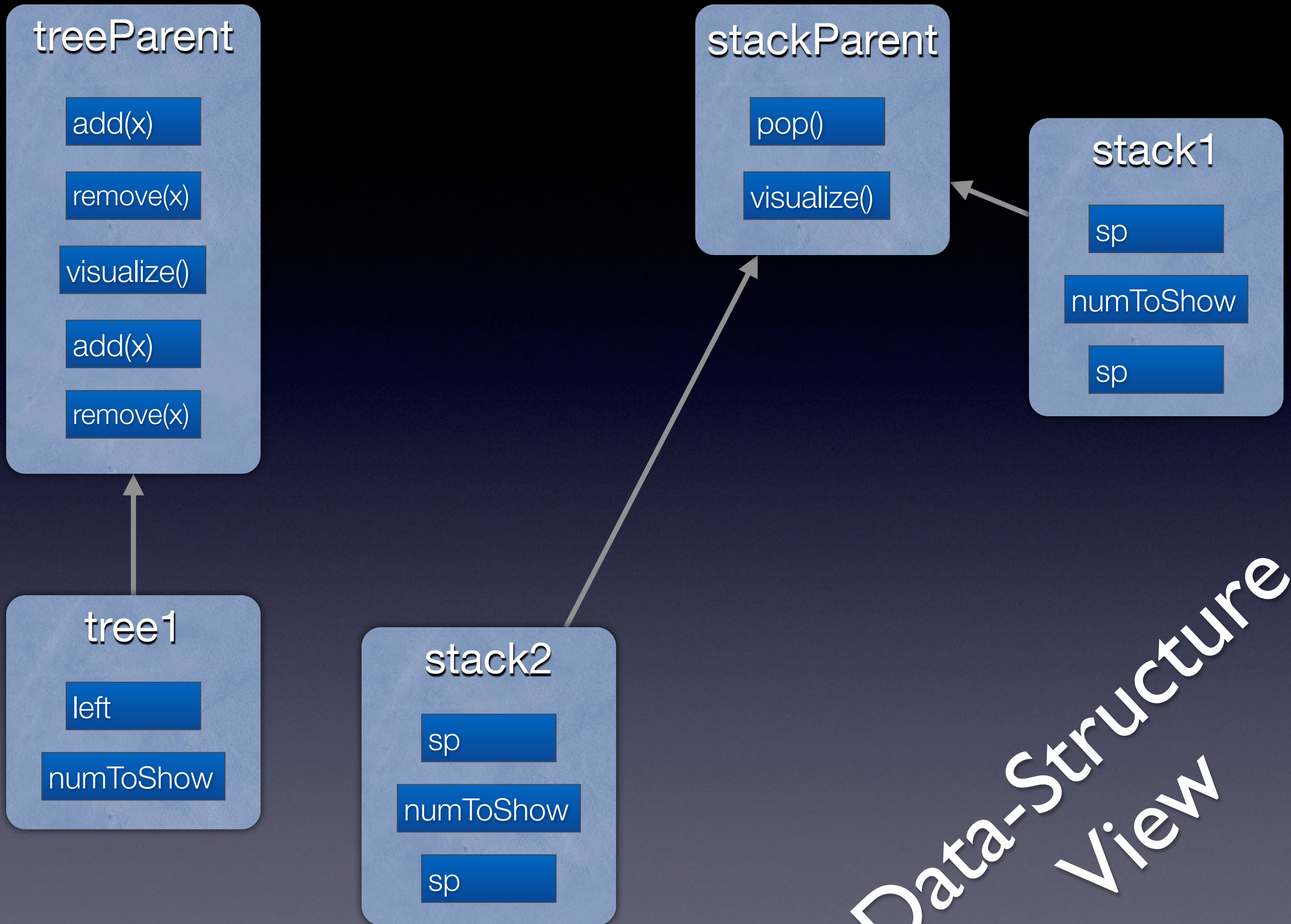
numToShow

assert fast

sp



© 2013, 2014 IBM Corporation




# Data-Structure View



© 2013, 2014 IBM Corporation

# Assertion View

## *none*

- pop()
- visualize()
- add(x)
- remove(x)
- visualize()
- sp
- numToShow
- sp
- numToShow
- left
- numToShow

## fast

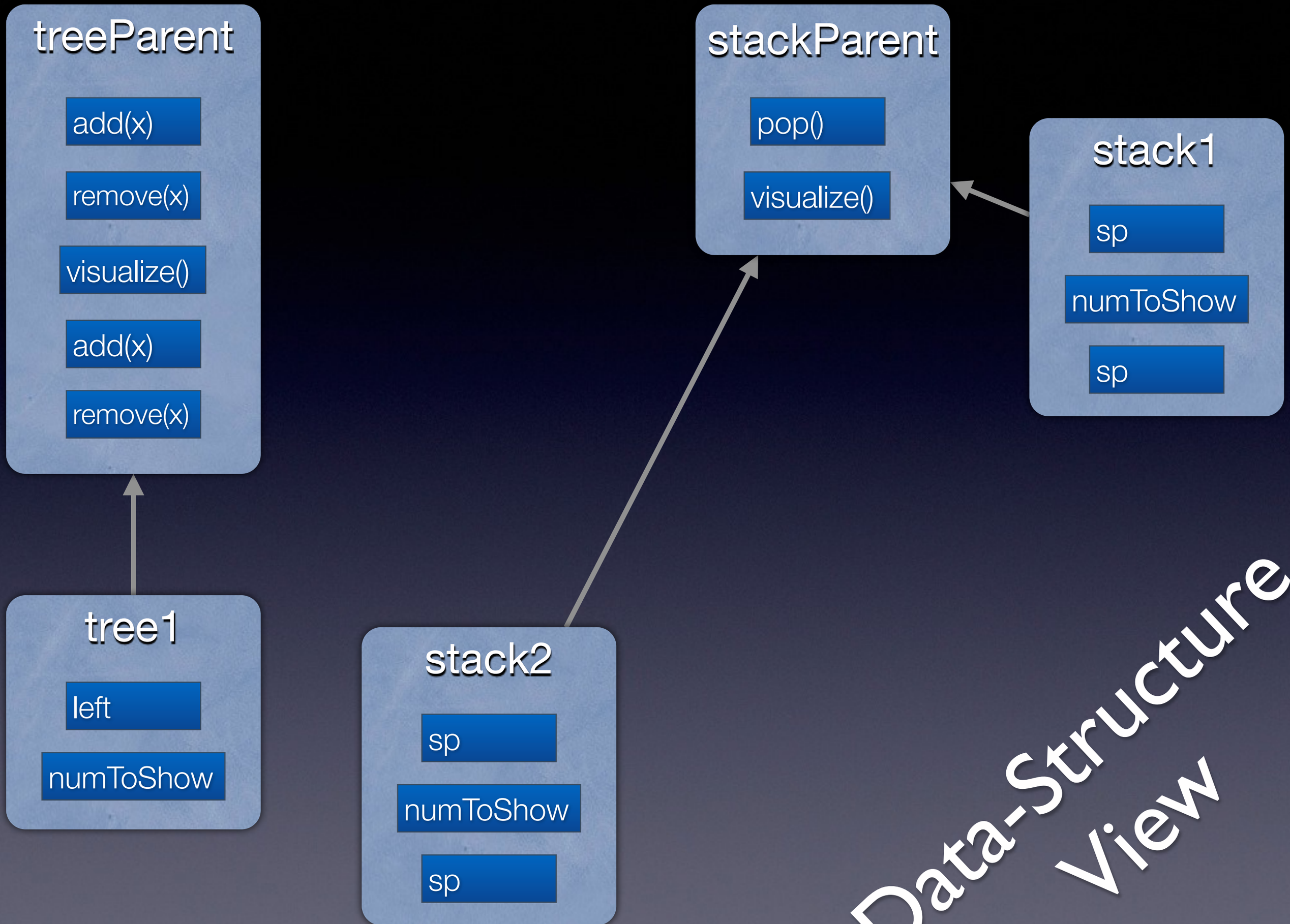
- sp
- sp
- remove(x)

## all

- add(x)



© 2013, 2014 IBM Corporation

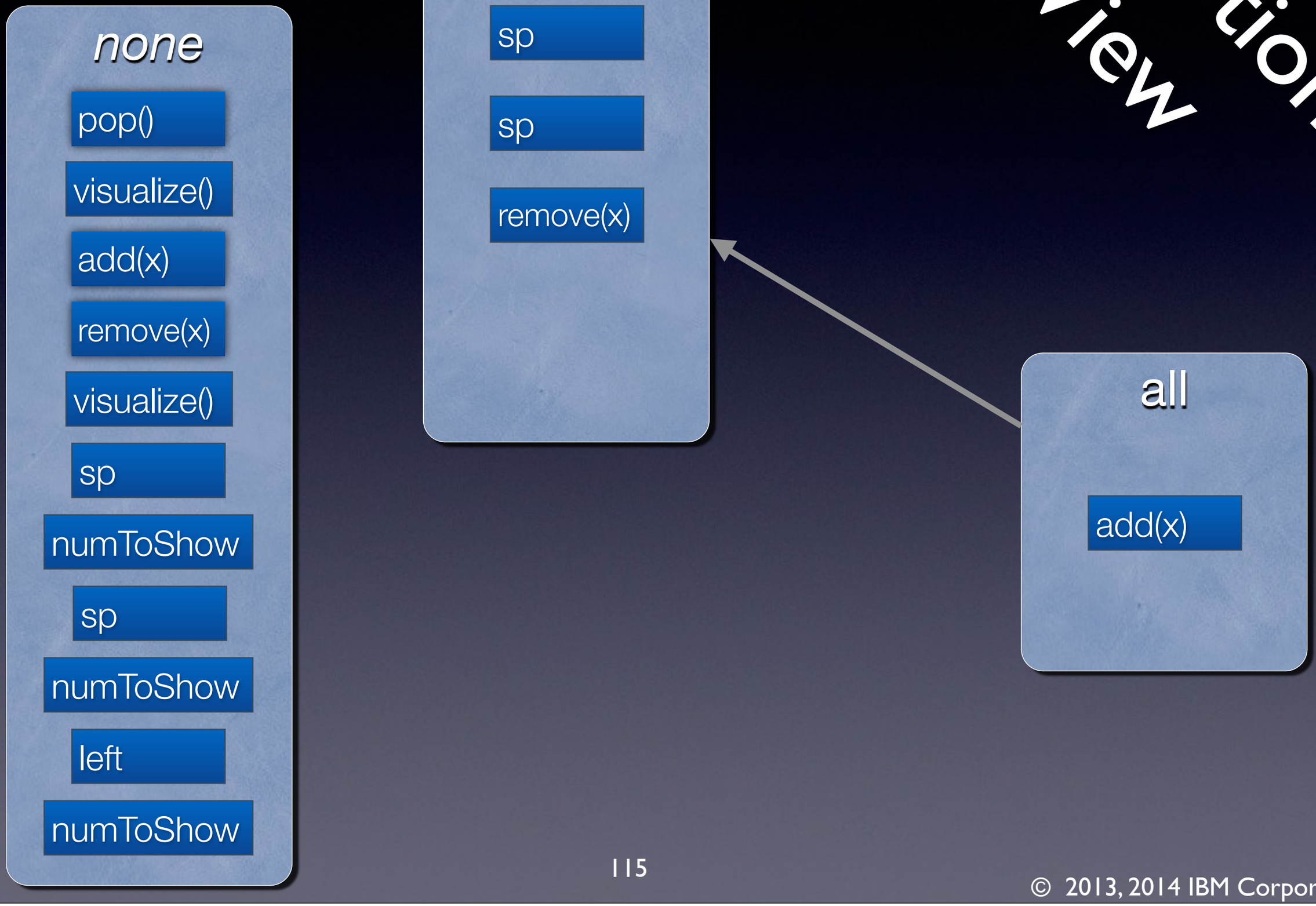



# Data-Structure View



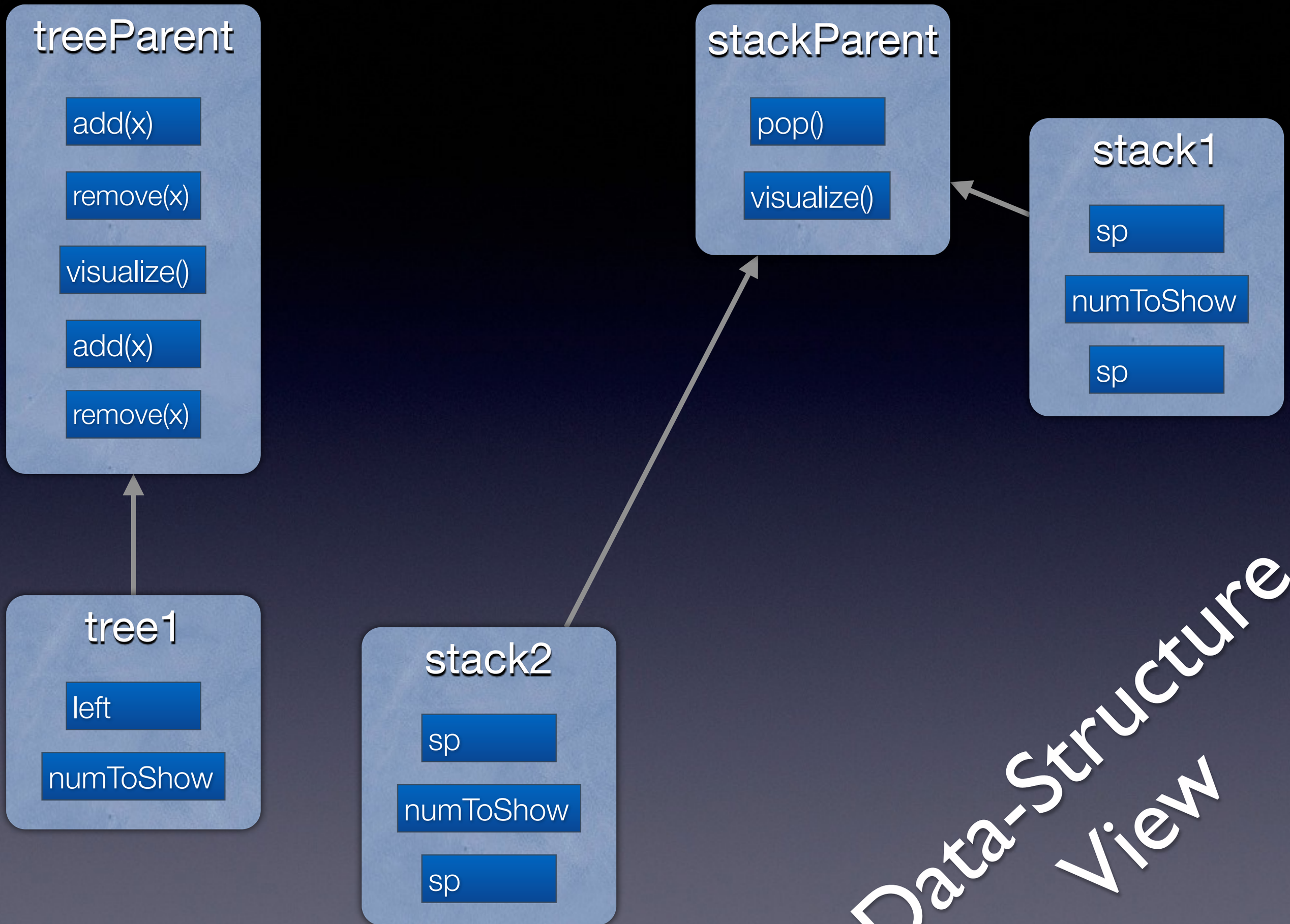
© 2013, 2014 IBM Corporation

# Assertion View






© 2013, 2014 IBM Corporation




# Data-Structure View



© 2013, 2014 IBM Corporation

add(x)

pop()

remove(x)

visualize()

add(x)

remove(x)

left

visualize()

sp

sp

numToShow

sp

numToShow

numToShow

sp



© 2013, 2014 IBM Corporation

# Sea of Slots

add(x)

pop()

remove(x)

visualize()

add(x)

remove(x)

left

visualize()

sp

sp

numToShow

numToShow

sp

numToShow

# Assertion View

## *none*

- pop()
- visualize()
- add(x)
- remove(x)
- visualize()
- sp
- numToShow
- sp
- numToShow
- left
- numToShow

## fast

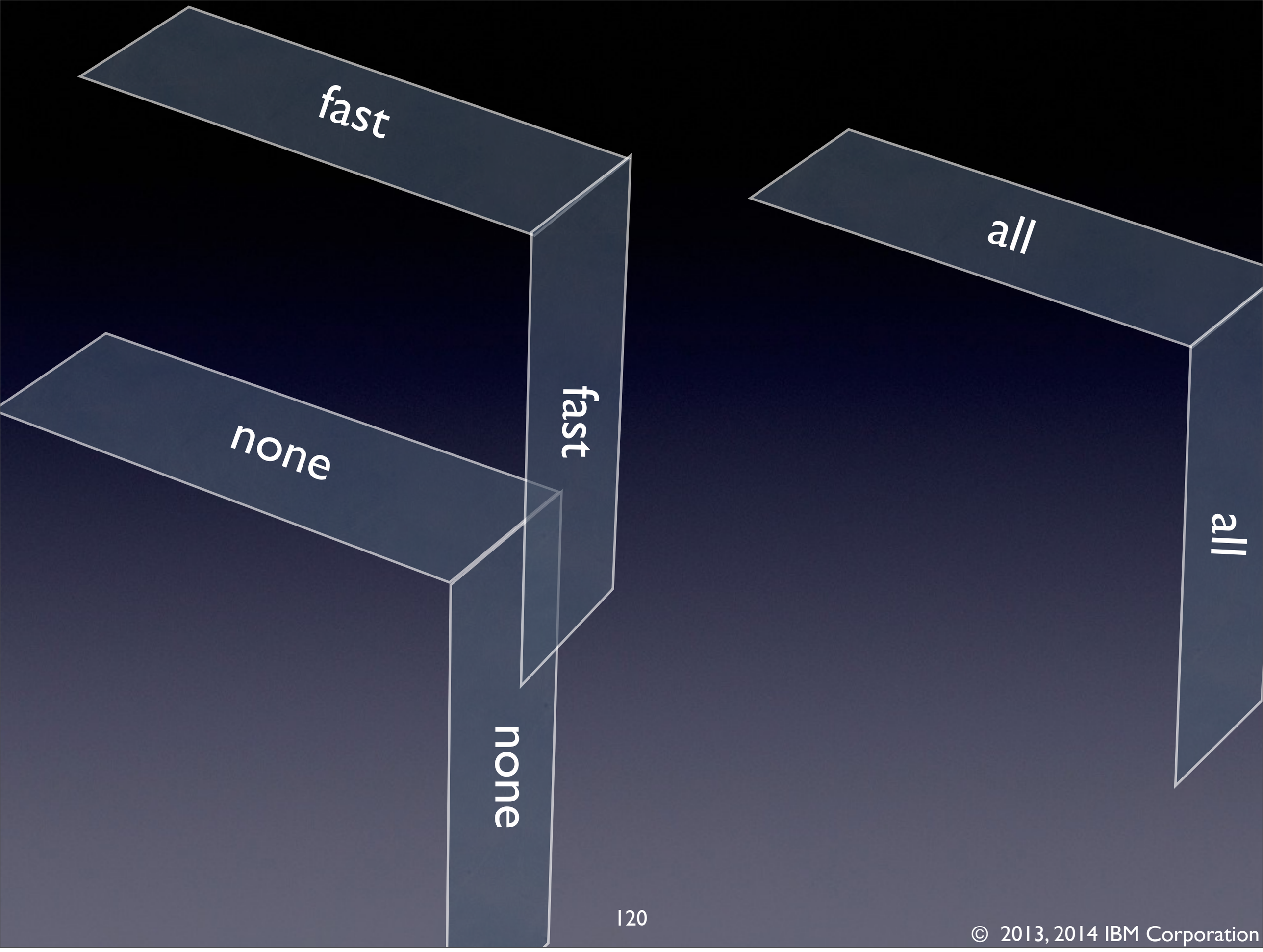
- sp
- sp
- remove(x)

## all

- add(x)

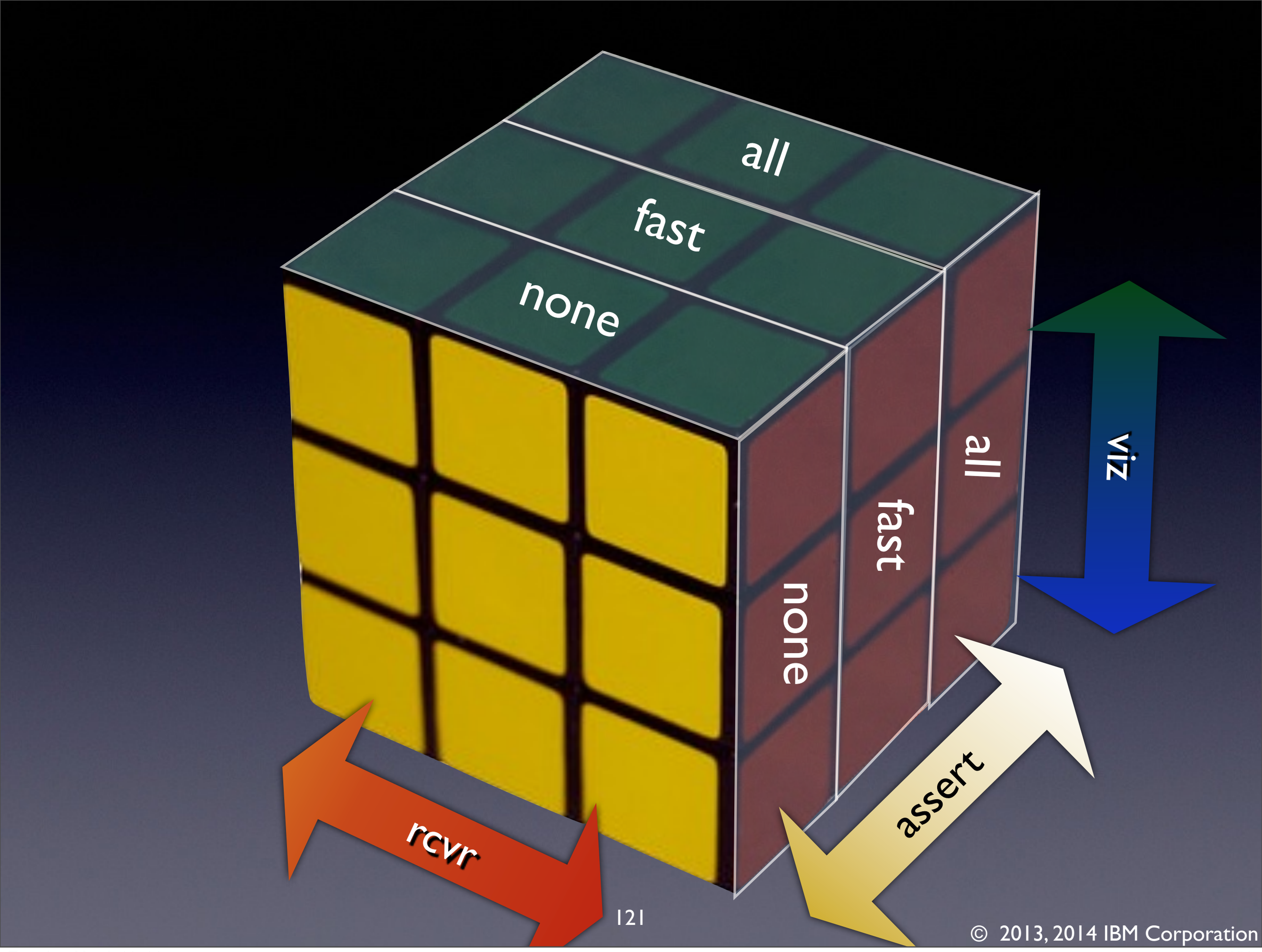


© 2013, 2014 IBM Corporation






© 2013, 2014 IBM Corporation






© 2013, 2014 IBM Corporation

# Dancing with Symmetry

- Space is symmetrical: no dominant decomposition
- Subjective views for different tasks
  - Message contexts: projections for specific *execution* tasks
  - IDE: projections for specific *programming* tasks
    - One-dimensional, with nested details
    - Two-dimensional pivot tables
    - ...
- Space contains lots of slots, many for very specific cases
  - Sparse
  - IDE must support finding, navigating, restricting, visualizing, ...
- Demo will show our early exploration



© 2013, 2014 IBM Corporation

# Further Speculation

- Development context: affects IDE
  - *Novice/expert* worlds, *assertion-checking* options, etc.
- Context affecting operation of the interpreter
  - *Ensembles*: run all matches
  - *Distribution*: context indicates where to run code
  - *Reflection*: context tells object to behave like a mirror
- Side-effect-free evaluation
- Numeric precision, IEEE Floating Point
- ... etc. ...

Needs research!



© 2013, 2014 IBM Corporation

# Contents

- What's the big idea?
- What's it good for?
- How real is it?
- How does it work?
- Are you guys through yet?

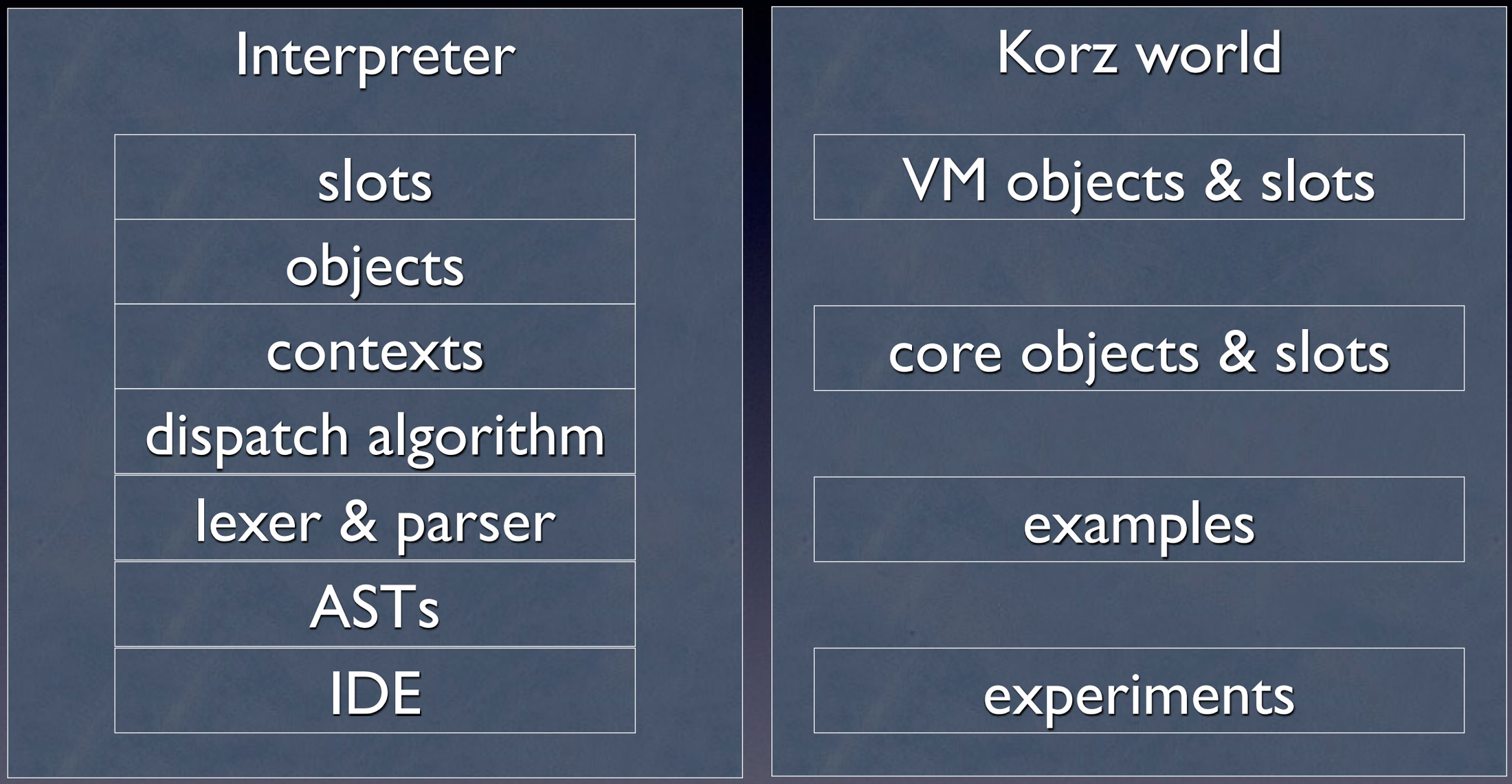
# Contents

- What's the big idea?
- What's it good for?
- How real is it?
- How does it work?
- Are you guys through yet?



© 2013, 2014 IBM Corporation

# Korz Prototype Pieces




20 K lines of Self + 10 K lines of Korz



© 2013, 2014 IBM Corporation

# Korz Demo

(Reminder to Self, err, Korz.)



© 2013, 2014 IBM Corporation

# Contents

- What's the big idea?
- What's it good for?
- How real is it?
- How does it work?
- Are you guys through yet?

# Contents

- What's the big idea?
- What's it good for?
- How real is it?
- How does it work?
- Are you guys through yet?



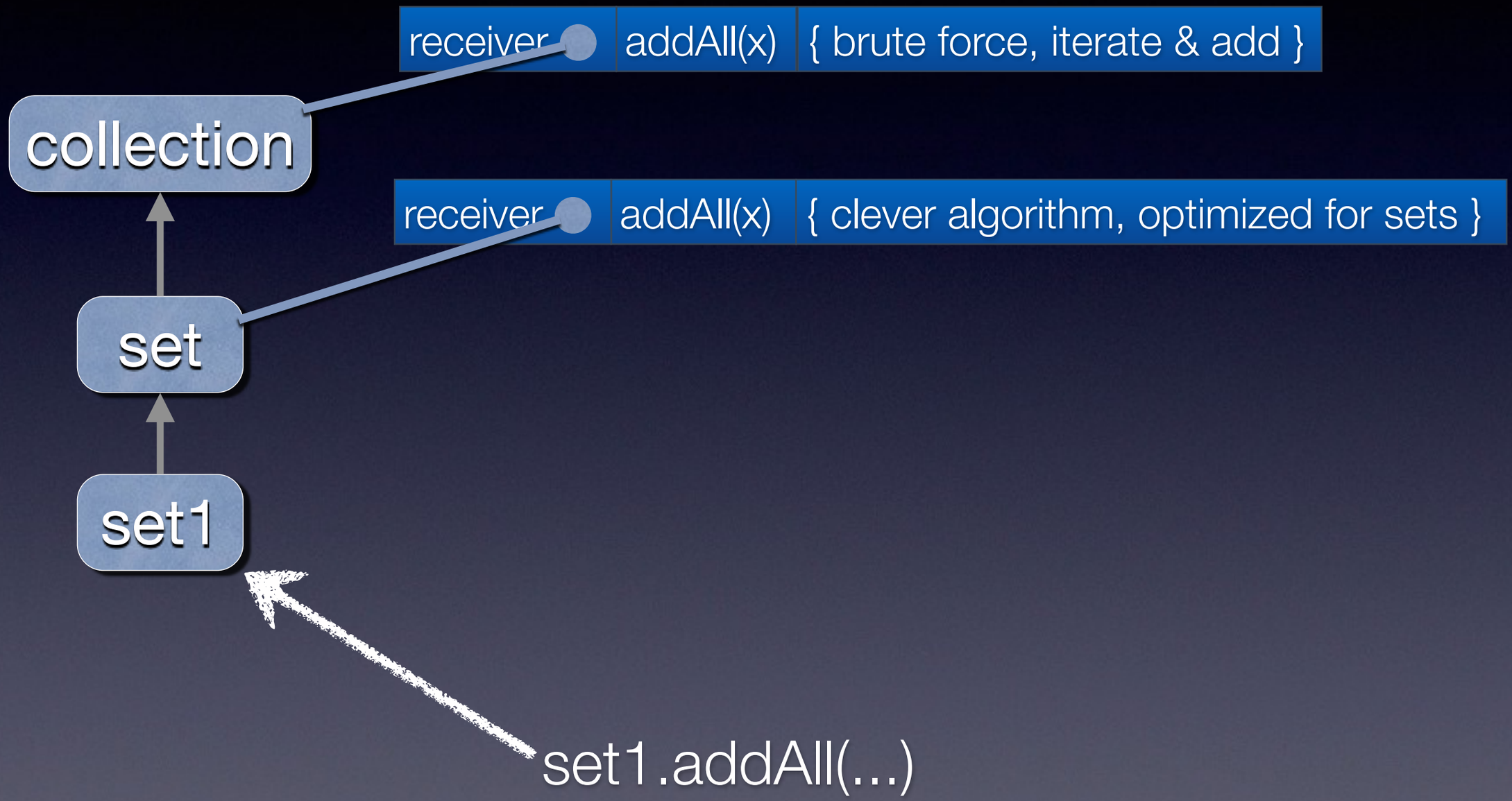
© 2013, 2014 IBM Corporation

# Dispatch principles



© 2013, 2014 IBM Corporation

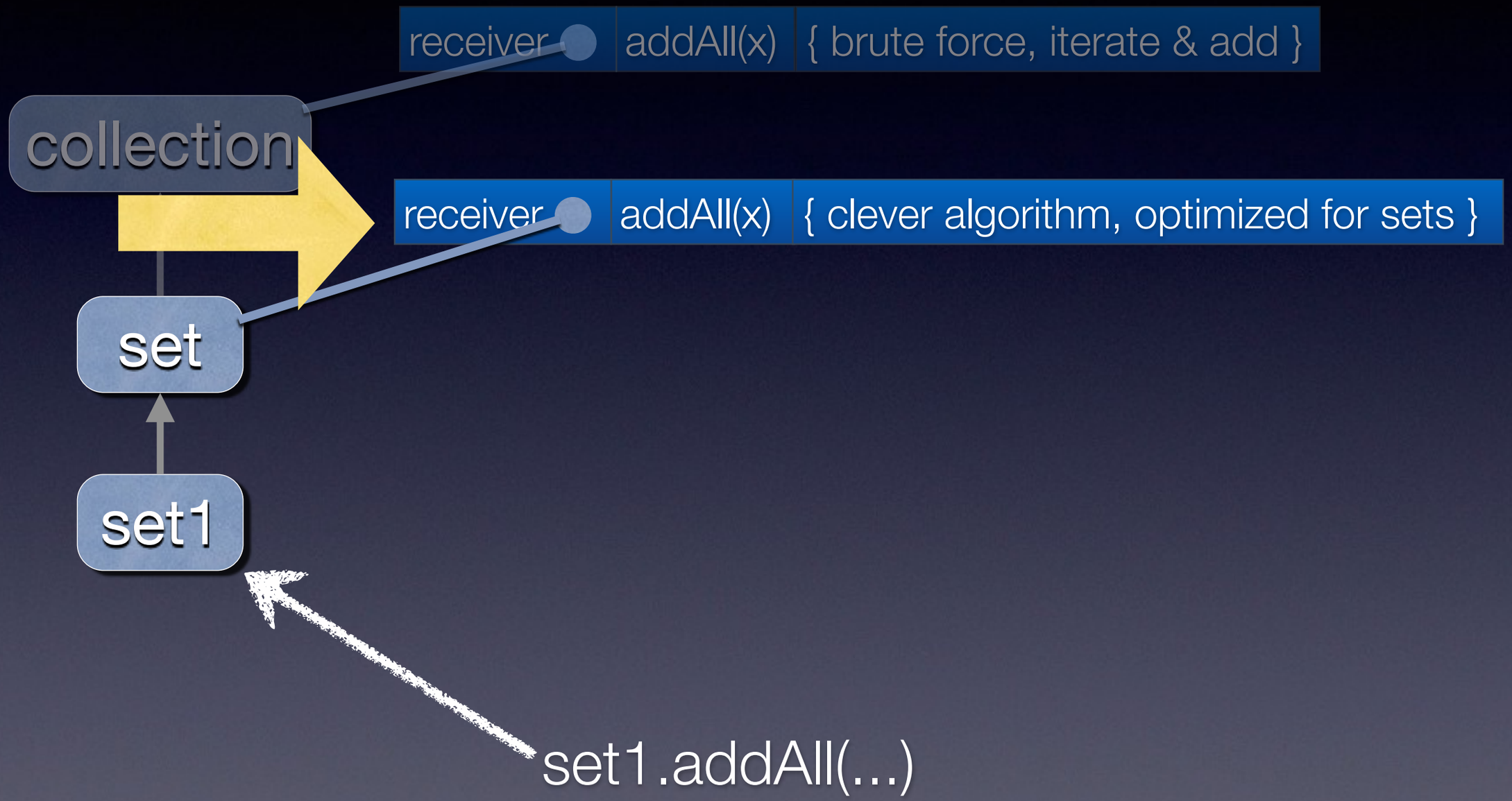
# Most specific slot wins






© 2013, 2014 IBM Corporation

# Most specific slot wins

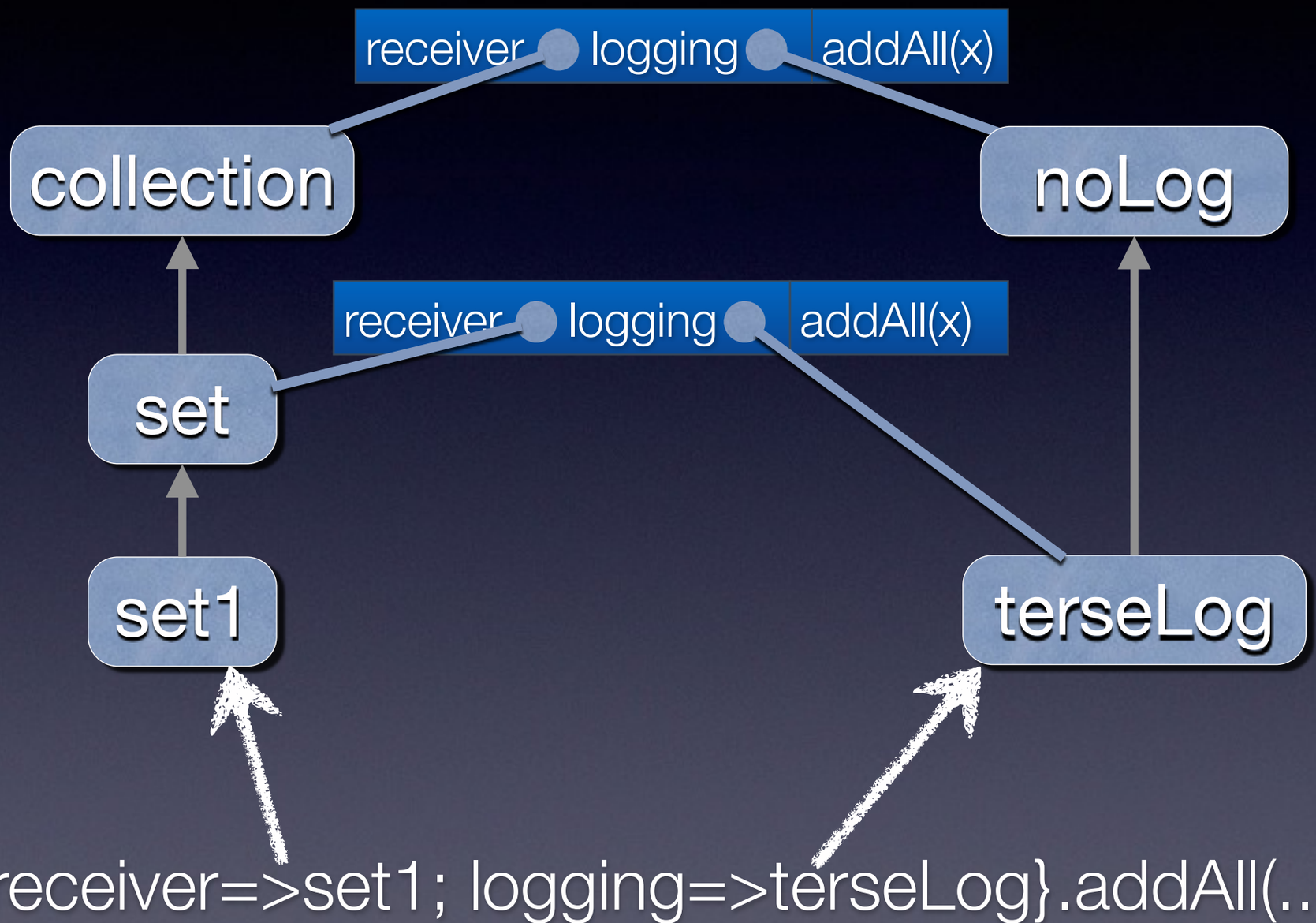



familiar inheritance: children beat parents



© 2013, 2014 IBM Corporation

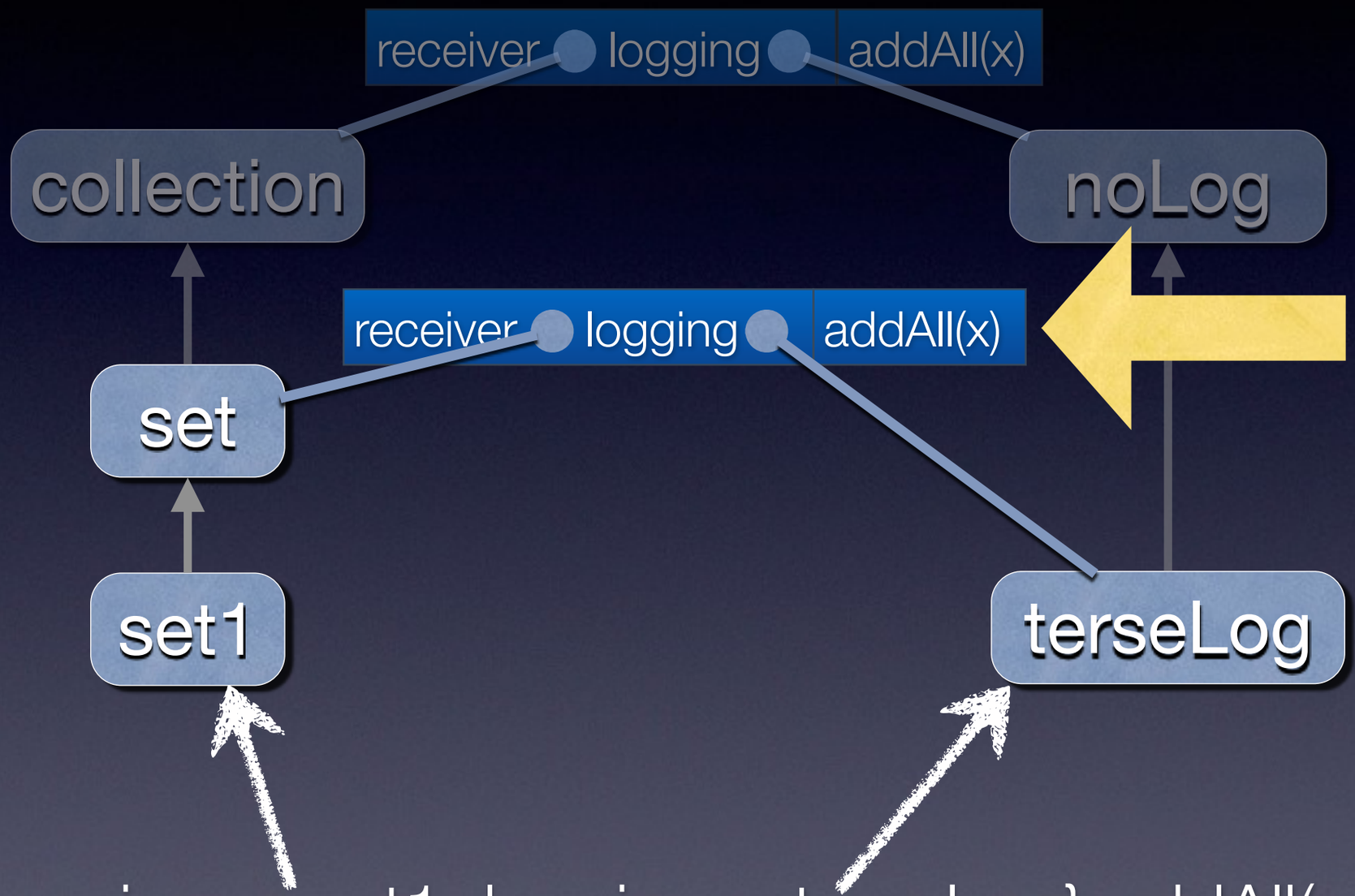
# Most specific slot wins






© 2013, 2014 IBM Corporation

# Most specific slot wins

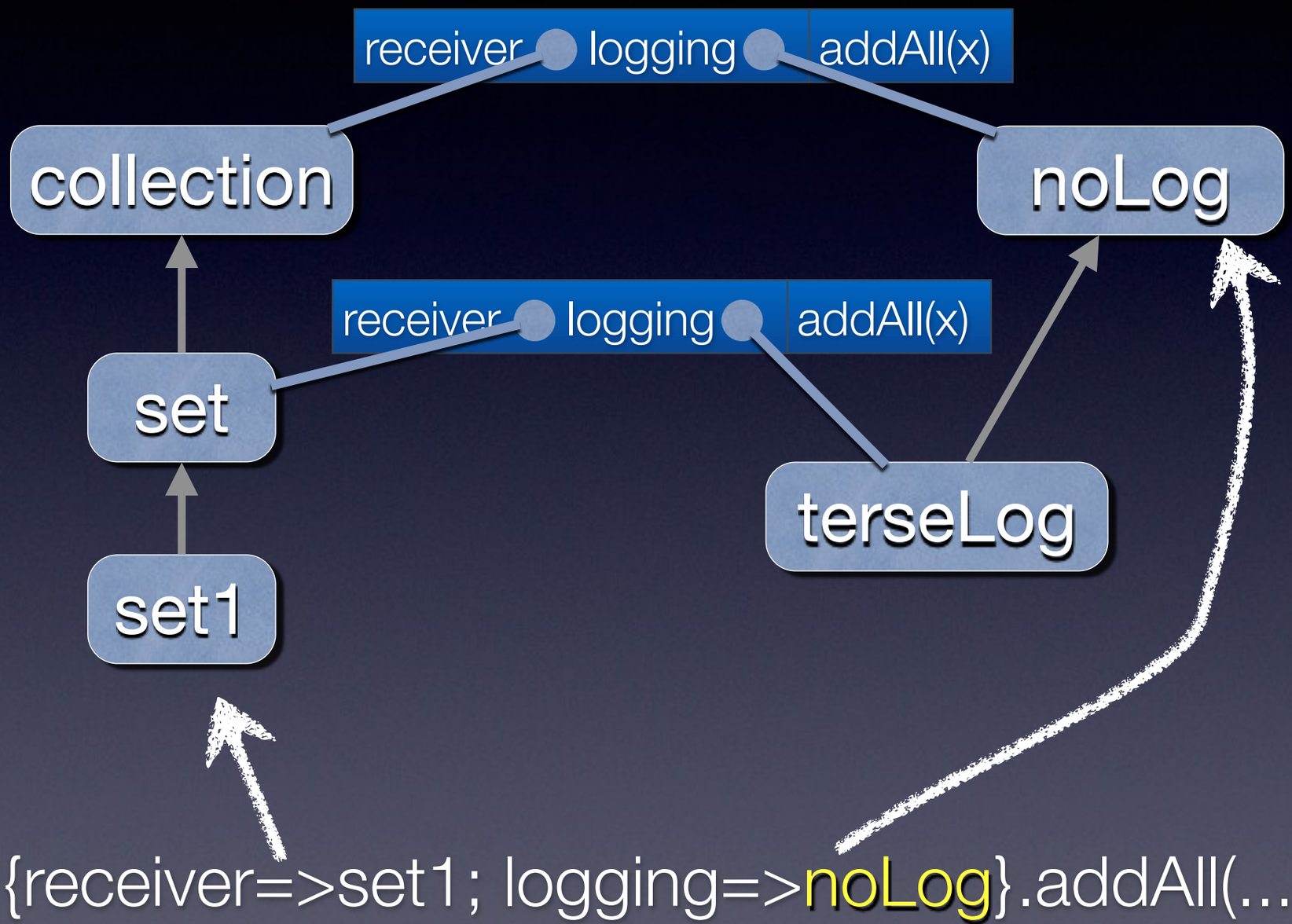



simple: most specific in all dimensions



© 2013, 2014 IBM Corporation

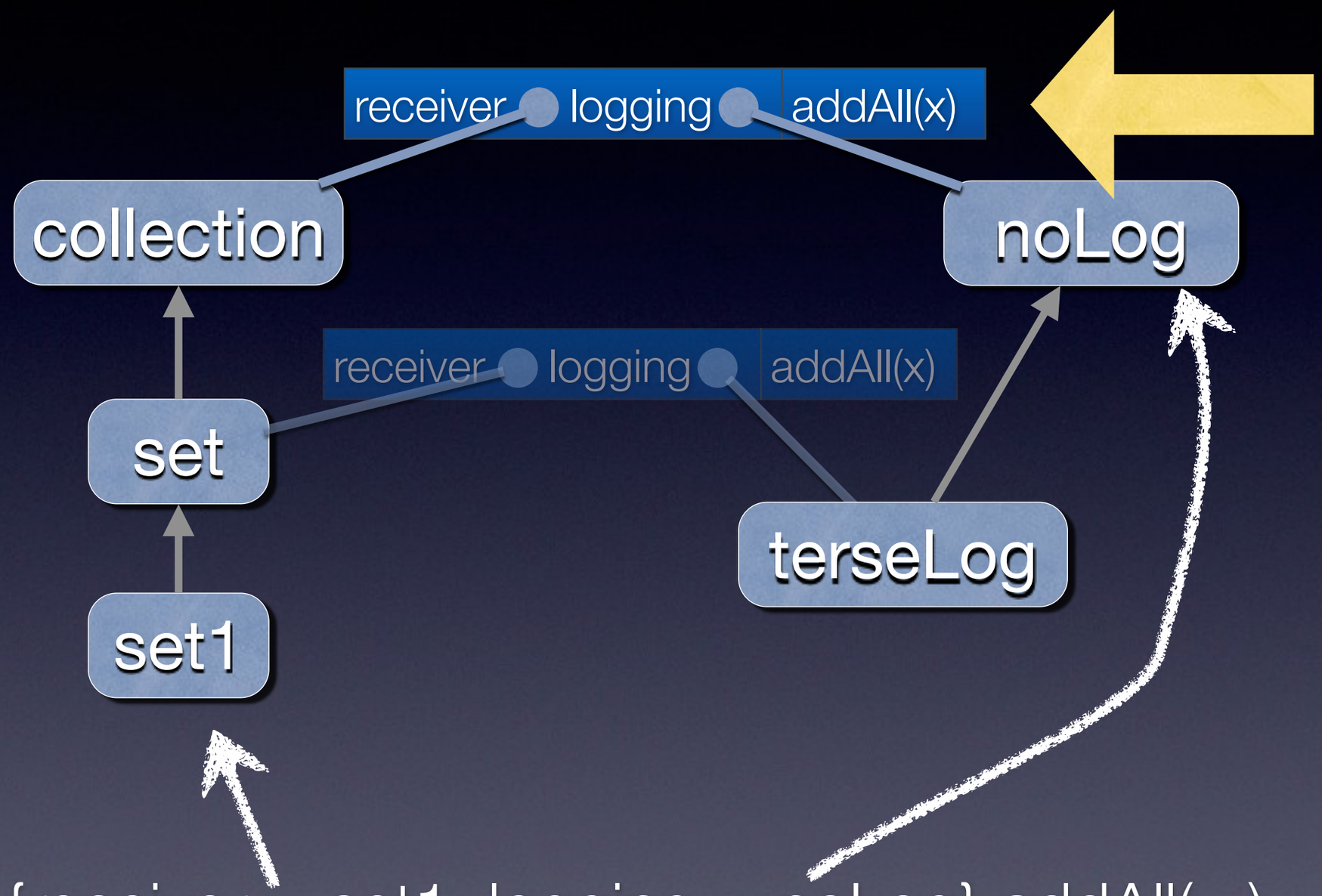
# Must match in all dimensions






© 2013, 2014 IBM Corporation

# Must match in all dimensions

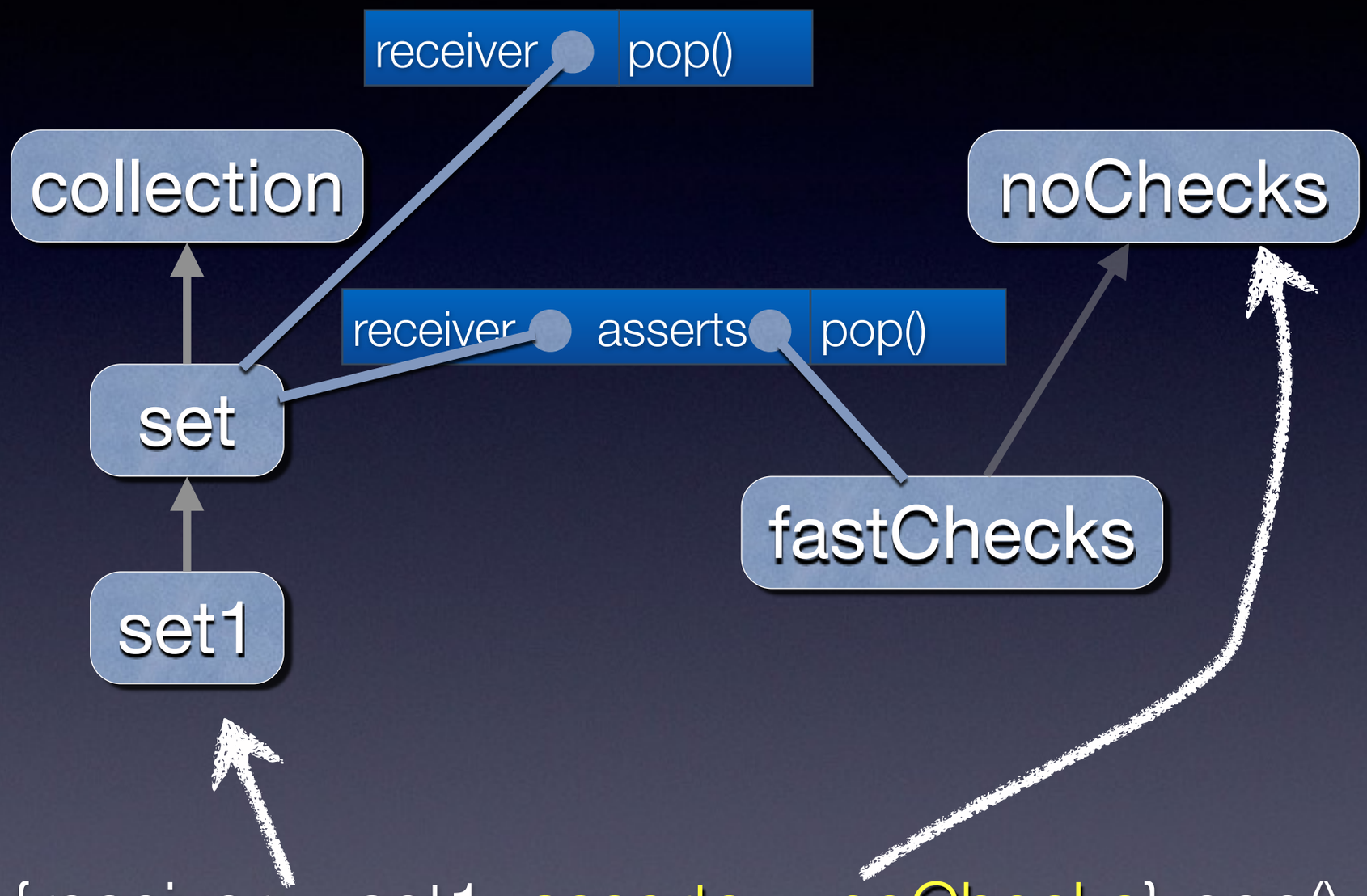



All parts of guard must be satisfied



© 2013, 2014 IBM Corporation

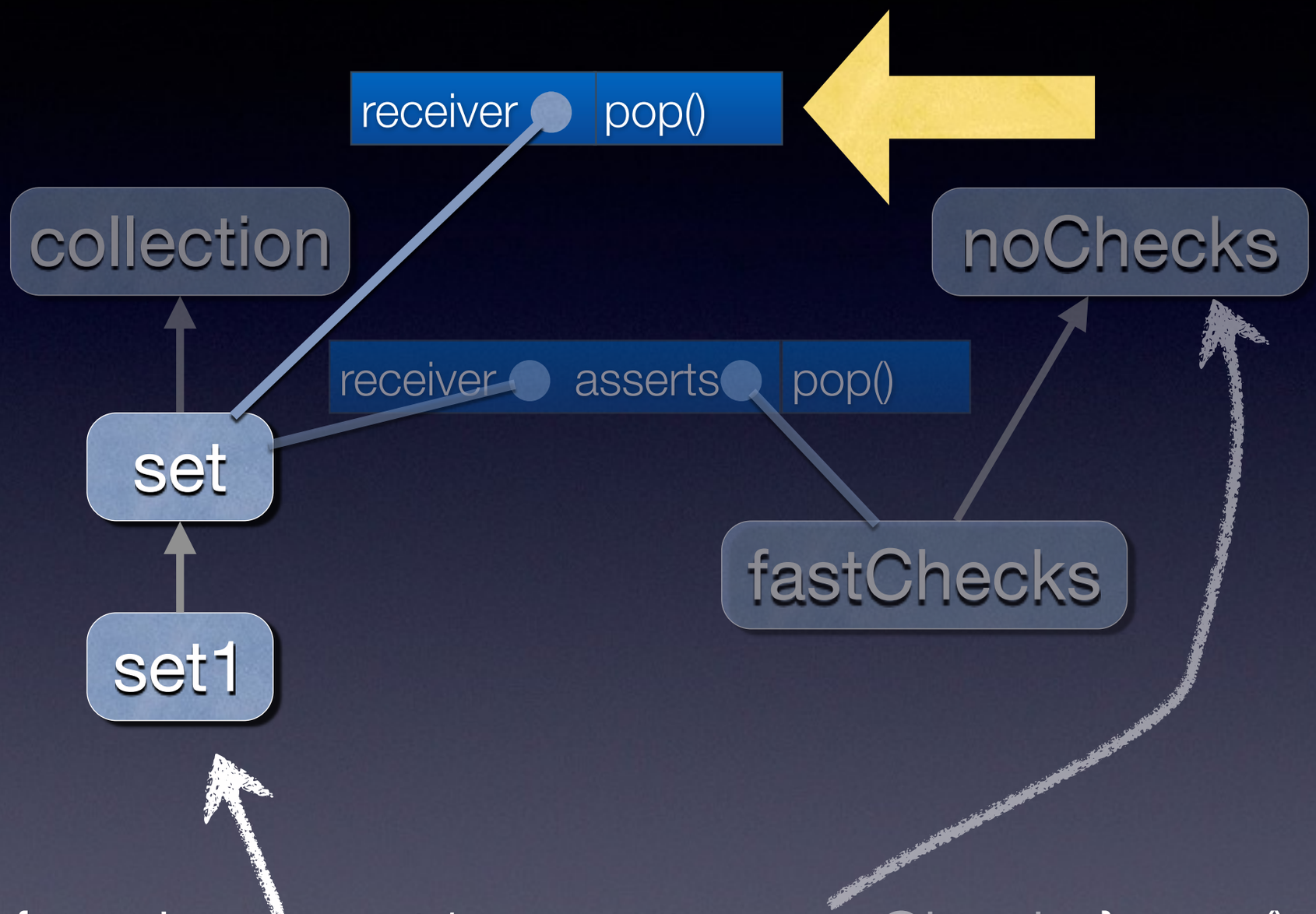
# Unguarded dimension ≡ Don't care






© 2013, 2014 IBM Corporation

# Unguarded dimension ≡ Don't care

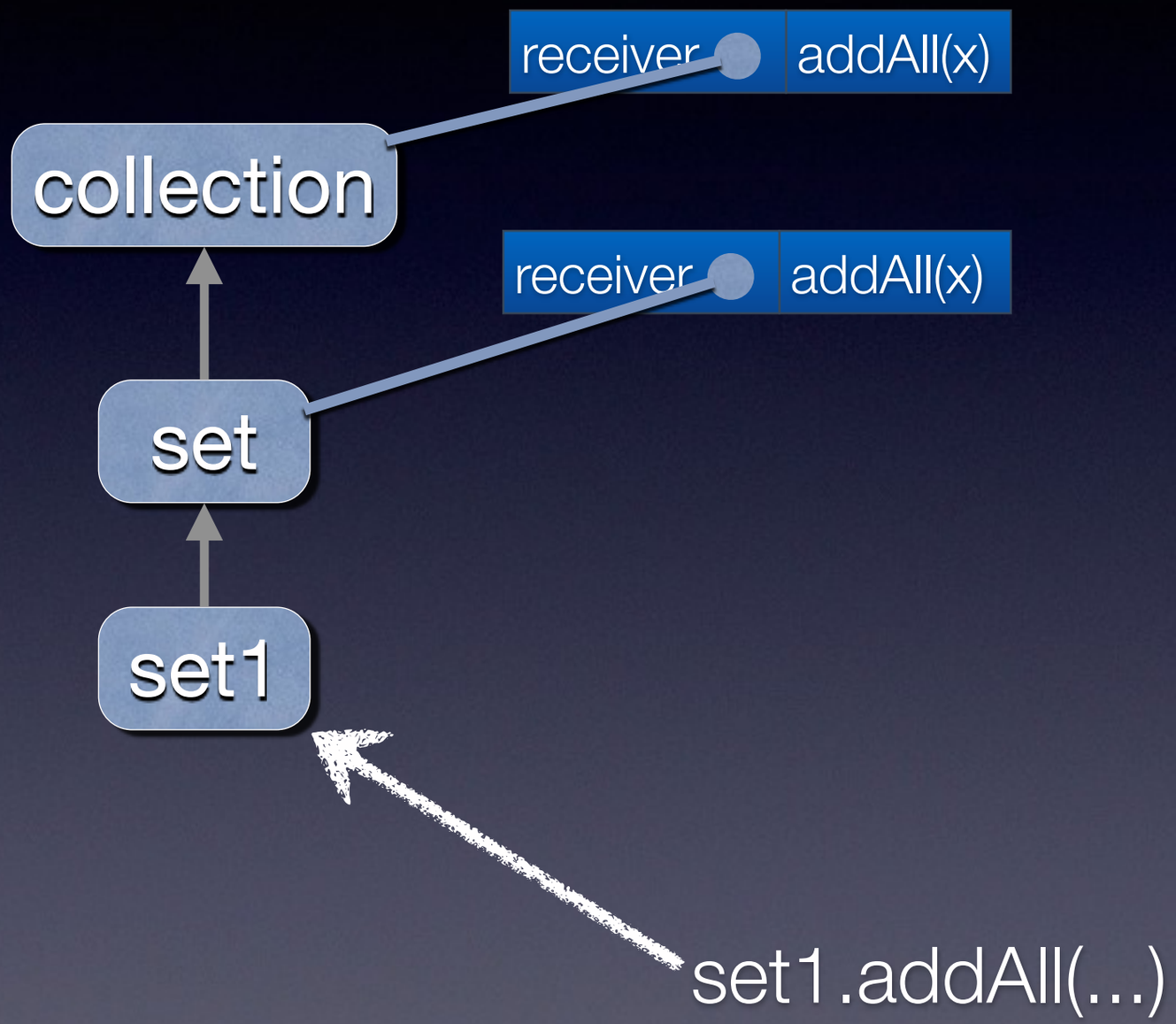



Eases adding dimensions for evolution



© 2013, 2014 IBM Corporation

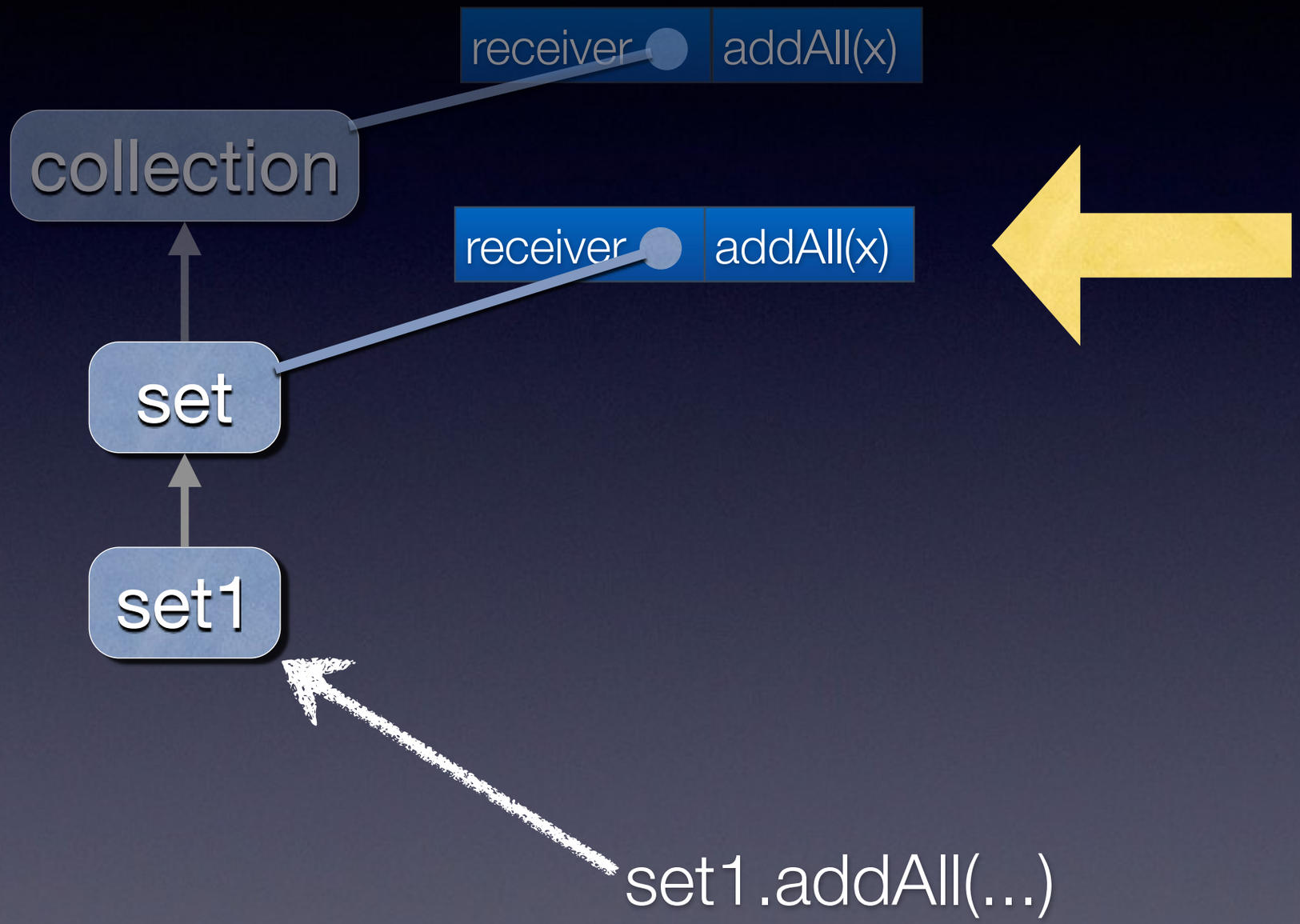
# Recall: Most specific slot wins






© 2013, 2014 IBM Corporation

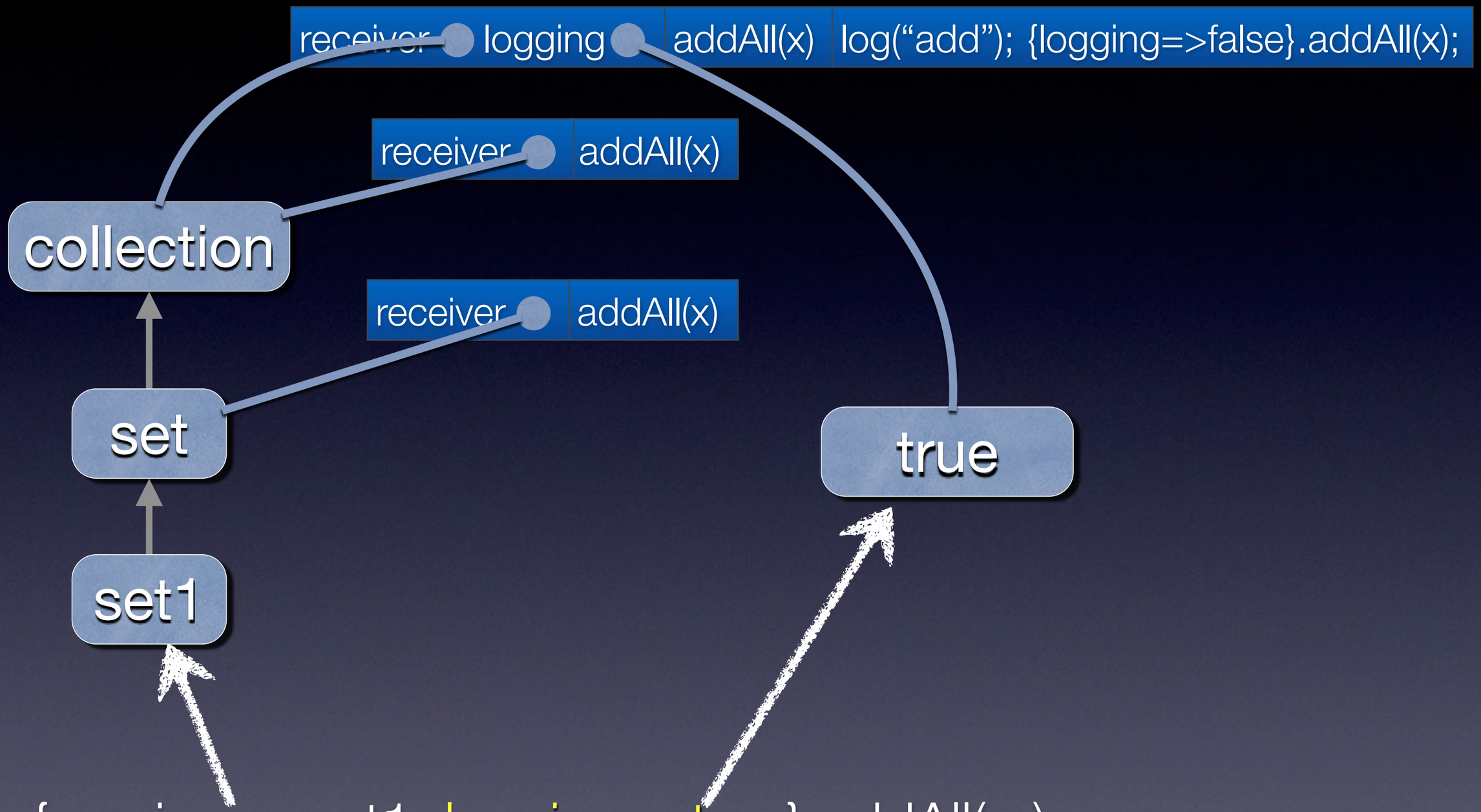
# Recall: Most specific slot wins






© 2013, 2014 IBM Corporation

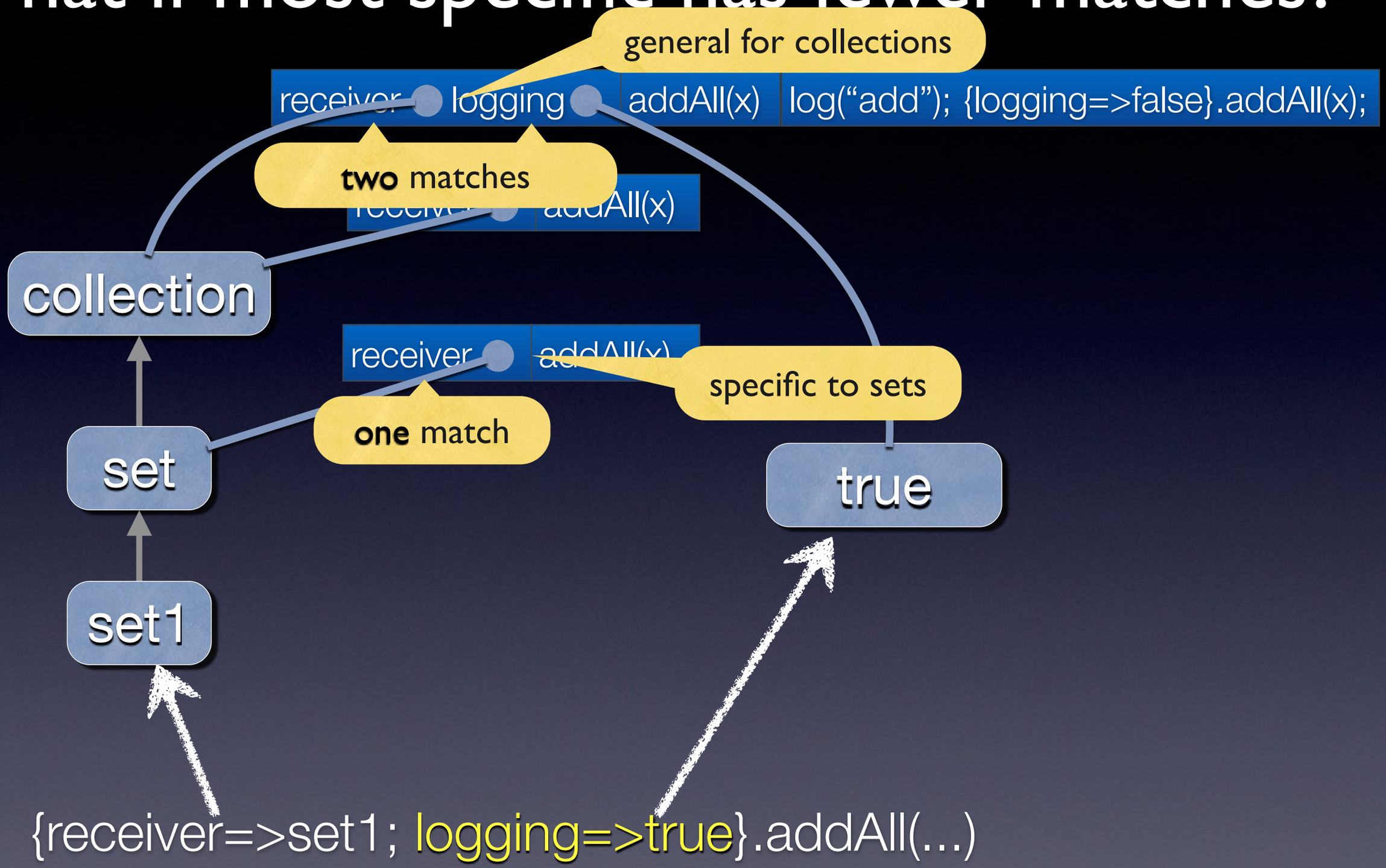
# What if most specific has fewer matches?






© 2013, 2014 IBM Corporation

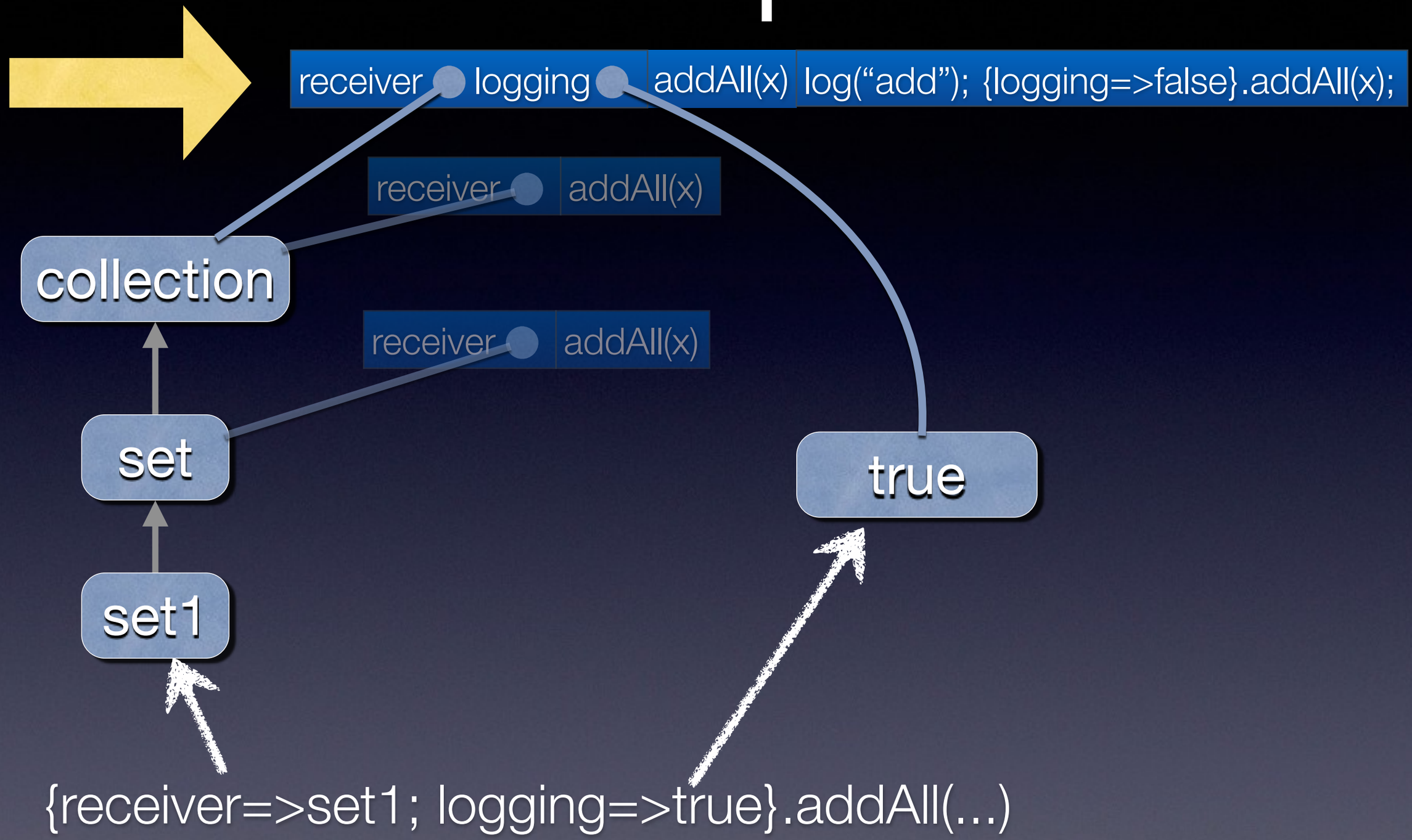
# What if most specific has fewer matches?






© 2013, 2014 IBM Corporation

# Match count trumps inheritance



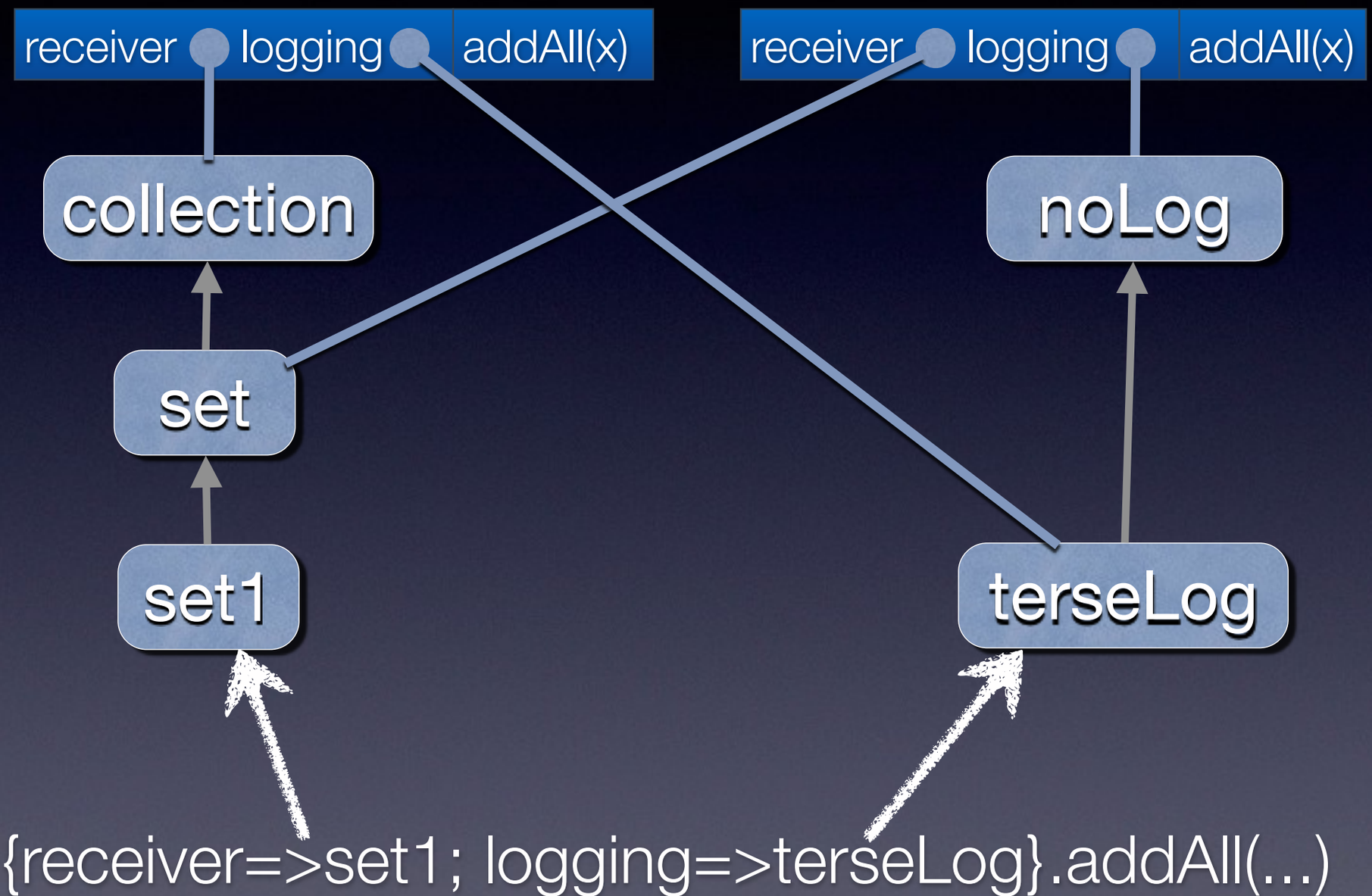

Supports overriding by adding dimensions

Tend to add child before adding dimensions



© 2013, 2014 IBM Corporation

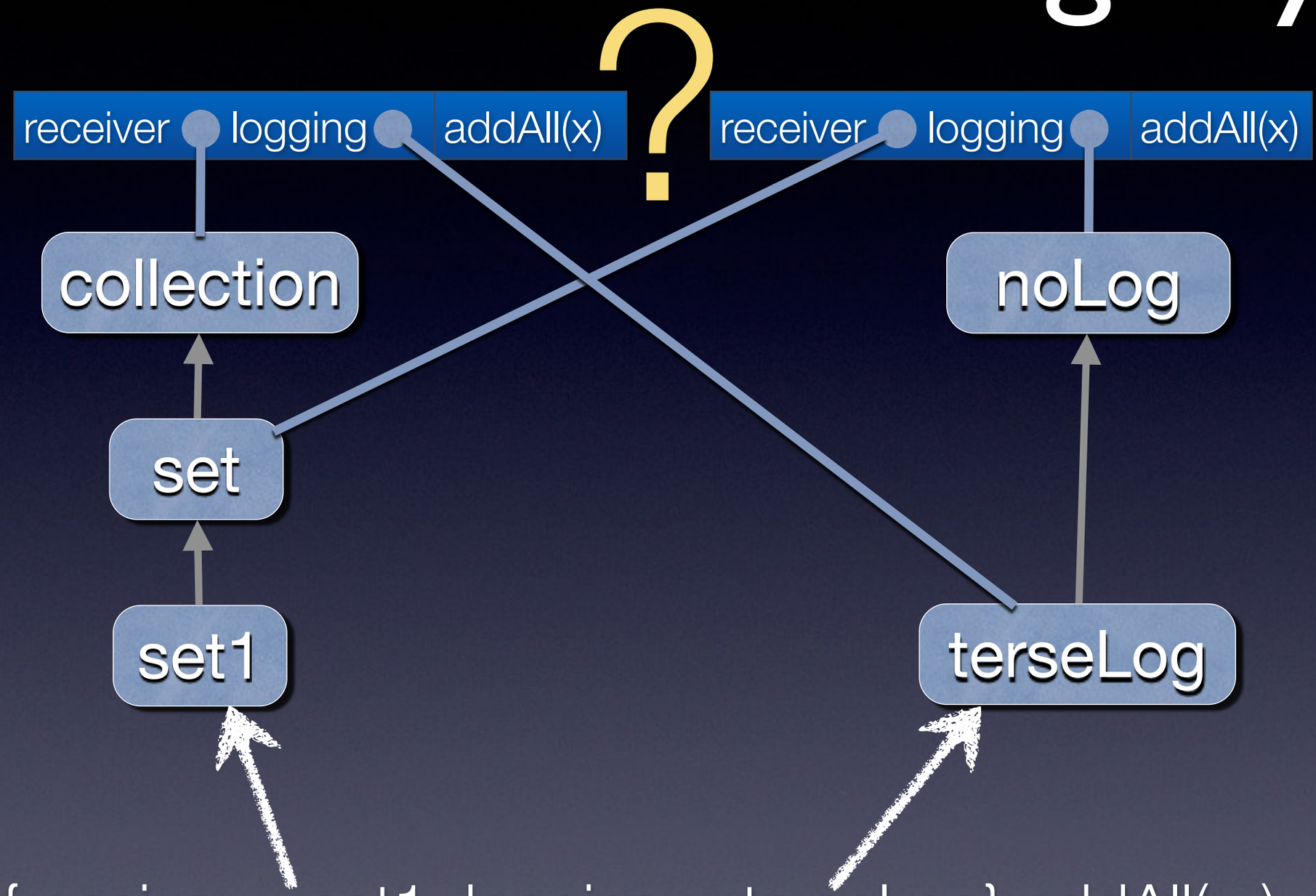
# New kind of ambiguity






© 2013, 2014 IBM Corporation

# New kind of ambiguity




{receiver=>set1; logging=>terseLog}.addAll(...)

All dimensions have equal weight



© 2013, 2014 IBM Corporation

# Summary

- Children beat parents
- New ambiguity results if dimensions conflict
- Omitted dimensions in guard match anything
- Every dimension in guard must be in context
- # of matching dimensions trumps specificity



© 2013, 2014 IBM Corporation

# Benefits

- Add new dimensions without changing every slot guard
- Dimensions can only apply to part of the space
- Reduces to familiar inheritance


© 2013, 2014 IBM Corporation

# Contents

- What's the big idea?
- What's it good for?
- How real is it?
- How does it work?
- Are you guys through yet?

# Contents

- What's the big idea?
- What's it good for?
- How real is it?
- How does it work?
- Are you guys through yet?



© 2013, 2014 IBM Corporation

# Earlier Work with Colleagues

- A Simple and Unifying Approach to Subjective Objects (Smith, Ungar, 1996)
  - Thought experiment
  - No named roles

Subject-Oriented Programming (Harrison, Ossher, 1993)

Multi-Dimensional Separation of Concerns (Tarr, Ossher, Harrison, Sutton, 1999)

  - Static composition rather than dynamic dispatch



© 2013, 2014 IBM Corporation

# Related Work

- Object-Oriented Multi-Methods in Cecil (Chambers, 1992)
  - Didn't look at multiple implicit arguments
- Aspect-Oriented Software Development
- Context-Oriented Programming
  - Layers on top of OOP rather than a symmetric, multi-dimensional model with subjectivity
- Much more...


© 2013, 2014 IBM Corporation

# Korz?!

- General Semantics, by Korzybski
  - Abstraction at all stages of thought
    - Determines perception of reality
      - edges => shapes => faces => friends



- Message context affects everything that runs for that message
  - Intervening code oblivious
  - Both code and data
    - Even changes "object" identity



© 2013, 2014 IBM Corporation

# Future work

- Debugger
- JavaScript-style syntax
- Richer, better syntax for setting context
- Super/resend
  - any dimension? directed on a dimension?
- Fix privileged dimension (rcvr) of linear syntax
- Progressive disclosure in environment
- Replace lexical scoping for scalability in IDE
- Selector just another dimension
- Faster implementation



© 2013, 2014 IBM Corporation

# Summary: Complexity Made Simple

- From one to many (implicit) "receivers"
  - each with a named role (dimension)

*Context*

- Sea-of-slots instead of objects
- Powerful, but complicated
- Interactive environment might tame complexity
  - Provide illusion of objects
  - Gathering slots according to user context



© 2013, 2014 IBM Corporation

# Conclusions

- *Context*
  unstated, fosters reuse & evolution
- *Unified symmetric multiple dimensions*
  **better fit for many problems**
- *Radical language exploration*
  **undiscovered treasures remain**



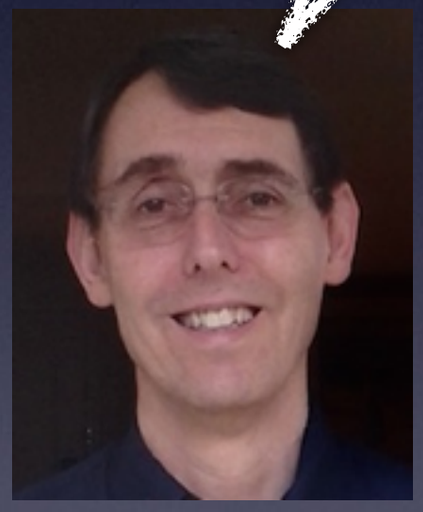
© 2013, 2014 IBM Corporation

# Thank you!

Replace the single implicit receiver with many implicit objects, each playing a different role.



Use multi-dimensional context to both organize and dispatch state and behavior.



Thanks to our colleagues:
Sam Adams, Suparna Bhattacharya, Bob Friedlander, Doug Kimelman, Jim Kraemer, John Richards, Mark Wegman



© 2013, 2014 IBM Corporation

# Backup



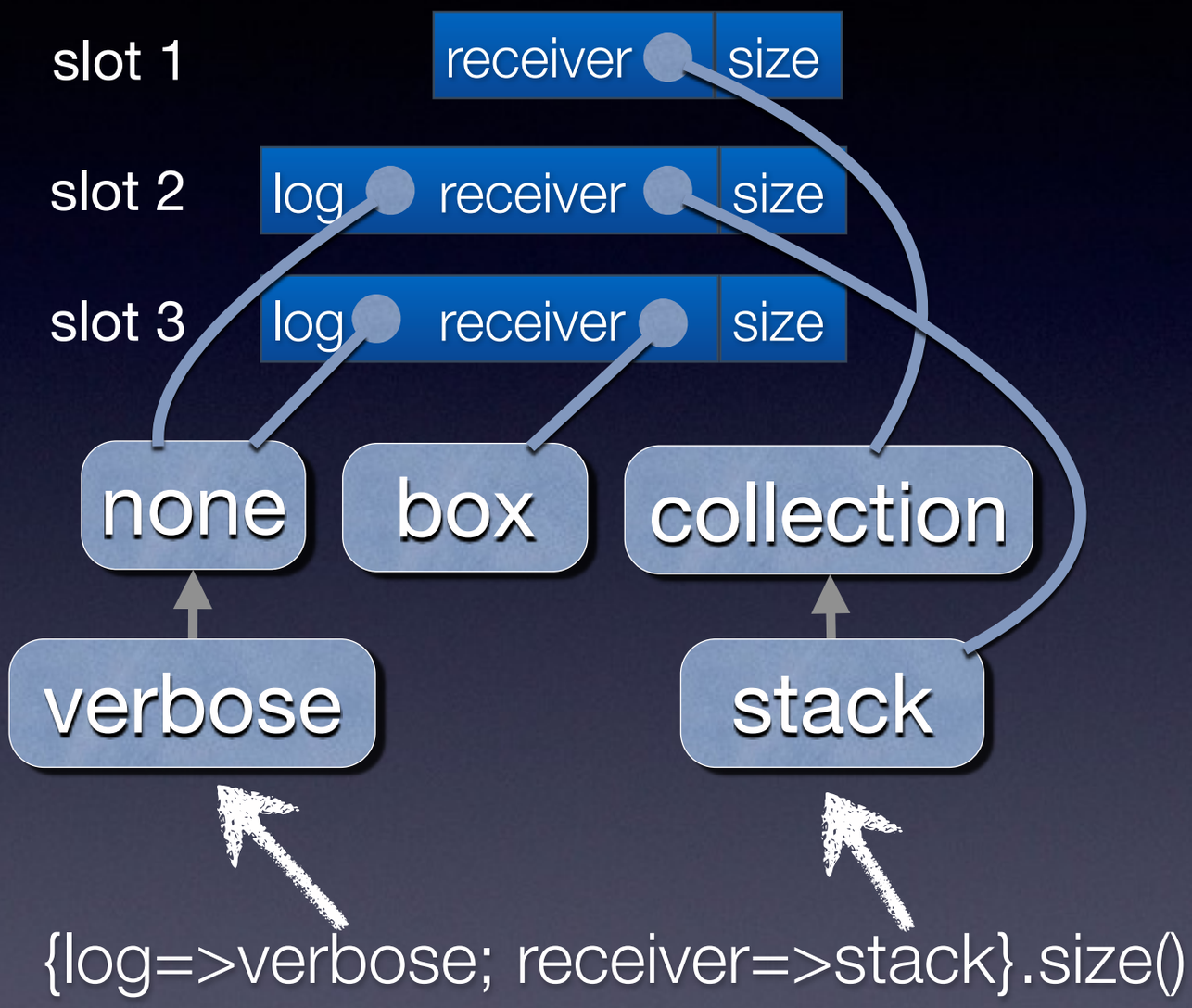
© 2013, 2014 IBM Corporation

# Dispatch algorithm



© 2013, 2014 IBM Corporation

# Lookup steps



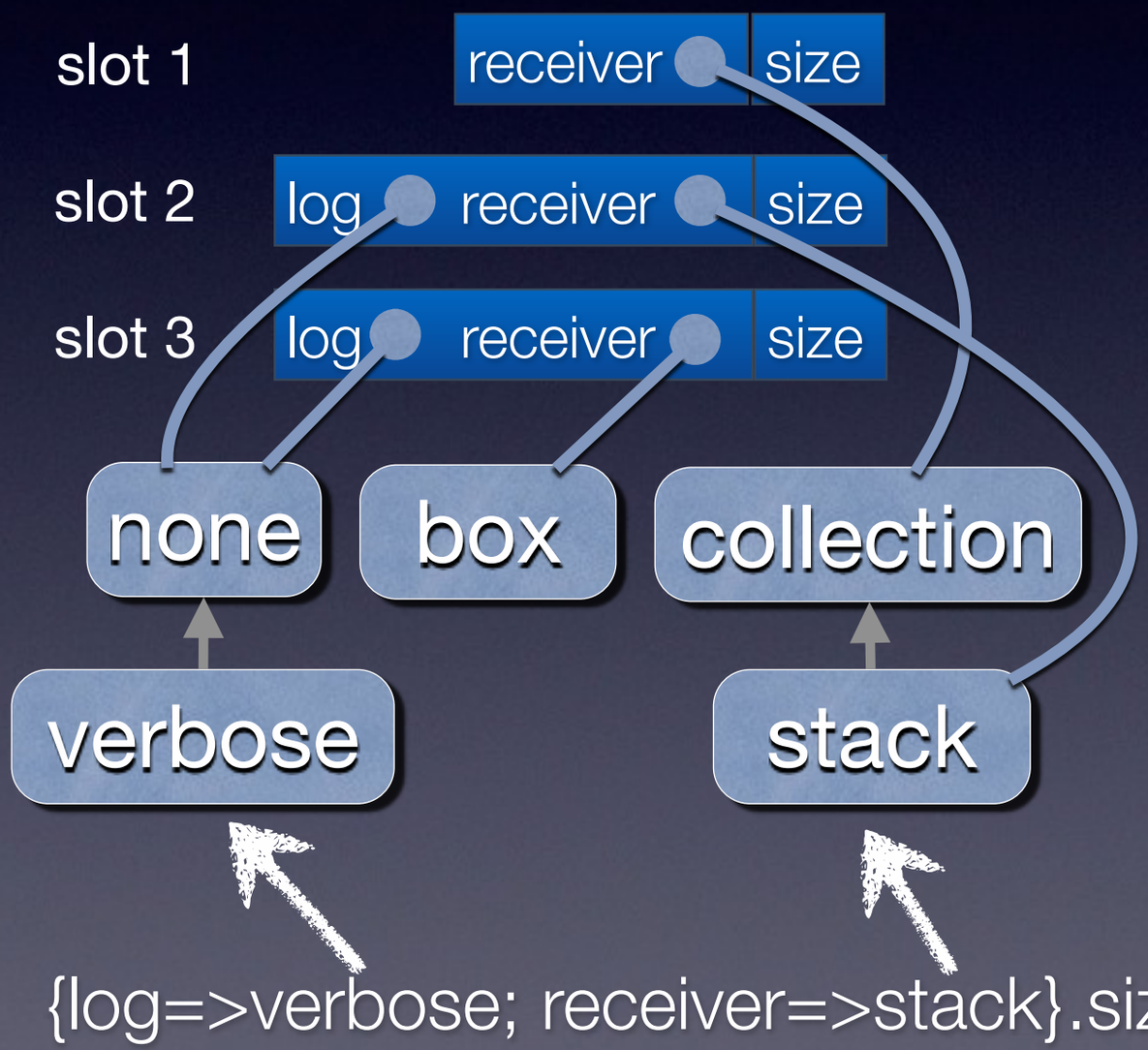

1. Find matches along each dimension of context
2. Consolidate slots
3. Each dimension in guard must be in context
4. Trim less specific slots



© 2013, 2014 IBM Corporation

# 1. Find matches along each dimension of context

| dimension | slot | distance | slot | distance | slot | distance |
|---|---|---|---|---|---|---|
| log | slot 2 | 1 | | | slot 3 | 1 |
| receiver | slot 2 | 0 | slot 1 | 1 | | |

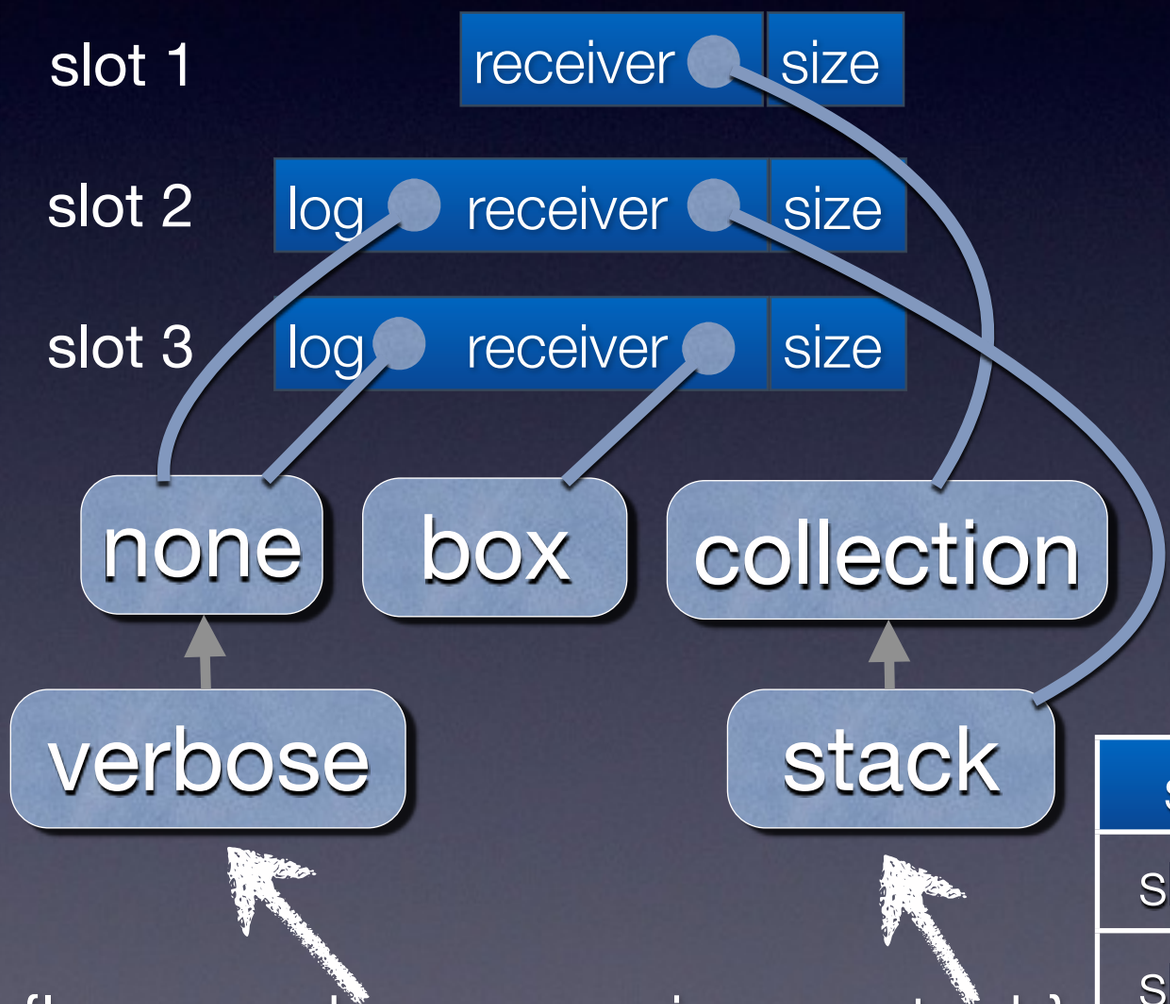





© 2013, 2014 IBM Corporation

# 2. Consolidate slots

| dimension | slot | distance | slot | distance | slot | distance |
|---|---|---|---|---|---|---|
| log | slot 2 | 1 | | | slot 3 | 1 |
| receiver | slot 2 | 0 | slot 1 | 1 | | |



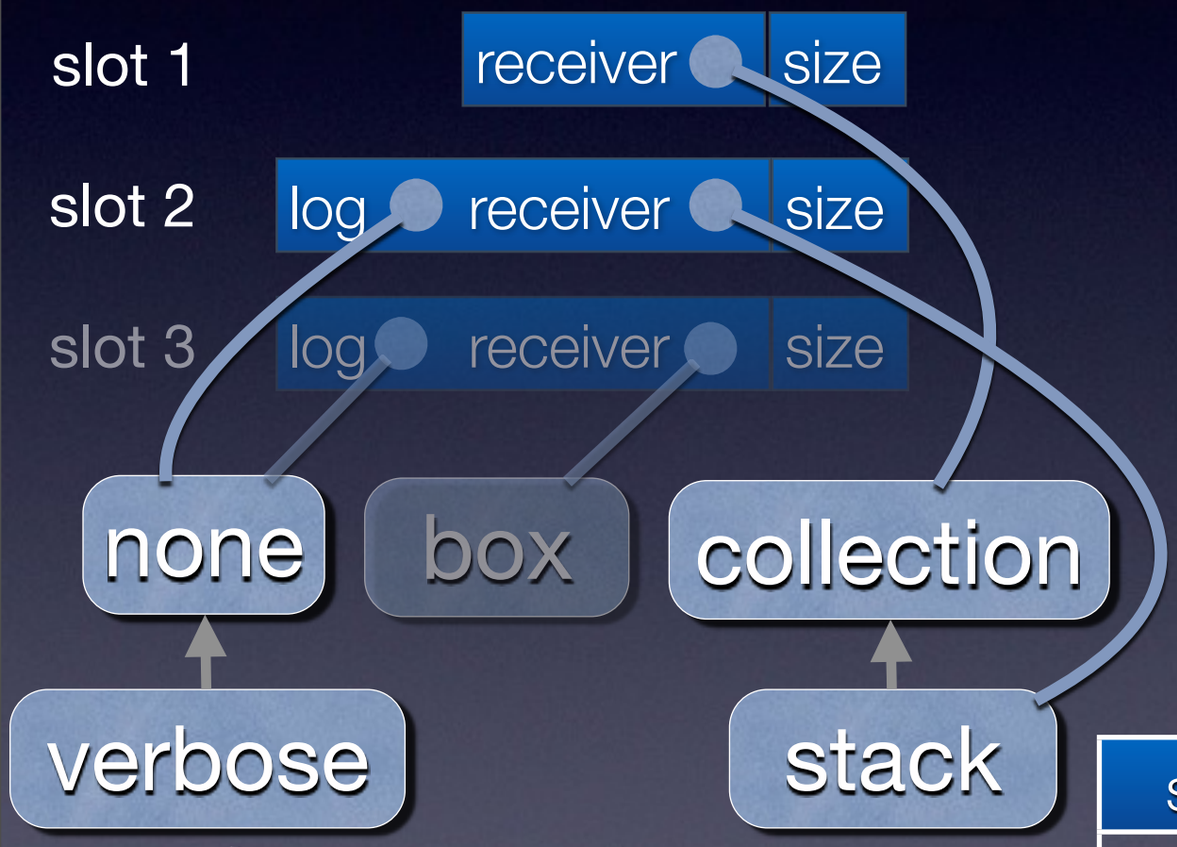

{log=>verbose; receiver=>stack}
.size()

| slot | dimension | distance | dimension | distance |
|---|---|---|---|---|
| slot 1 | receiver | 1 | | |
| slot 2 | receiver | 0 | log | 1 |
| slot 3 | | | log | 1 |



© 2013, 2014 IBM Corporation

# 3. Each dimension in guard must be in context

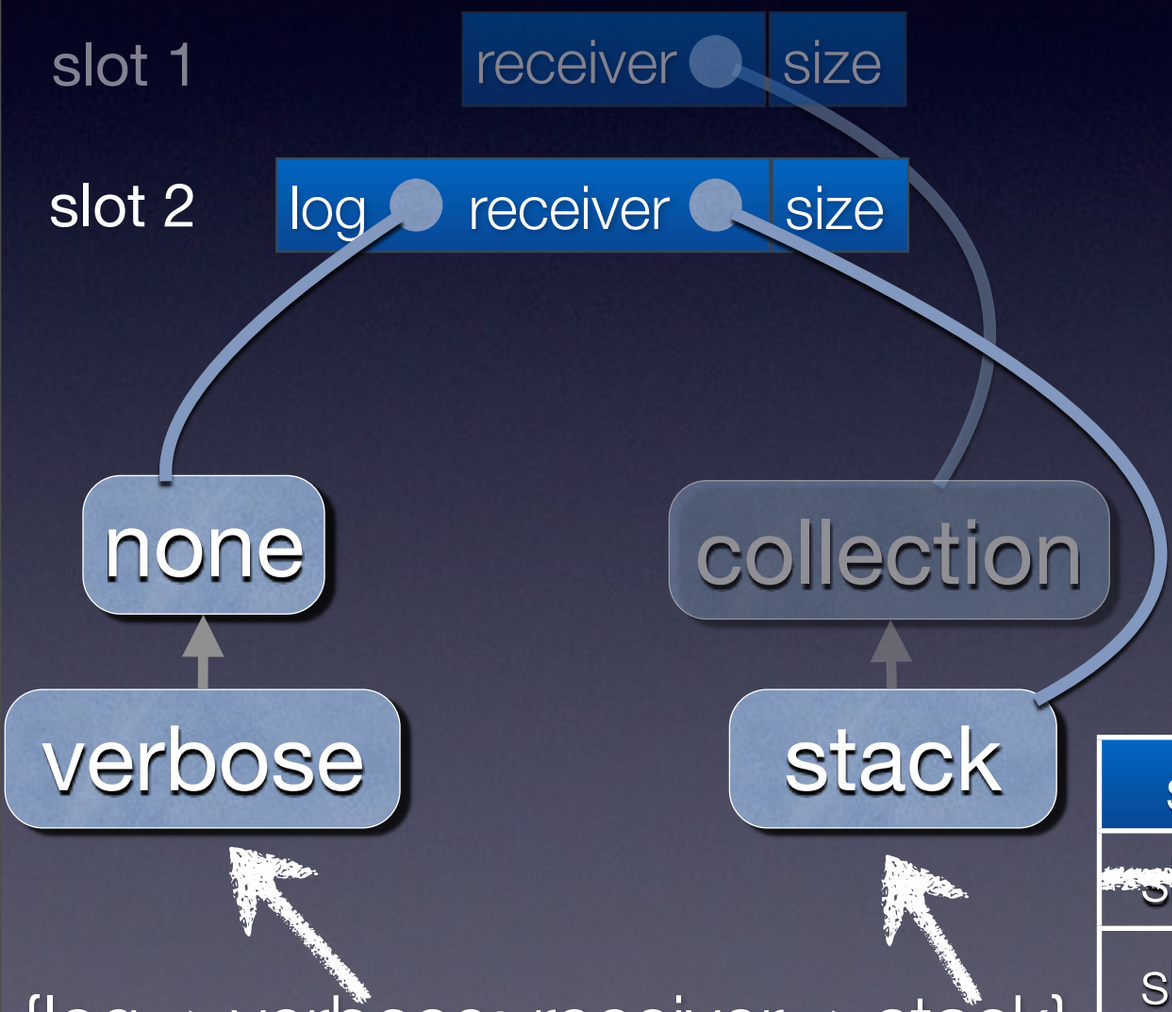



| slot | dimension | distance | dimension | distance |
|---|---|---|---|---|
| slot 1 | receiver | 1 | | |
| slot 2 | receiver | 0 | log | 1 |
| slot 3 | | | log | 1 |



© 2013, 2014 IBM Corporation

# 4. Trim less specific slots



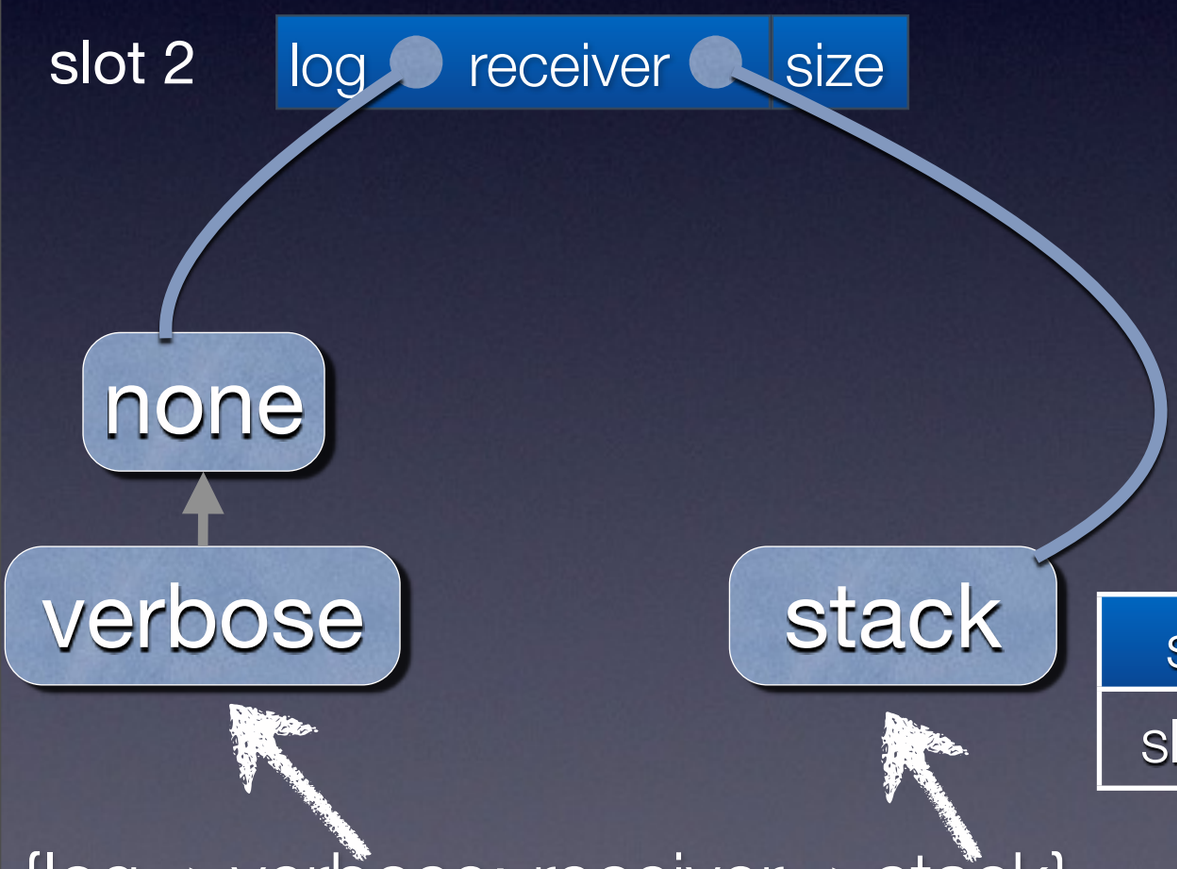

{log=>verbose; receiver=>stack}
.size()

| slot | dimension | distance | dimension | distance |
|---|---|---|---|---|
| ~~slot 1~~ | ~~receiver~~ | ~~1~~ | | |
| slot 2 | receiver | 0 | log | 1 |



© 2013, 2014 IBM Corporation

# Done



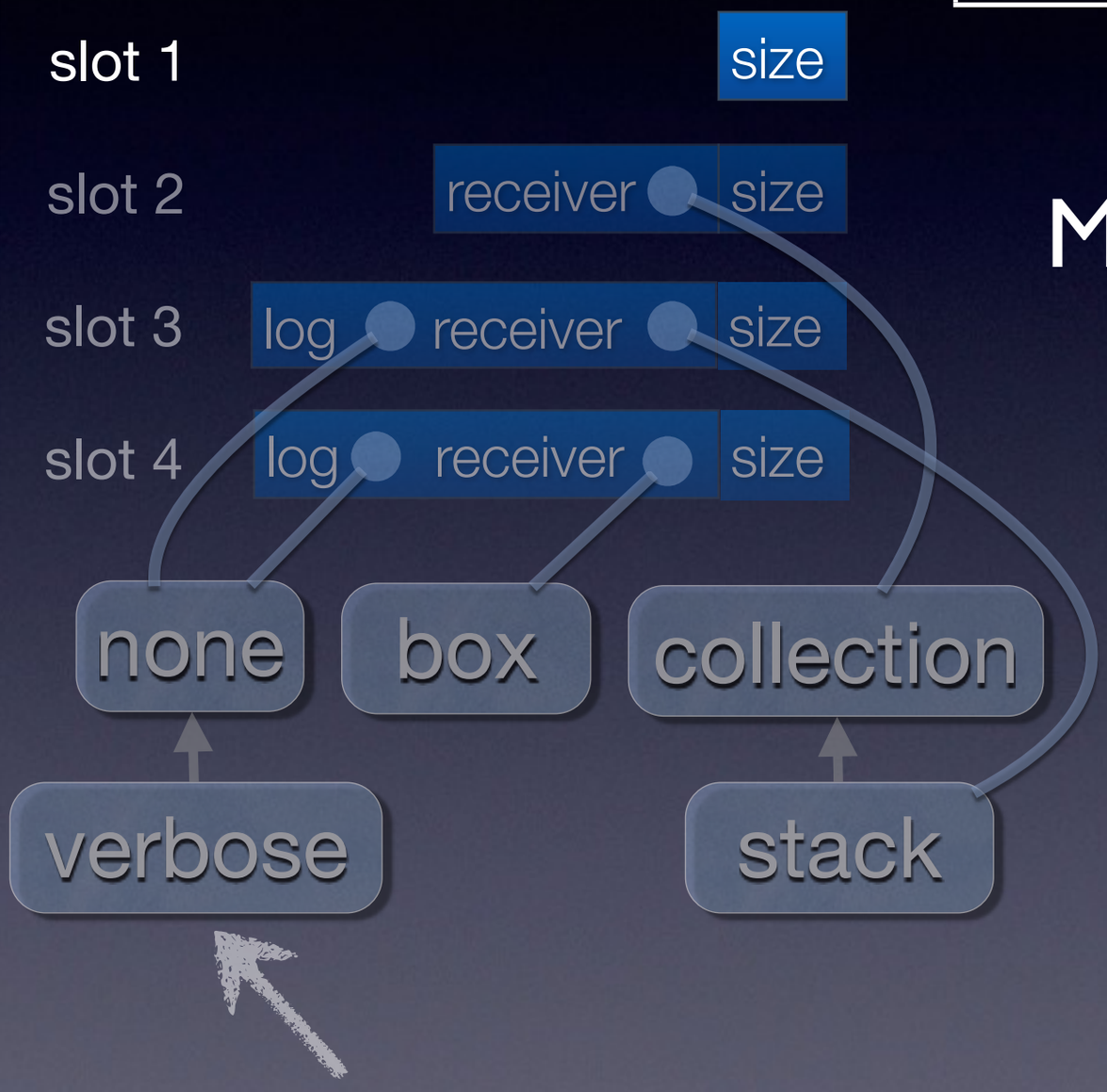

| slot | dimension | distance | dimension | distance |
|---|---|---|---|---|
| slot 2 | receiver | 0 | log | 1 |



© 2013, 2014 IBM Corporation

# 2.1 Undispatched Slots

| slot | dimension | distance | dimension | distance |
|---|---|---|---|---|
| slot 1 | receiver | 1,000 | log | 1,000 |




Must be completely undispatched:
no coordinates in guard

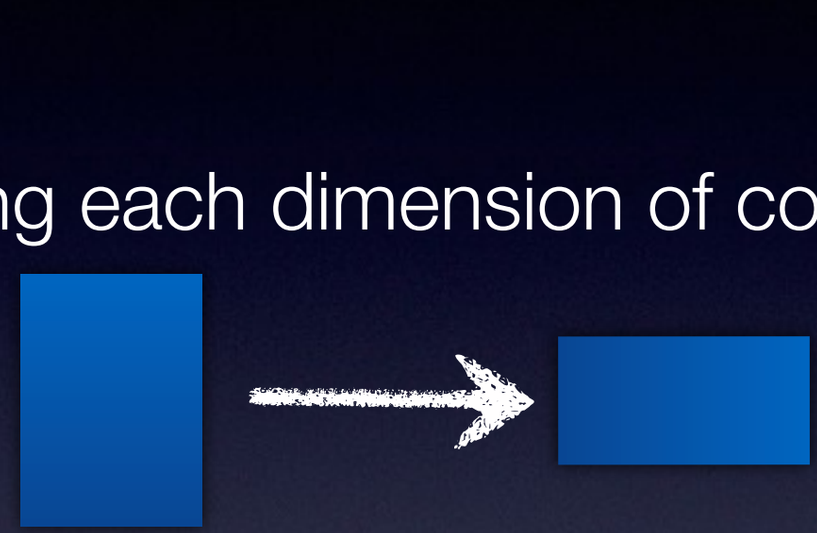
{log=>verbose; receiver=>wobulator}.size()



© 2013, 2014 IBM Corporation

# Lookup summary

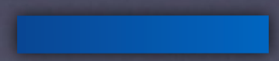
1. Find matches along each dimension of context



2. Consolidate slots
   2.1. Add in undispatched slots
3. Each dimension in guard must be in context



4. Trim less specific slots





© 2013, 2014 IBM Corporation

# AOSD



© 2013, 2014 IBM Corporation

# Aspect-Orientation

- Evolution examples with crosscutting concerns are typical of AOSD
- Many approaches, with special mechanisms, e.g.:
  - adaptive plug-and-play components (Mezini, Lieberherr, et. al.)
  - aspect-oriented programming (AOP) (Kiczales, Lamping, Lopes, et. al.)
  - composition filters (Aksit, Bergmans, et. al.)
  - context-oriented programming (COP) (Hirschfeld, Costanza & Nierstrasz)
  - feature-oriented programming (FOP) (Prehofer) (Batory)
  - multi-dimensional separation of concerns (MDSOC) (Tarr, Ossher, Harrison & Sutton)
  - object teams (Hermann)
  - subject-oriented programming (SOP) (Harrison & Ossher)
  - ...



© 2013, 2014 IBM Corporation

# Korz for AOSD

- Korz can do much *without special mechanisms,* but:
  - Limited *method combination*
    - Overriding only: No most-specific slot → ambiguity error
  - Limited *pointcuts* (common code at multiple join points)
    - Flexible parenting, but not selector-based patterns
- Possible Korz extensions:
  - Open point for custom handling of ambiguity
  - Selector as a dimension with pattern matching
- Aren't these extensions special mechanisms?
  - No! Fit the overall model, general, learned incrementally
- Good basis for AOSD, but more research is needed!



© 2013, 2014 IBM Corporation