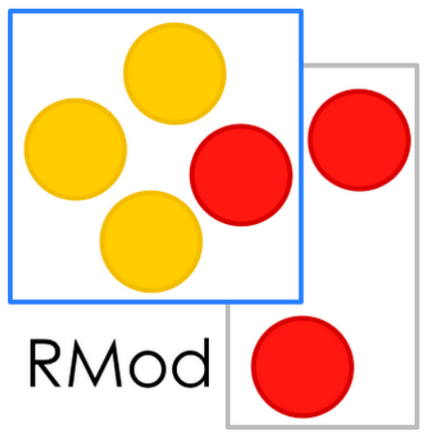
Stack frames and contexts relationships

VM presentation

Théo Rogliano
Inria, Univ. Lille, CNRS, Centrale Lille,
UMR 9189 - CRIStAL
Lille France

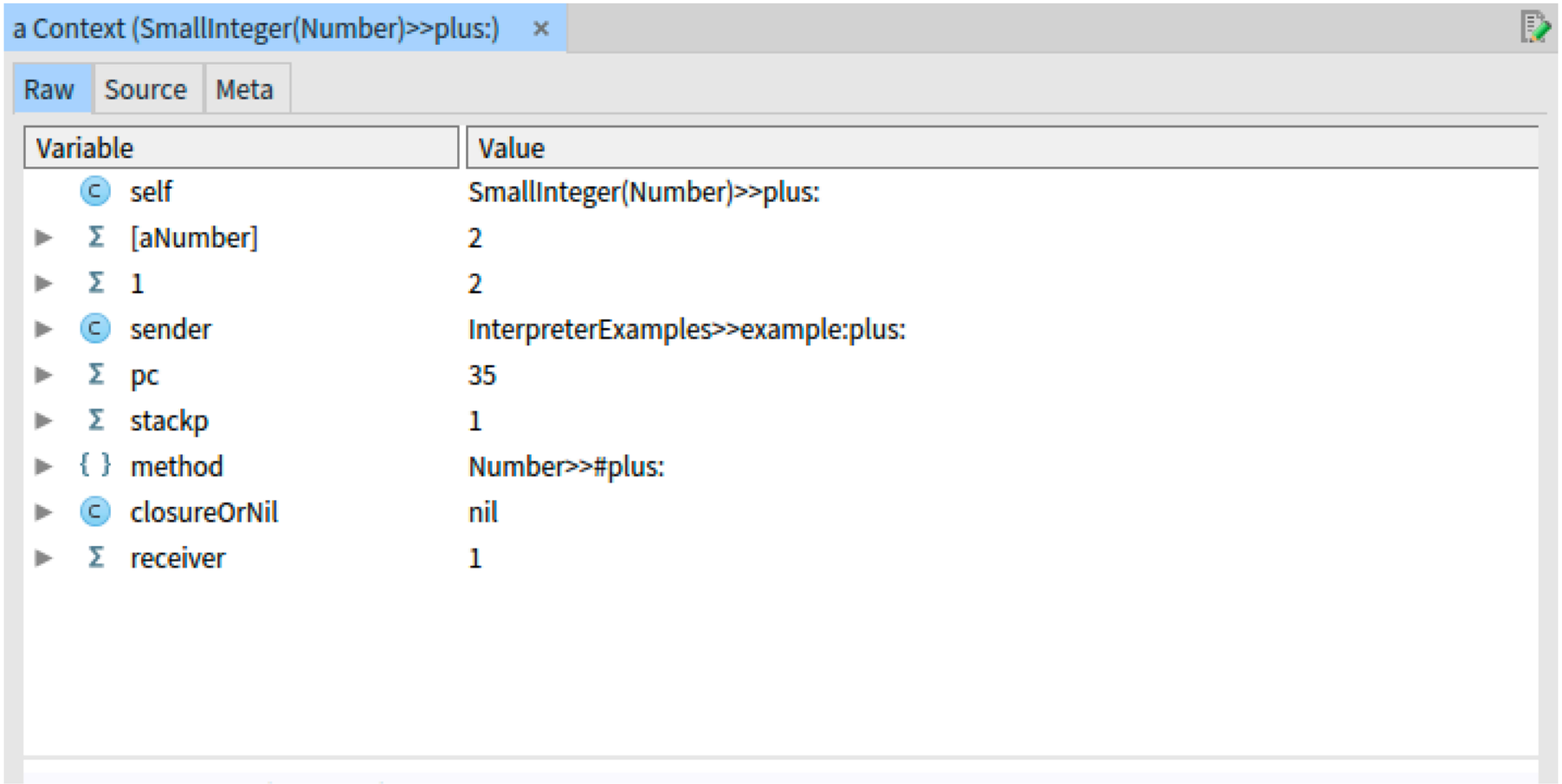


Plan

- Basis
- Stack frames versus list of contexts
- Mapping between stack frame and context
- Stack frame and context lives (better that tele novelas)
- Some examples

Basis : Context

An object representing method or closure activation



Basis : Execution = linked list of contexts

Basis : Reminder

http://rmod-files.lille.inria.fr/Team/Presentations/2020-VM-Rogliano-interpreterBytecode.pdf

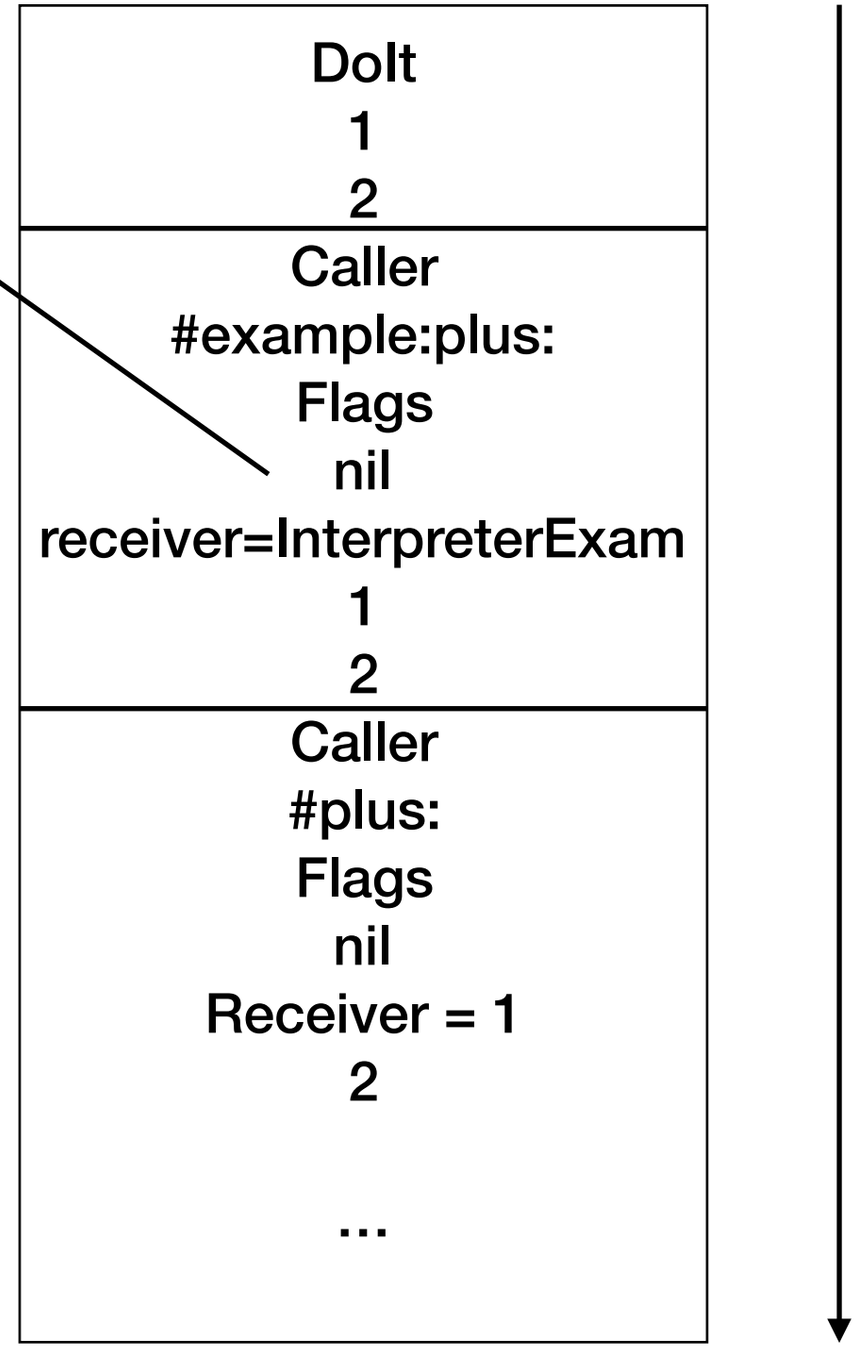
- **Execution as stack**
- **Stack frames**

```
InterpreterExamples new example: 1 plus: 2
```

```
example: aNumber plus: aNumberToAdd

    ^aNumber plus: aNumberToAdd
```

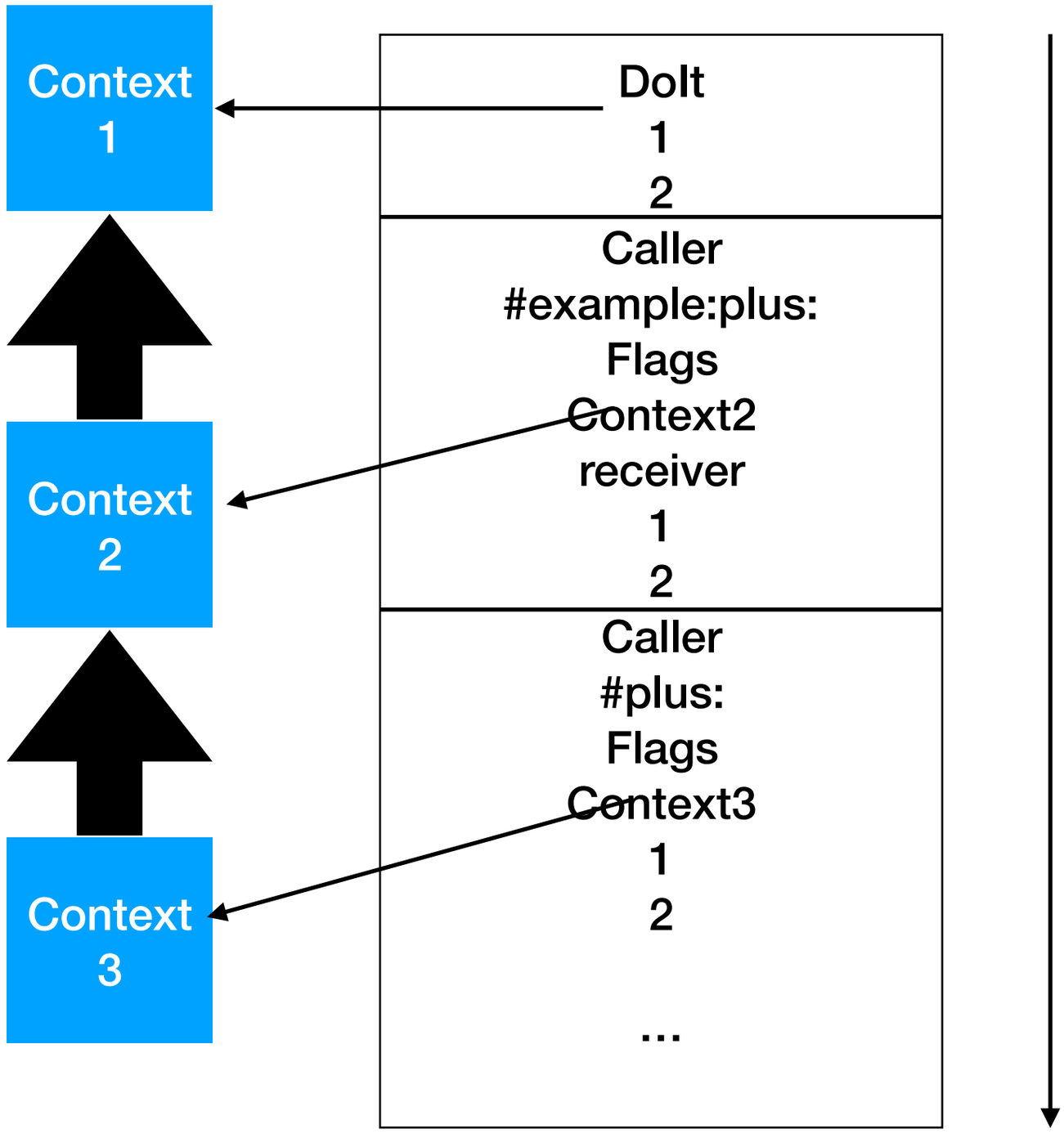
Wherels

Or a context



Two visions of the execution

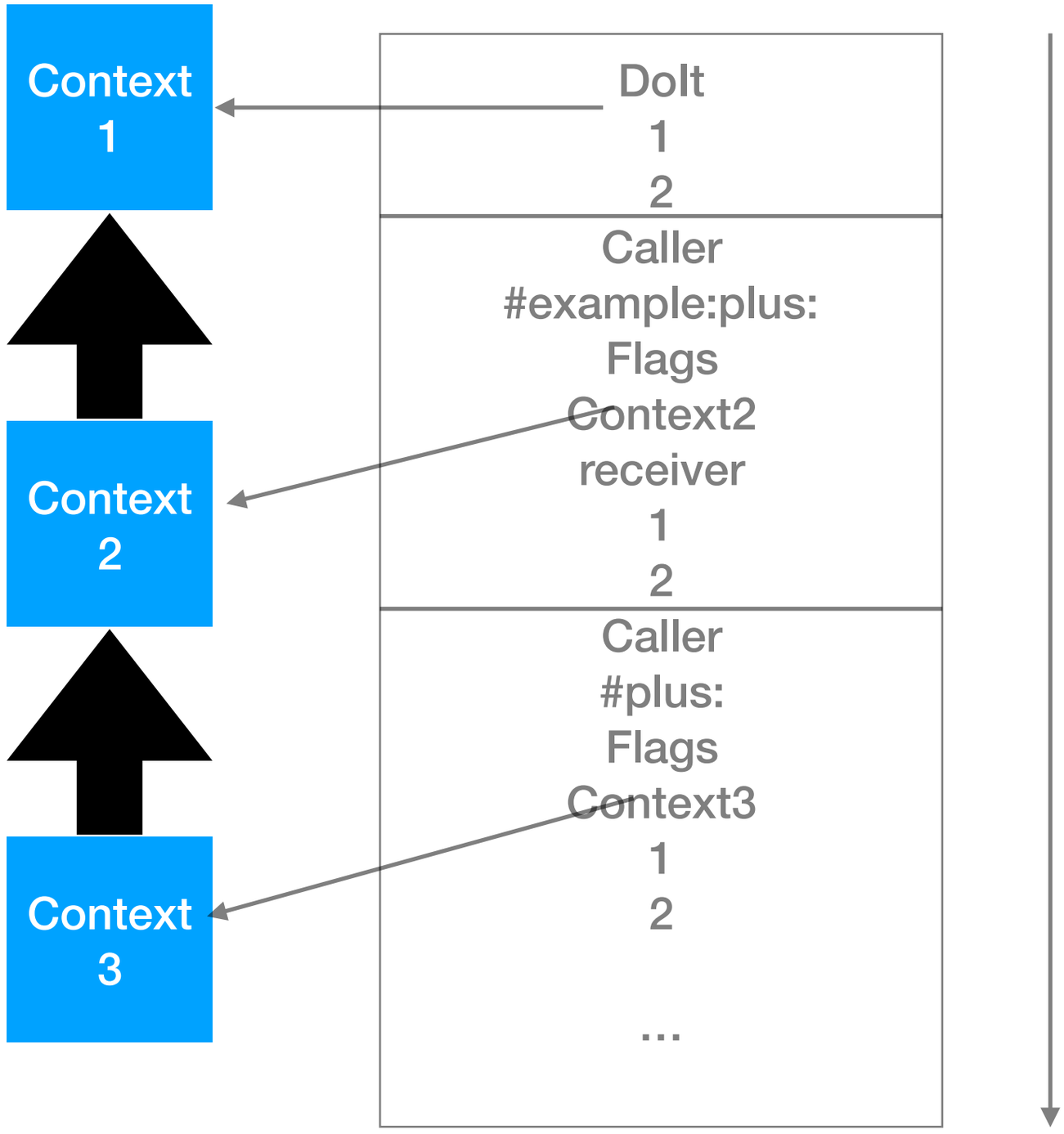
Why two representations of execution ?



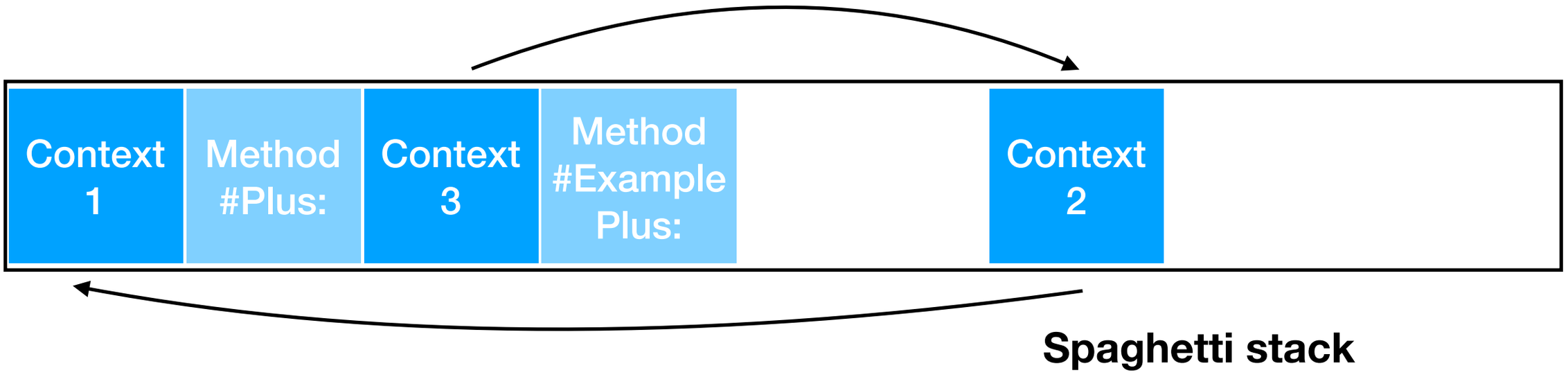
Usage of contexts

Objects (send messages)

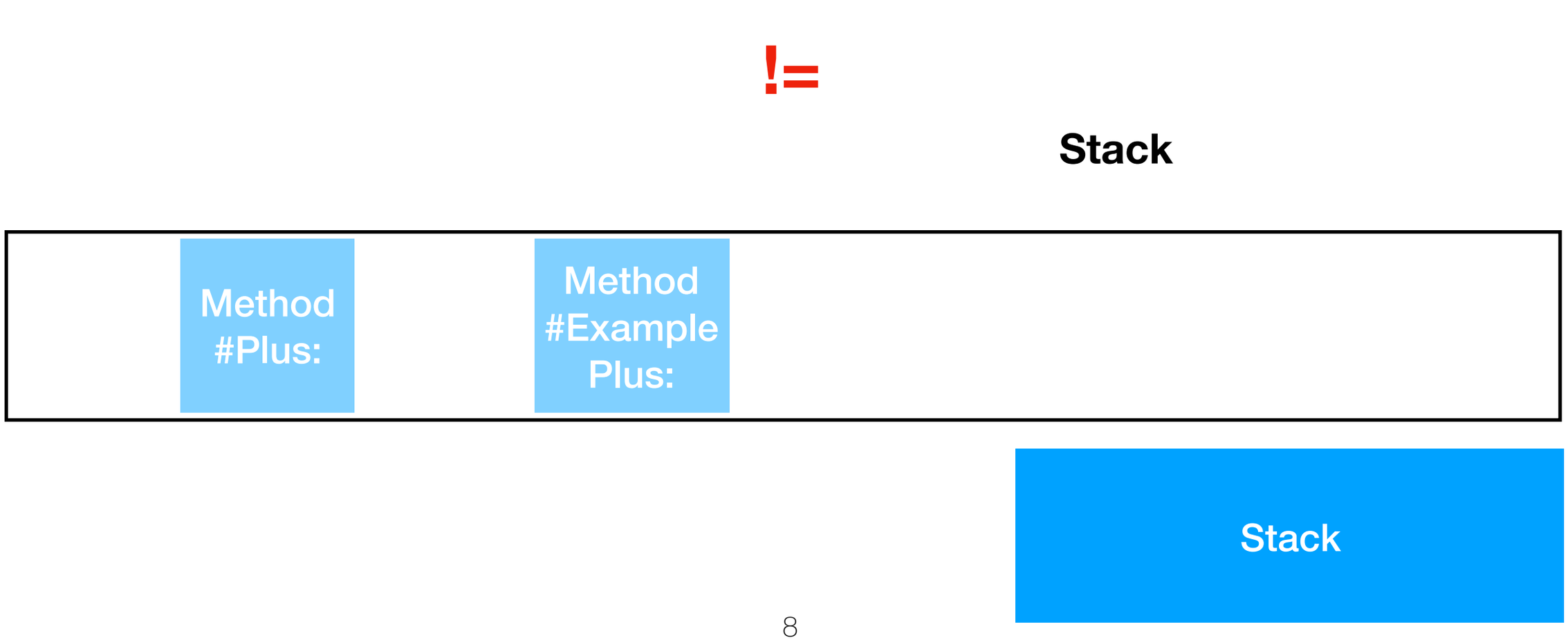
1. **Debugging**
2. **Exception**
3. **Snapshot**
4. **…**



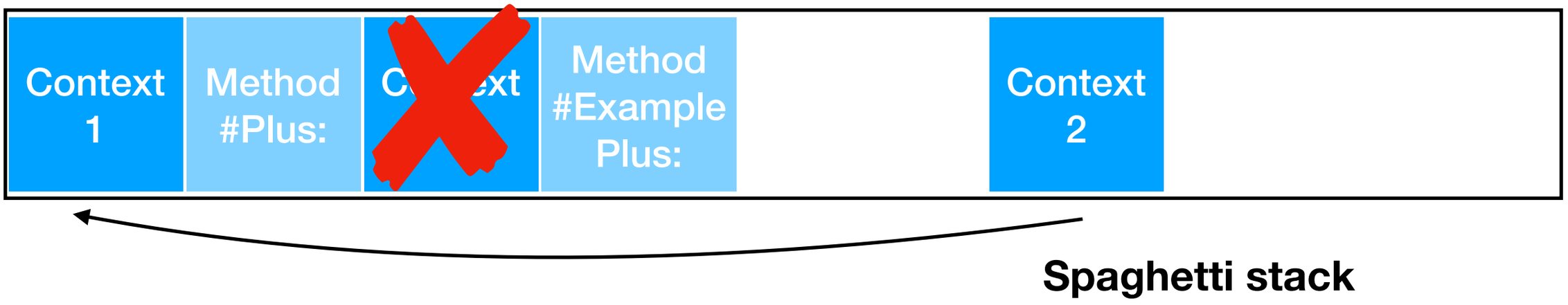
Stack is localised (!= spaghetti)



!=

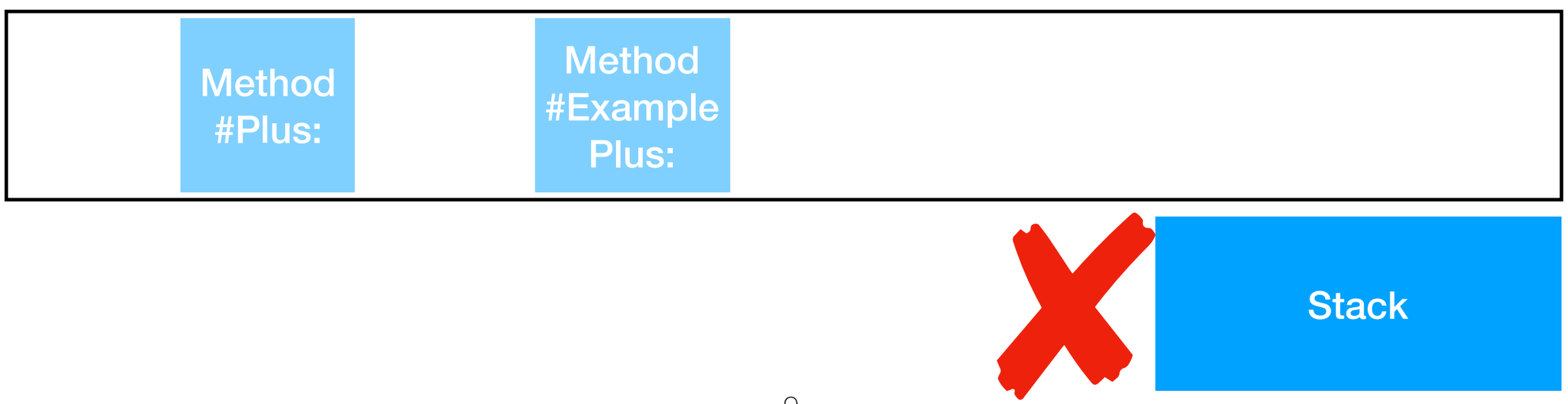


Return operation

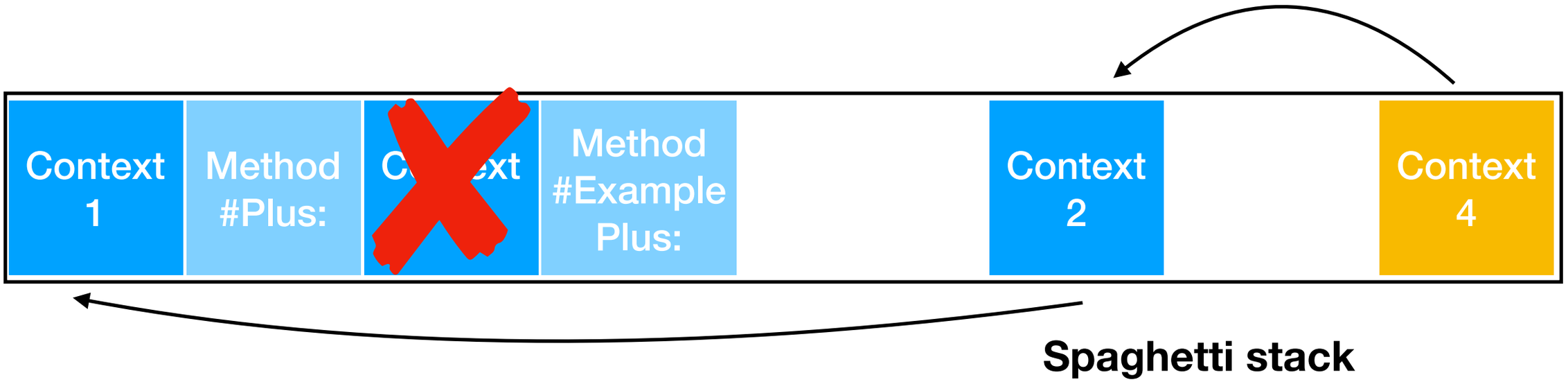


!=

Stack

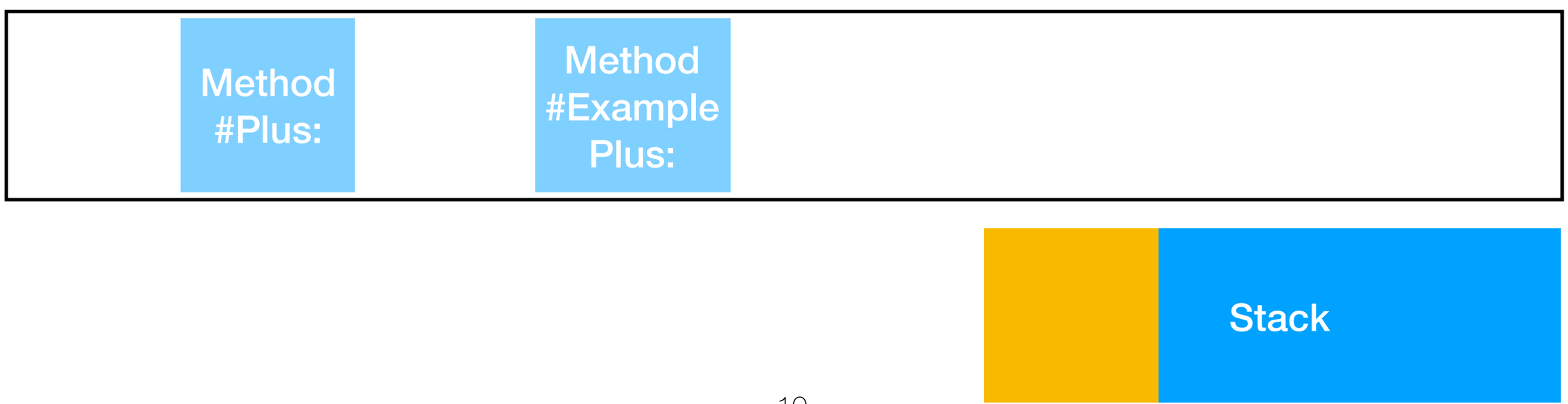


Stack reuses memory



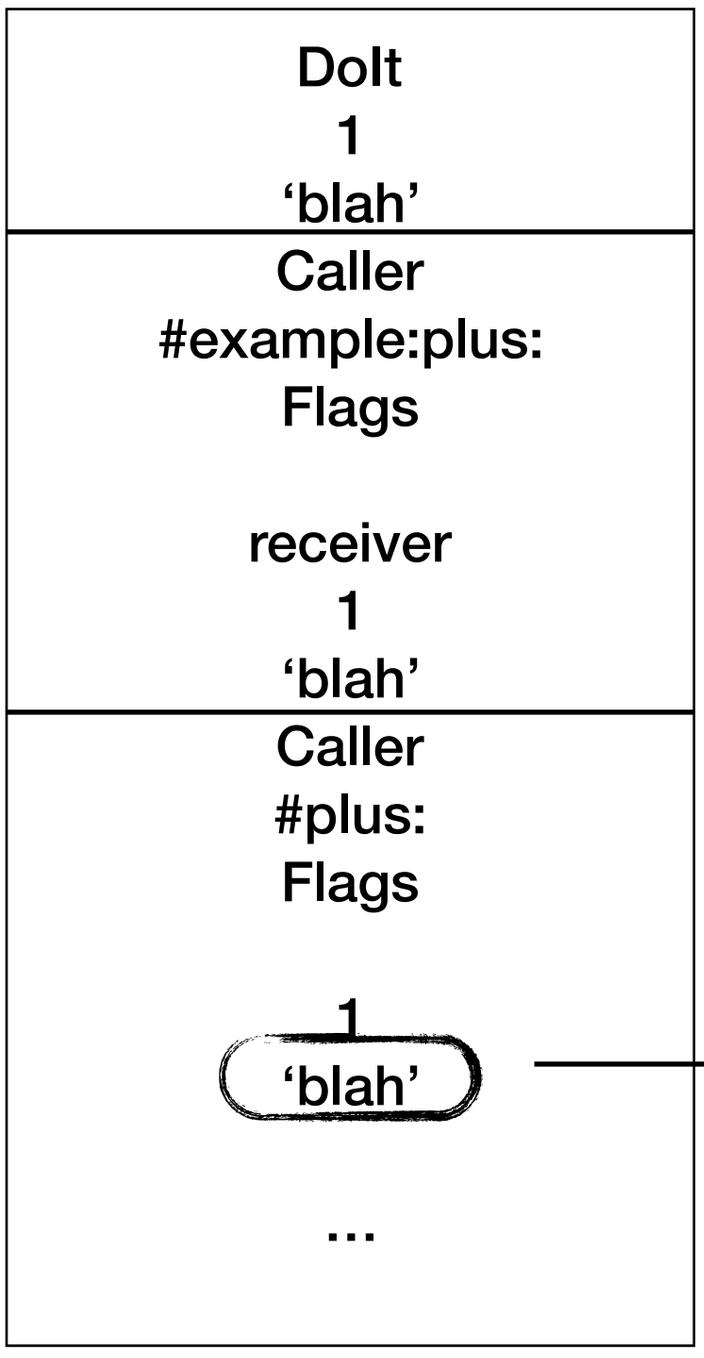
!=

Stack



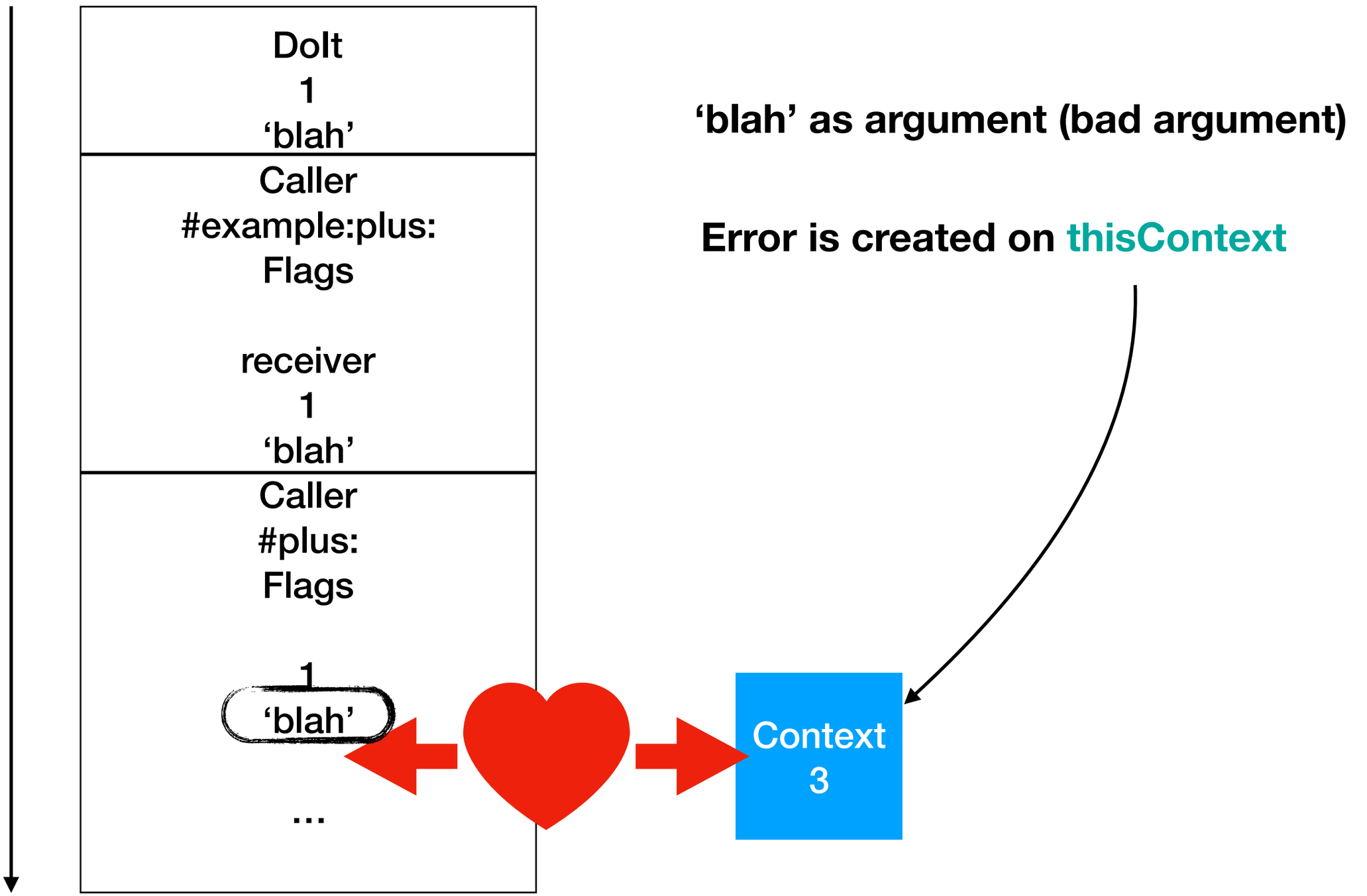
Example debugging

```
InterpreterExamples new example: 1 plus: 'blah'
```

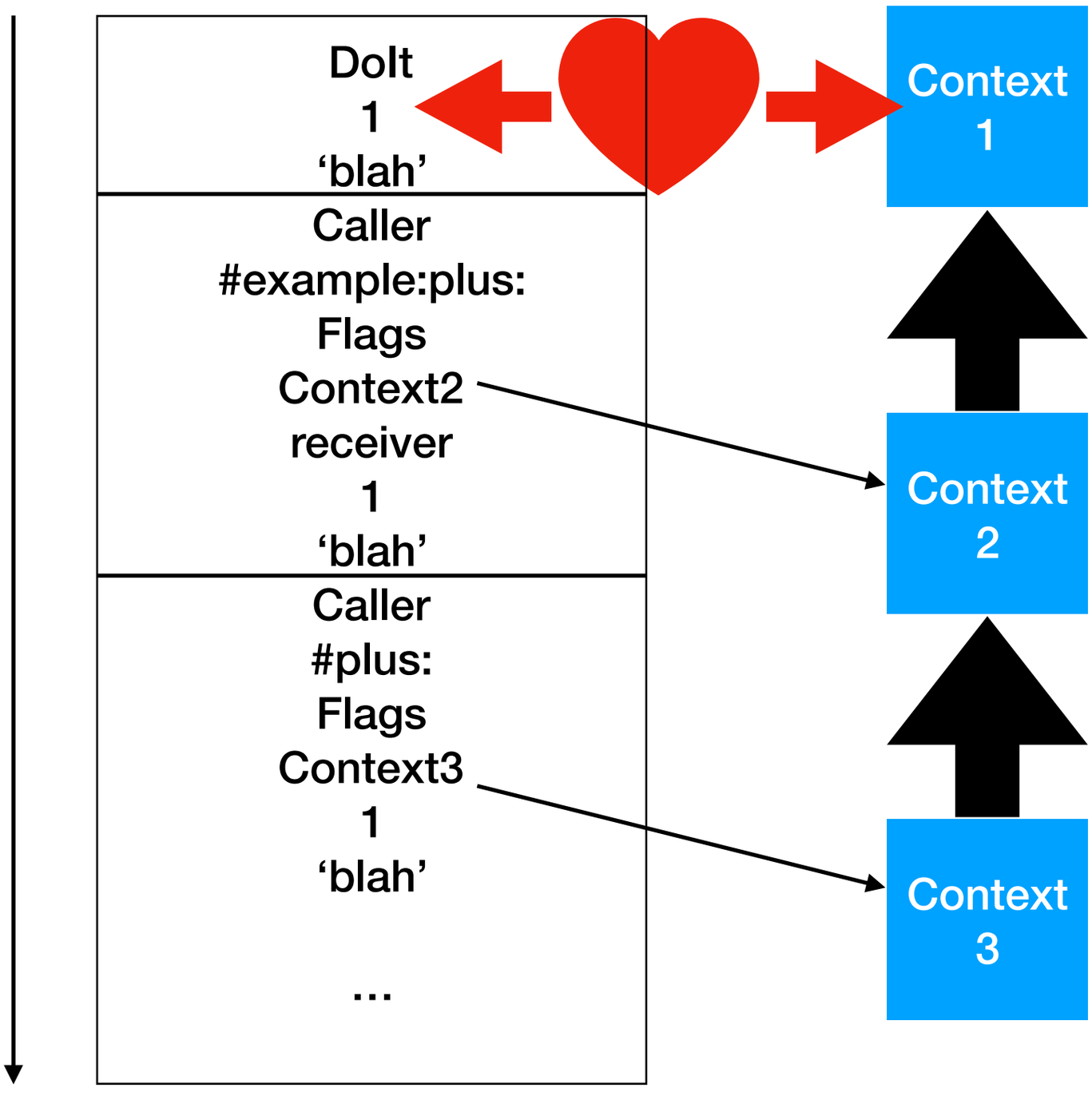


primitiveAdd don’t know how to handle String argument

Example debugging



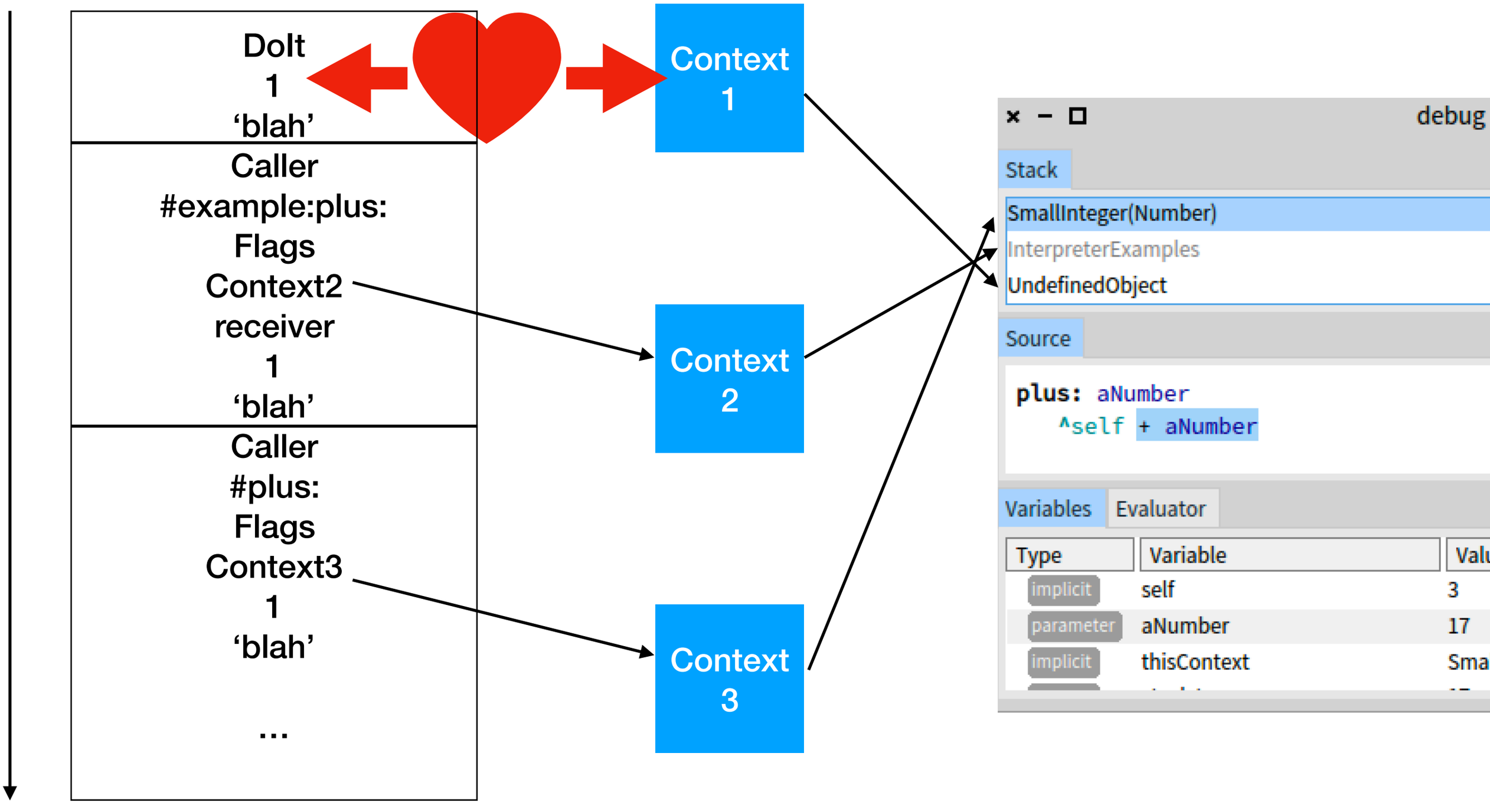
Example debugging



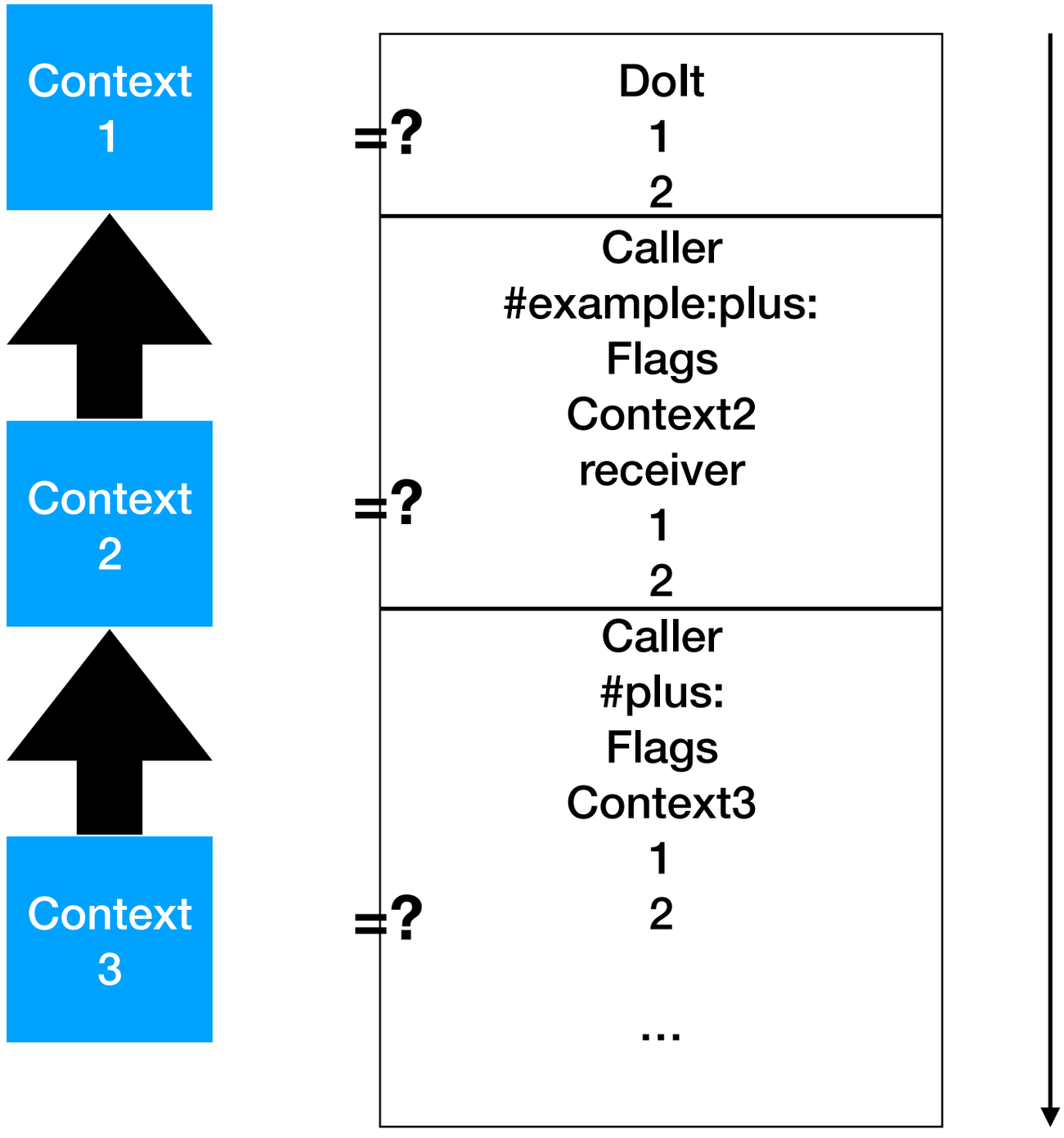
Debugger catches the error.

Debugger asks Context>>#sender of contexts recursively

Example debugging



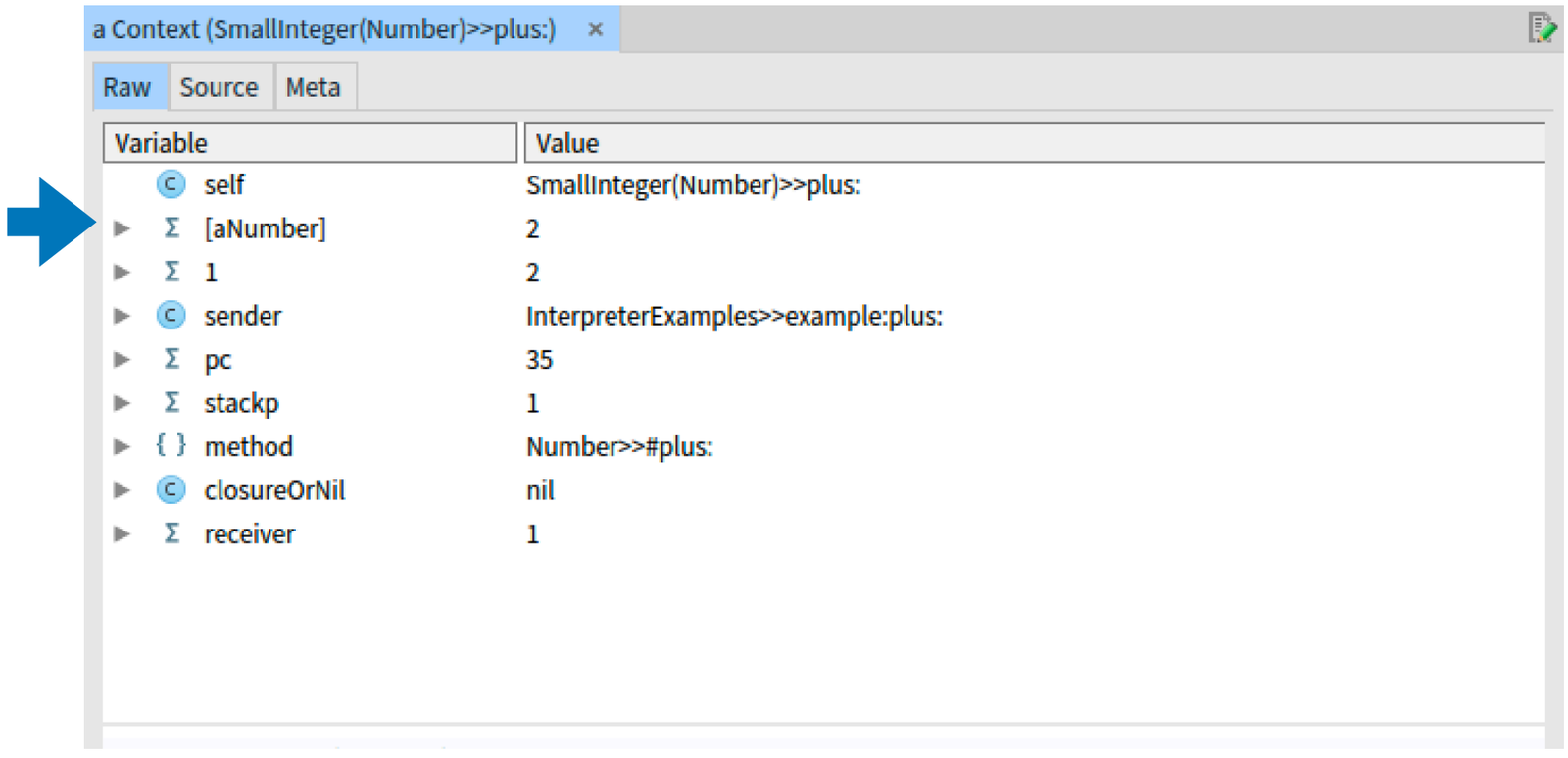
How the contexts map to the stack frames ?

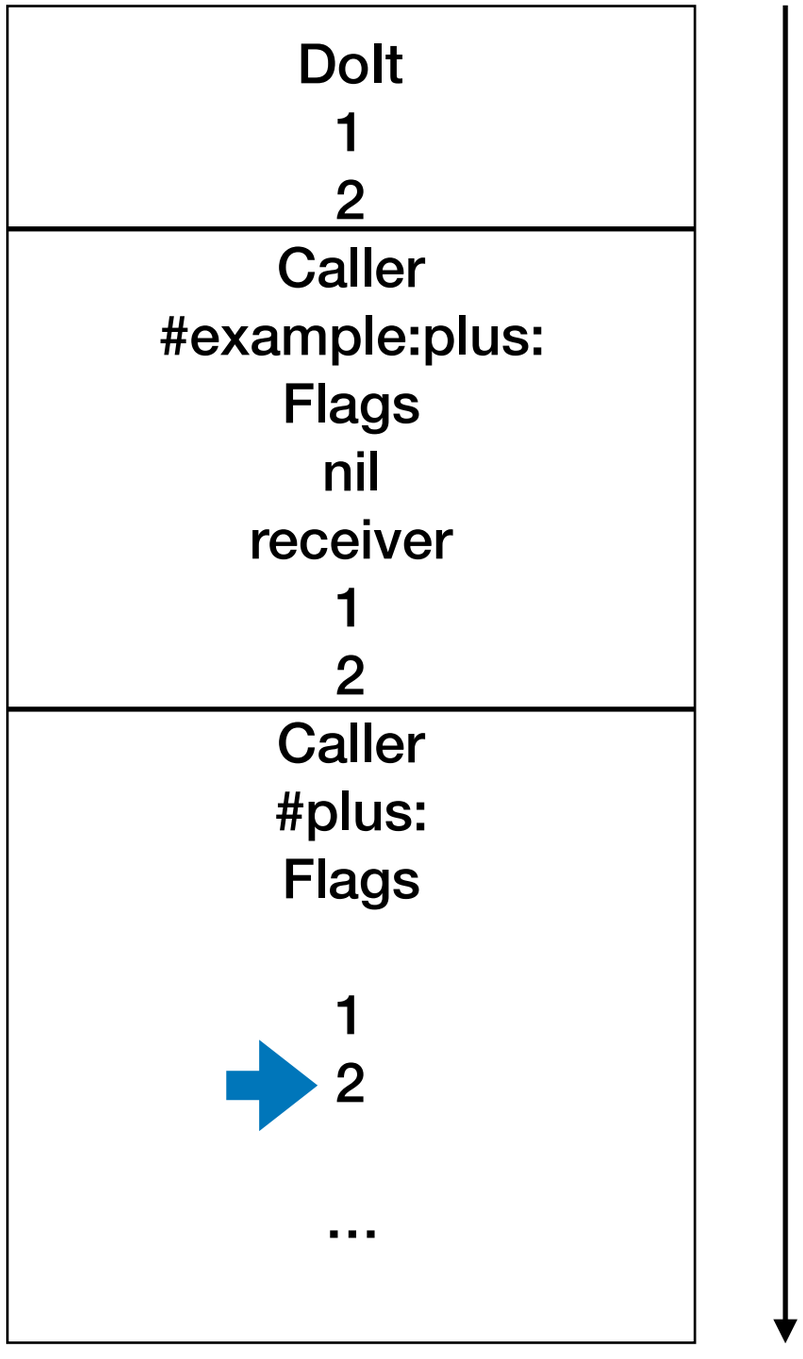


Context to stack frame

Argument(s) and temporaries:

1. **With names**

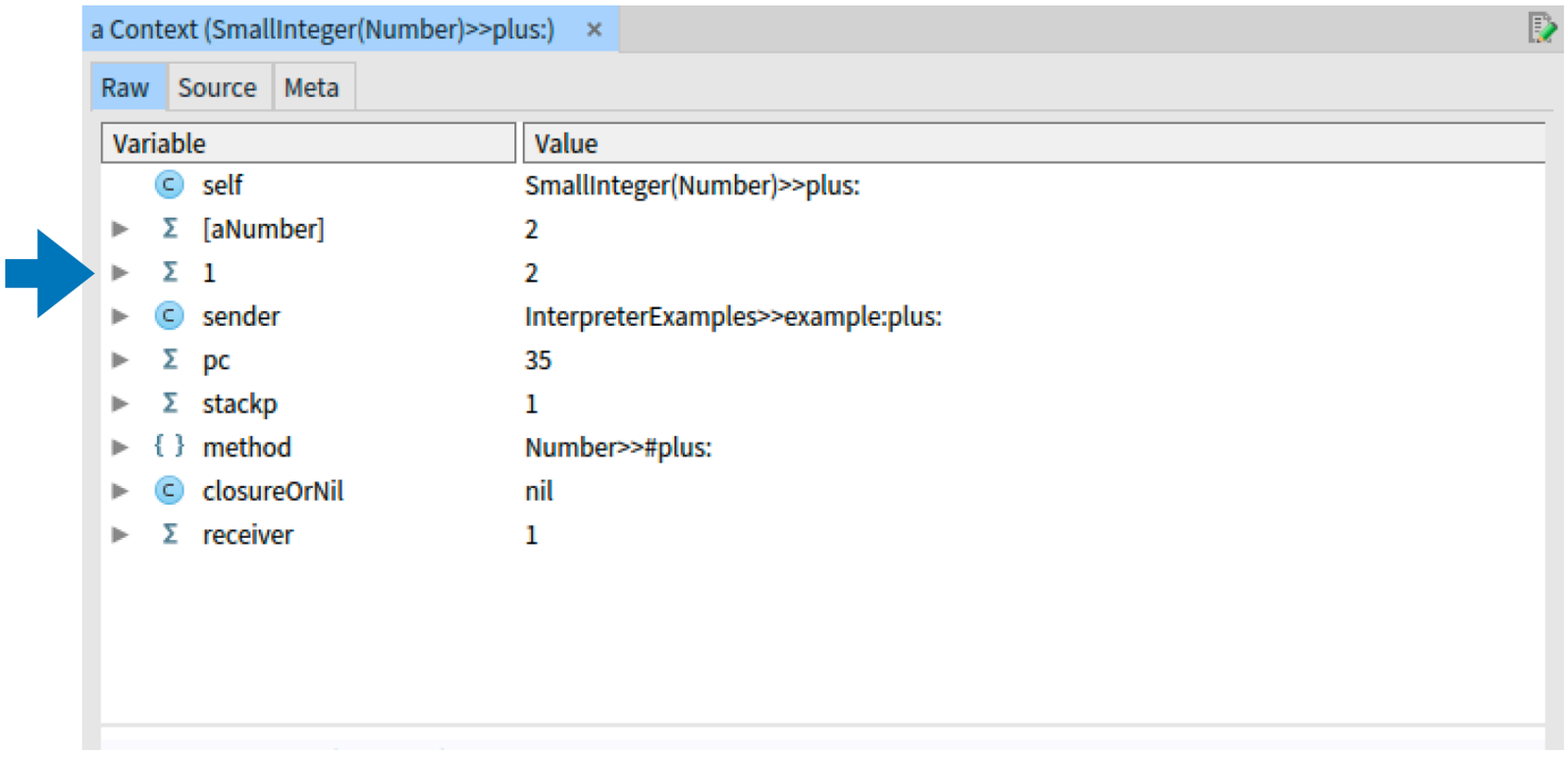


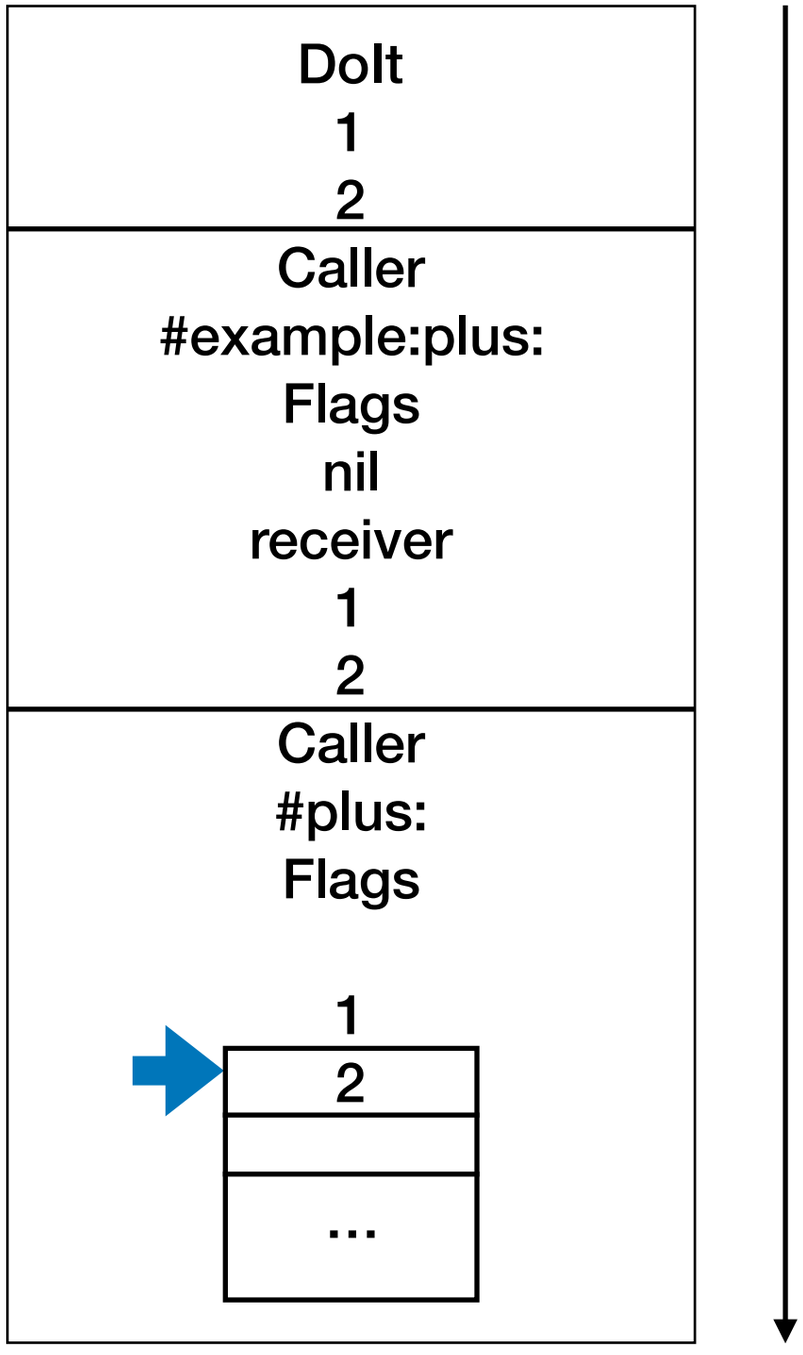


Context to stack frame

Argument(s) and temporaries:

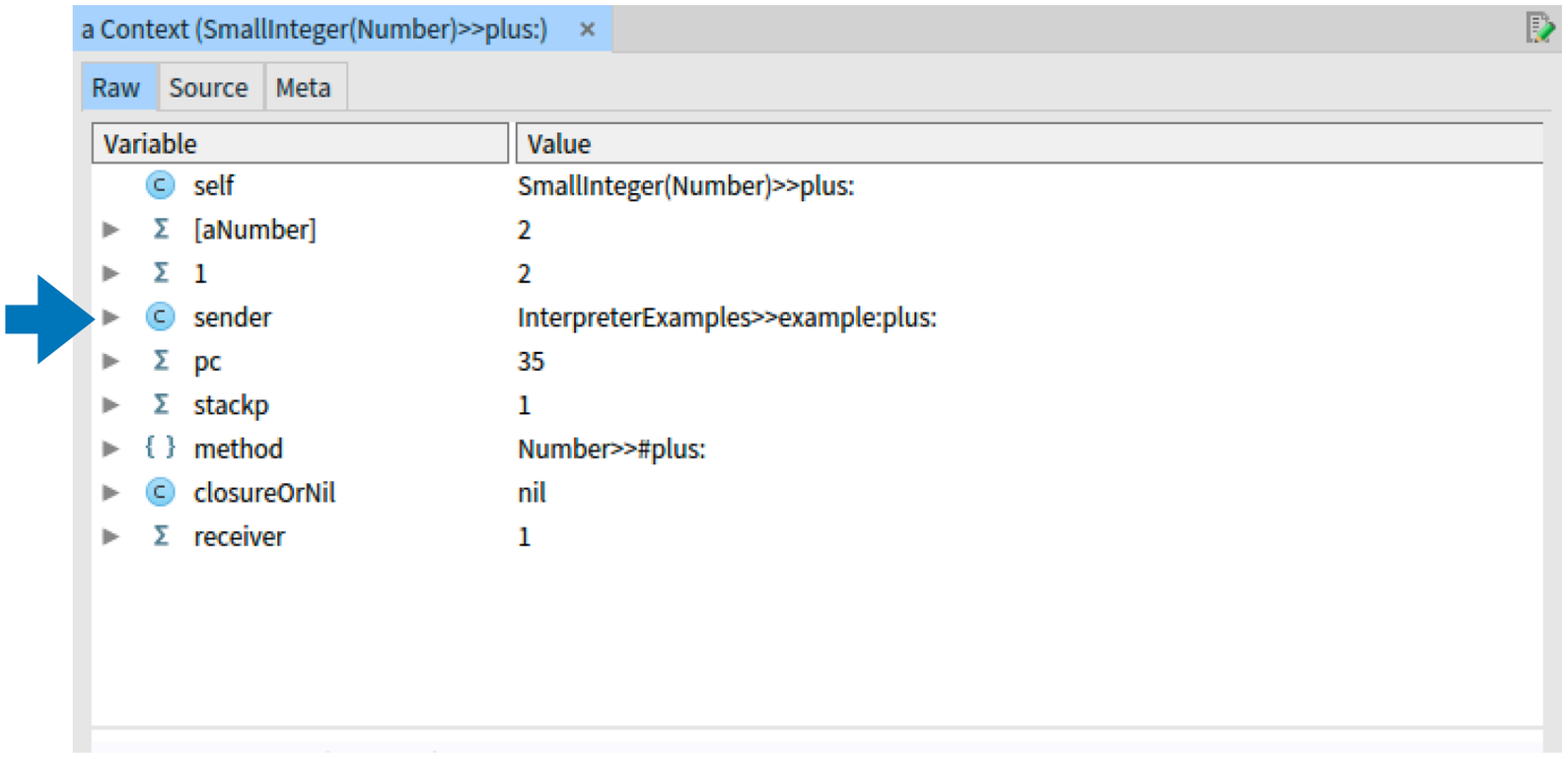
1. **With names**
2. **Stack representation**

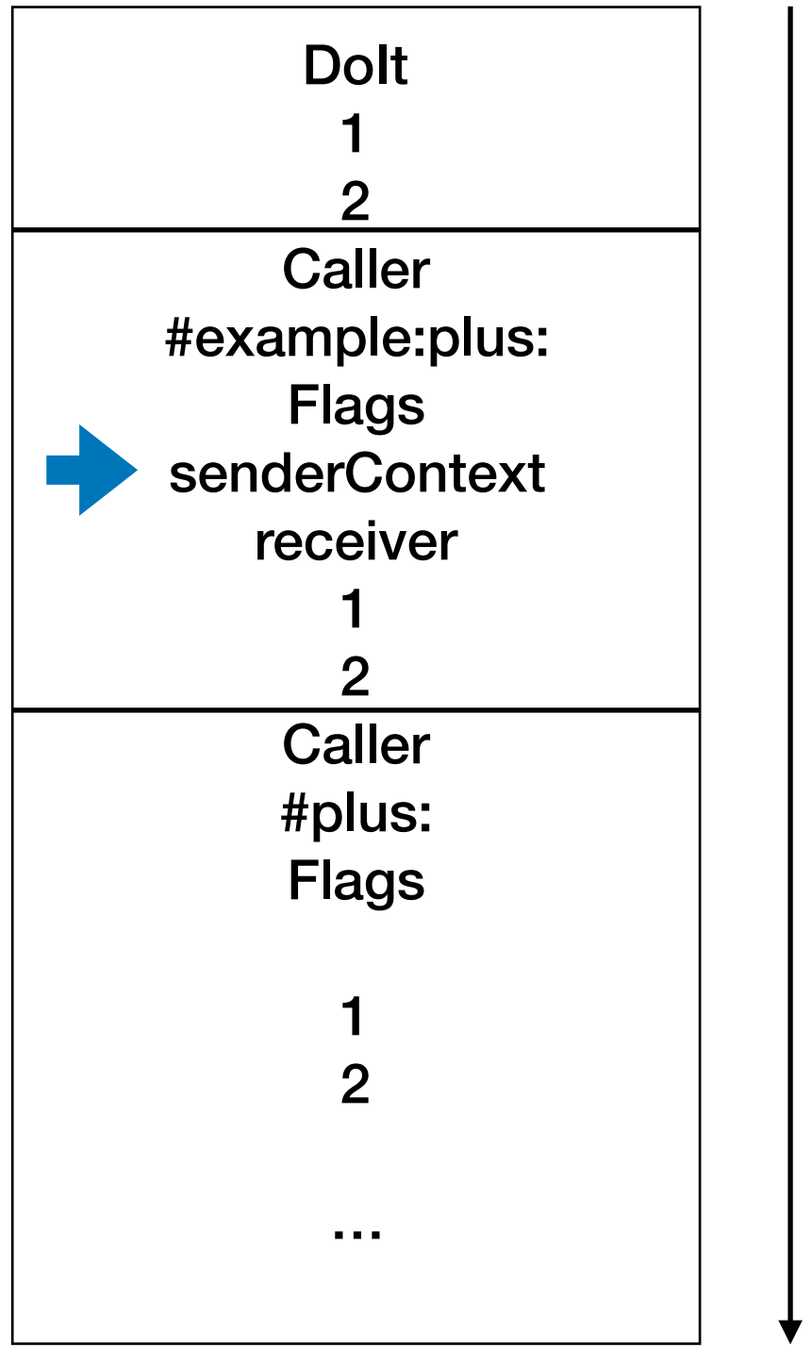




Context to stack frame

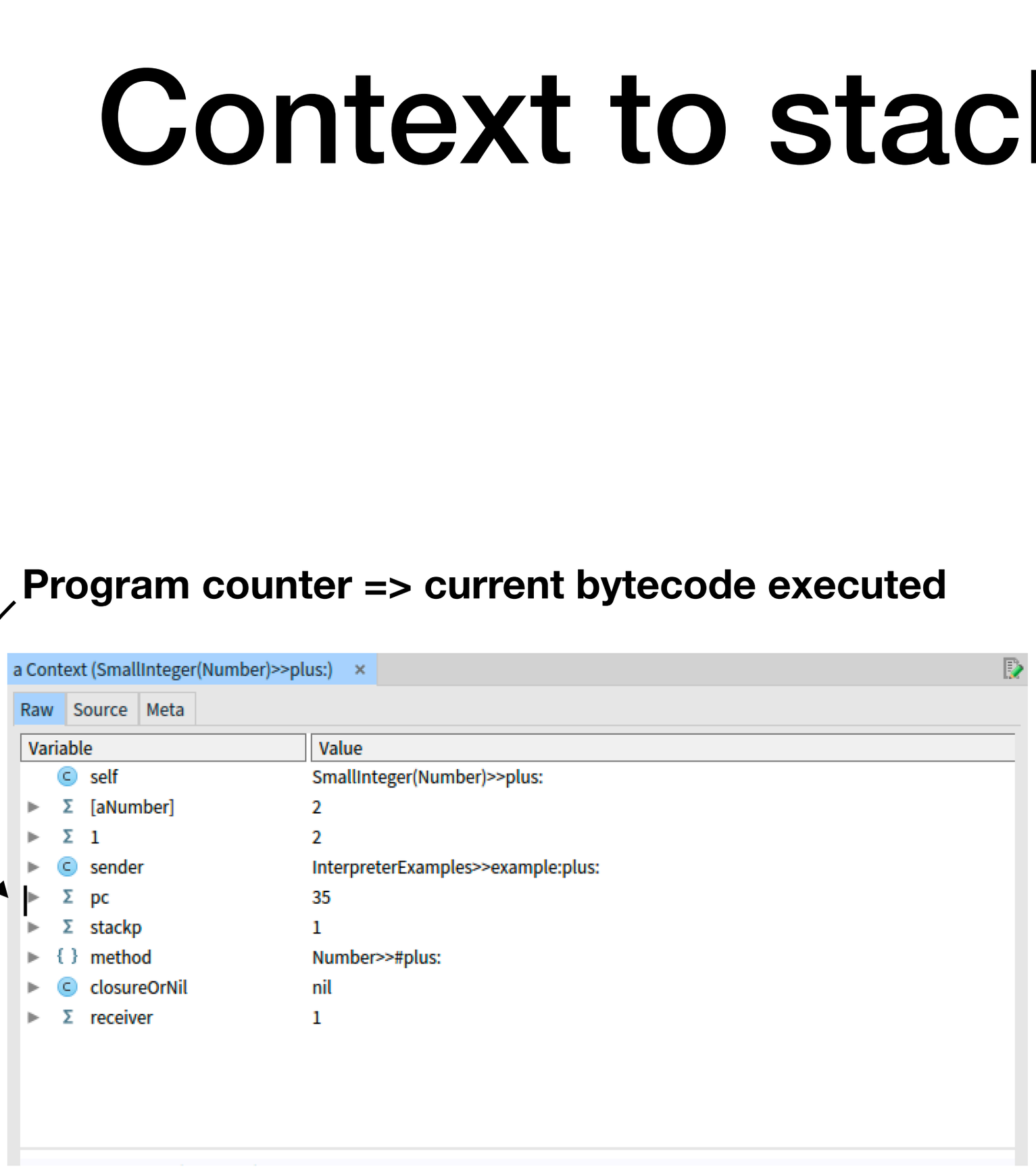
Sender

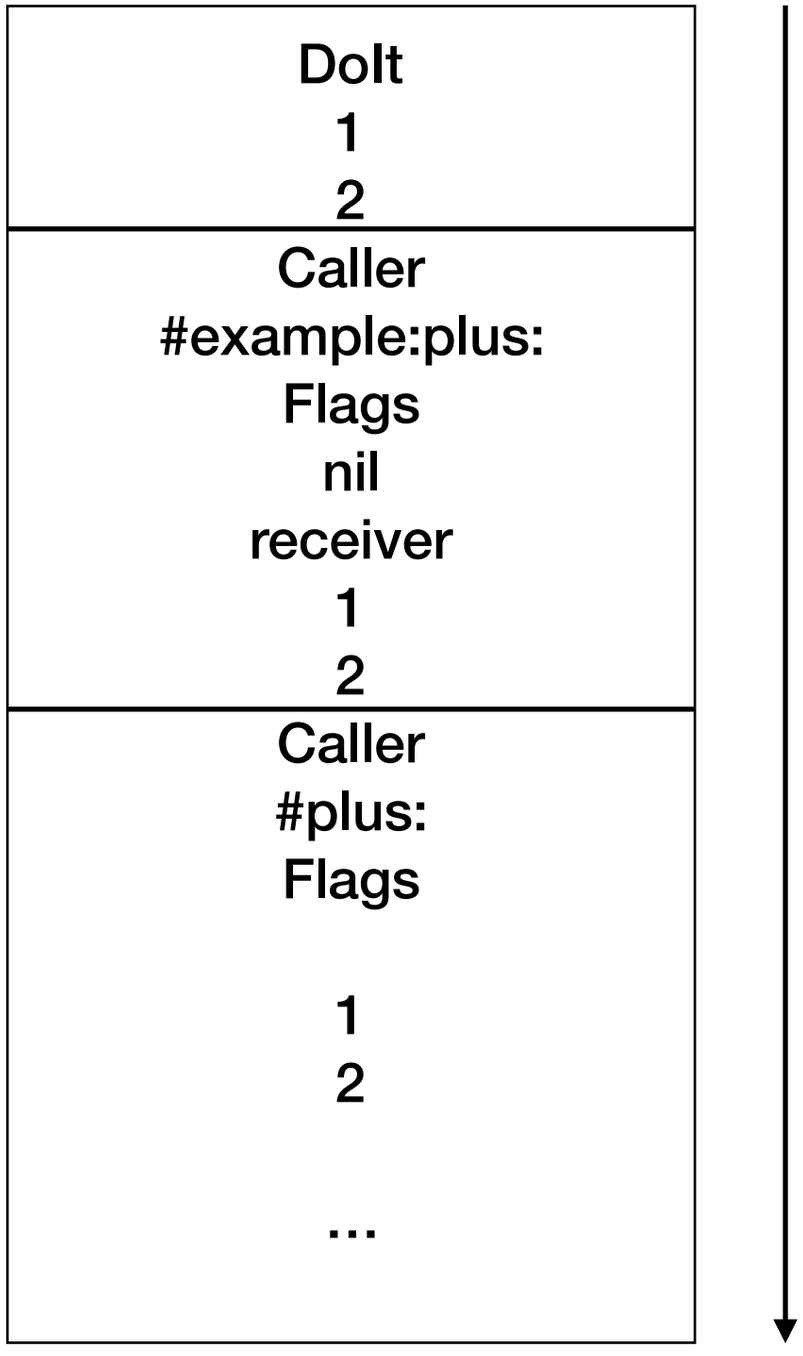




Context to stack frame

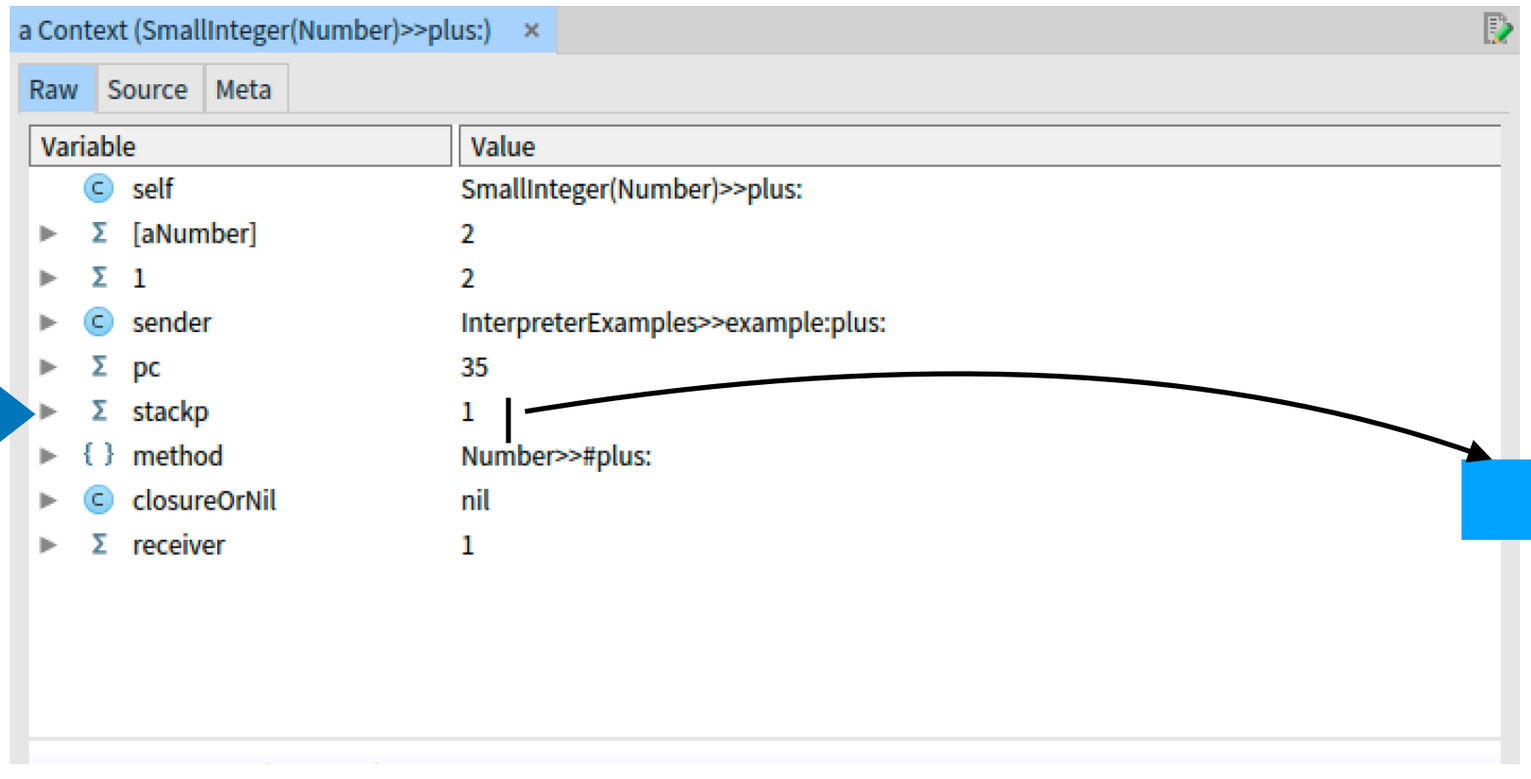
Program counter => current bytecode executed

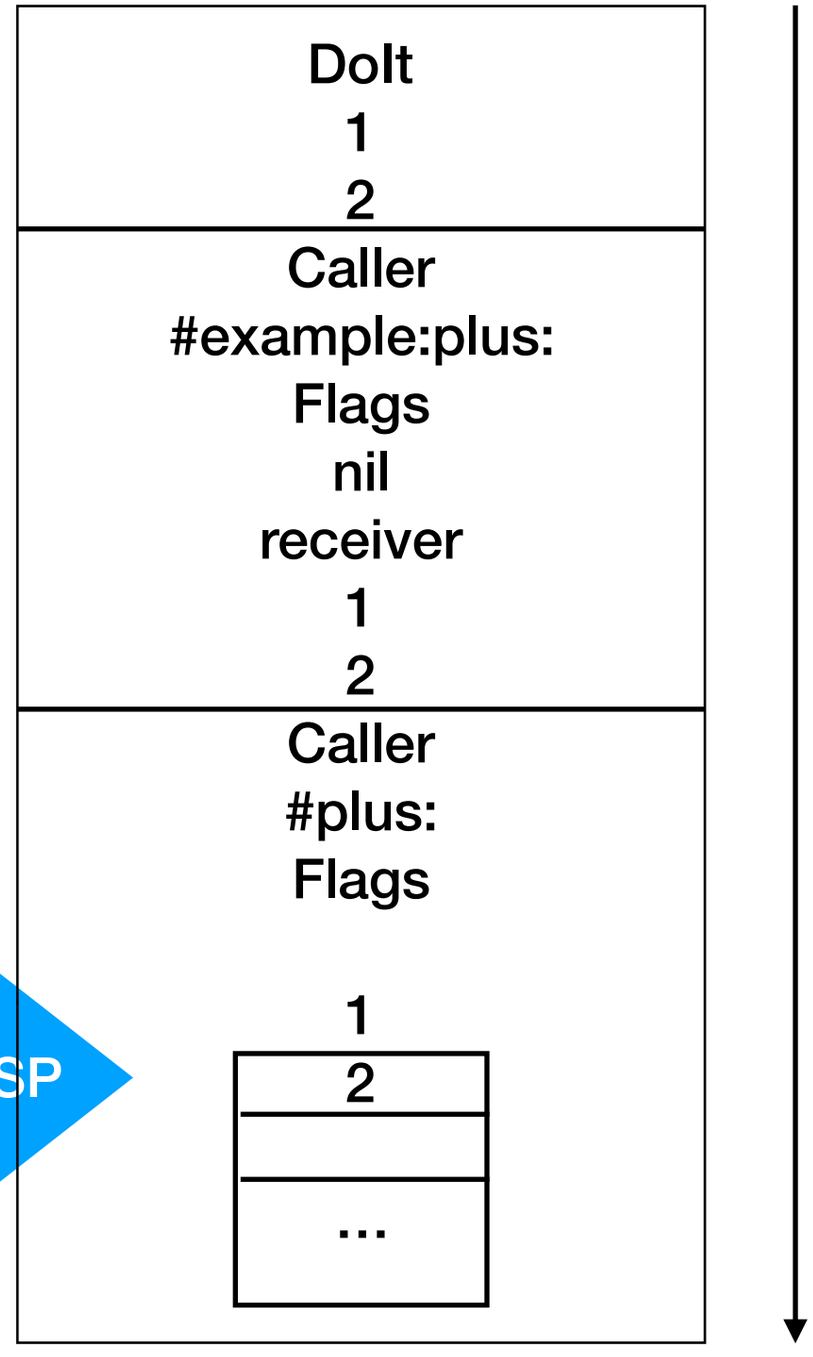




Context to stack frame

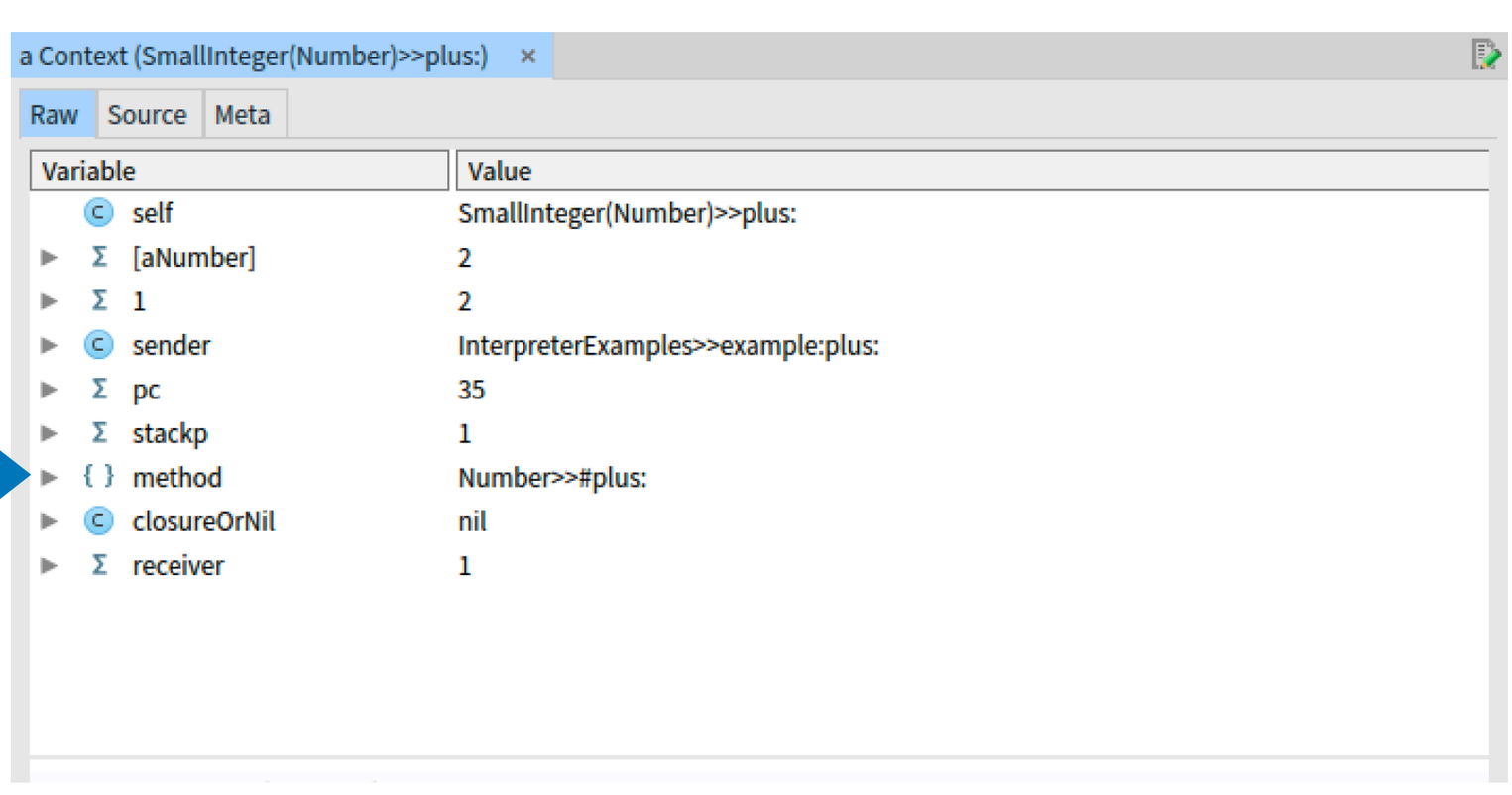
Stack pointer => Position of the current stack top

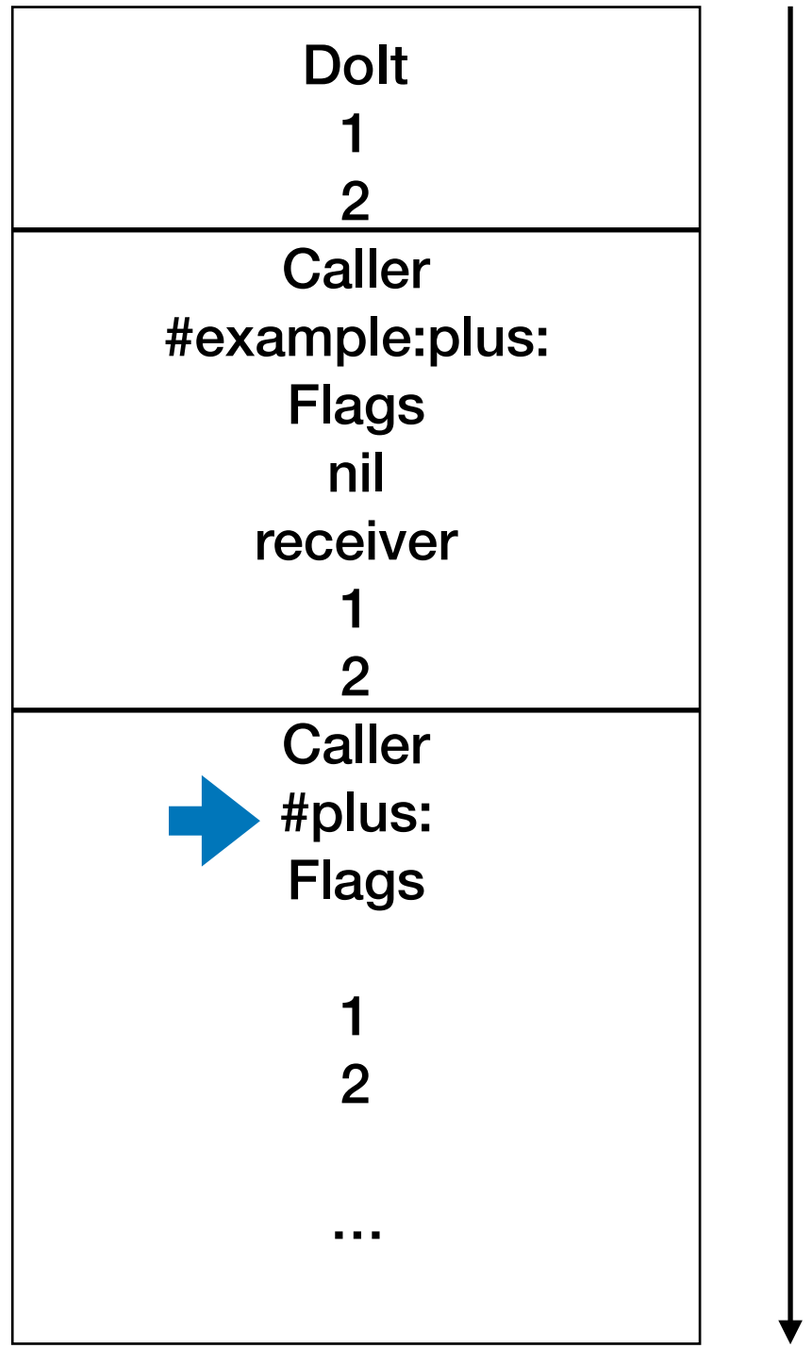




Context to stack frame

Method

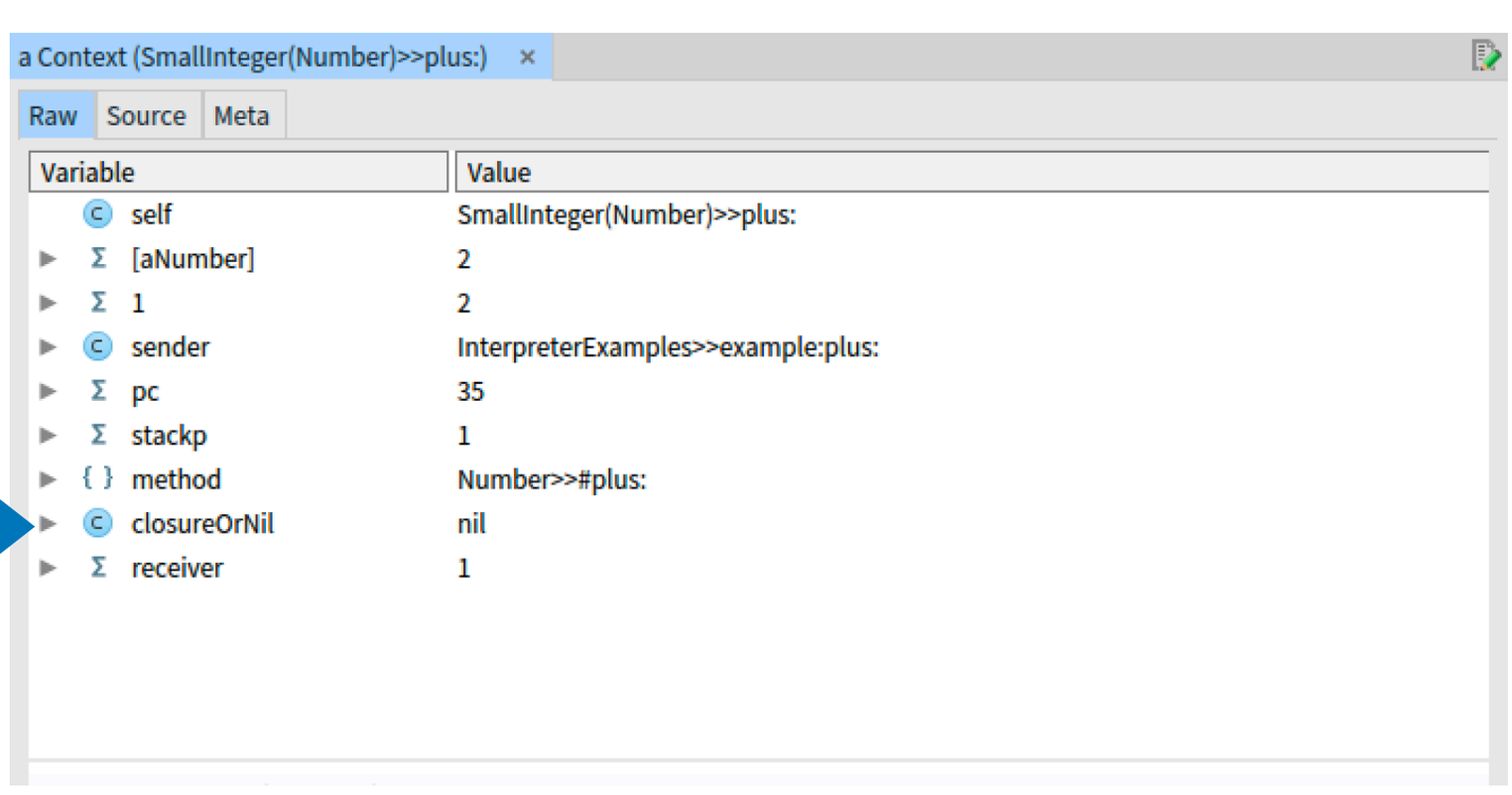


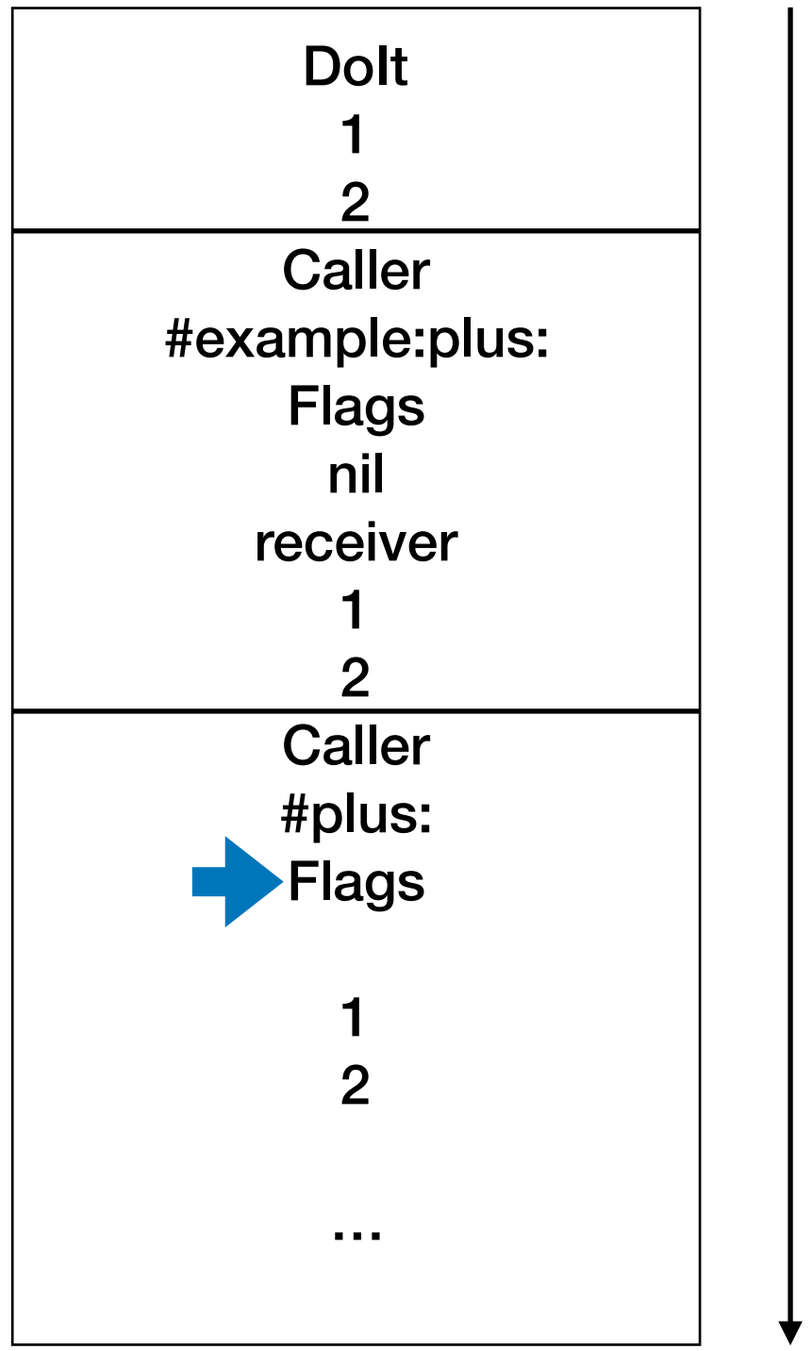


Context to stack frame

Closure…=> if this the execution of a block
…OrNil => for a method

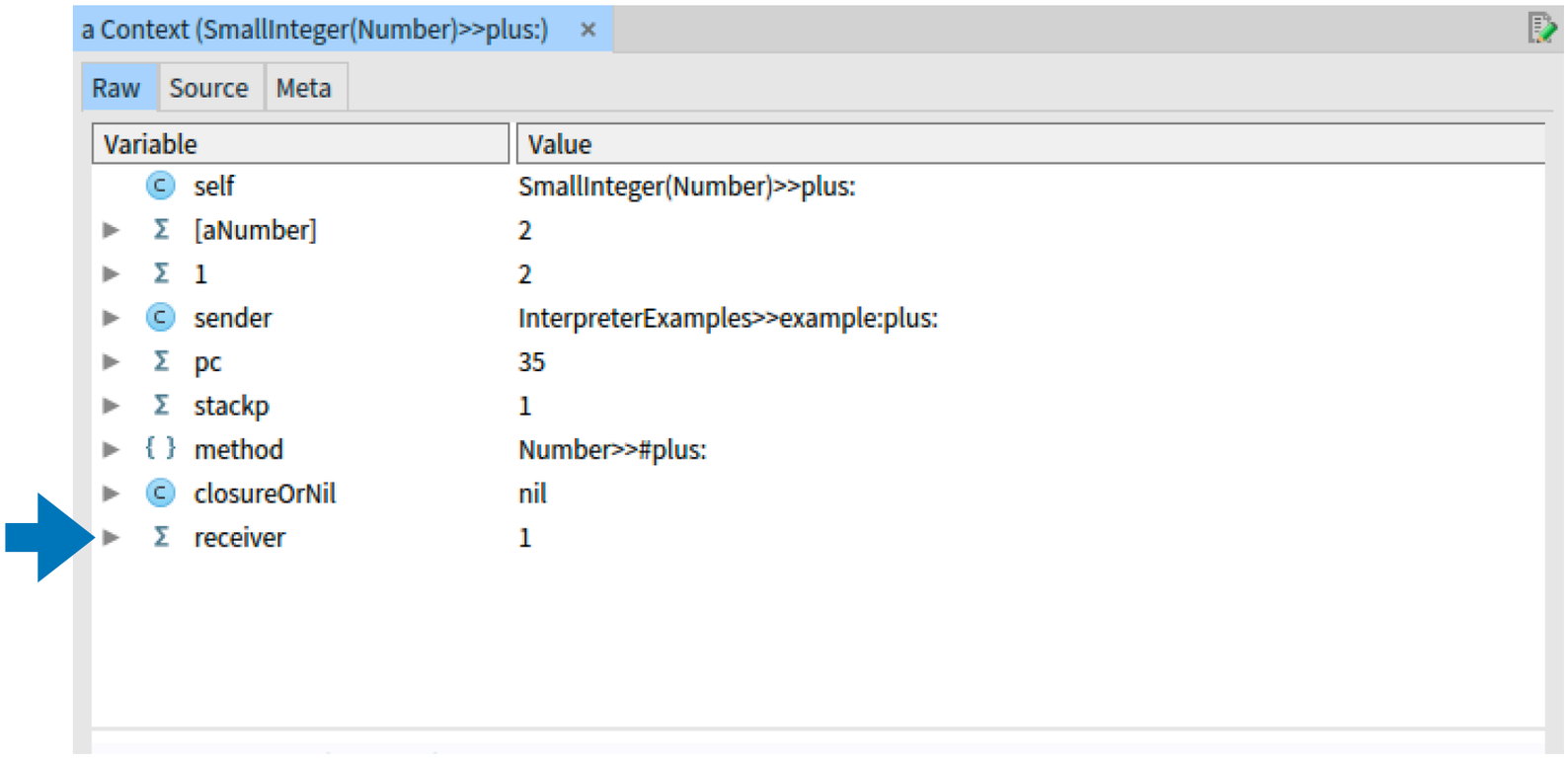
Mask of the flags

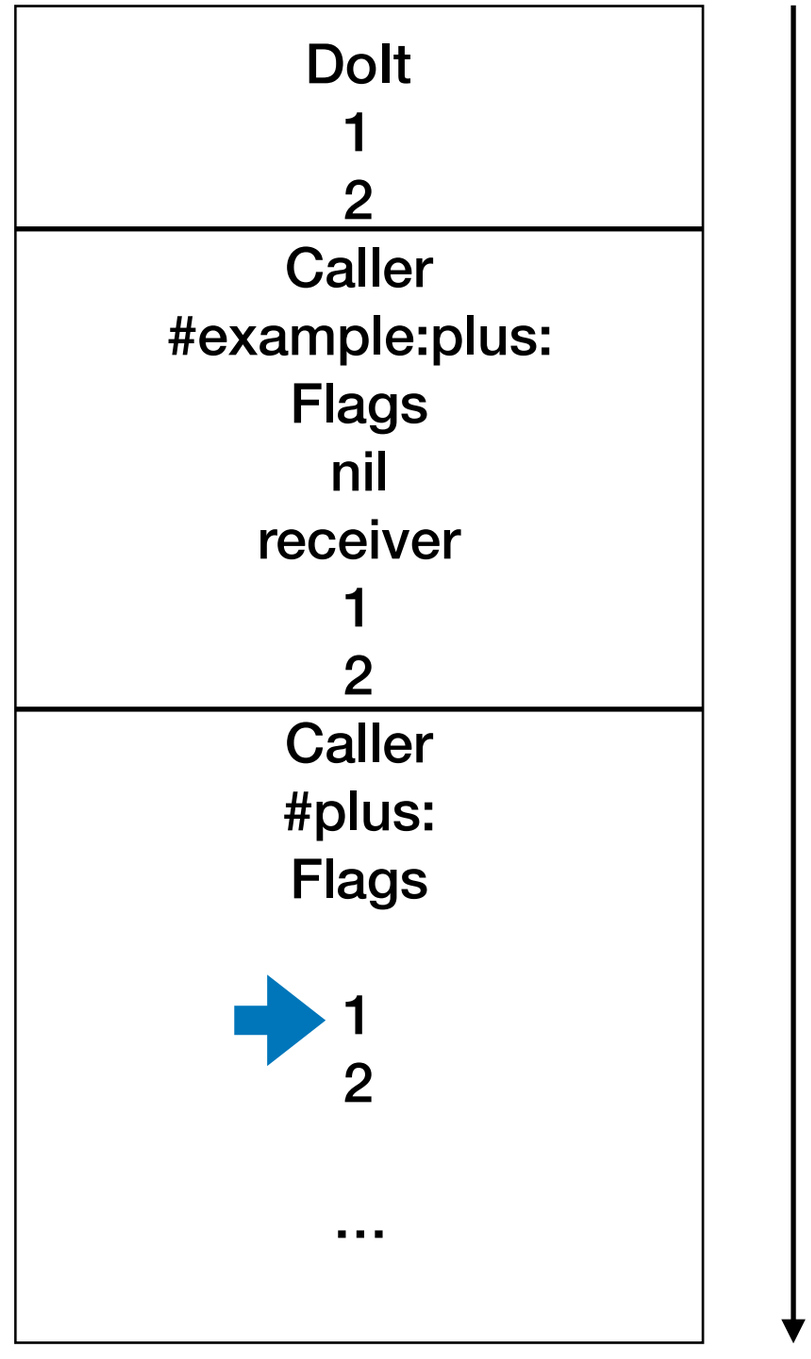




Context to stack frame

Receiver

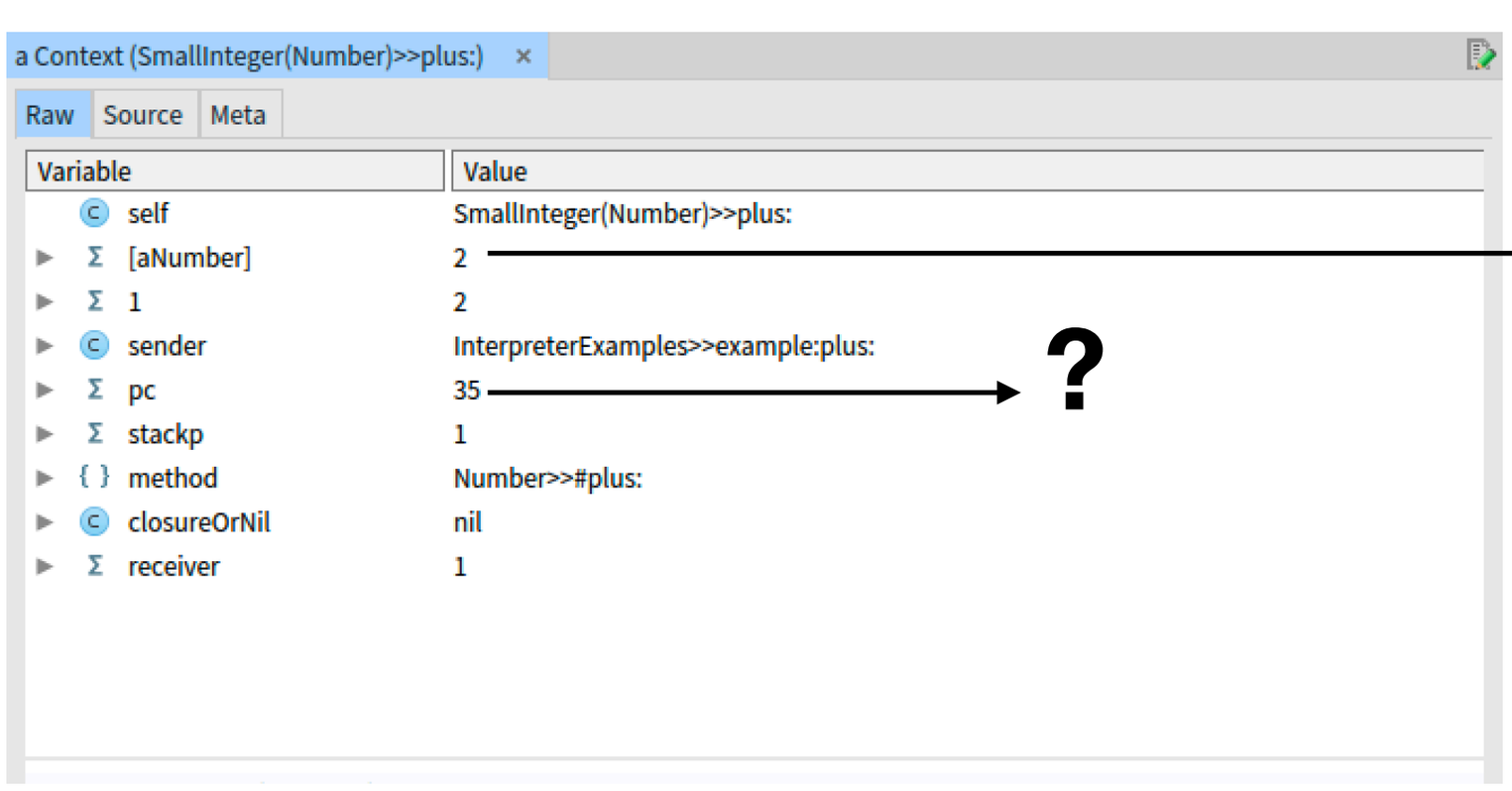


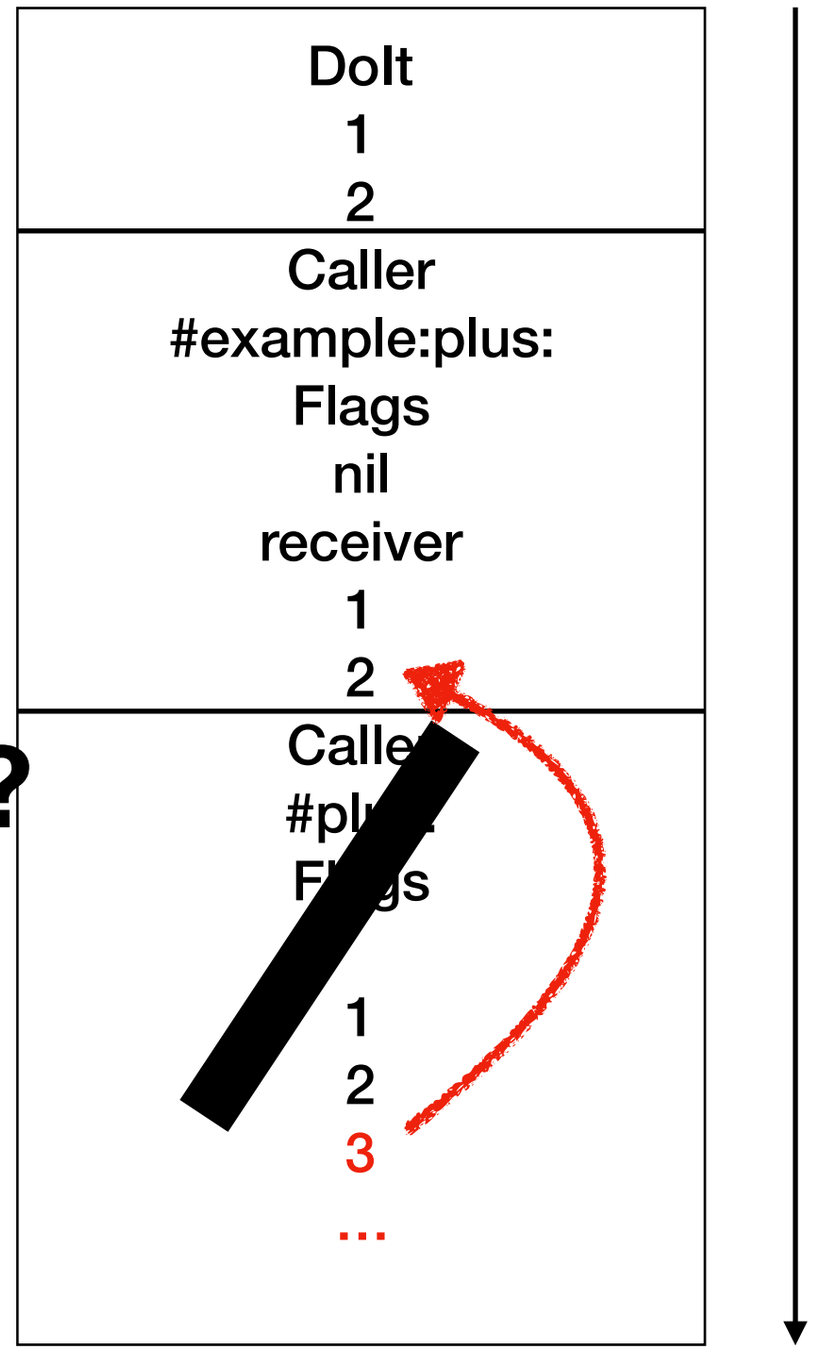


What happens when ?

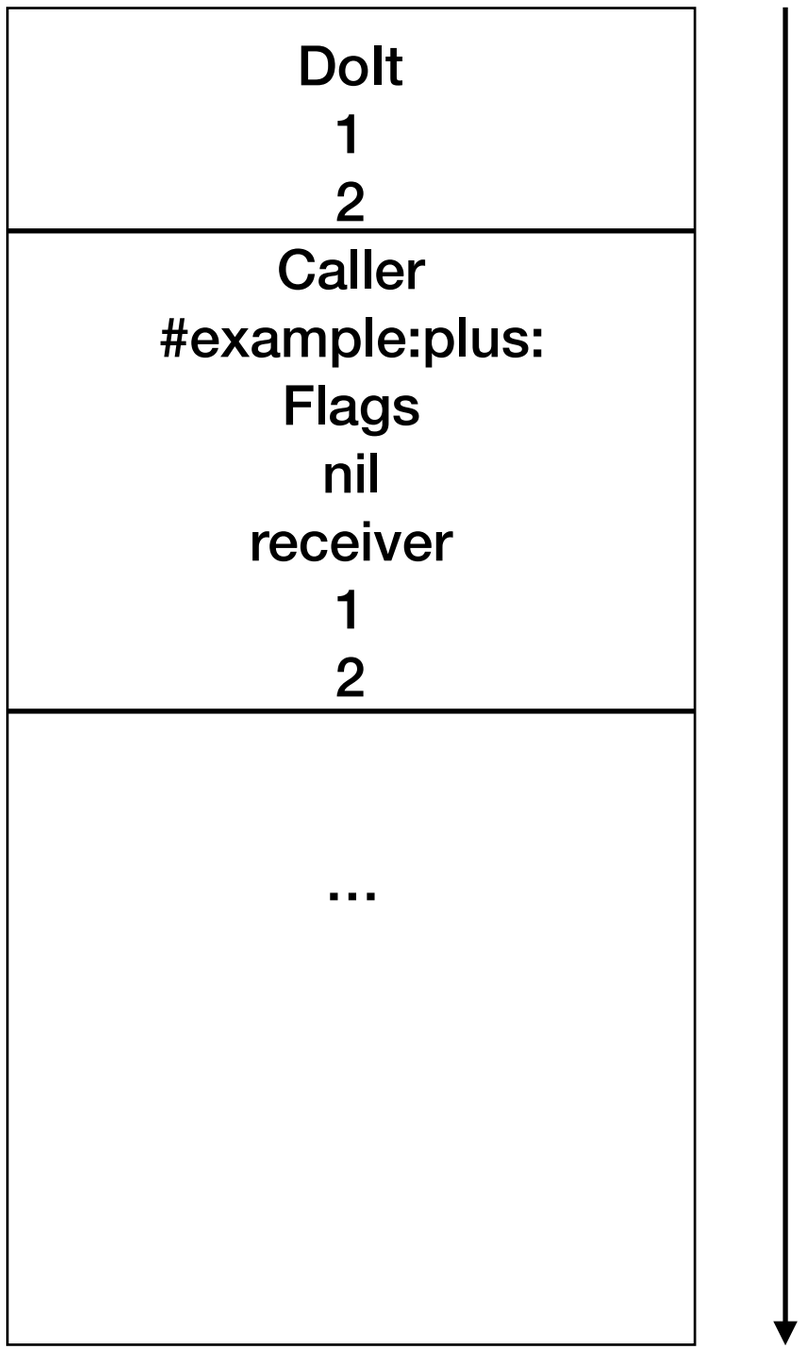
#Plus: return and ?

Stack frames and contexts have different life cycles

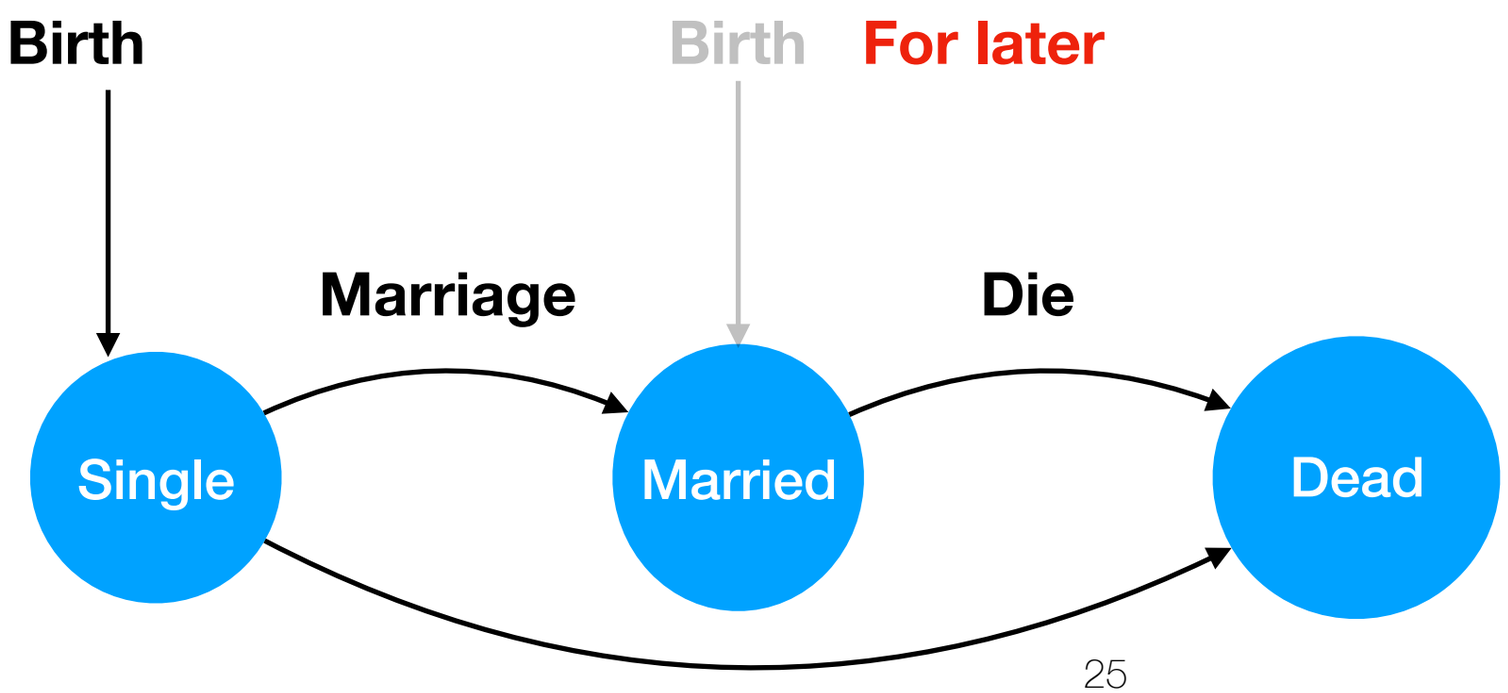




Stack frame life



Automaton for a frame

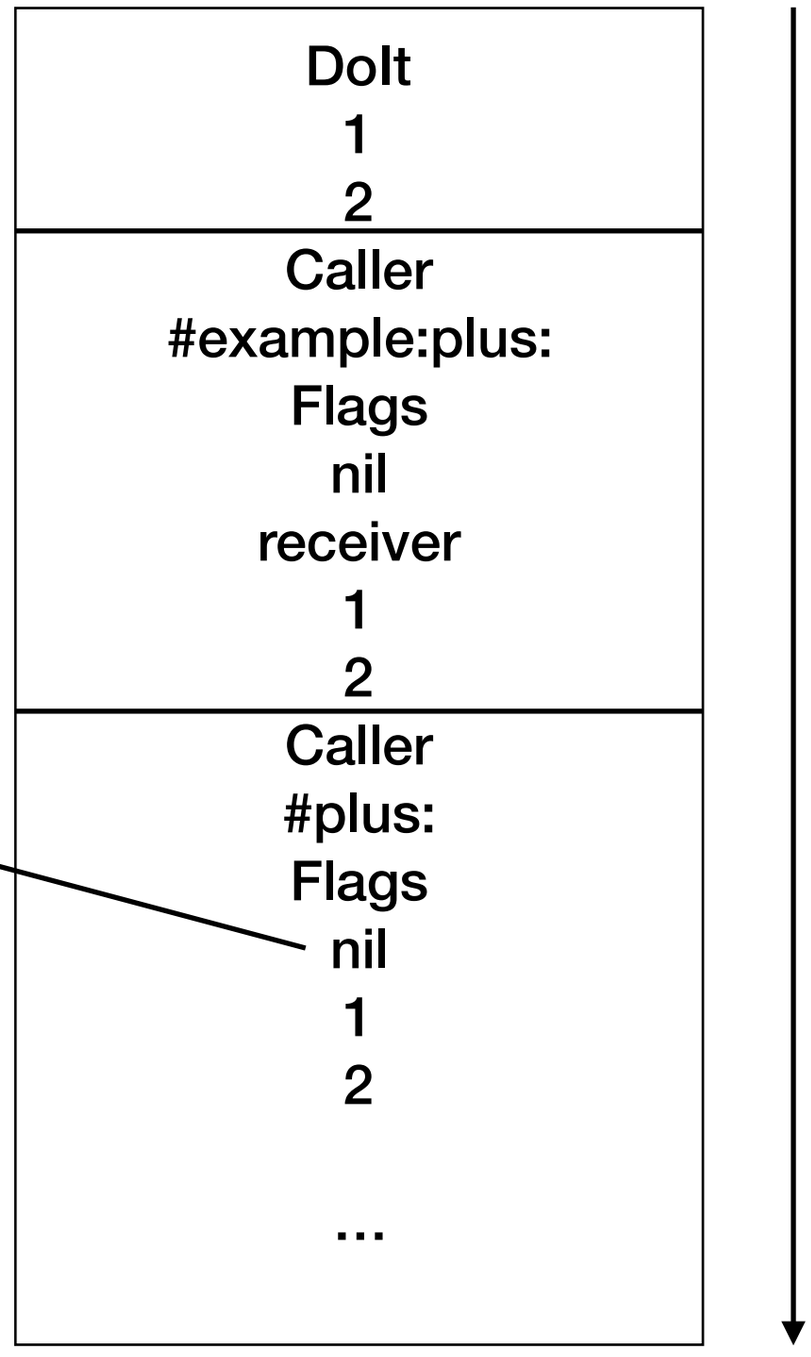


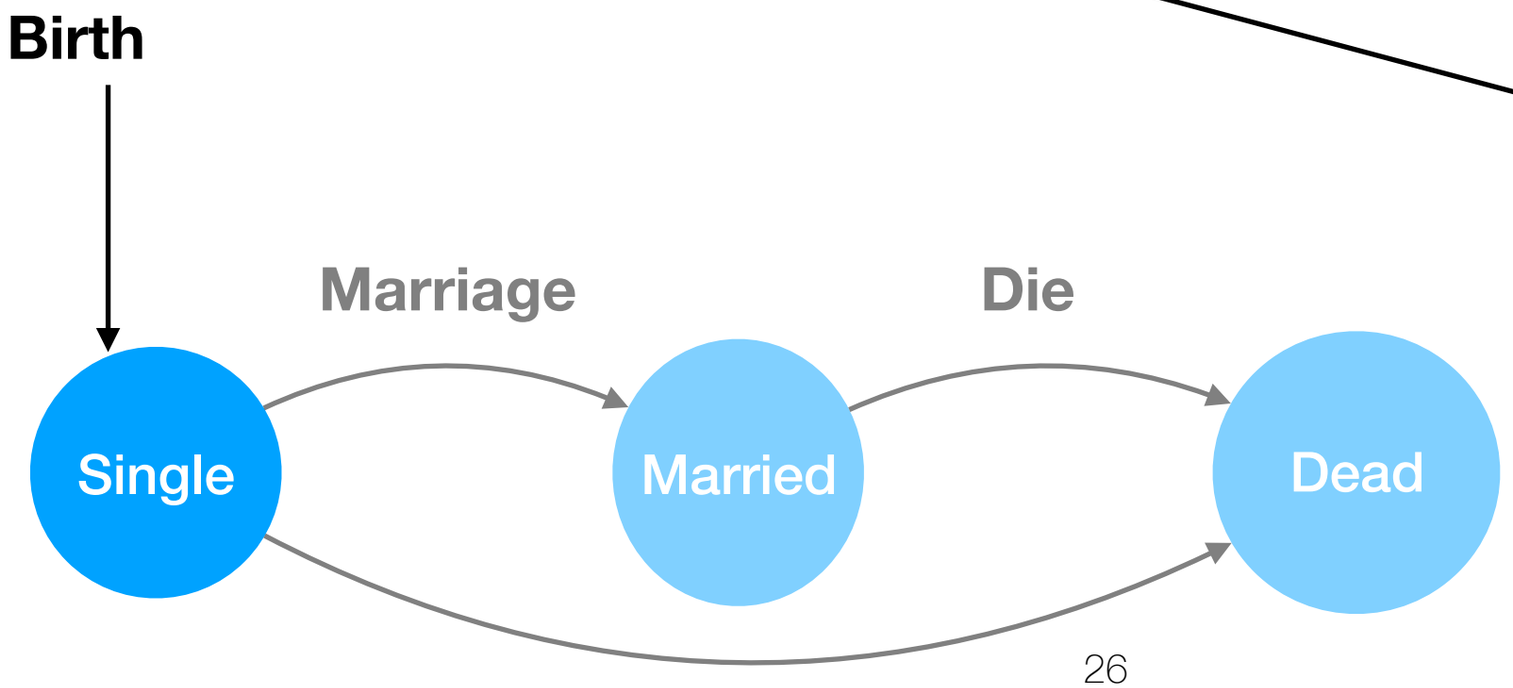
Stack frame life: birth

```
example: aNumber plus: aNumberToAdd

    ^aNumber plus: aNumberToAdd
```

Frame is created single

No context linked





Stack frame life: marriage

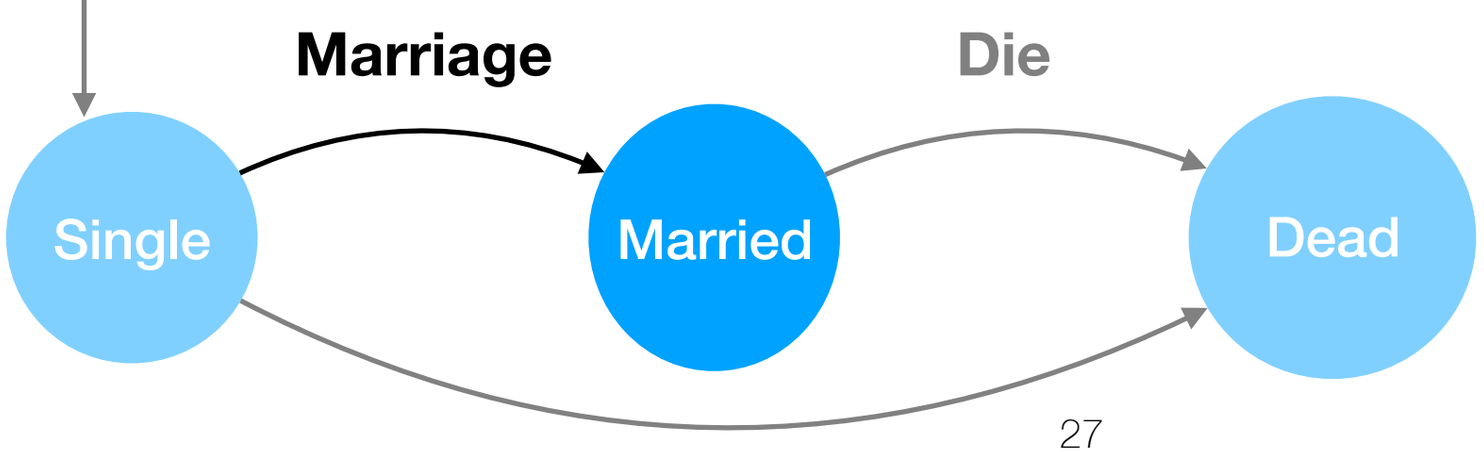
```
plus: aNumber
    thisContext.
    ^self + aNumber
```

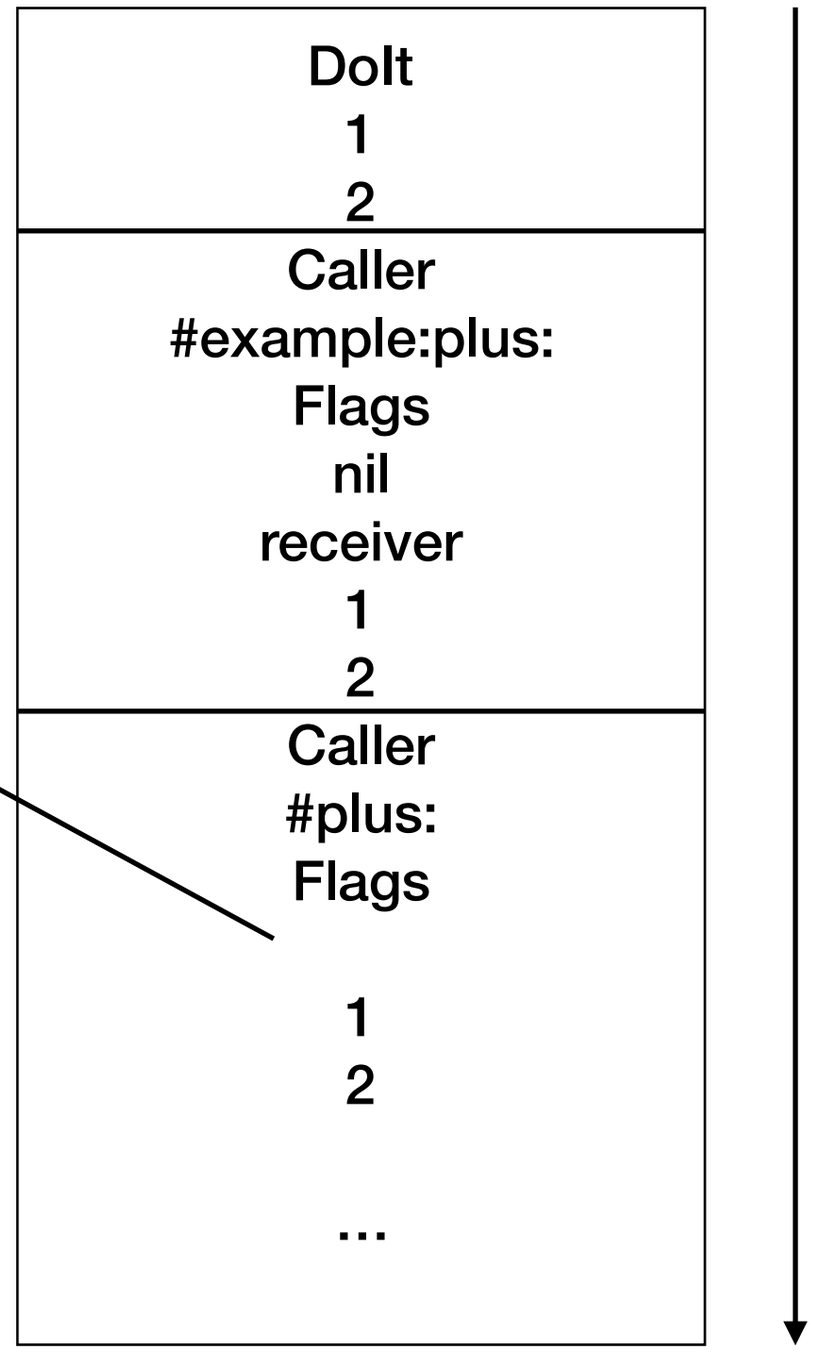
a Context (SmallInteger(Number)>>plus:)

Raw | Source | Meta

Variable	Value
self	SmallInteger(Number)>>plus:
[aNumber]	2
1	2
sender	InterpreterExamples>>example:plus:
pc	35
stackp	1
method	Number>>#plus:
closureOrNil	nil
receiver	1

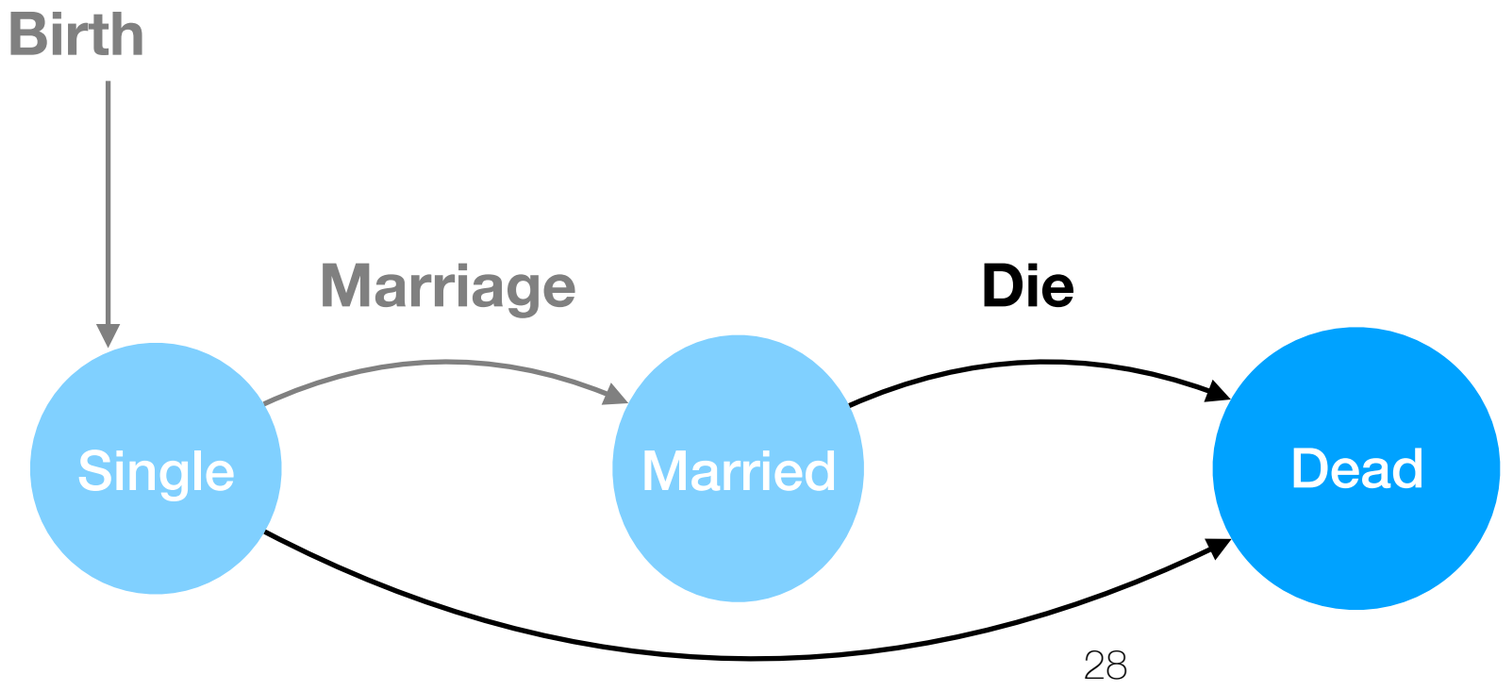
Birth

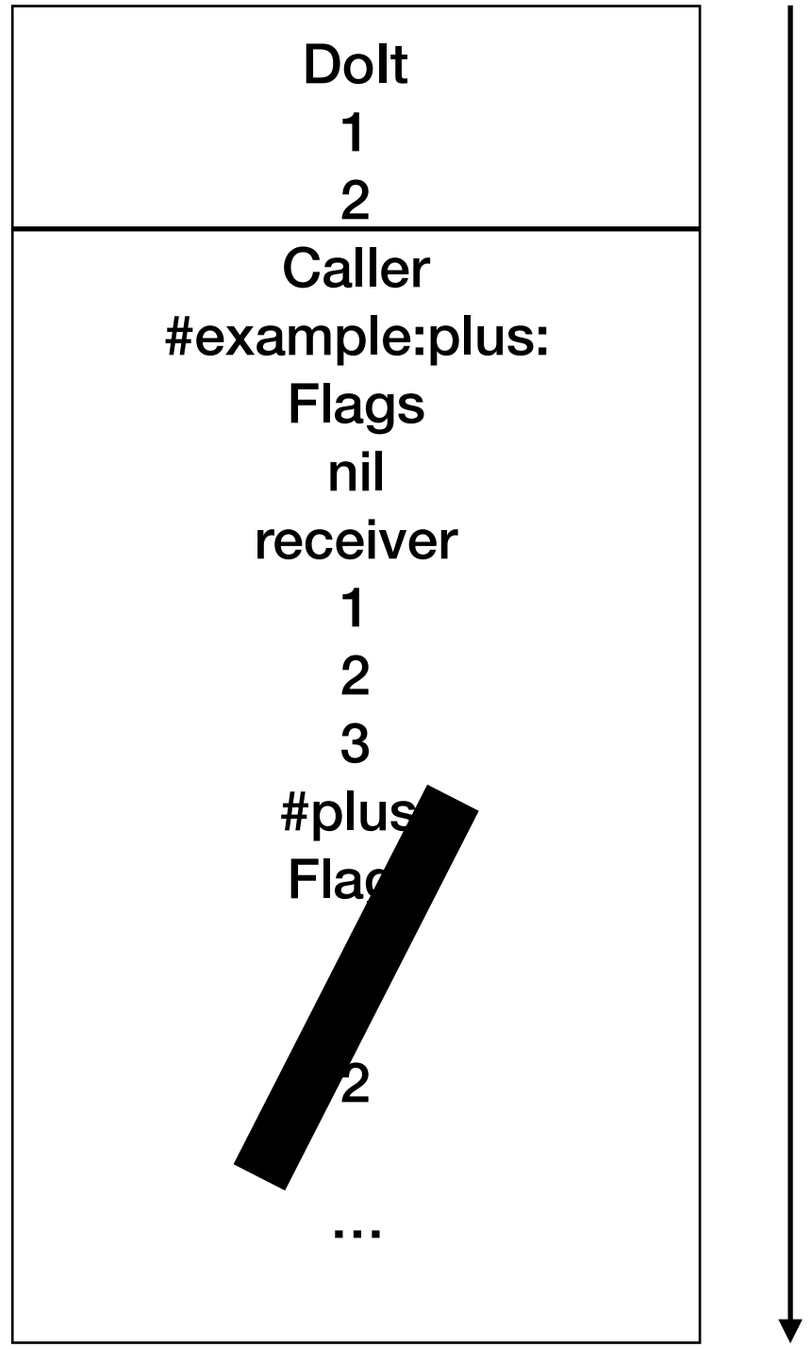




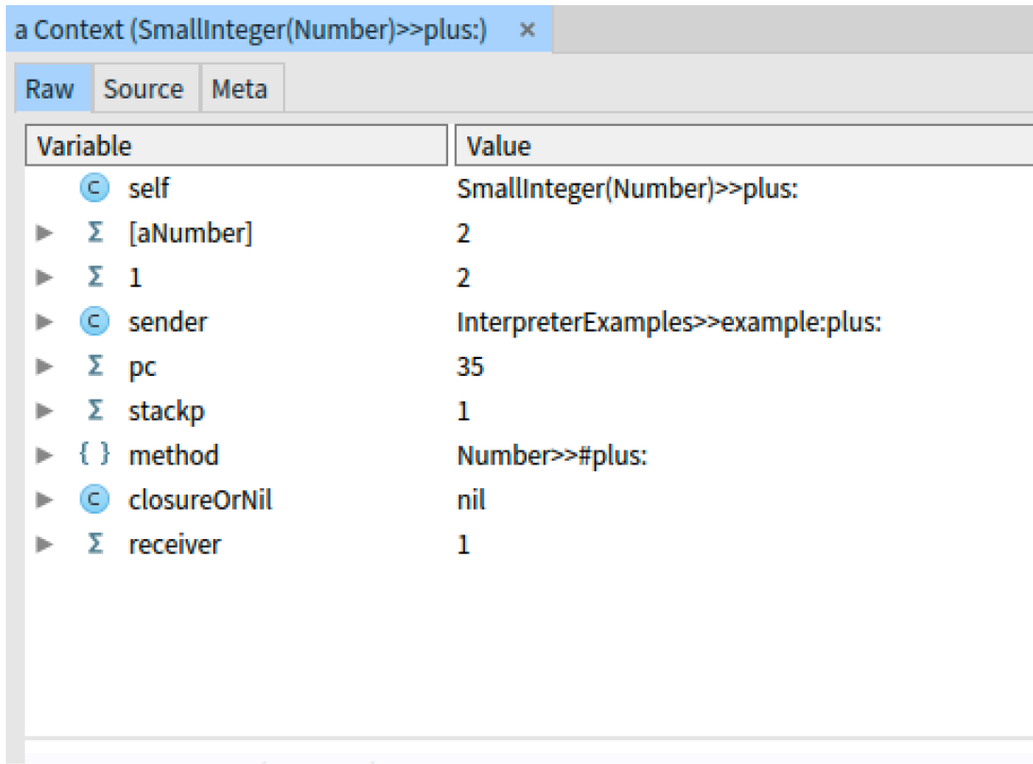
Stack frame life: death

```
example: aNumber plus: aNumberToAdd

    ^aNumber plus: aNumberToAdd
```

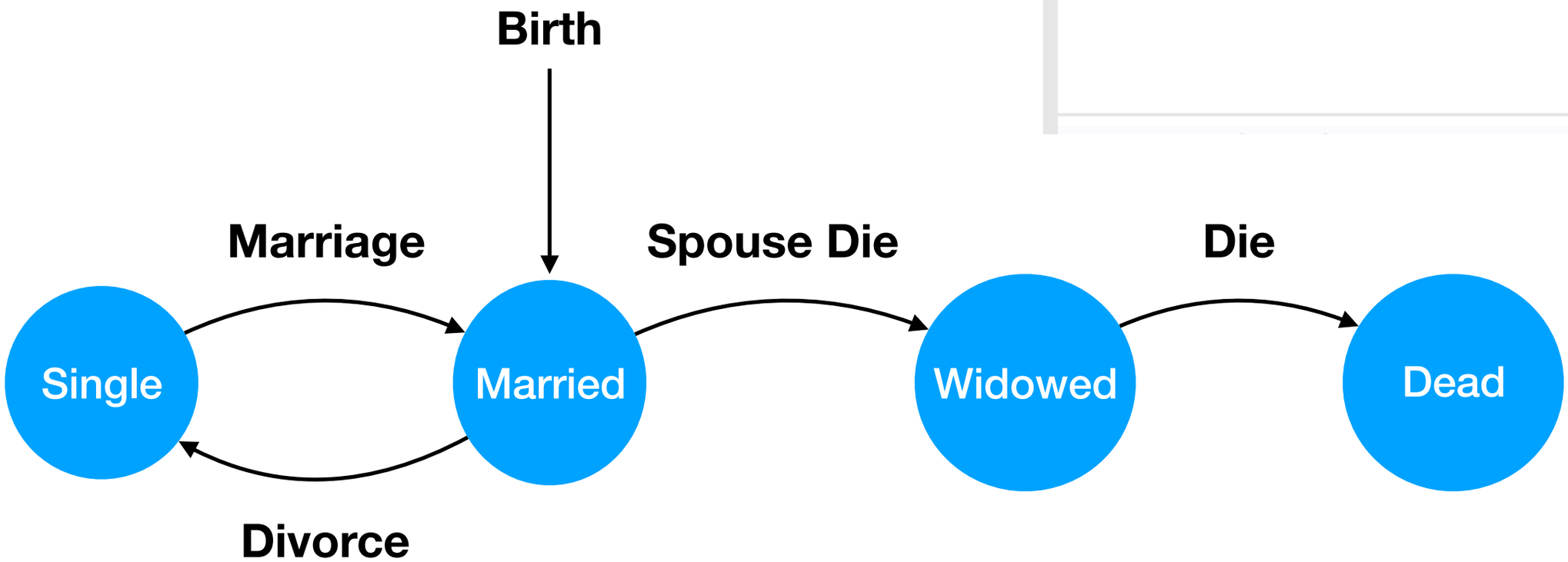




Context life



Automaton for a context



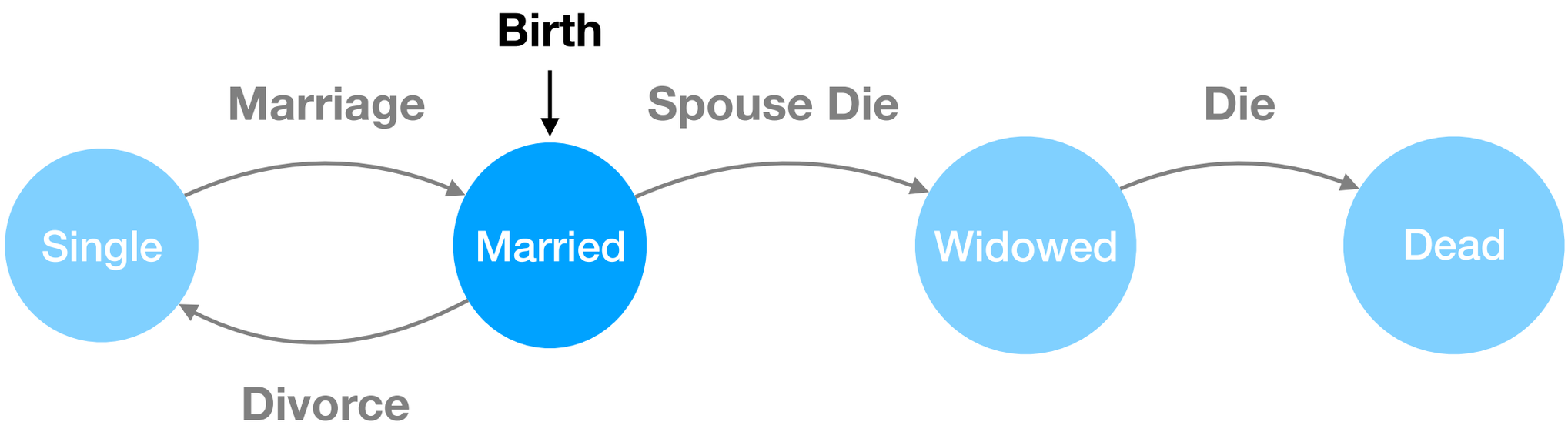
Context life: birth

```
plus: aNumber
    thisContext.
    ^self + aNumber
```

User code

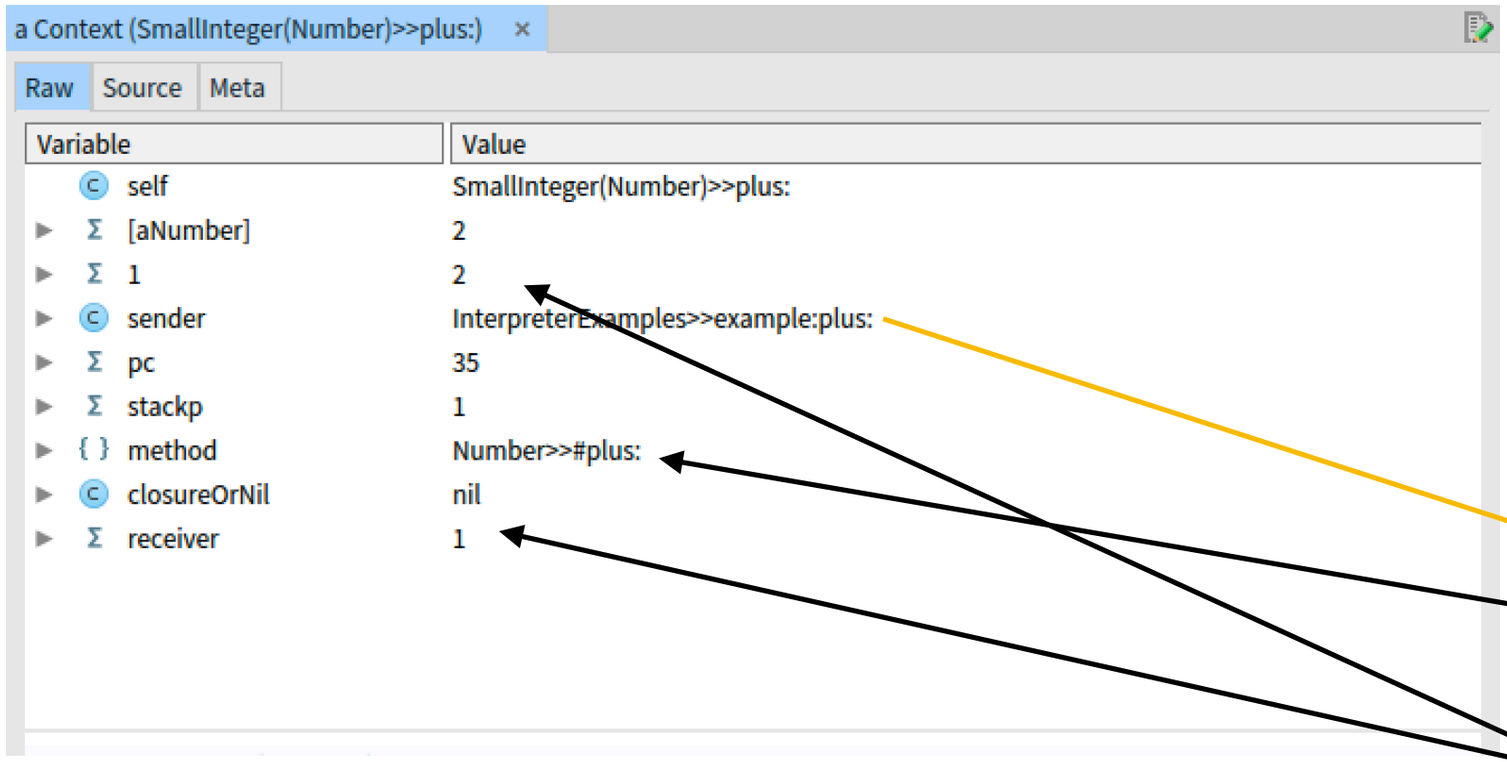
```
marryFrame: theFP SP: theSP copyTemps: copyTemps
```

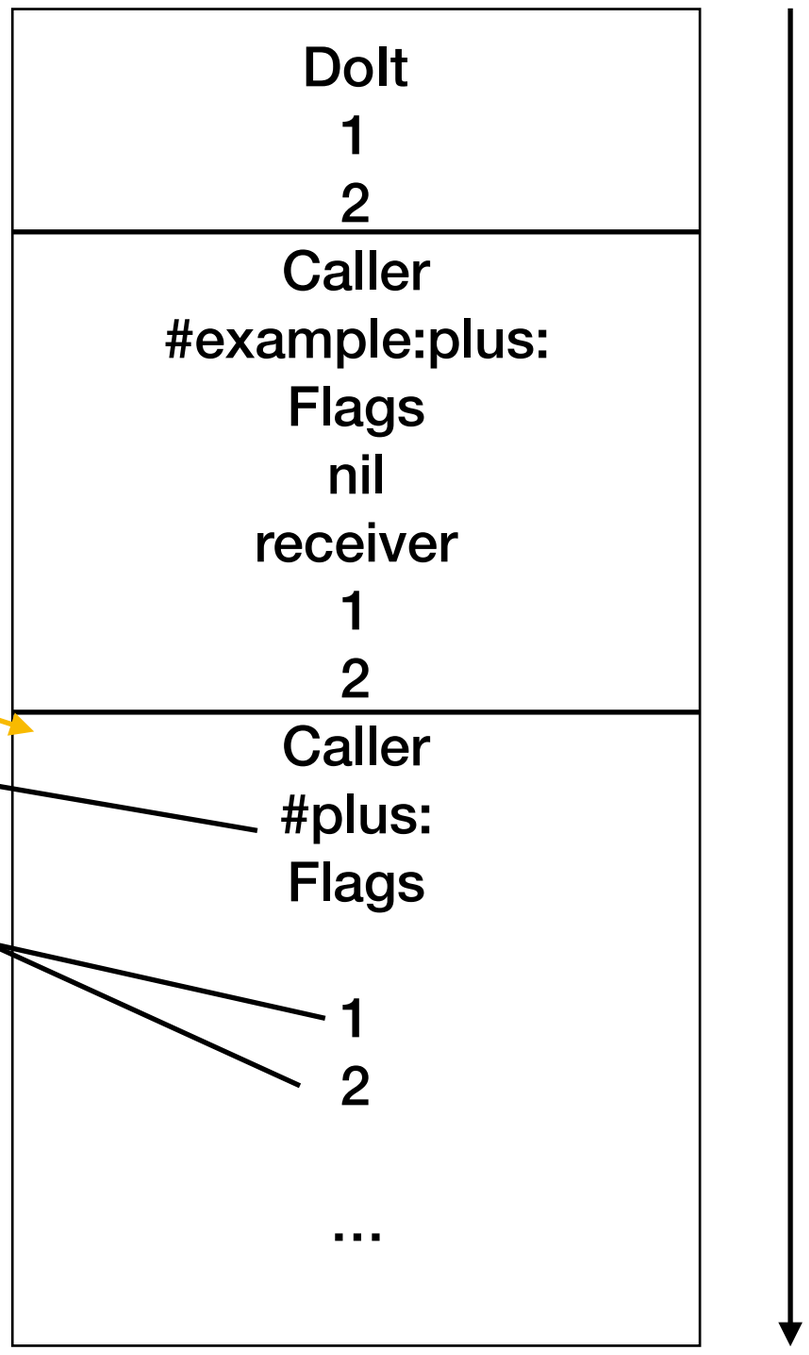
VM code



Context life: birth

```
marryFrame: theFP SP: theSP copyTemps: copyTemps
```





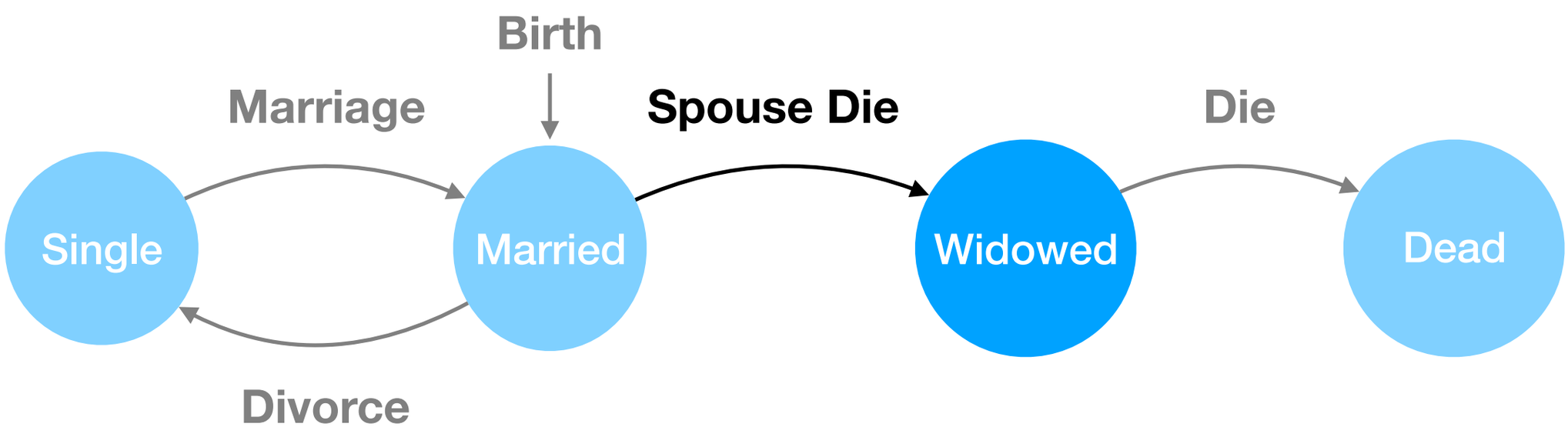
Pointer to the frame.

Context life: widow

```
example: aNumber plus: aNumberToAdd

    ^aNumber plus: aNumberToAdd
```

Method returns = nothing happens for the context

Different for blocks. Not covered in this presentation.



Context life: widow

How does the context know it is widowed ?

```
isWidowedContext: aOnceMarriedContext
   "See if the argument is married to a live frame or not.
    If it is not, turn it into a bereaved single context.."
   | theFrame thePage shouldBeFrameCallerField |
   <var: #theFrame type: #'char *'>
   <var: #thePage type: #'StackPage *'>
   <var: #shouldBeFrameCallerField type: #'char *'>
   self assert: ((objectMemory isContext: aOnceMarriedContext)
              and: [self isMarriedOrWidowedContext: aOnceMarriedContext]).
   theFrame := self frameOfMarriedContext: aOnceMarriedContext.
   thePage := stackPages stackPageFor: theFrame.
   ((stackPages isFree: thePage)
    or: [theFrame < thePage headFP]) ifFalse:
      ["The frame pointer is within the bounds of a live page.
        Now check if it matches a frame."
       shouldBeFrameCallerField := self withoutSmallIntegerTags:
                                   (objectMemory
                                      fetchPointer: InstructionPointerIndex
                                      ofObject: aOnceMarriedContext).
       ((self frameCallerFP: theFrame) = shouldBeFrameCallerField
        and: [self frameHasContext: theFrame]) ifTrue:
         [self deny: (((self isFrame: theFrame onPage: thePage))
                   and: [objectMemory isForwarded: (self frameContext: theFrame)]).
          (self frameContext: theFrame) = aOnceMarriedContext ifTrue: "It is still married!"
            [^false]]].
   "It is out of range or doesn't match the frame's context.
    It is widowed. Time to wear black."
   self markContextAsDead: aOnceMarriedContext.
   ^true
```

Retrieve the pointer to the supposed frame

Context life: widow

How does the context know it is widowed ?

```
isWidowedContext: aOnceMarriedContext
   "See if the argument is married to a live frame or not.
    If it is not, turn it into a bereaved single context.."
   | theFrame thePage shouldBeFrameCallerField |
   <var: #theFrame type: #'char *'>
   <var: #thePage type: #'StackPage *'>
   <var: #shouldBeFrameCallerField type: #'char *'>
   self assert: ((objectMemory isContext: aOnceMarriedContext)
              and: [self isMarriedOrWidowedContext: aOnceMarriedContext]).
   theFrame := self frameOfMarriedContext: aOnceMarriedContext.
   thePage := stackPages stackPageFor: theFrame.
   ((stackPages isFree: thePage)
    or: [theFrame < thePage headFP]) ifFalse:
      ["The frame pointer is within the bounds of a live page.
        Now check if it matches a frame."
       shouldBeFrameCallerField := self withoutSmallIntegerTags:
                                   (objectMemory
                                      fetchPointer: InstructionPointerIndex
                                      ofObject: aOnceMarriedContext).
       ((self frameCallerFP: theFrame) = shouldBeFrameCallerField
        and: [self frameHasContext: theFrame]) ifTrue:
         [self deny: (((self isFrame: theFrame onPage: thePage))
                   and: [objectMemory isForwarded: (self frameContext: theFrame)]).
          (self frameContext: theFrame) = aOnceMarriedContext ifTrue: "It is still married!"
            [^false]]].
   "It is out of range or doesn't match the frame's context.
    It is widowed. Time to wear black."
   self markContextAsDead: aOnceMarriedContext.
   ^true
```

Check if the supposed frame is in a live execution

Context life: widow

How does the context know it is widowed ?

```
isWidowedContext: aOnceMarriedContext
   "See if the argument is married to a live frame or not.
    If it is not, turn it into a bereaved single context.."
   | theFrame thePage shouldBeFrameCallerField |
   <var: #theFrame type: #'char *'>
   <var: #thePage type: #'StackPage *'>
   <var: #shouldBeFrameCallerField type: #'char *'>
   self assert: ((objectMemory isContext: aOnceMarriedContext)
              and: [self isMarriedOrWidowedContext: aOnceMarriedContext]).
   theFrame := self frameOfMarriedContext: aOnceMarriedContext.
   thePage := stackPages stackPageFor: theFrame.
   ((stackPages isFree: thePage)
    or: [theFrame < thePage headFP]) ifFalse:
      ["The frame pointer is within the bounds of a live page.
        Now check if it matches a frame."
       shouldBeFrameCallerField := self withoutSmallIntegerTags:
                                   (objectMemory
                                      fetchPointer: InstructionPointerIndex
                                      ofObject: aOnceMarriedContext).
       ((self frameCallerFP: theFrame) = shouldBeFrameCallerField
        and: [self frameHasContext: theFrame]) ifTrue:
         [self deny: (((self isFrame: theFrame onPage: thePage))
                   and: [objectMemory isForwarded: (self frameContext: theFrame)]).
          (self frameContext: theFrame) = aOnceMarriedContext ifTrue: "It is still married!"
            [^false]]].
   "It is out of range or doesn't match the frame's context.
    It is widowed. Time to wear black."
   self markContextAsDead: aOnceMarriedContext.
   ^true
```

Check if the supposed frame is still live and has a context

Context life: widow

How does the context know it is widowed ?

```
isWidowedContext: aOnceMarriedContext
   "See if the argument is married to a live frame or not.
    If it is not, turn it into a bereaved single context.."
   | theFrame thePage shouldBeFrameCallerField |
   <var: #theFrame type: #'char *'>
   <var: #thePage type: #'StackPage *'>
   <var: #shouldBeFrameCallerField type: #'char *'>
   self assert: ((objectMemory isContext: aOnceMarriedContext)
              and: [self isMarriedOrWidowedContext: aOnceMarriedContext]).
   theFrame := self frameOfMarriedContext: aOnceMarriedContext.
   thePage := stackPages stackPageFor: theFrame.
   ((stackPages isFree: thePage)
    or: [theFrame < thePage headFP]) ifFalse:
      ["The frame pointer is within the bounds of a live page.
        Now check if it matches a frame."
       shouldBeFrameCallerField := self withoutSmallIntegerTags:
                                   (objectMemory
                                      fetchPointer: InstructionPointerIndex
                                      ofObject: aOnceMarriedContext).
       ((self frameCallerFP: theFrame) = shouldBeFrameCallerField
        and: [self frameHasContext: theFrame]) ifTrue:
         [self deny: (((self isFrame: theFrame onPage: thePage))
                   and: [objectMemory isForwarded: (self frameContext: theFrame)]).
          (self frameContext: theFrame) = aOnceMarriedContext ifTrue: "It is still married!"
            [^false]]].
   "It is out of range or doesn't match the frame's context.
    It is widowed. Time to wear black."
   self markContextAsDead: aOnceMarriedContext.
   ^true
```

Check if the supposed frame points back to the context.

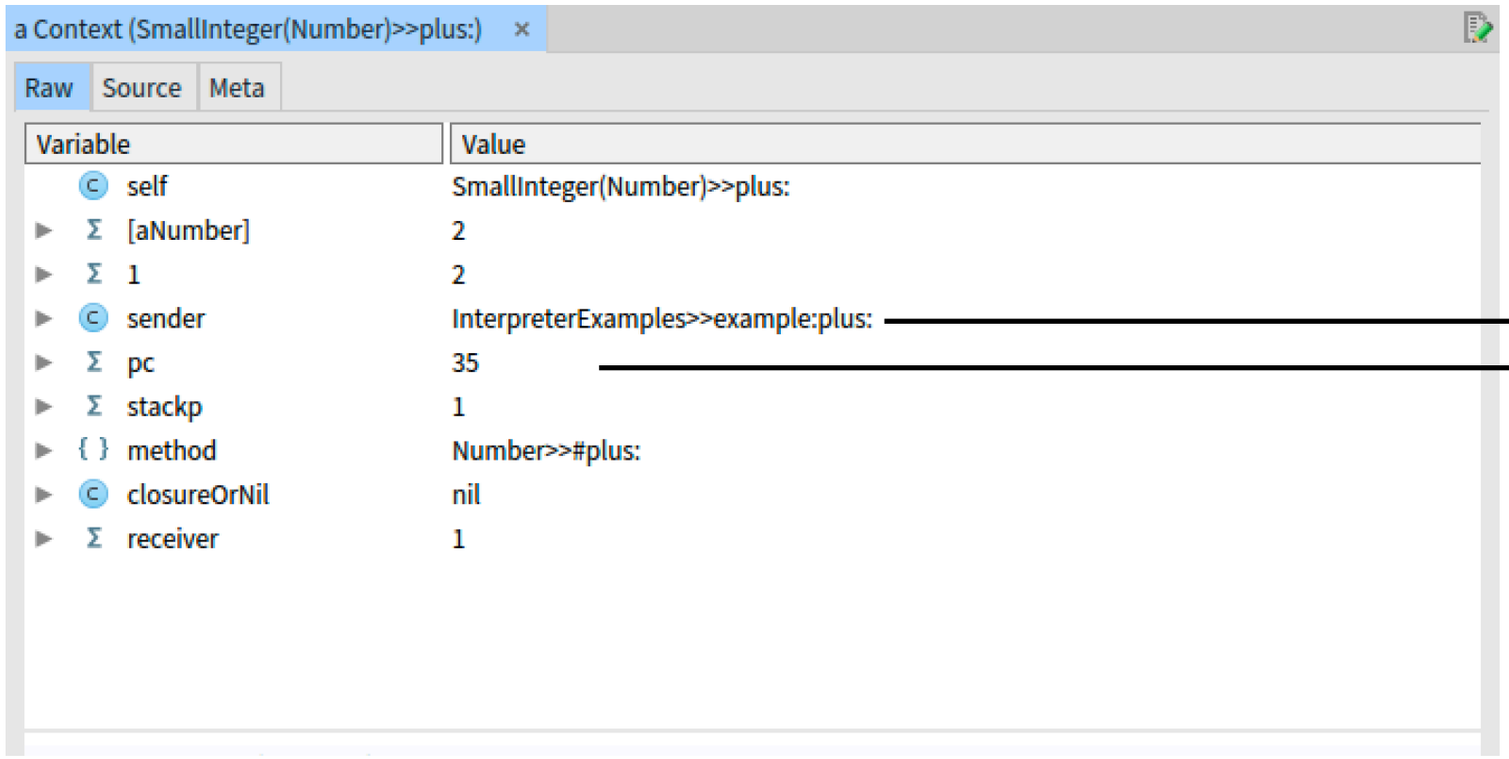
Context life: widow

How does the context know it is widowed ?

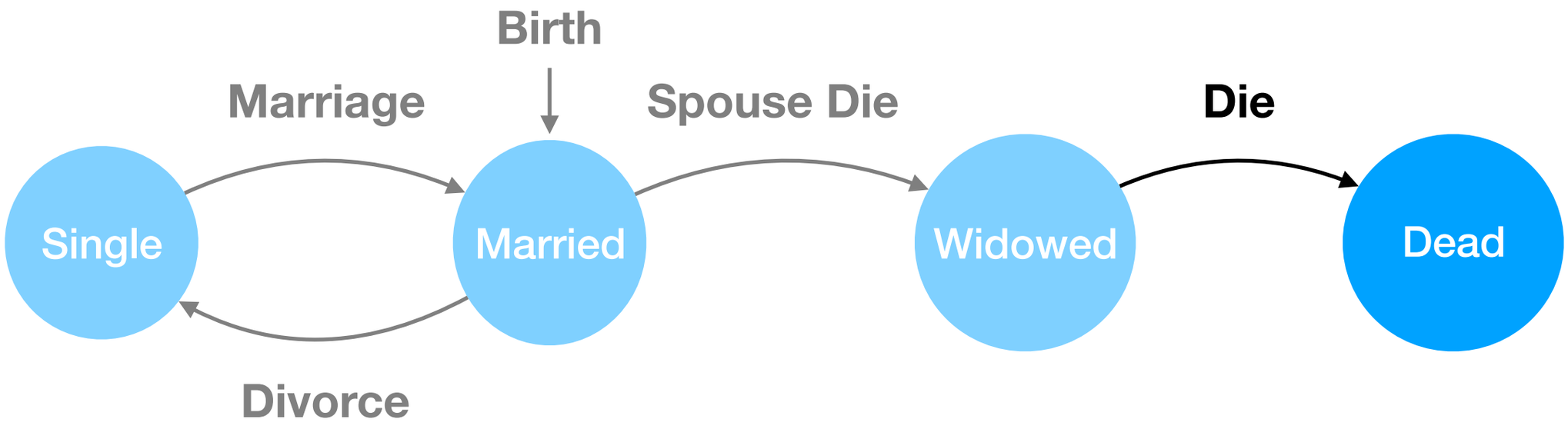
```
isWidowedContext: aOnceMarriedContext
   "See if the argument is married to a live frame or not.
    If it is not, turn it into a bereaved single context.."
   | theFrame thePage shouldBeFrameCallerField |
   <var: #theFrame type: #'char *'>
   <var: #thePage type: #'StackPage *'>
   <var: #shouldBeFrameCallerField type: #'char *'>
   self assert: ((objectMemory isContext: aOnceMarriedContext)
              and: [self isMarriedOrWidowedContext: aOnceMarriedContext]).
   theFrame := self frameOfMarriedContext: aOnceMarriedContext.
   thePage := stackPages stackPageFor: theFrame.
   ((stackPages isFree: thePage)
    or: [theFrame < thePage headFP]) ifFalse:
      ["The frame pointer is within the bounds of a live page.
        Now check if it matches a frame."
       shouldBeFrameCallerField := self withoutSmallIntegerTags:
                                   (objectMemory
                                      fetchPointer: InstructionPointerIndex
                                      ofObject: aOnceMarriedContext).
       ((self frameCallerFP: theFrame) = shouldBeFrameCallerField
        and: [self frameHasContext: theFrame]) ifTrue:
         [self deny: (((self isFrame: theFrame onPage: thePage))
                   and: [objectMemory isForwarded: (self frameContext: theFrame)]).
          (self frameContext: theFrame) = aOnceMarriedContext ifTrue: "It is still married!"
            [^false]]].
   "It is out of range or doesn't match the frame's context.
    It is widowed. Time to wear black."
   self markContextAsDead: aOnceMarriedContext.
   ^true
```

Update the context

Context life: death

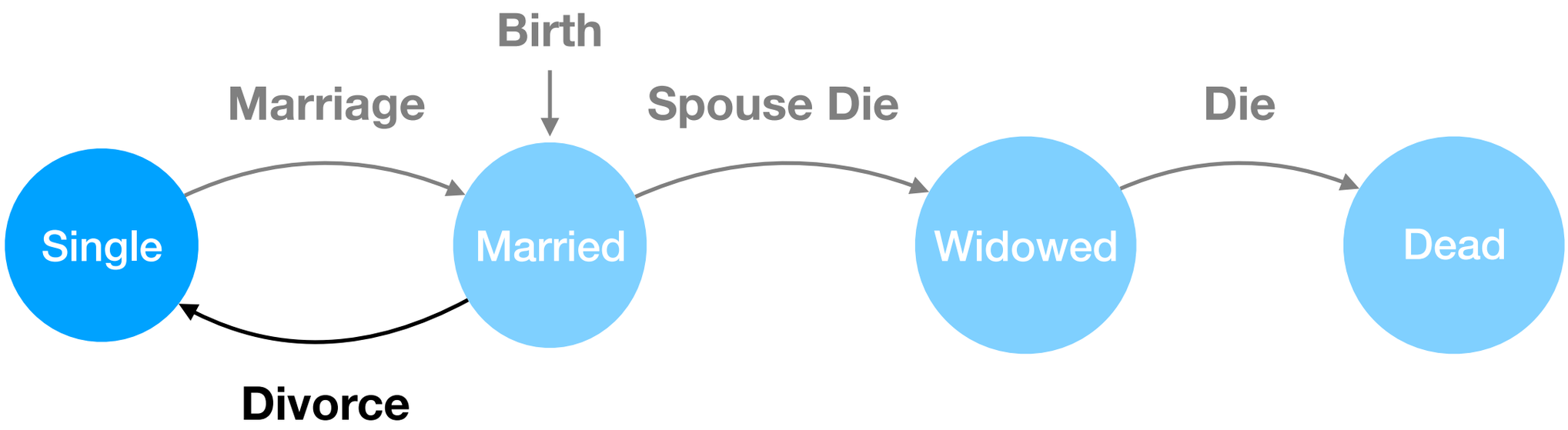


Nobody hold a reference to the context anymore, it is garbage collected.



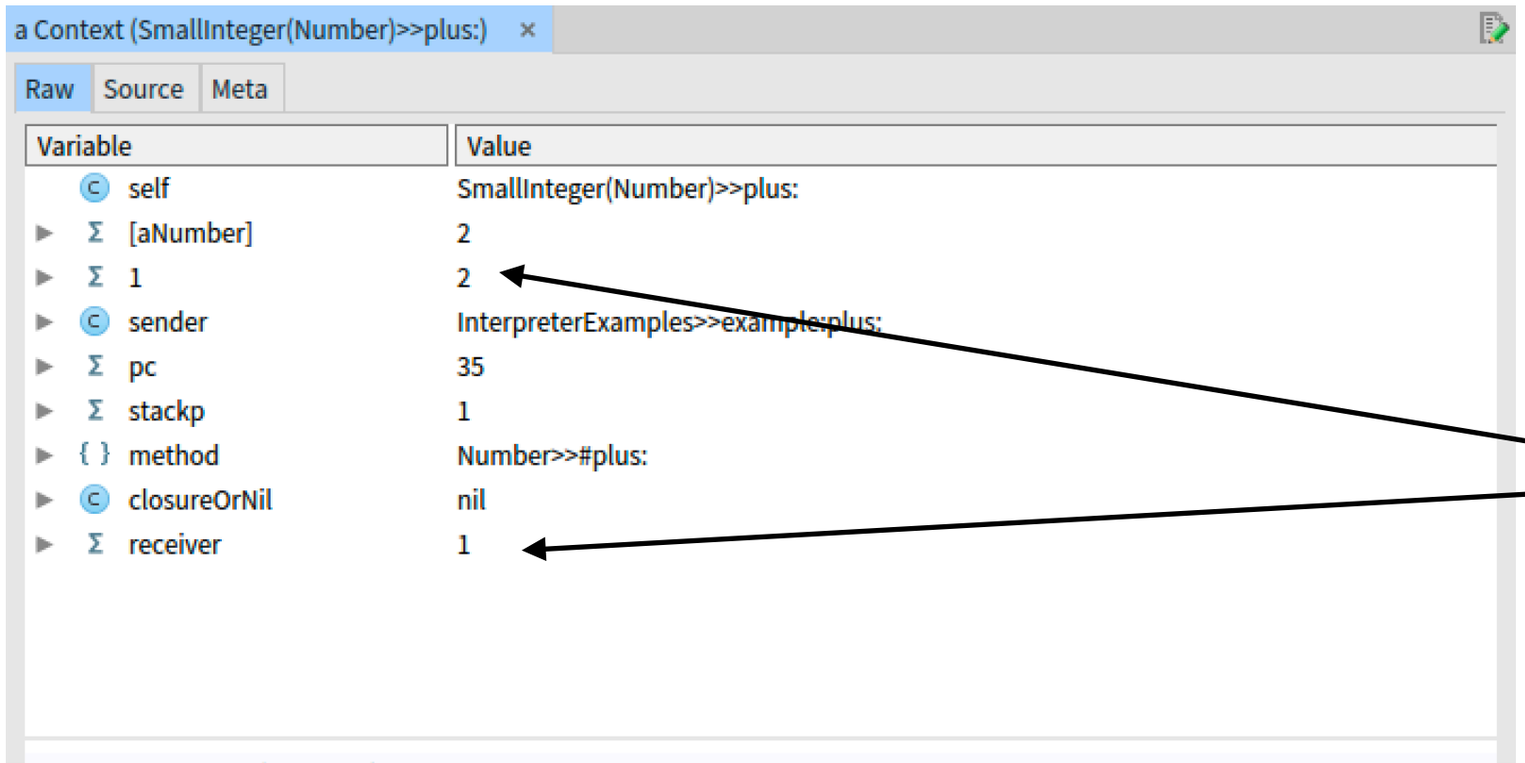
Context life: divorce

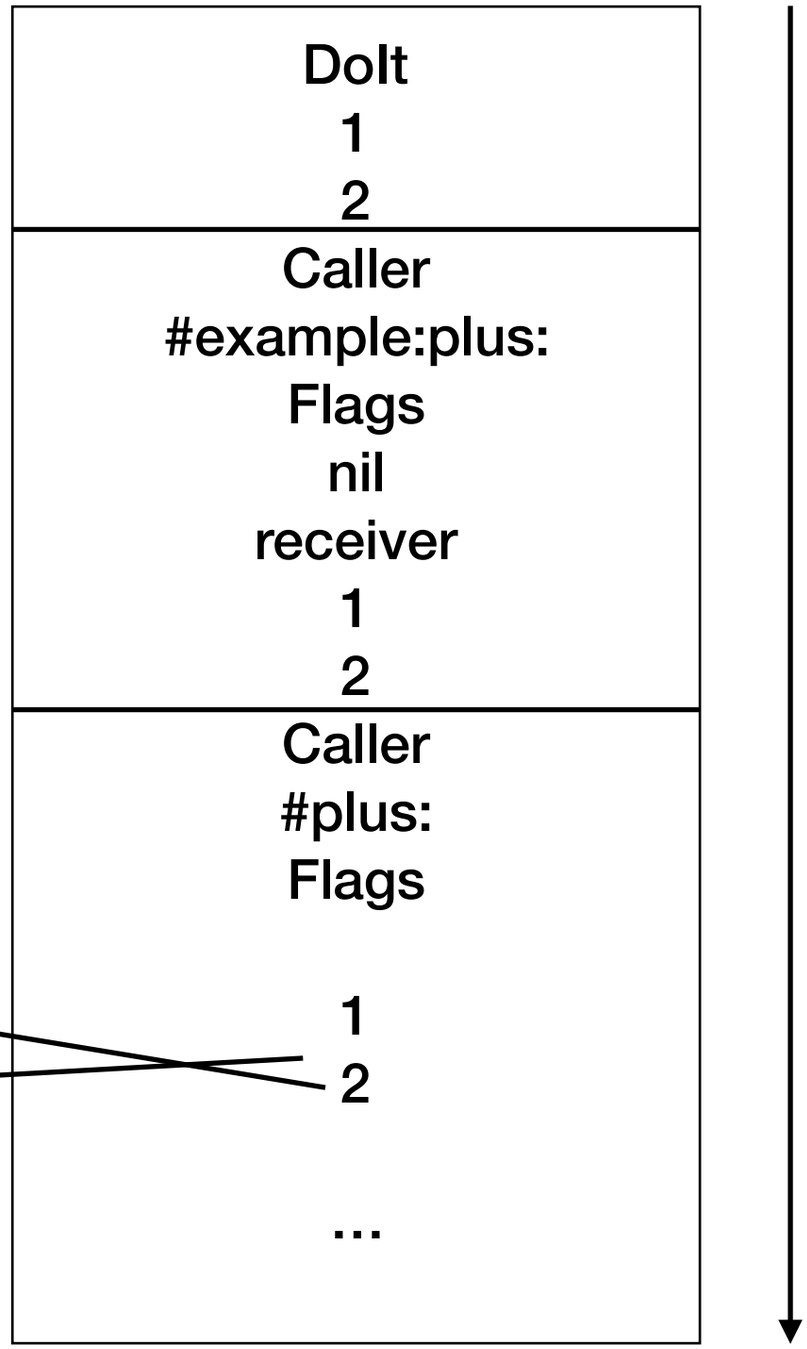
```
plus: aNumber                                   = receiverIndex
    thisContext at: 6 put: 2.                   User code
    ^self + aNumber

externalDivorceFrame: theFP andContext: ctxt    VM code
```



Context life: divorce

Save state





Context life: divorce

a Context (SmallInteger(Number)>>plus:)

Raw | Source | Meta

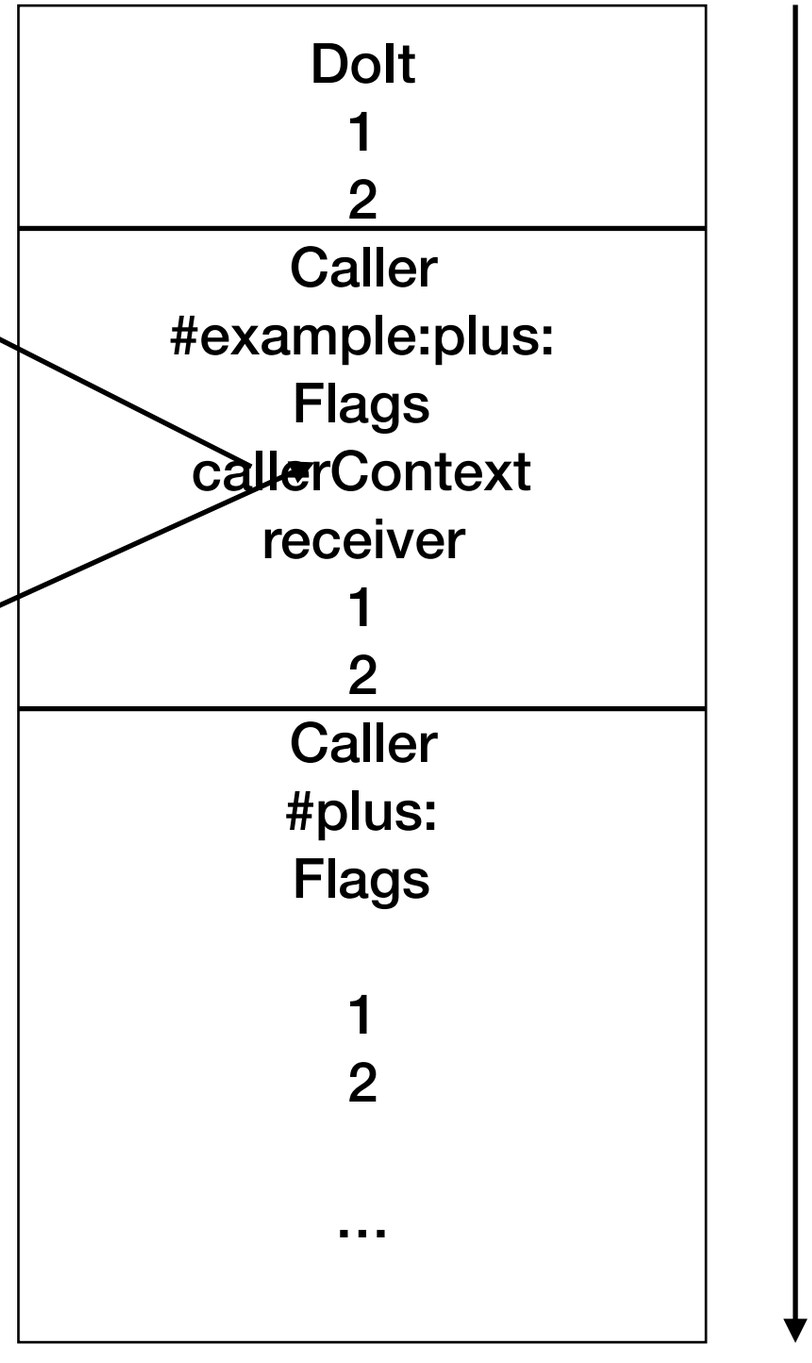
Variable	Value
self	SmallInteger(Number)>>plus:
[aNumber]	2
1	2
2	1
3	2
sender	InterpreterExamples>>example:plus:
pc	27
stackp	3
method	Number>>#plus:
closureOrNil	nil
receiver	1

Create caller context and put it as sender

a Context (SmallInteger(Number)>>plus:)

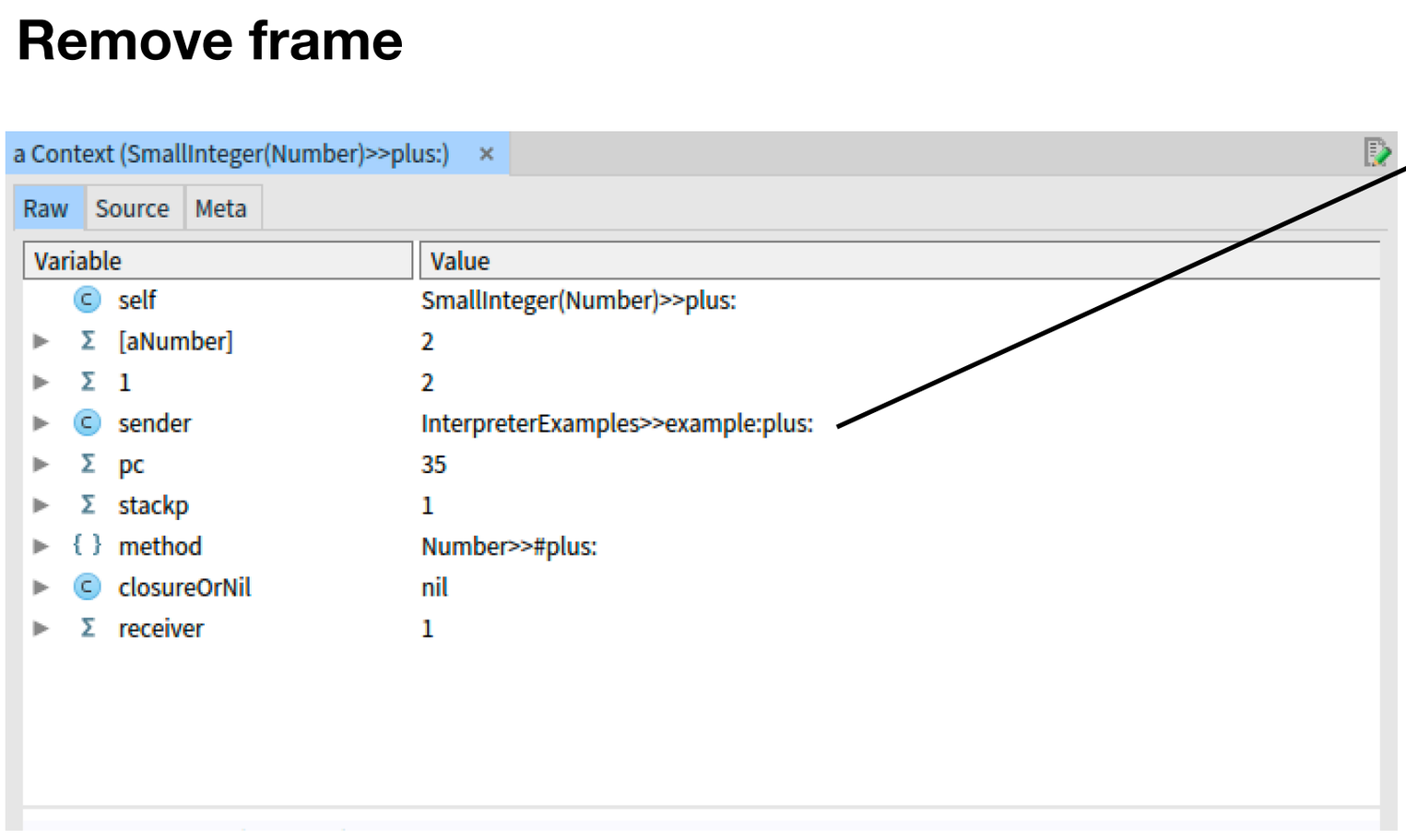
Raw | Source | Meta

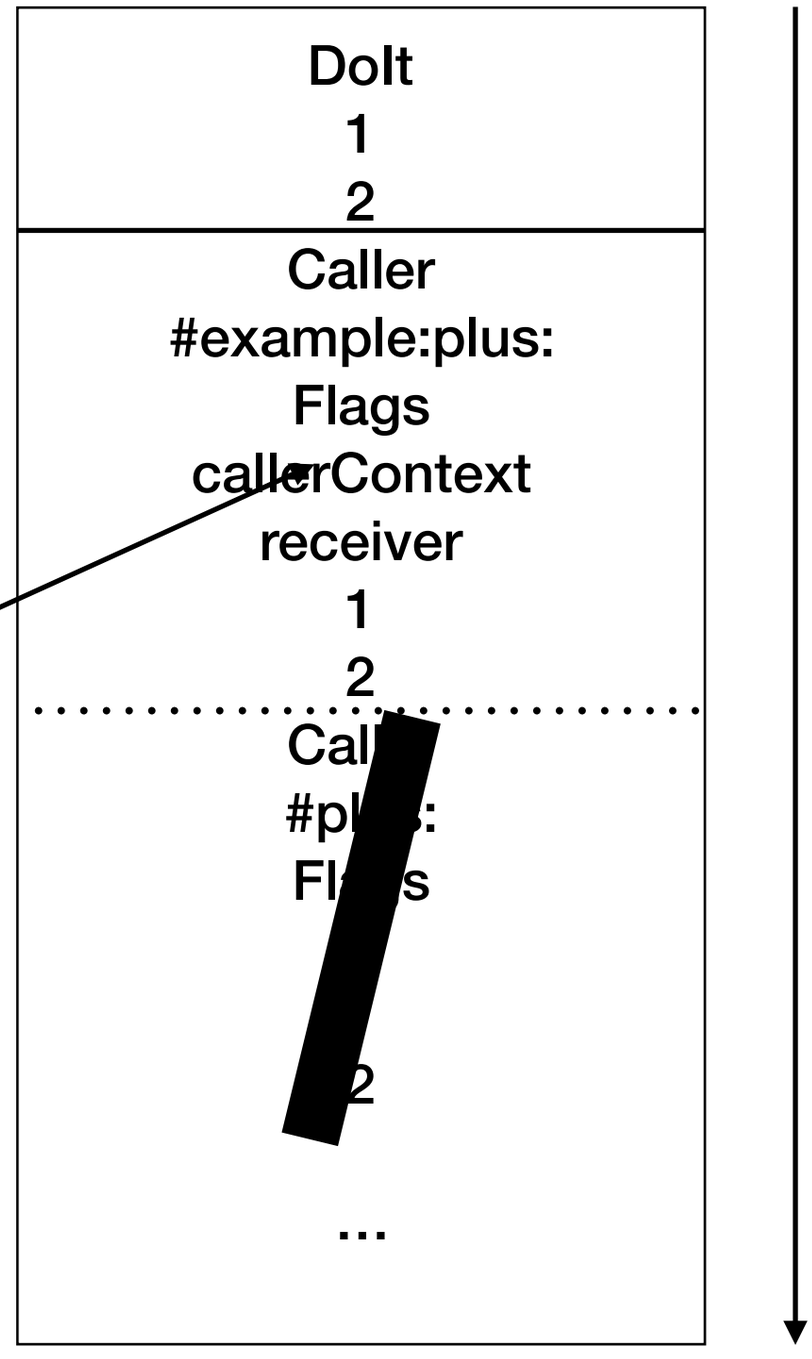
Variable	Value
self	SmallInteger(Number)>>plus:
[aNumber]	2
1	2
sender	InterpreterExamples>>example:plus:
pc	35
stackp	1
method	Number>>#plus:
closureOrNil	nil
receiver	1



Context life: divorce

Remove frame

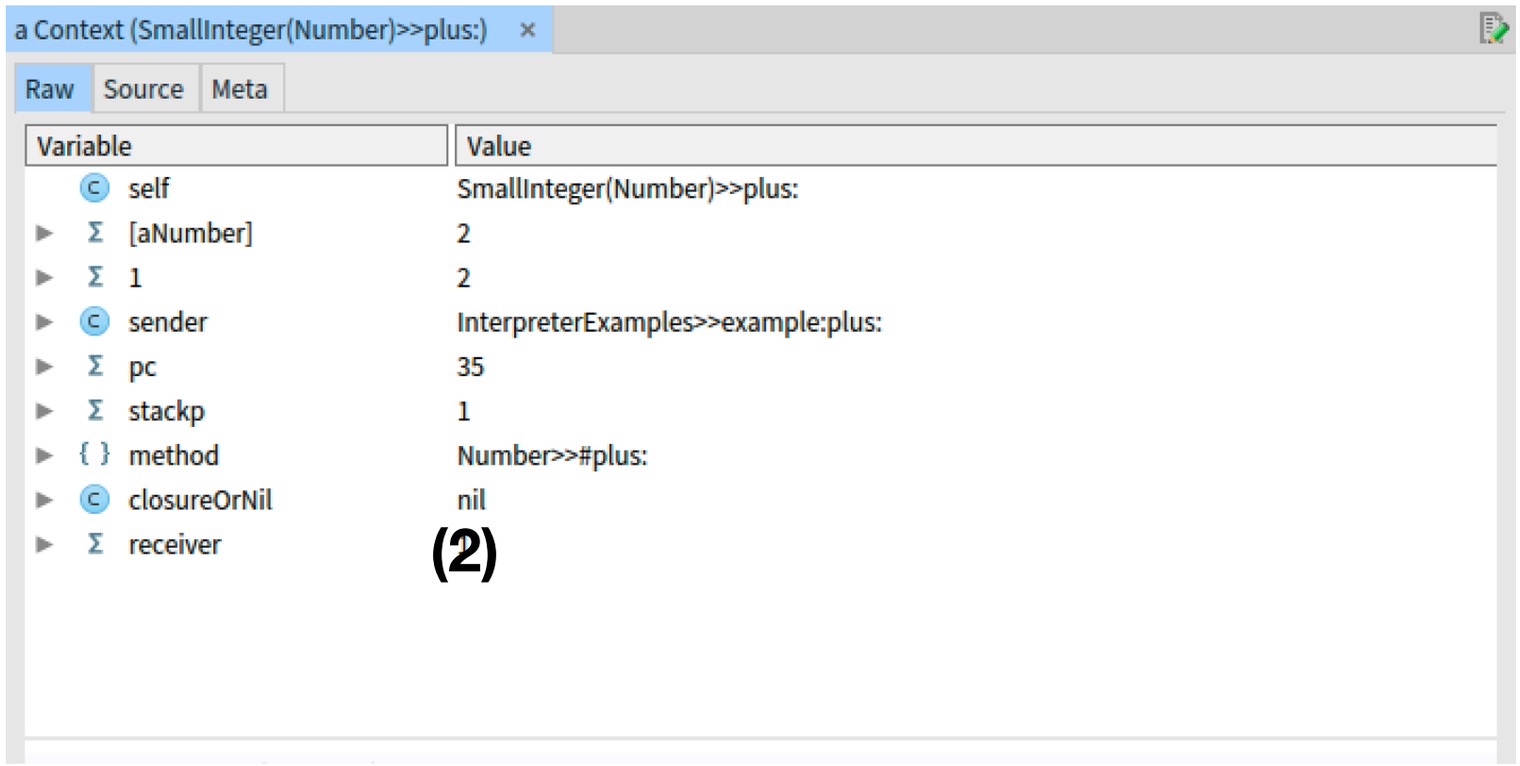




Context life: divorce

```
plus: aNumber
    thisContext at: 6 put: 2.
    ^self + aNumber
```

Apply the change to the context !

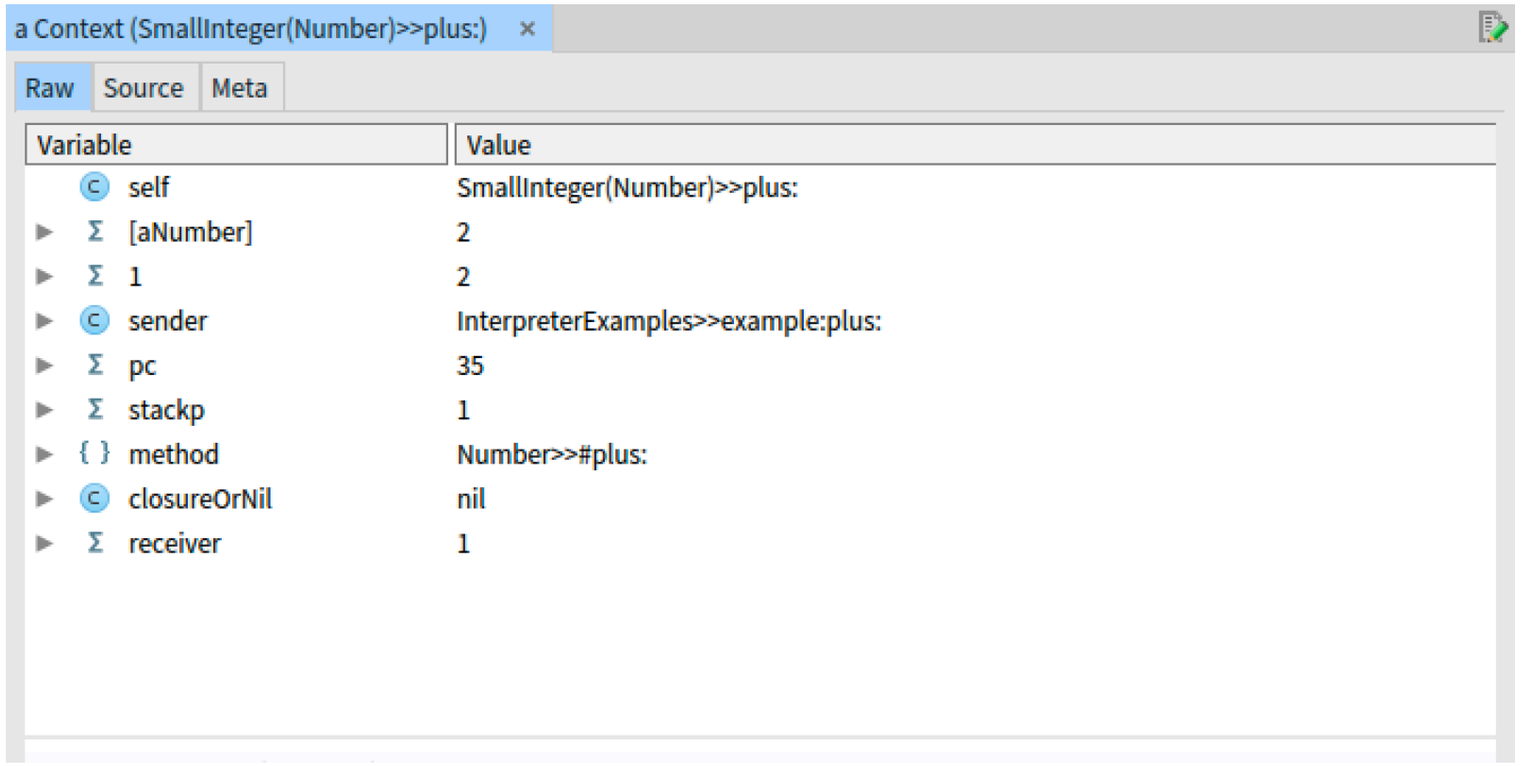


Context life: single

How does the context know it is single ?

```
isSingleContext: aContext
    ^objectMemory isNonImmediate: (objectMemory fetchPointer: SenderIndex ofObject: aContext)
```

The sender is a reference to another context

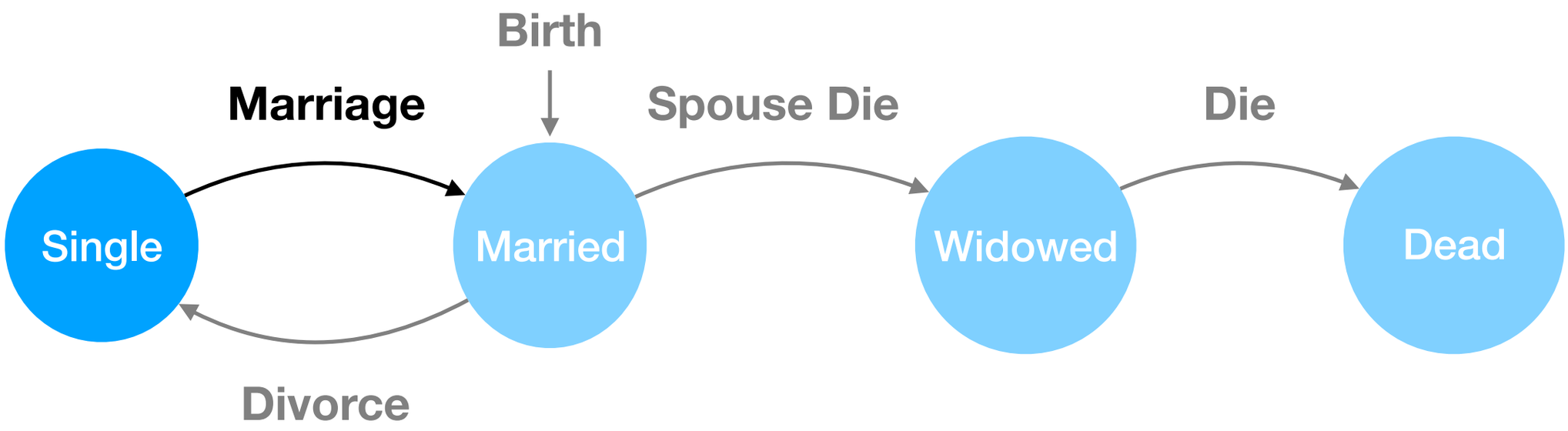


Context life: marriage

```
plus: aNumber
    | aContext |
    aContext := thisContext at: 6 put: 2.
    aContext jump.
    ^ self + aNumber
```

User code

We will create a frame according to the context.



(

How does the context know it is married ?

```
isStillMarriedContext: aContext
    "Answer if aContext is married or widowed and still married.
     If a context is widowed then turn it into a single dead context."
    ^(self isMarriedOrWidowedContext: aContext)
        and: [(self isWidowedContext: aContext) not]
```

```
isMarriedOrWidowedContext: aContext
    ^objectMemory isIntegerObject: (objectMemory fetchPointer: SenderIndex ofObject: aContext)
```

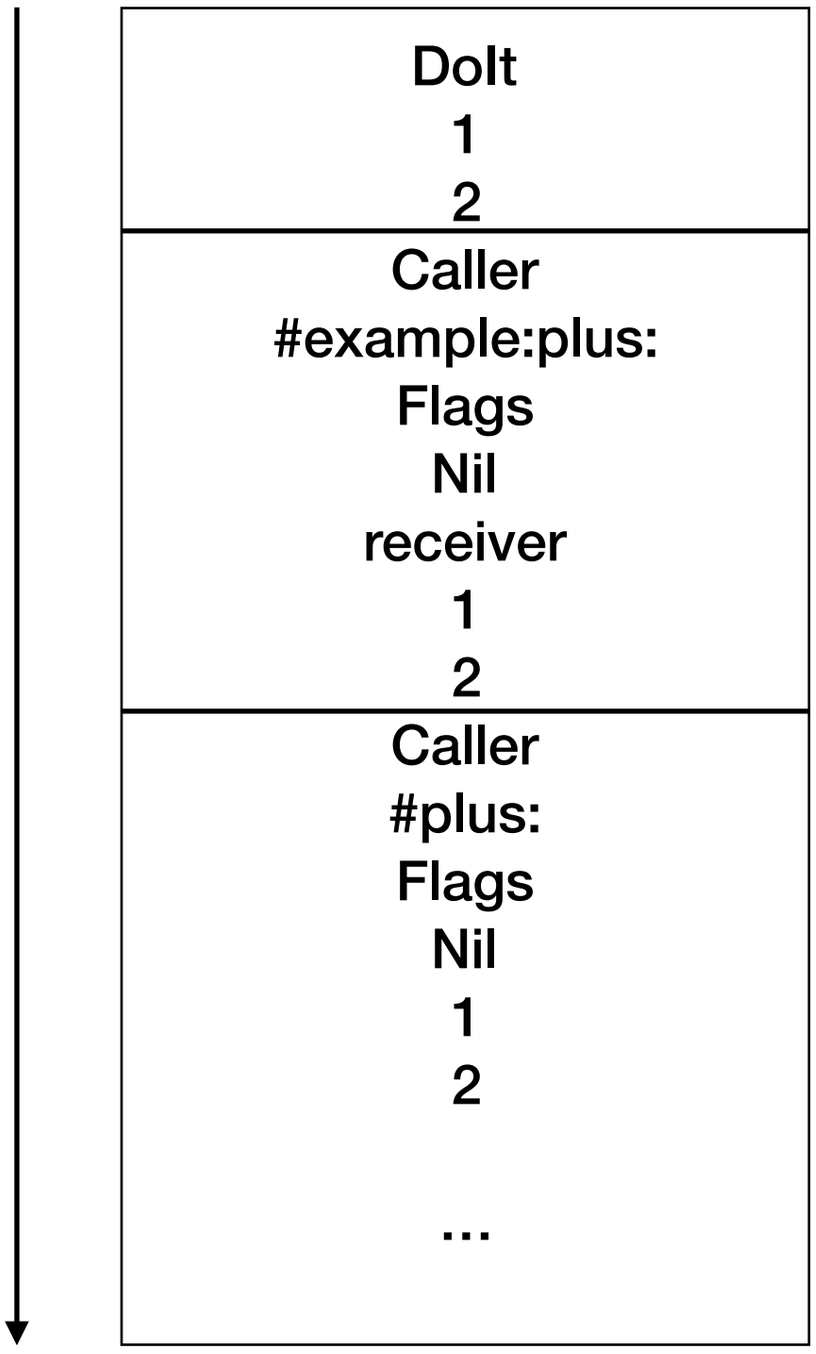
= (isSingleContext: aContext) not

It is a pointer inside the stack frame disguised as an integer object.

```
isStillMarriedContext: aContext
    "Answer if aContext is married or widowed and still married.
     If a context is widowed then turn it into a single dead context."
    ^(self isMarriedOrWidowedContext: aContext)
        and: [(self isWidowedContext: aContext) not]
```

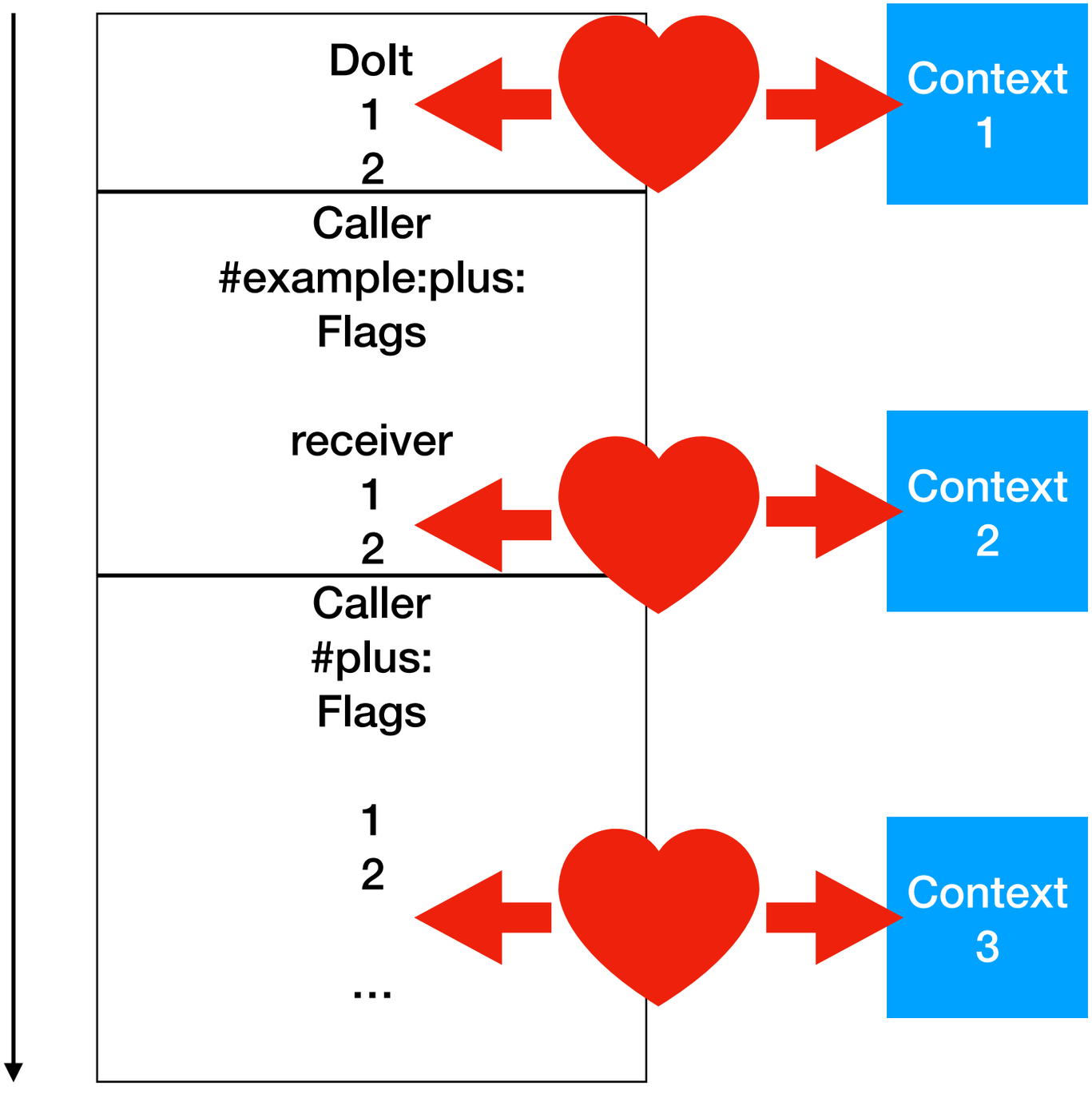
)

Other usage of context mapping



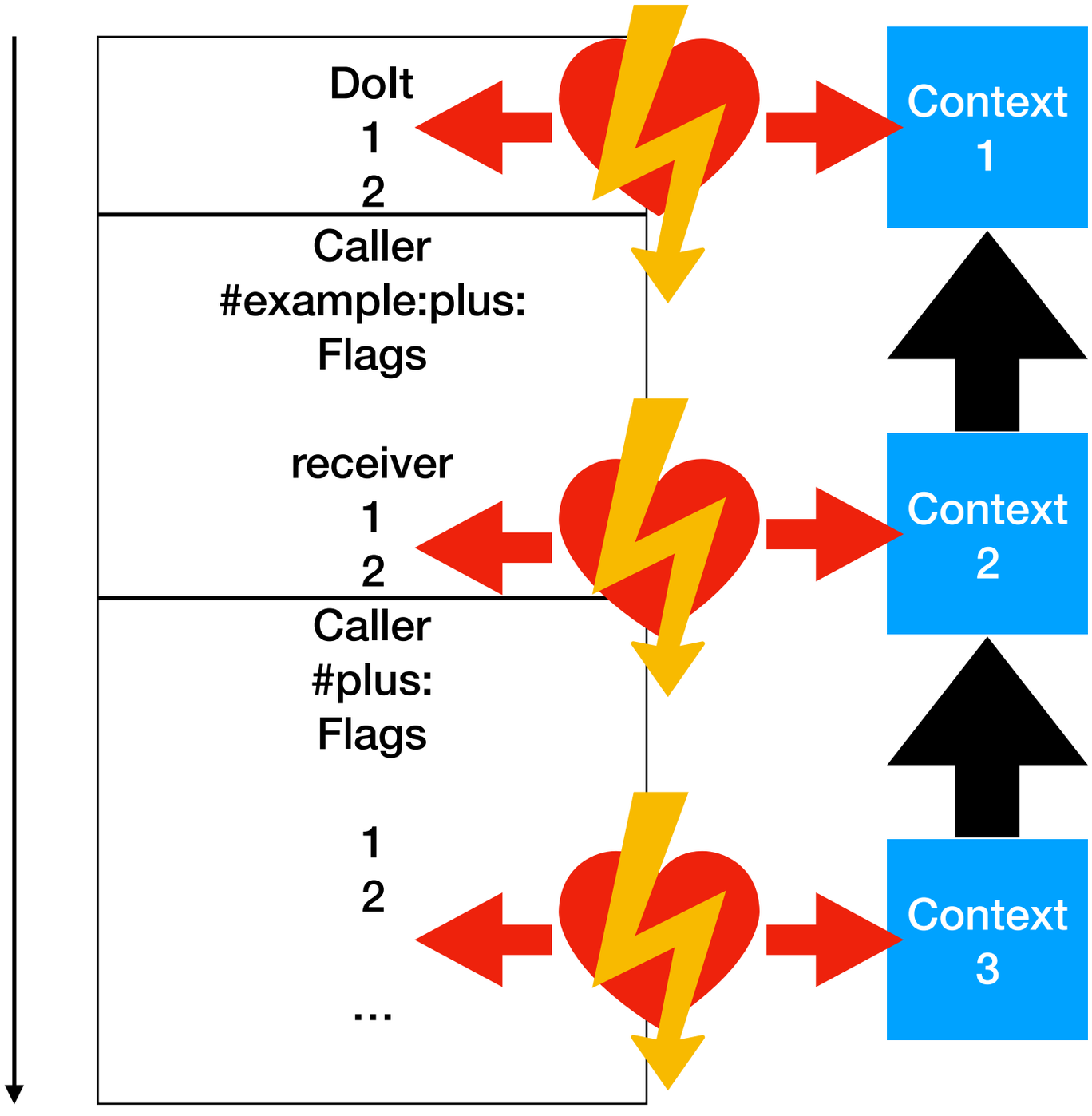
How to save the execution ?

Example snapshot



Marry all frames

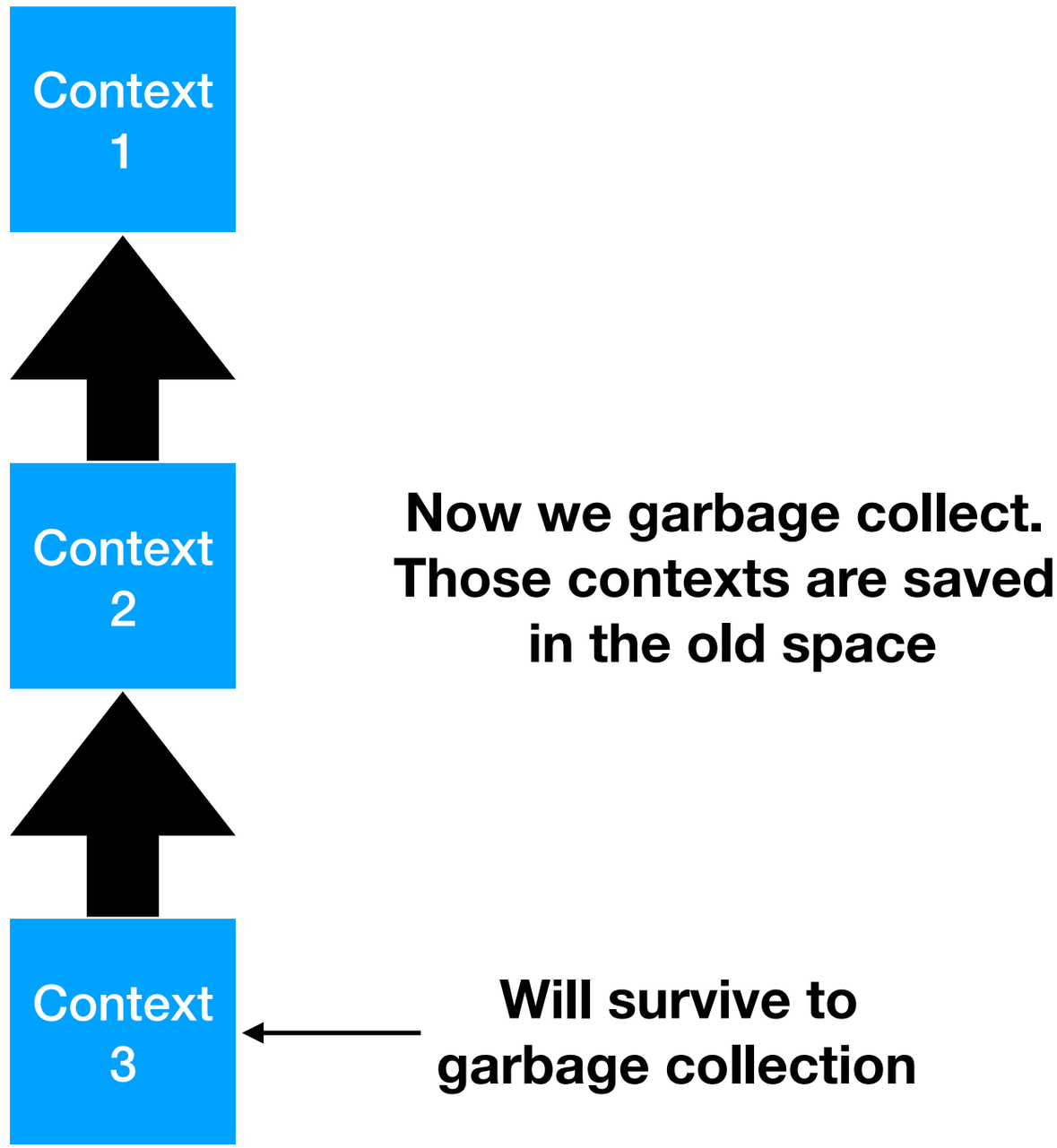
Example snapshot



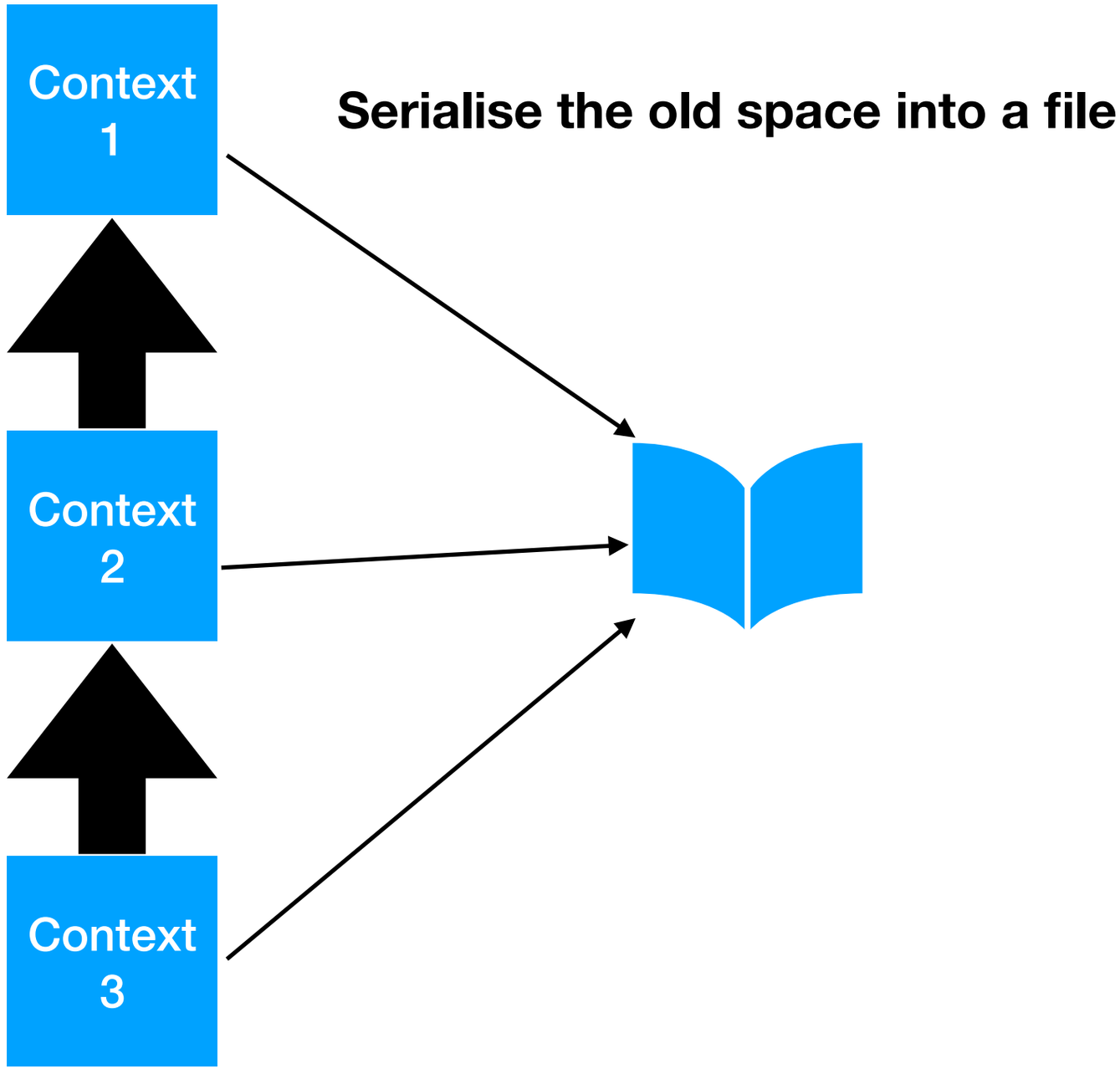
Divorce all frames

Now the stack is discarded

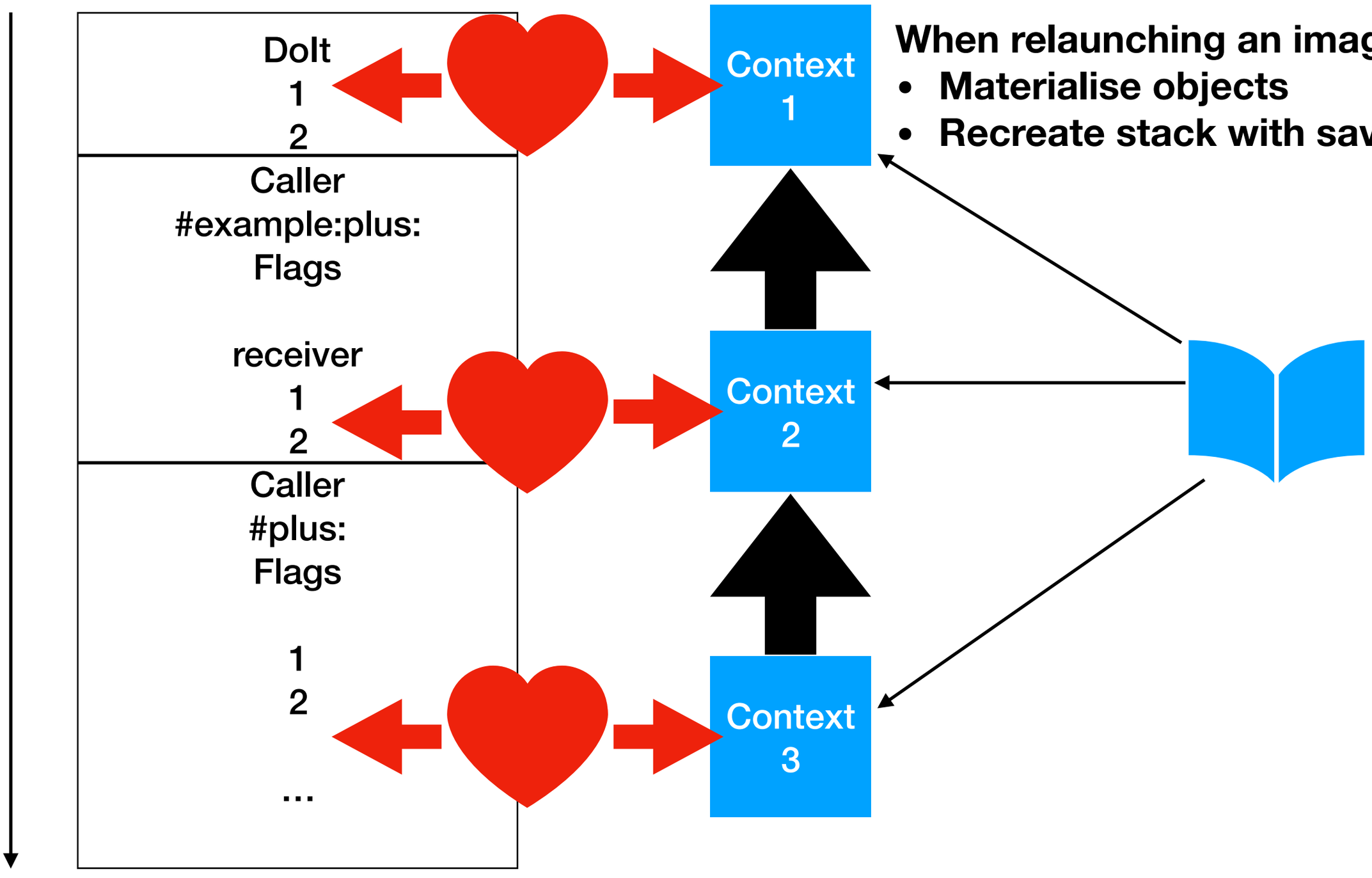
Example snapshot



Example snapshot



Example snapshot



References

Elliot Miranda article

https://www.researchgate.net/publication/2614781_Context_Management_in_VisualWorks_5i

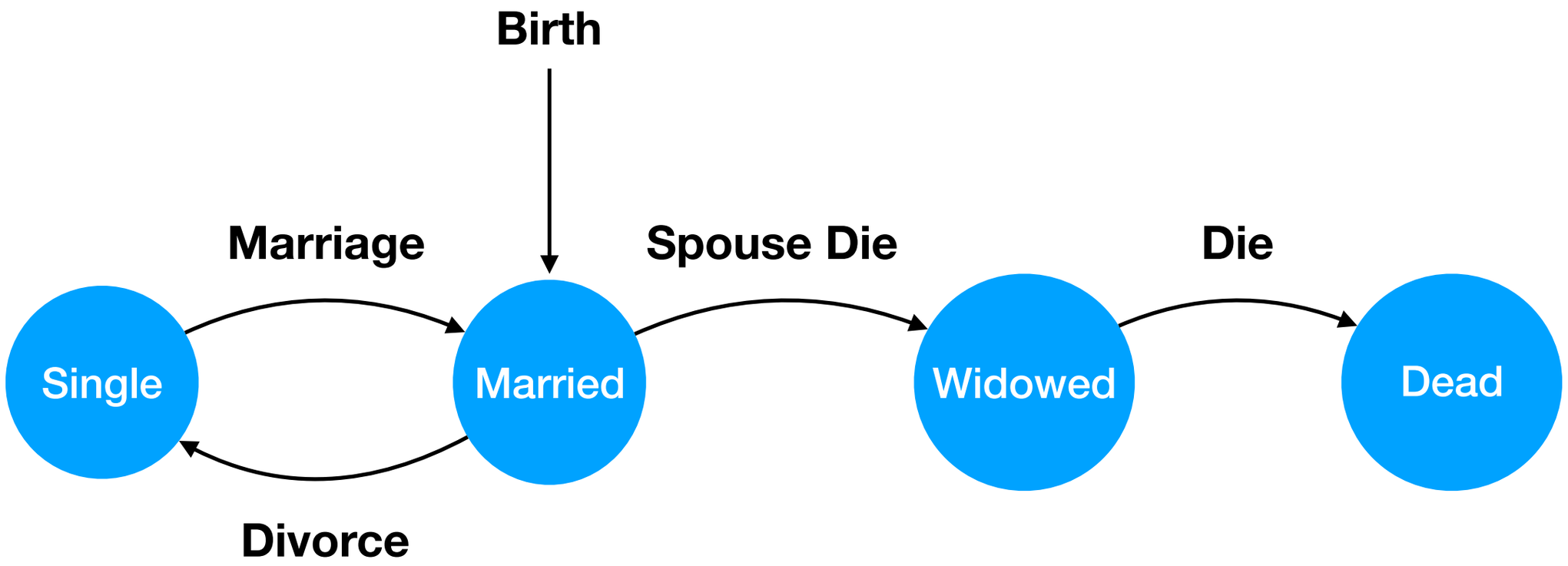
Elliot Miranda blogpost

http://www.mirandabanda.org/cogblog/2011/03/04/an-arranged-marriage/

StackMappingTest

https://github.com/pharo-project/opensmalltalk-vm/tree/headless

Automaton for a context



Automaton for a frame