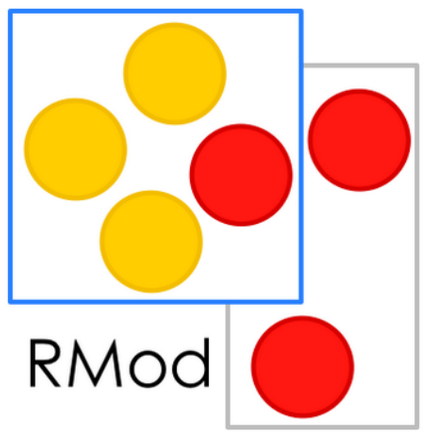
BlockClosures viewed by the stack interpreter

VM presentation

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



Plan

- BlockClosure creation
- The value message
- Return and non local return
- Unwind protect

BlockClosure

= Deferred sequence of operations between 2 brackets.

```
methodReturningABlock

  ^[ ]
```

PushClosure bytecode

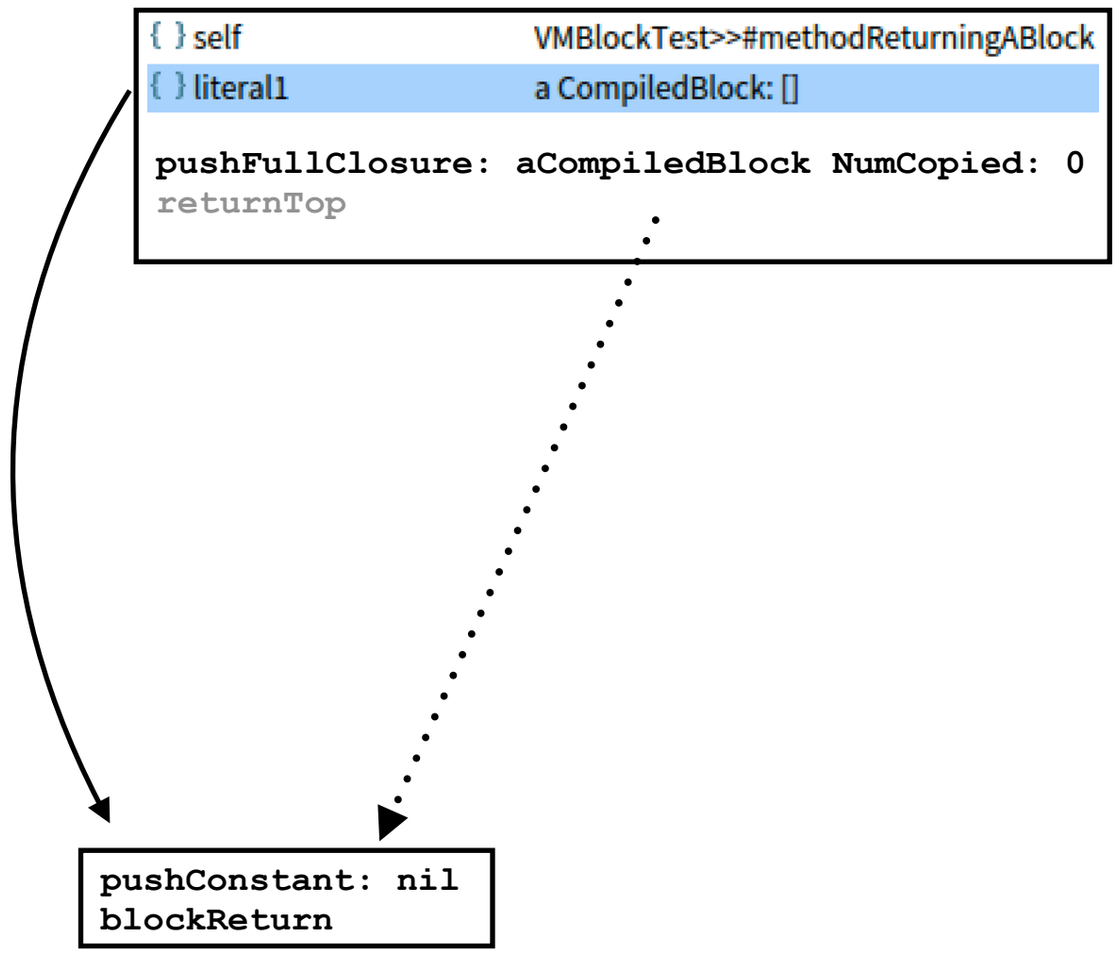
```
methodReturningABlock

  ^[ ]
```

=

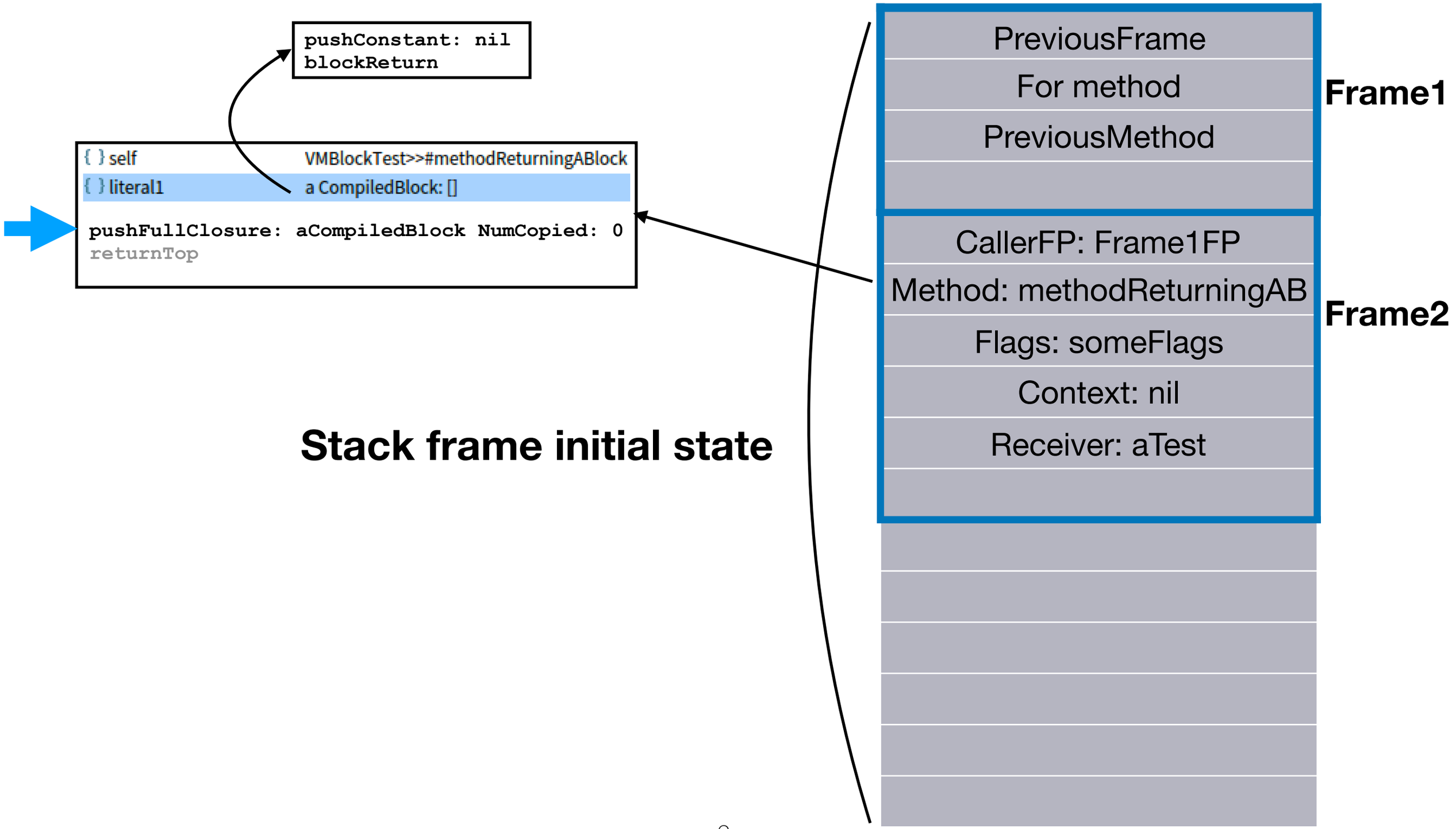
```
pushFullClosure: aCompiledBlock NumCopied: 0
returnTop
```

Where is the BlockClosure's code stored ?

```
methodReturningABlock

    ^[ ]
```

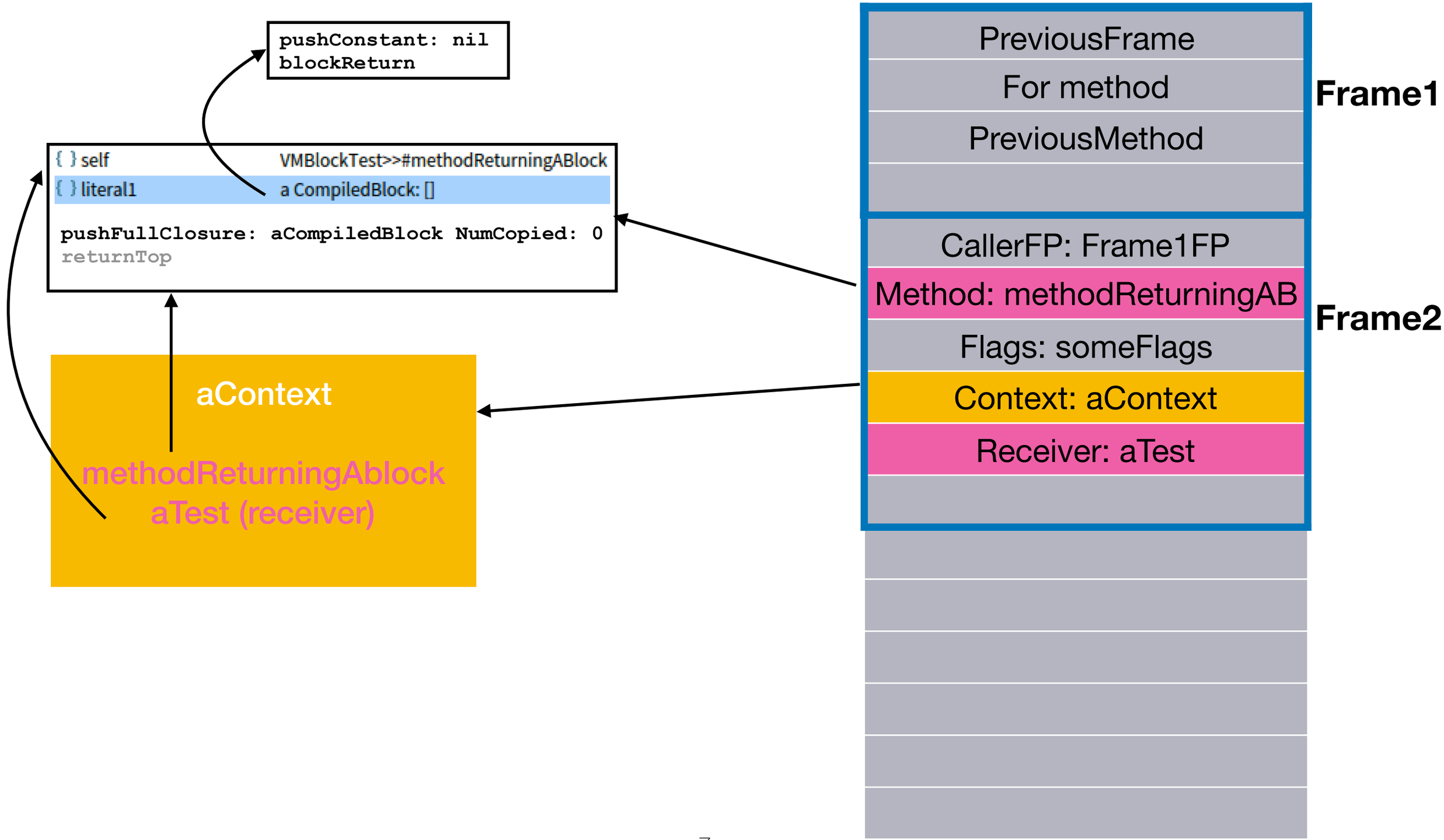


Executing the bytecode

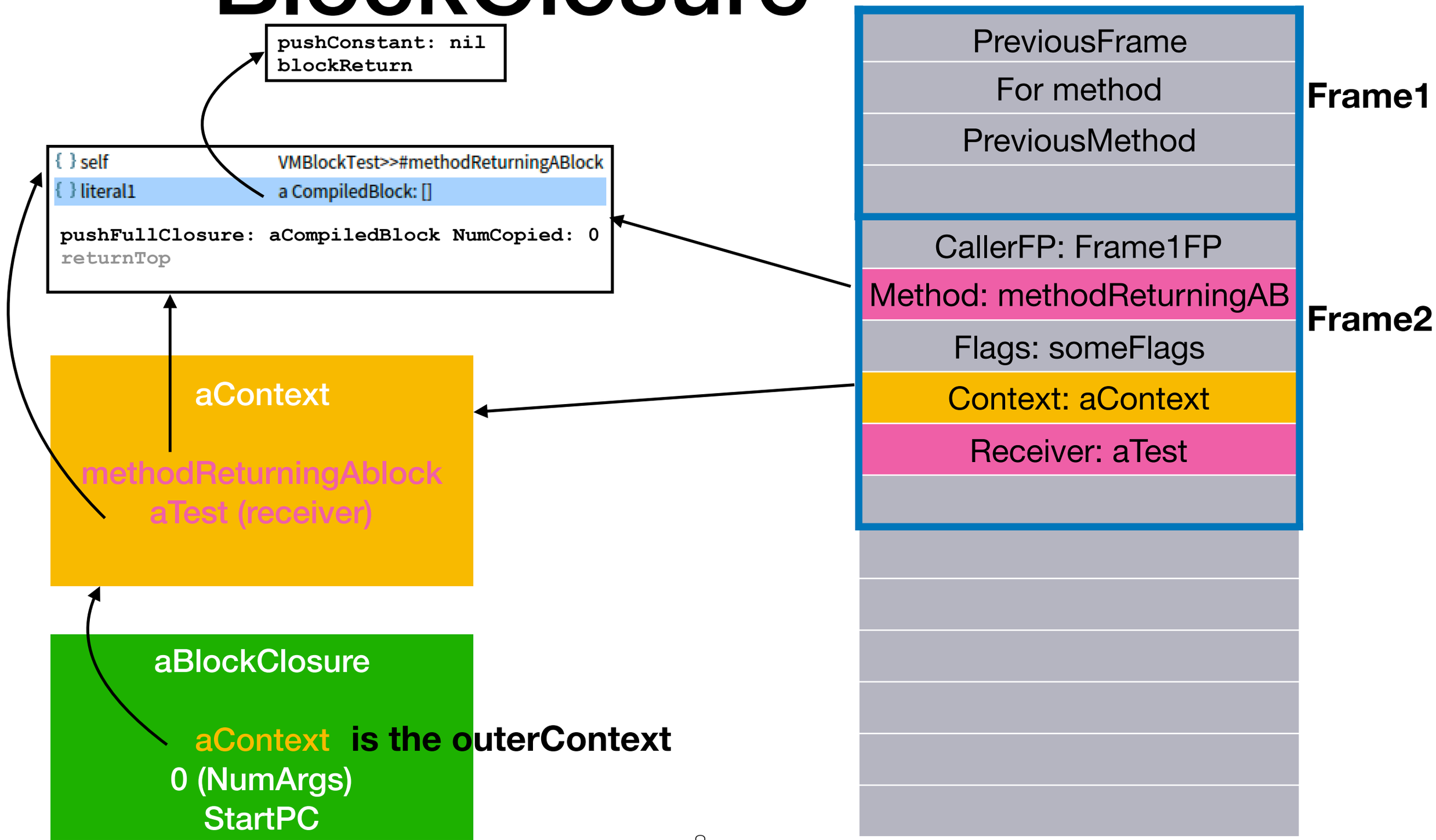


Stack frame initial state

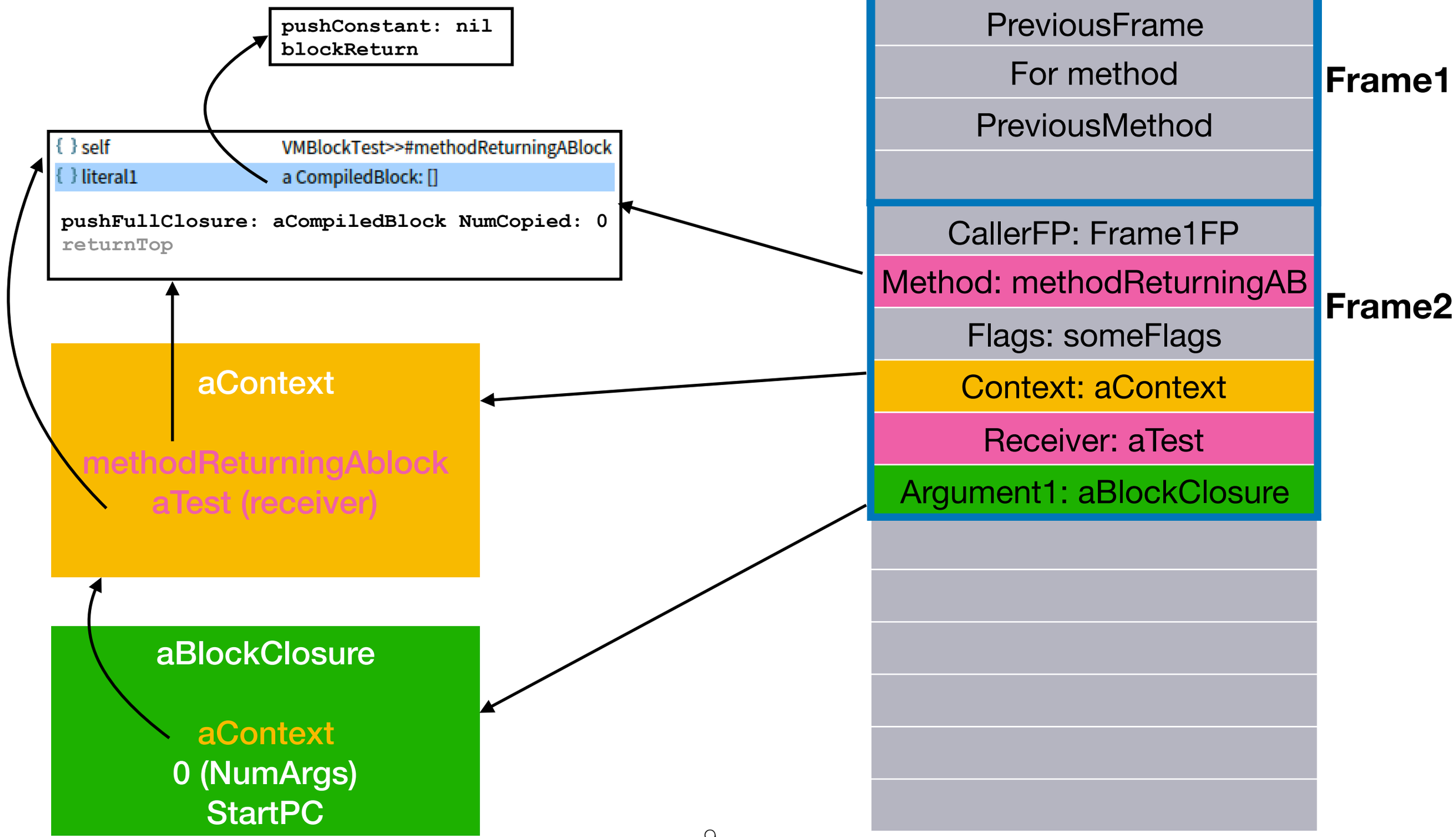
Marry frame


pushConstant: nil
blockReturn
{ } self
VMBlockTest>>#methodReturningABlock
{ } literal1
a CompiledBlock: []
pushFullClosure: aCompiledBlock NumCopied: 0
returnTop
aContext
methodReturningAblock
aTest (receiver)
PreviousFrame
For method
PreviousMethod
Frame1
CallerFP: Frame1FP
Method: methodReturningAB
Flags: someFlags
Context: aContext
Receiver: aTest
Frame2

Create an instance of BlockClosure



Push closure on top



Example with a variable

```
evaluateBlockWritingAndReadingOuterMethodVAriable

  |aVariable|
  ^[aVariable := 42.
  aVariable ] value
```

Method and block share a Variable

=

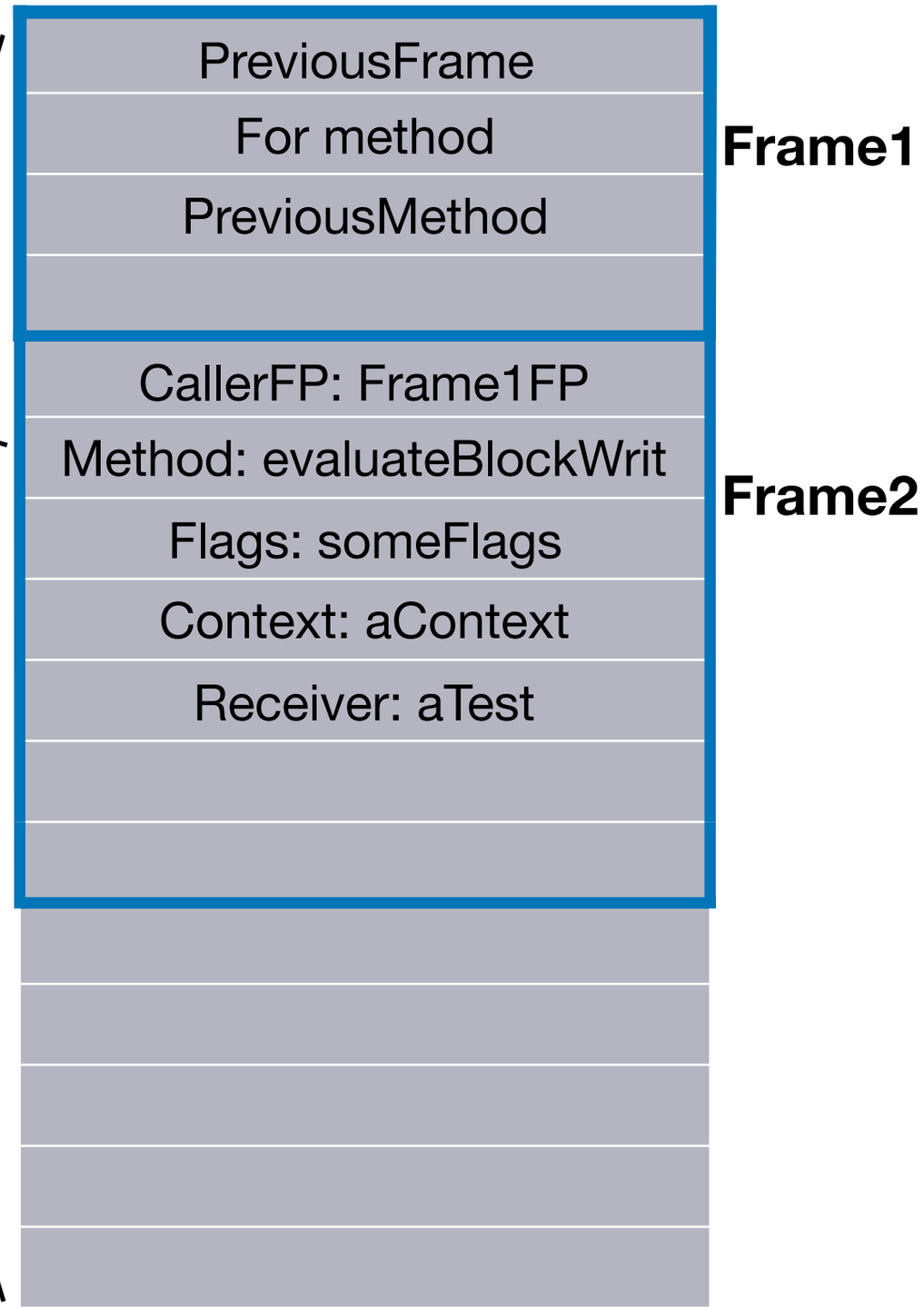
```
VMBlockTest>>evaluateBlockWritingAndReadingO
aCompiledBlock

push: (Array new:1)
popIntoTemp: 0
pushTemp: 0
fullClosure: aCompiledBlock numCopied: 1
send: value
returnTop
```

```
pushConstant: 42
popIntoTemp: 0 inVectorAt: 0
pushTemp: 0 inVectorAt: 0
blockReturn
```

New frame creation

```
pushConstant: 42
popIntoTemp: 0 inVectorAt: 0
pushTemp: 0 inVectorAt: 0
blockReturn
```

```
VMBlockTest>>evaluateBlockWritingAndReadingO
aCompiledBlock

push: (Array new:1)
popIntoTemp: 0
pushTemp: 0
fullClosure: aCompiledBlock numCopied: 1
send: value
returnTop
```

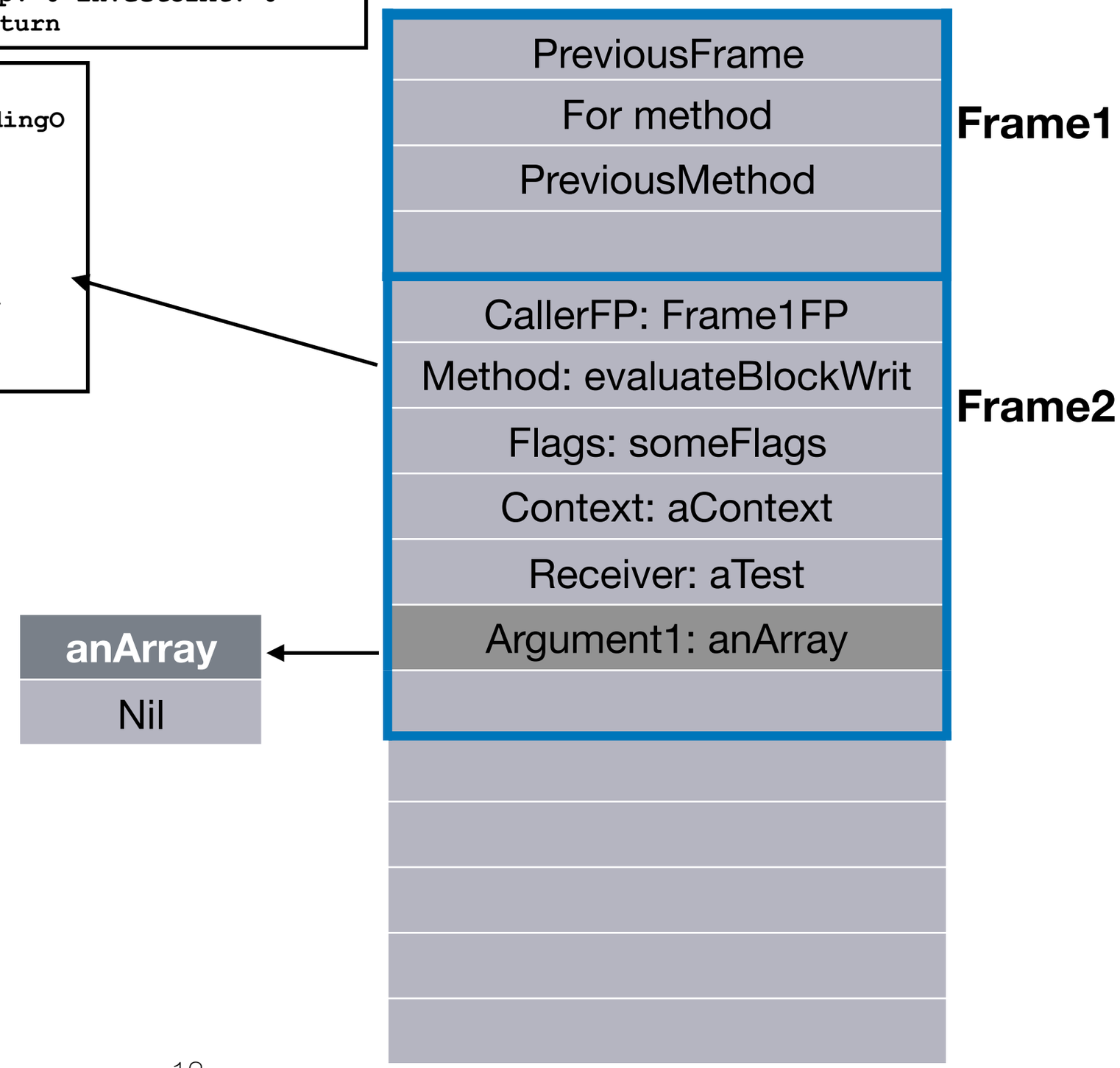


Stack frame initial state

Create then push array

```
pushConstant: 42
popIntoTemp: 0 inVectorAt: 0
pushTemp: 0 inVectorAt: 0
blockReturn
```

```
VMBlockTest>>evaluateBlockWritingAndReadingO
aCompiledBlock

push: (Array new:1)
popIntoTemp: 0
pushTemp: 0
fullClosure: aCompiledBlock numCopied: 1
send: value
returnTop
```



Marry frame

```
pushConstant: 42
popIntoTemp: 0 inVectorAt: 0
pushTemp: 0 inVectorAt: 0
blockReturn
```

```
VMBlockTest>>evaluateBlockWritingAndReadingO
aCompiledBlock

push: (Array new:1)
popIntoTemp: 0
pushTemp: 0
fullClosure: aCompiledBlock numCopied: 1
send: value
returnTop
```

aContext

evaluateBlockWriting
aTest (receiver)

anArray
Nil

PreviousFrame
For method
PreviousMethod

Frame1

CallerFP: Frame1FP
Method: evaluateBlockWrit
Flags: someFlags
Context: aContext
Receiver: aTest
Argument1: anArray

Frame2

Create blockClosure instance

```
pushConstant: 42
popIntoTemp: 0 inVectorAt: 0
pushTemp: 0 inVectorAt: 0
blockReturn
```

```
VMBlockTest>>evaluateBlockWritingAndReadingO
aCompiledBlock

push: (Array new:1)
popIntoTemp: 0
pushTemp: 0
fullClosure: aCompiledBlock numCopied: 1
send: value
returnTop
```

aContext

evaluateBlockWriting
aTest (receiver)

aBlockClosure

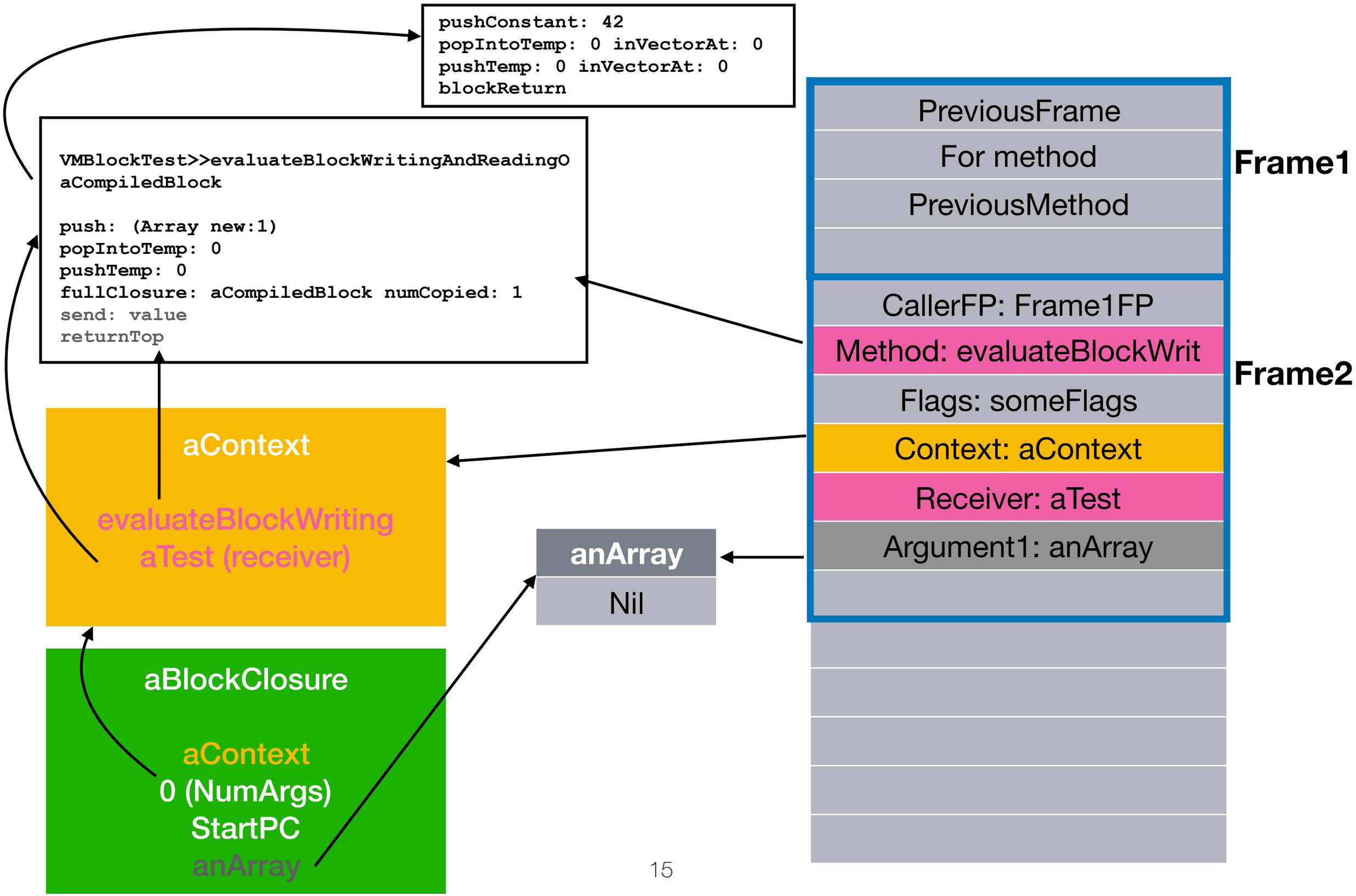
aContext
0 (NumArgs)
StartPC

anArray

Nil

PreviousFrame	
For method	
PreviousMethod	
	Frame1
CallerFP: Frame1FP	
Method: evaluateBlockWrit	
Flags: someFlags	**Frame2**
Context: aContext	
Receiver: aTest	
Argument1: anArray	

BlockClosure keeps a reference to the array



Push closure on top

```
pushConstant: 42
popIntoTemp: 0 inVectorAt: 0
pushTemp: 0 inVectorAt: 0
blockReturn
```

```
VMBlockTest>>evaluateBlockWritingAndReadingO
aCompiledBlock

push: (Array new:1)
popIntoTemp: 0
pushTemp: 0
fullClosure: aCompiledBlock numCopied: 1
send: value
returnTop
```

aContext

evaluateBlockWriting
aTest (receiver)

anArray

Nil

aBlockClosure

aContext
0 (NumArgs)
StartPC
anArray

Frame1
PreviousFrame
For method
PreviousMethod

Frame2
CallerFP: Frame1FP
Method: evaluateBlockWrit
Flags: someFlags
Context: aContext
Receiver: aTest
Argument1: anArray
Argument2: aBlockClosure

Plan

- BlockClosure creation
- The value message
- Return and non local return
- Unwind protect

Value message

```
evaluateBlockWritingAndReadingOuterMethodVAriable

  |aVariable|
  ^[aVariable := 42.
  aVariable ] value
```

=

```
VMBlockTest>>evaluateBlockWritingAndReadingO
aCompiledBlock

push: (Array new:1)
popIntoTemp: 0
pushTemp: 0
fullClosure: aCompiledBlock numCopied: 1
send: value
returnTop
```

```
pushConstant: 42
popIntoTemp: 0 inVectorAt: 0
pushTemp: 0 inVectorAt: 0
blockReturn
```

Not a normal send

Optimization for BlockClosures.

```
If receiver is block closure
then directly do value
else do a normal message send
(look up selector, new frame, etc …)
```

We were here

```
pushConstant: 42
popIntoTemp: 0 inVectorAt: 0
pushTemp: 0 inVectorAt: 0
blockReturn
```

```
VMBlockTest>>evaluateBlockWritingAndReadingO
aCompiledBlock

push: (Array new:1)
popIntoTemp: 0
pushTemp: 0
fullClosure: aCompiledBlock numCopied: 1
send: value
returnTop
```

aContext

evaluateBlockWriting
aTest (receiver)

anArray

Nil

aBlockClosure

aContext
0 (NumArgs)
StartPC
anArray

Frame1
PreviousFrame
For method
PreviousMethod

Frame2
CallerFP: Frame1FP
Method: evaluateBlockWrit
Flags: someFlags
Context: aContext
Receiver: aTest
Argument1: anArray
Argument2: aBlockClosure

Frame creation

```
pushConstant: 42
popIntoTemp: 0 inVectorAt: 0
pushTemp: 0 inVectorAt: 0
blockReturn
```

```
VMBlockTest>>evaluateBlockWritingAndReadingO
aCompiledBlock

push: (Array new:1)
popIntoTemp: 0
pushTemp: 0
fullClosure: aCompiledBlock numCopied: 1
send: value
returnTop
```

aContext

evaluateBlockWriting
aTest (receiver)

anArray

Nil

aBlockClosure

aContext
0 (NumArgs)
StartPC
anArray

PreviousFrame
For method
PreviousMethod

Frame1

CallerFP: Frame1FP
Method: evaluateBlockWrit
Flags: someFlags
Context: aContext
Receiver: aTest
Argument1: anArray
Argument2: aBlockClosure

Frame2

CallerFP: Frame2FP
Method: evaluateBlockWrit
Flags: isBlock := True
Context: nil
Receiver: aTest
Argument1: anArray

Block Frame

Continue execution

```
pushConstant: 42
popIntoTemp: 0 inVectorAt: 0
pushTemp: 0 inVectorAt: 0
blockReturn
```

```
VMBlockTest>>evaluateBlockWritingAndReadingO
aCompiledBlock

push: (Array new:1)
popIntoTemp: 0
pushTemp: 0
fullClosure: aCompiledBlock numCopied: 1
send: value
returnTop
```

aContext

evaluateBlockWriting
aTest (receiver)

aBlockClosure

aContext
0 (NumArgs)
StartPC
anArray

anArray
42

PreviousFrame
For method
PreviousMethod

Frame1

CallerFP: Frame1FP
Method: evaluateBlockWrit
Flags: someFlags
Context: aContext
Receiver: aTest
Argument1: anArray
Argument2: aBlockClosure

Frame2

CallerFP: Frame2FP
Method: evaluateBlockWrit
Flags: isBlock := True
Context: nil
Receiver: aTest
Argument1: anArray

Block Frame

Plan

- BlockClosure creation
- The value message
- Return and non local return
- Unwind protect

2 return operations

```
evaluateBlock

  ^[42] value
```

Normal return

=

A block with no explicit return statement returns the value of its last expression to its caller.

```
evaluateBlockWithNonLocalReturn

  [^42] value
```

Non local return

=

A block with an explicit return statement returns its home method. It returns the value in the return statement.

Block local return

```
evaluateBlock

  ^[42] value
```

Normal return

=

A block with no explicit return statement returns the value of its last expression to its caller.

```
pushContant: 42
blockReturn
```

Special bytecode

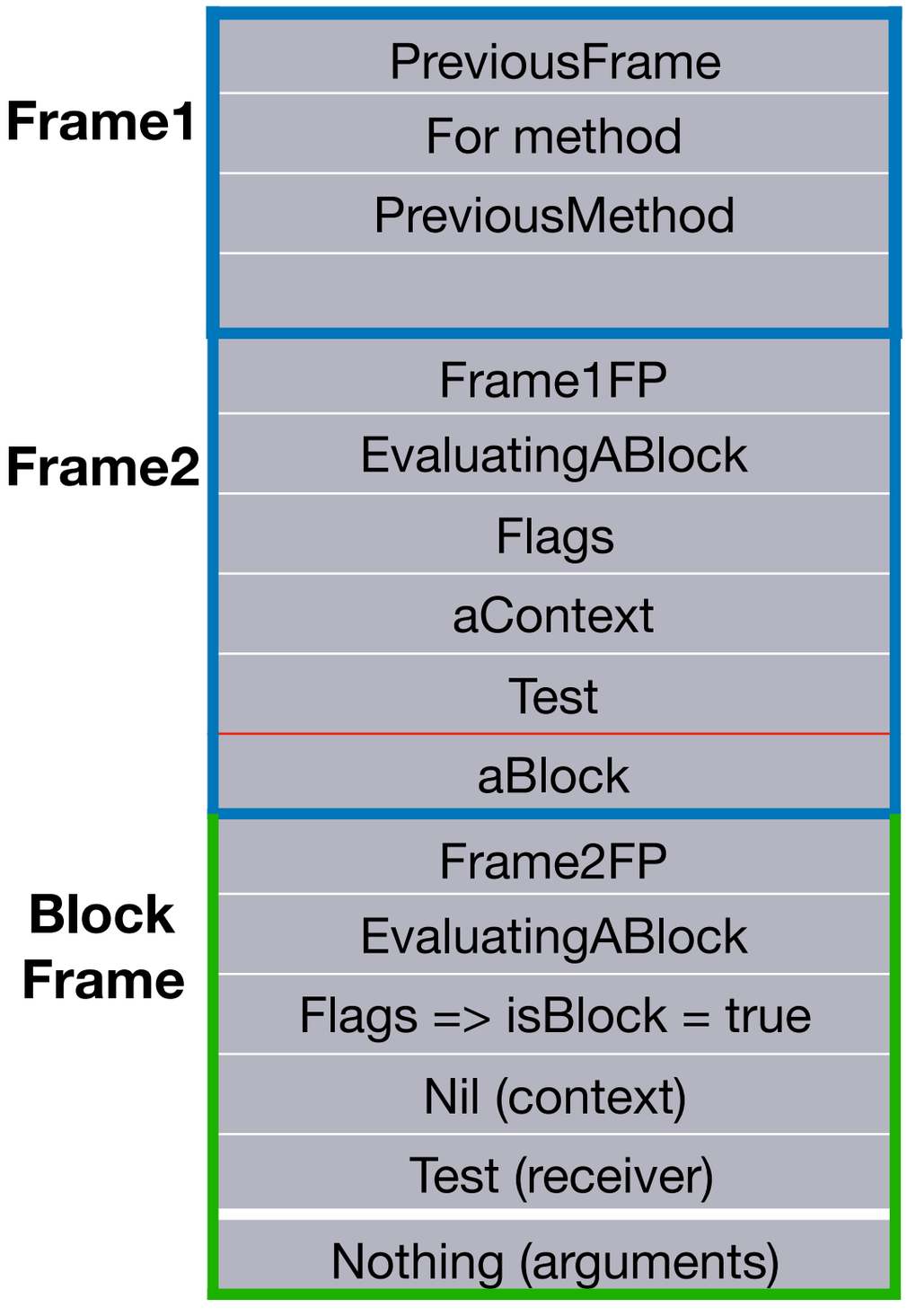
```
evaluateBlockWithNonLocalReturn

  [^42] value
```

Non local return

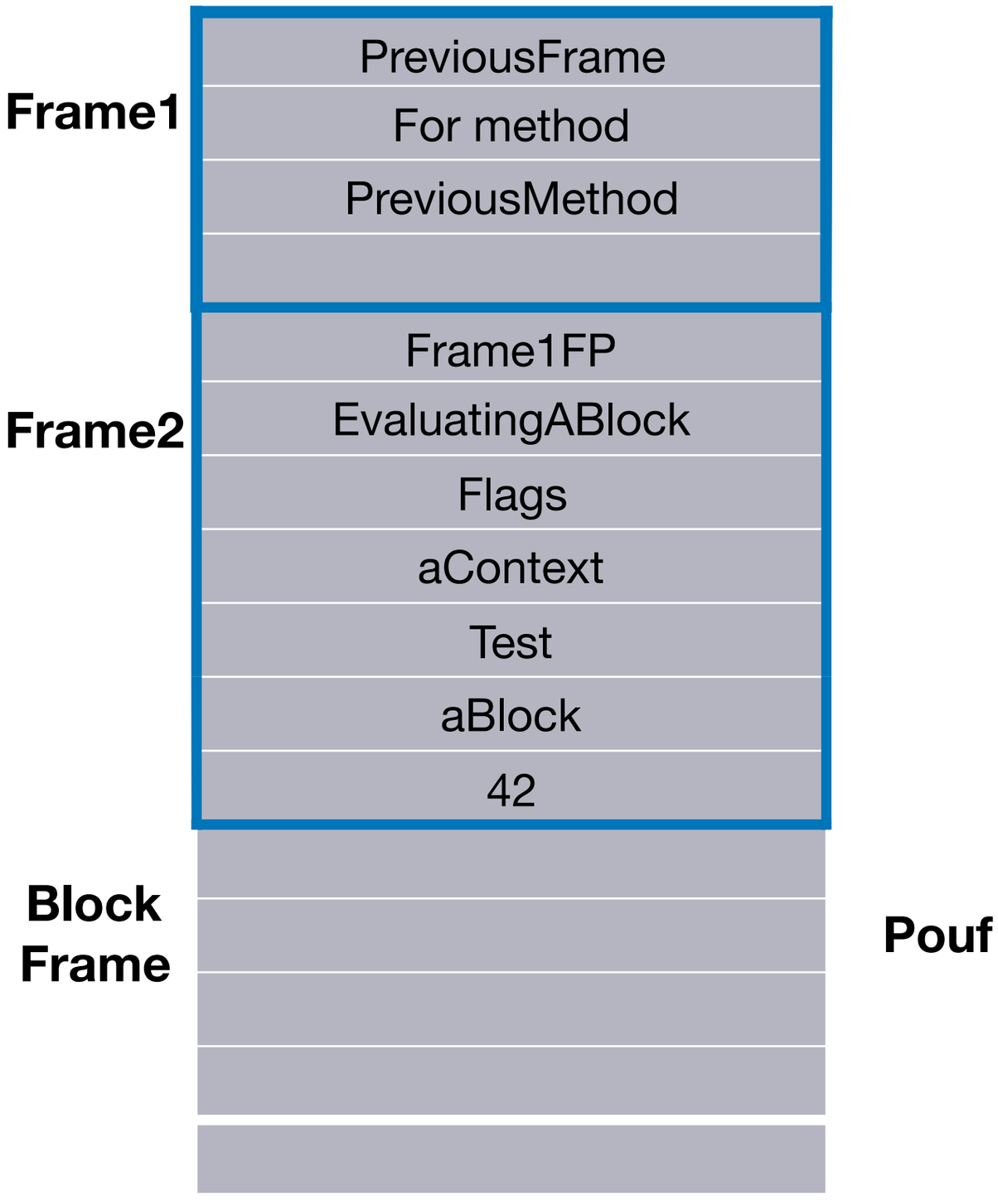
=

A block with an explicit return statement returns its home method. It returns the value in the return statement.

Before return



After normal return



Block non local return

```
evaluateBlock

  ^[42] value
```

Normal return

=

A block with no explicit return statement returns the value of its last expression to its caller.

```
pushContant: 42
blockReturn
```

Special bytecode

```
evaluateBlockWithNonLocalReturn

  [^42] value
```

Non local return

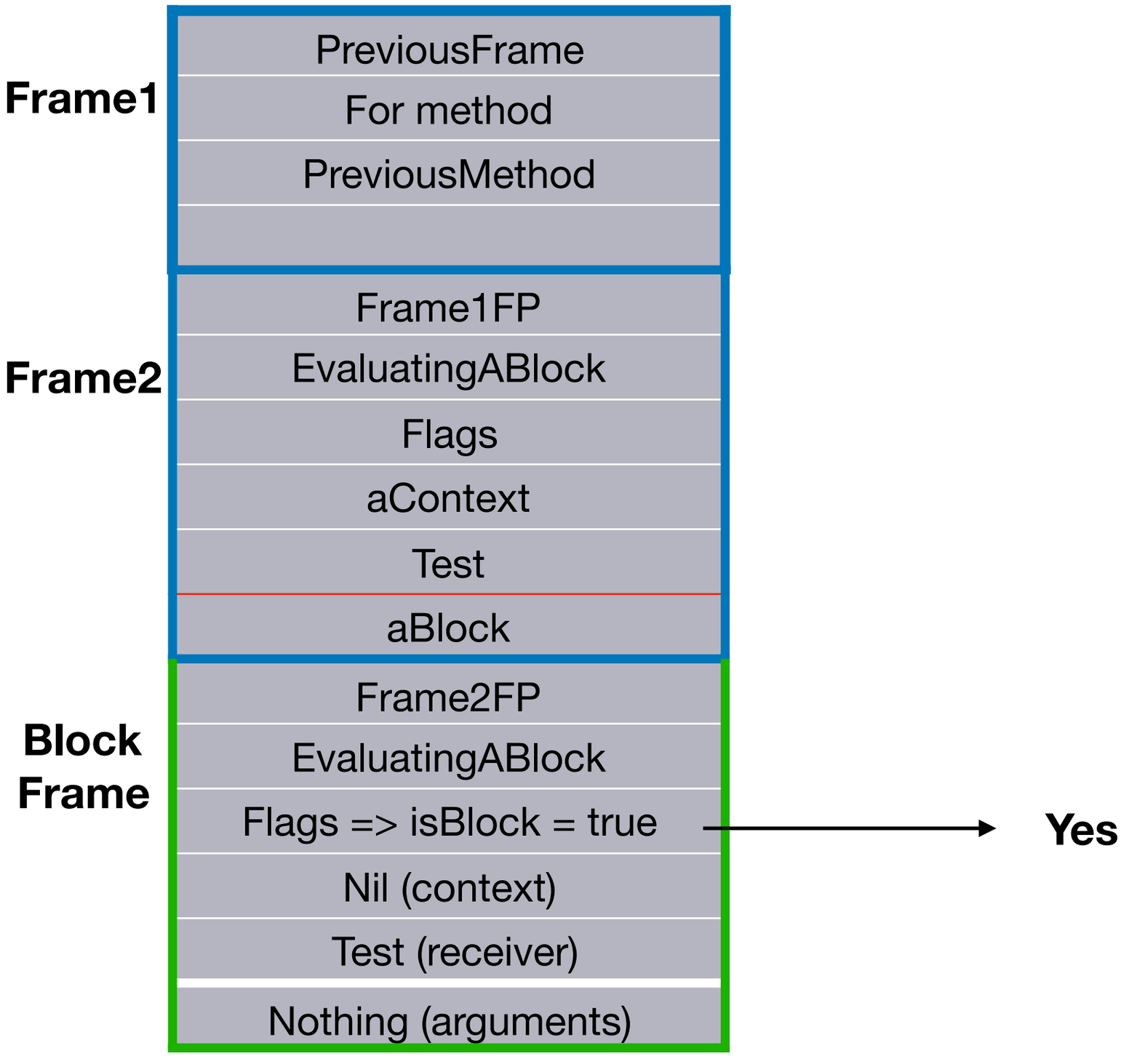
=

A block with an explicit return statement returns its home method. It returns the value in the return statement.

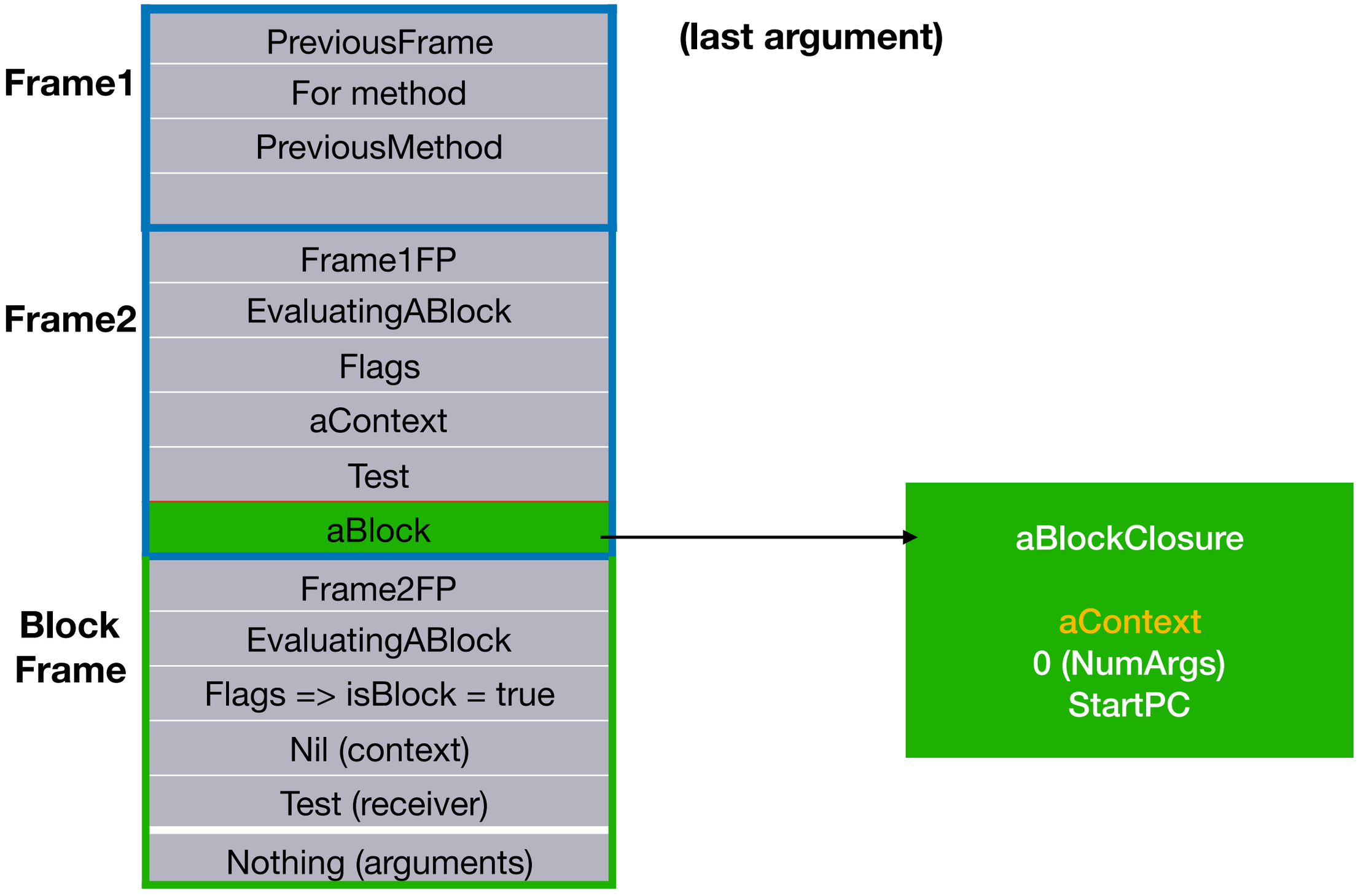
```
pushContant: 42
returnTop
```

Same bytecode as method

Block activation ?



Retrieve block



Retrieve outer context

Frame1

PreviousFrame
For method
PreviousMethod

Frame2

Frame1FP
EvaluatingABlock
Flags
aContext
Test
aBlock

Block Frame

Frame2FP
EvaluatingABlock
Flags => isBlock = true
Nil (context)
Test (receiver)
Nothing (arguments)

Take outer context.

aContext

MethodReturningAblock
Test (receiver)

aBlockClosure

aContext
0 (NumArgs)
StartPC

Home context ?

Frame1

PreviousFrame
For method
PreviousMethod

Frame2

Frame1FP
EvaluatingABlock
Flags
aContext
Test
aBlock

Block Frame

Frame2FP
EvaluatingABlock
Flags => isBlock = true
Nil (context)
Test (receiver)
Nothing (arguments)

Home context

=

context of the method creating the block

aContext

MethodReturningAblock
Test (receiver)

aBlockClosure

aContext
0 (NumArgs)
StartPC

Here outer

Look for home context

Frame1

PreviousFrame
For method
PreviousMethod

Frame2

Frame1FP
EvaluatingABlock
Flags
aContext
Test
aBlock

Block Frame

Frame2FP
EvaluatingABlock
Flags => isBlock = true
Nil (context)
Test (receiver)
Nothing (arguments)

Is home context? **Yes,**
doesn't contain another Block
=
context of the method

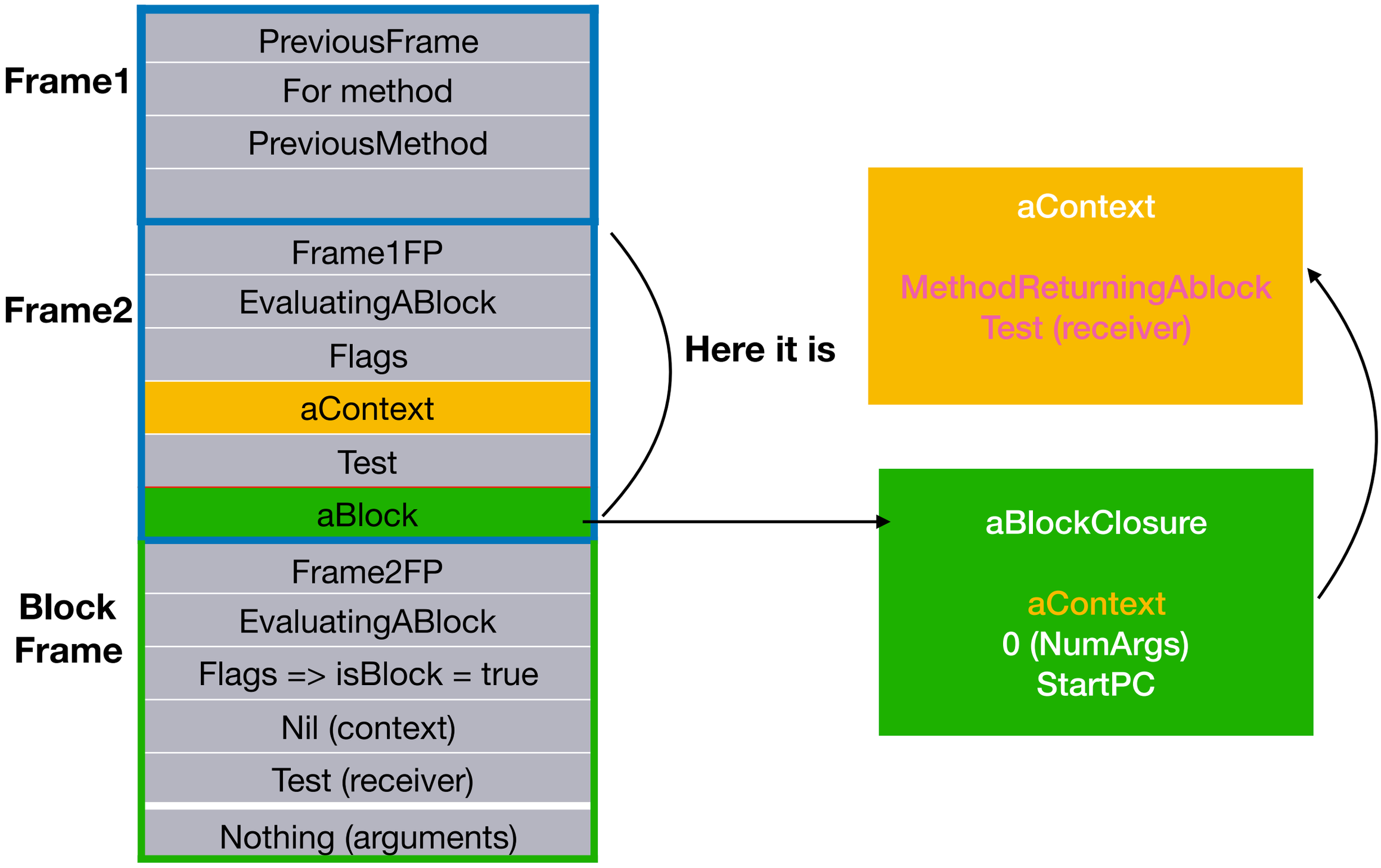
aContext

MethodReturningAblock
Test (receiver)

aBlockClosure

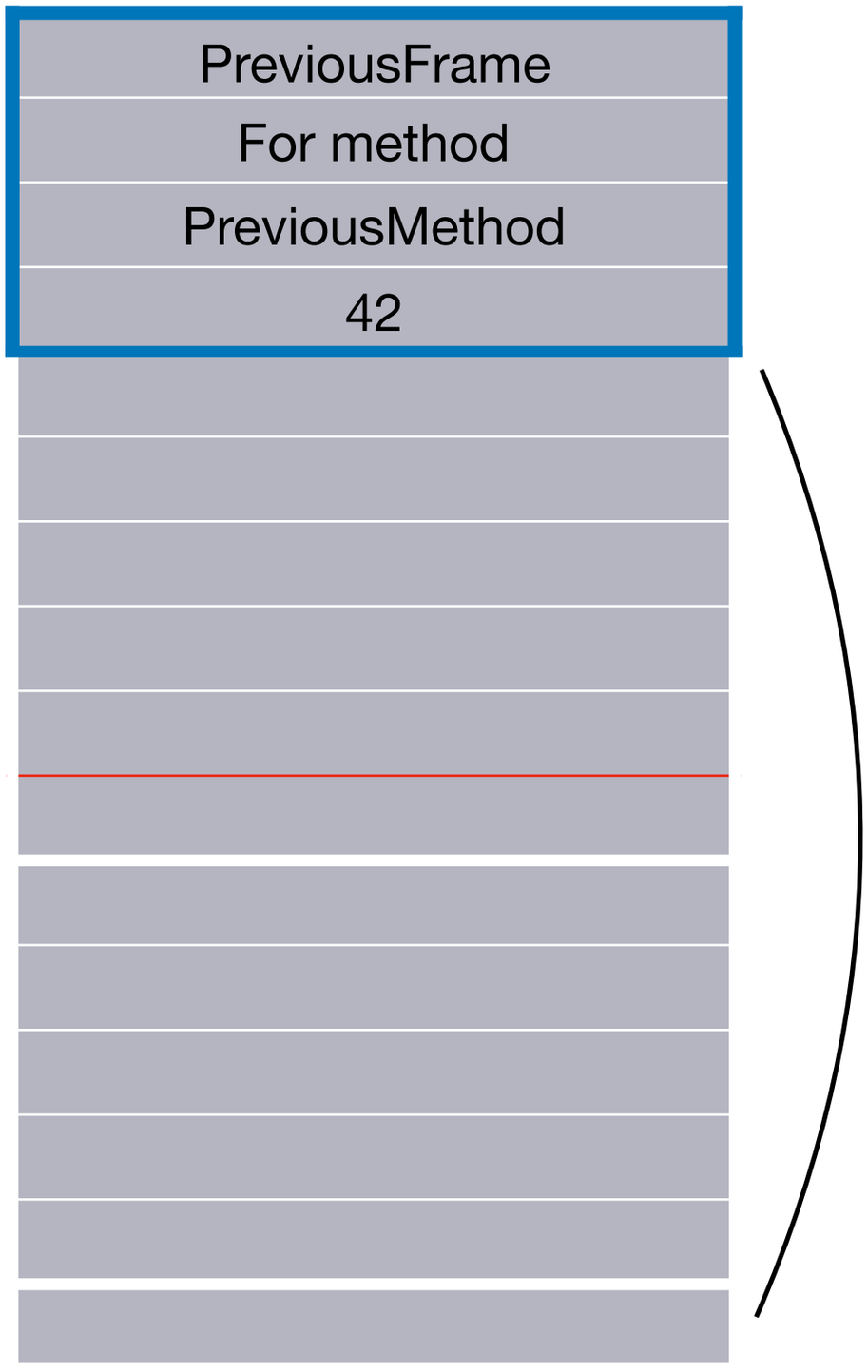
aContext
0 (NumArgs)
StartPC

Where is frame ?



Non local return

Frame1



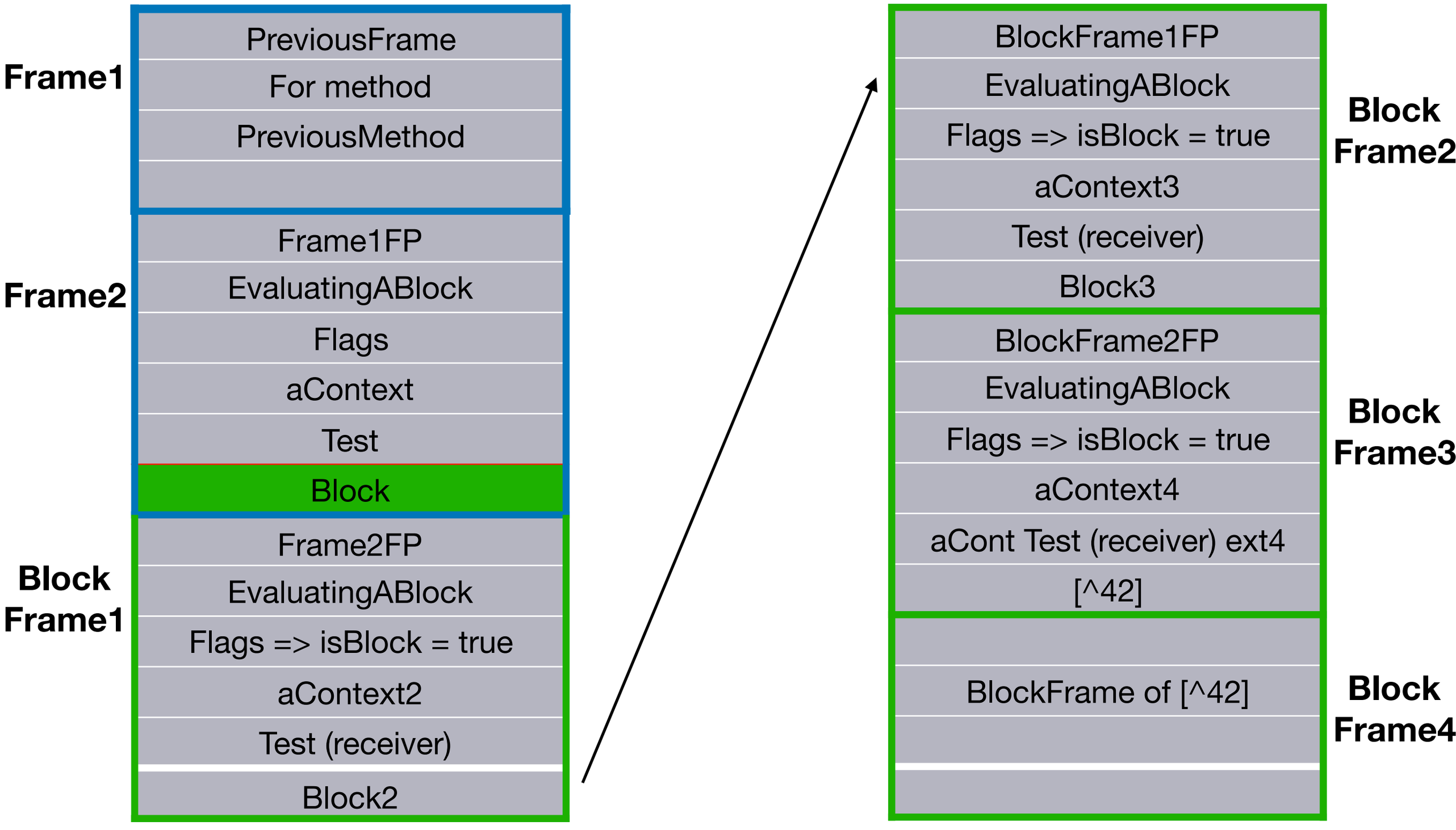
Pouf

Little more complex example

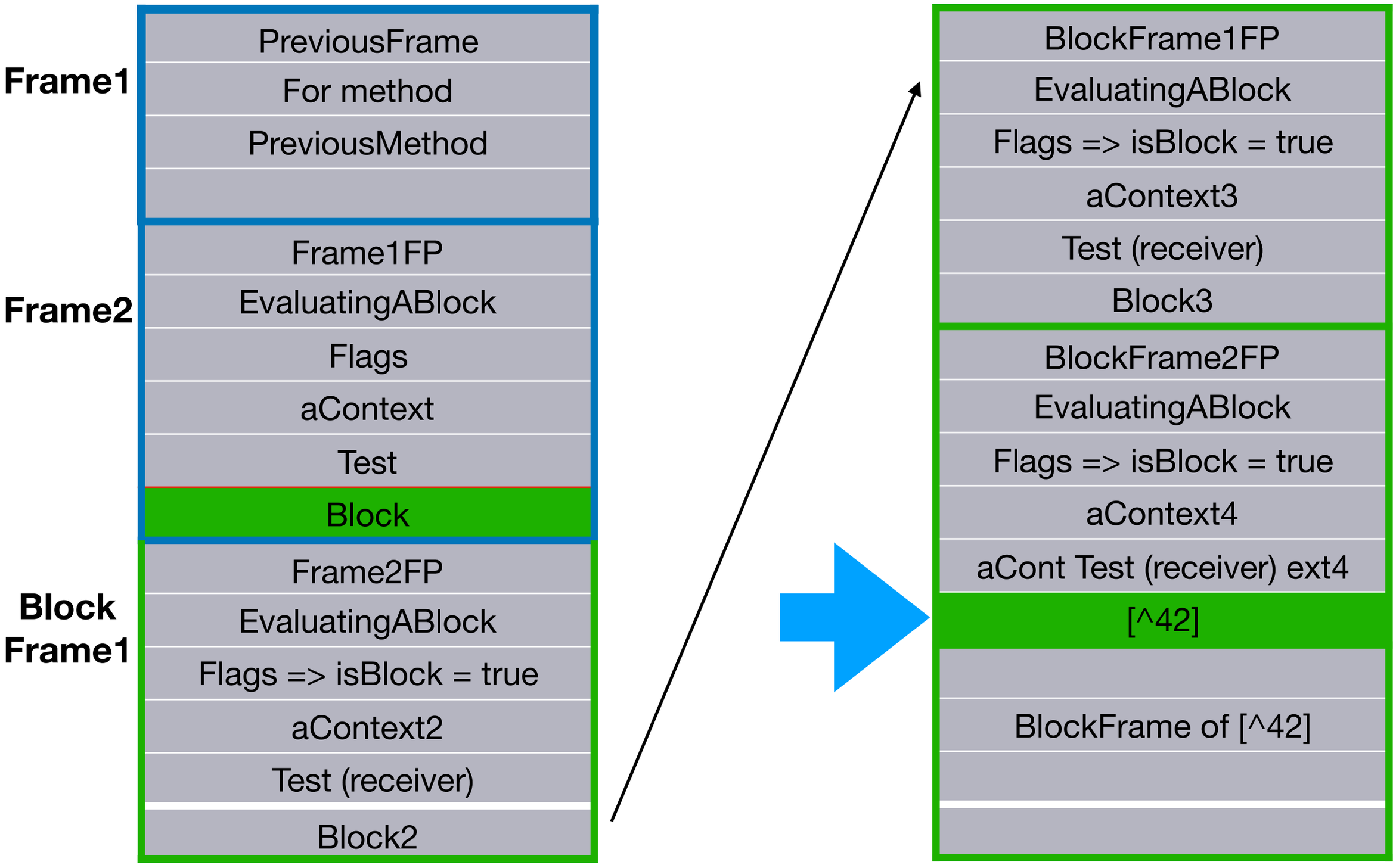
```
longNonLocalReturn

  [[[[^42] value ] value ] value ] value ]
```

Non local return inside 4 blocks

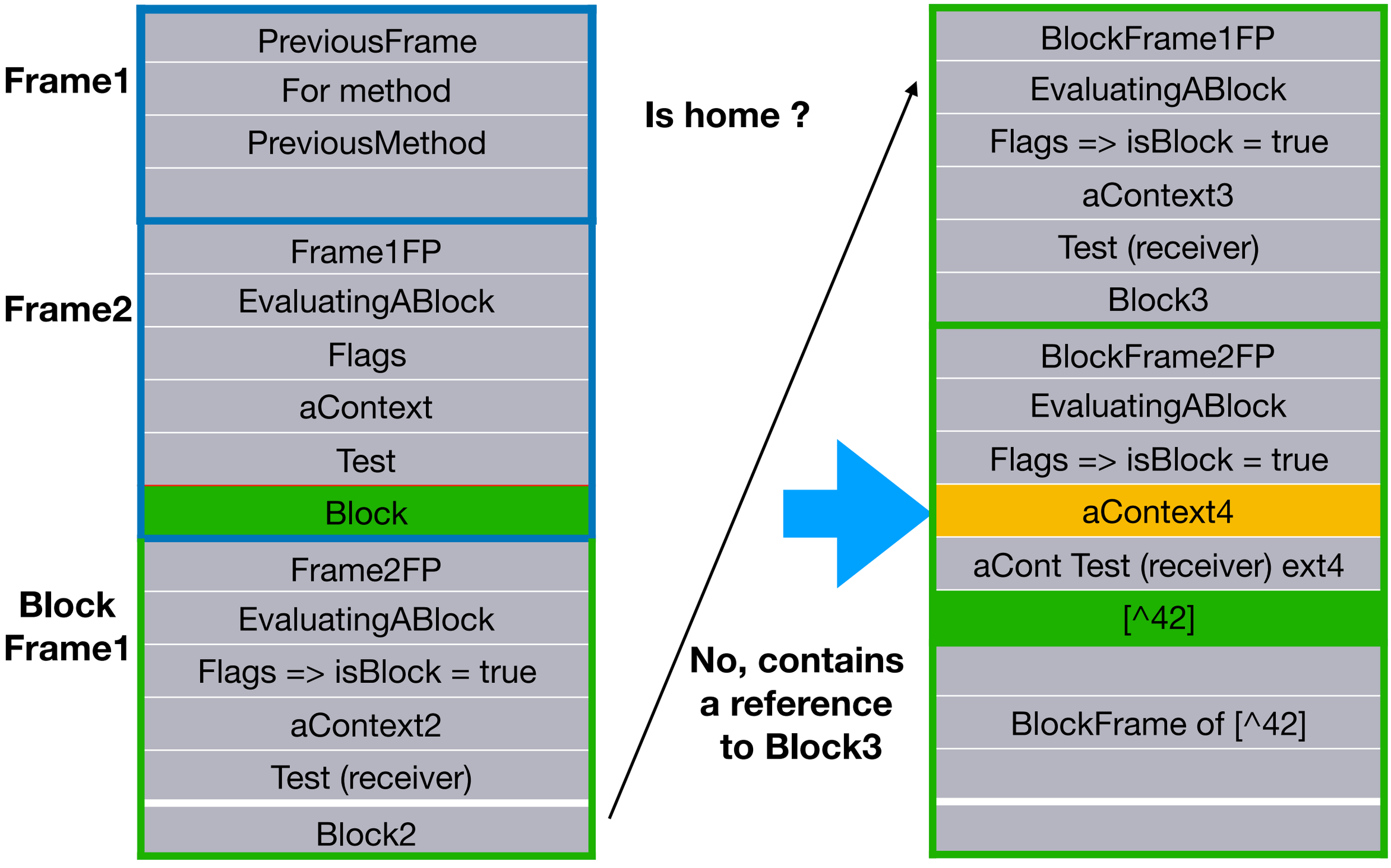
Look for home context



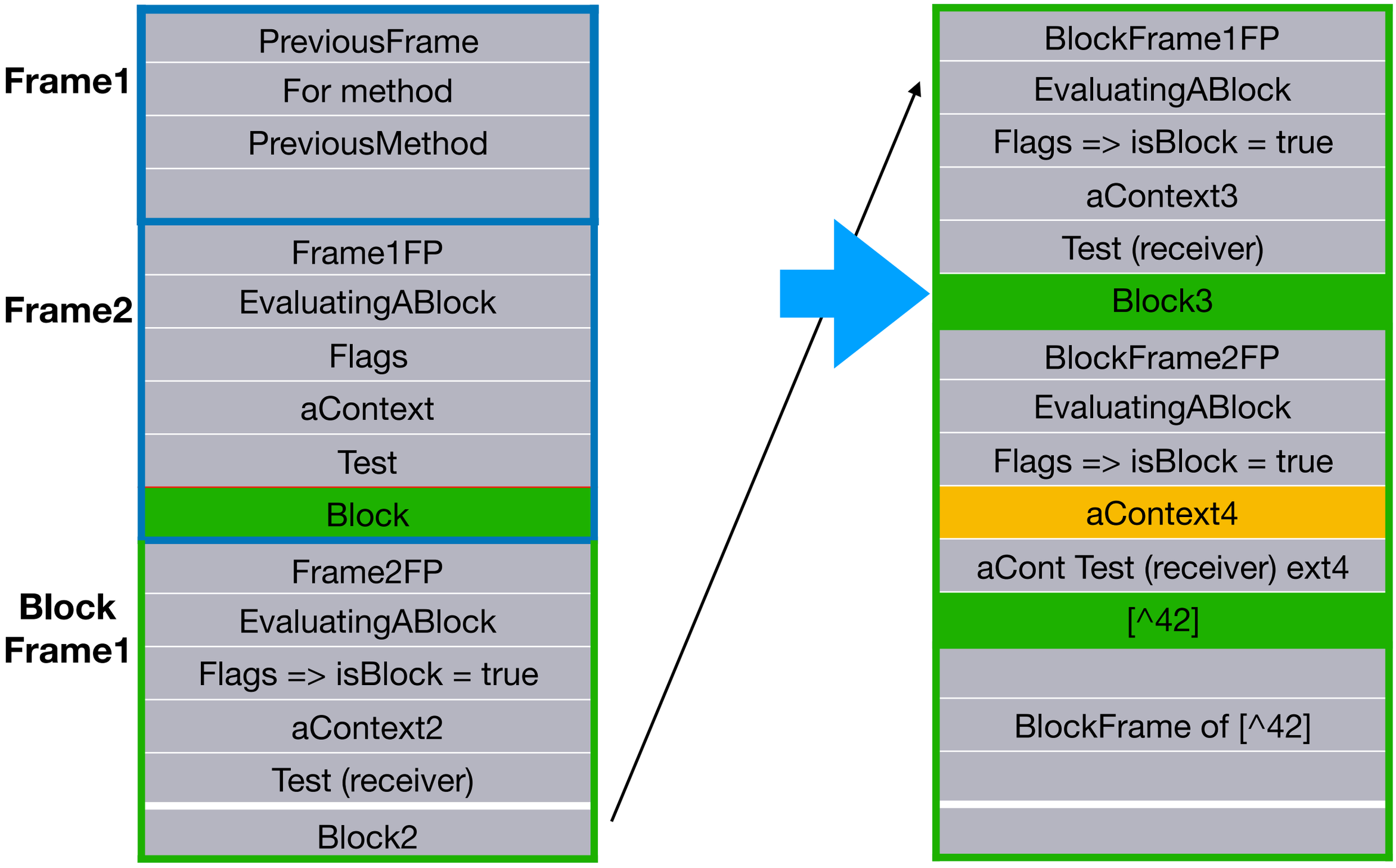
Take Block


Frame1
PreviousFrame
For method
PreviousMethod
Frame2
Frame1FP
EvaluatingABlock
Flags
aContext
Test
Block
Block Frame1
Frame2FP
EvaluatingABlock
Flags => isBlock = true
aContext2
Test (receiver)
Block2
BlockFrame1FP
EvaluatingABlock
Flags => isBlock = true
aContext3
Test (receiver)
Block3
BlockFrame2FP
EvaluatingABlock
Flags => isBlock = true
aContext4
aCont Test (receiver) ext4
[^42]
BlockFrame of [^42]

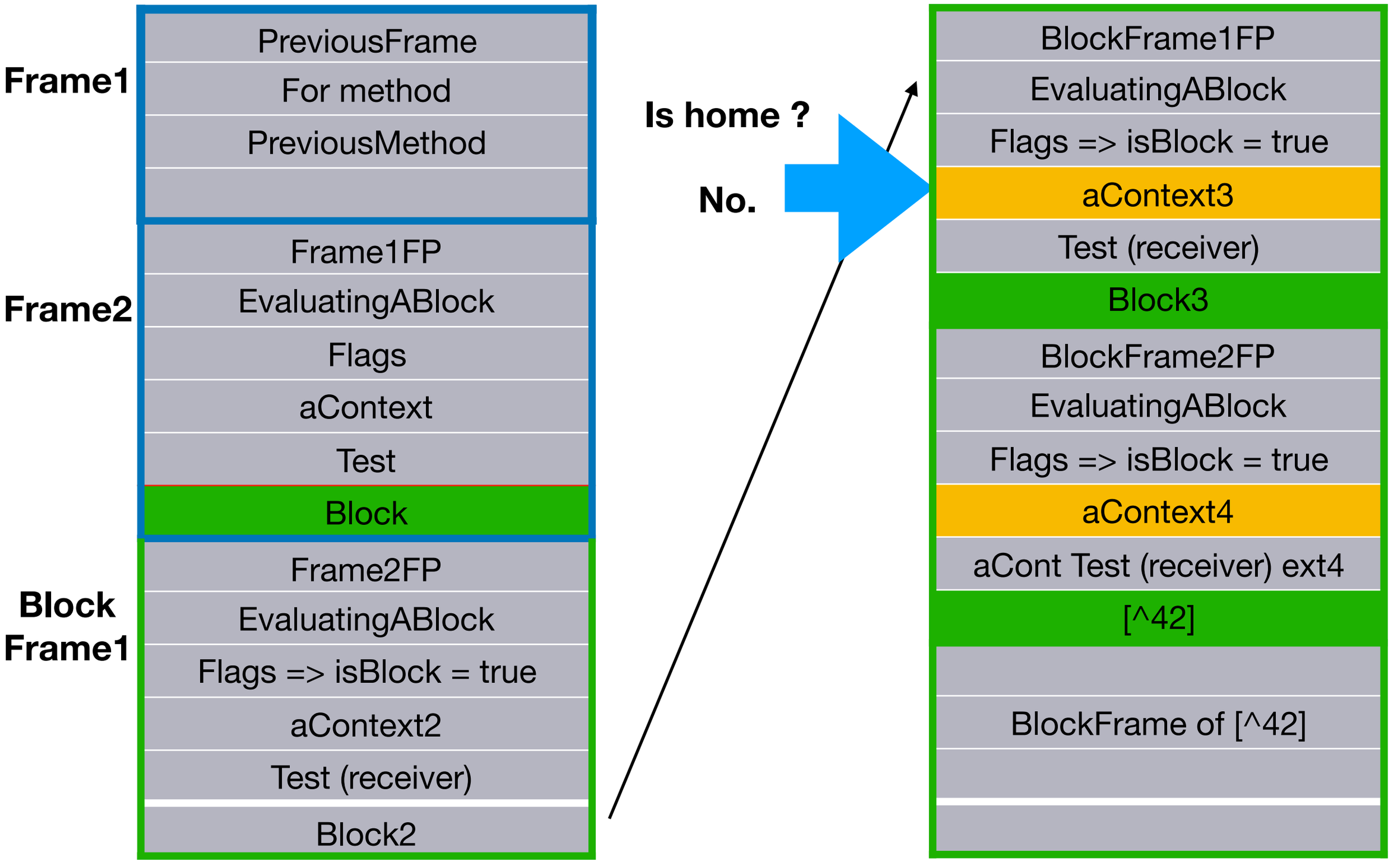
Take outer context



Rince and repeat



Rince and repeat



Rince and repeat

Frame1

PreviousFrame
For method
PreviousMethod

Frame2

Frame1FP
EvaluatingABlock
Flags
aContext
Test
Block

Block Frame1

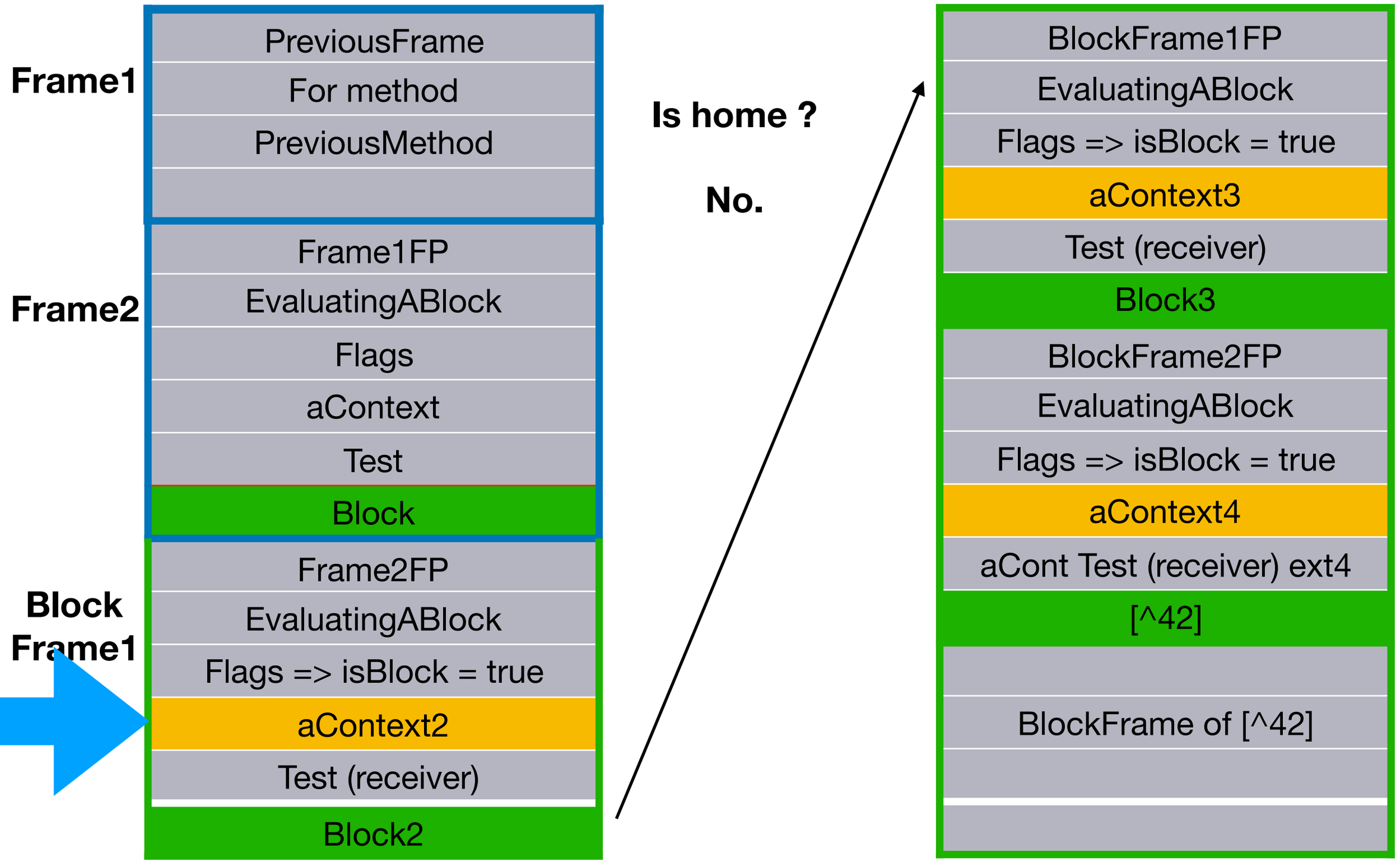
Frame2FP
EvaluatingABlock
Flags => isBlock = true
aContext2
Test (receiver)
Block2

BlockFrame1FP
EvaluatingABlock
Flags => isBlock = true
aContext3
Test (receiver)
Block3

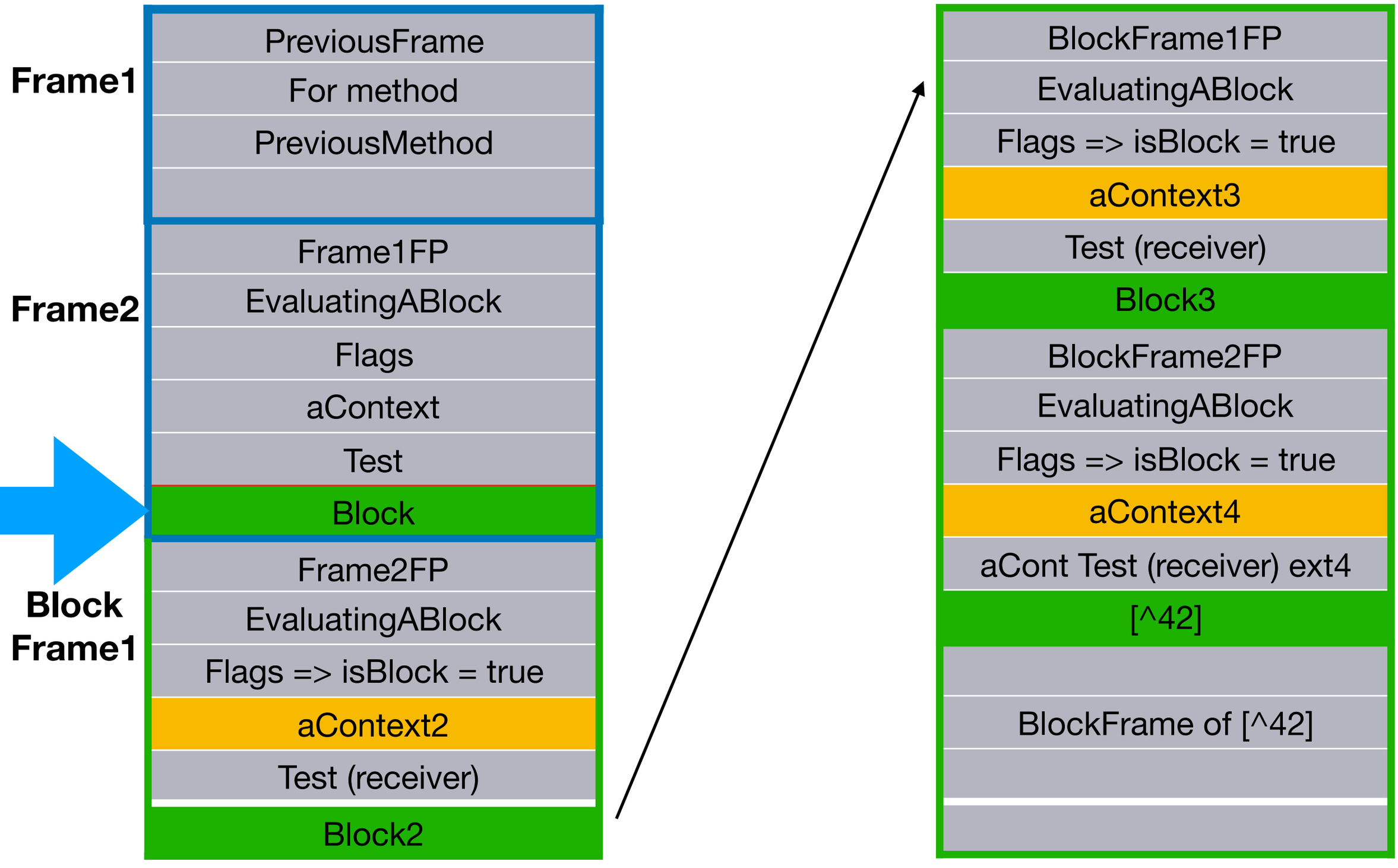
BlockFrame2FP
EvaluatingABlock
Flags => isBlock = true
aContext4
aCont Test (receiver) ext4
[^42]

BlockFrame of [^42]

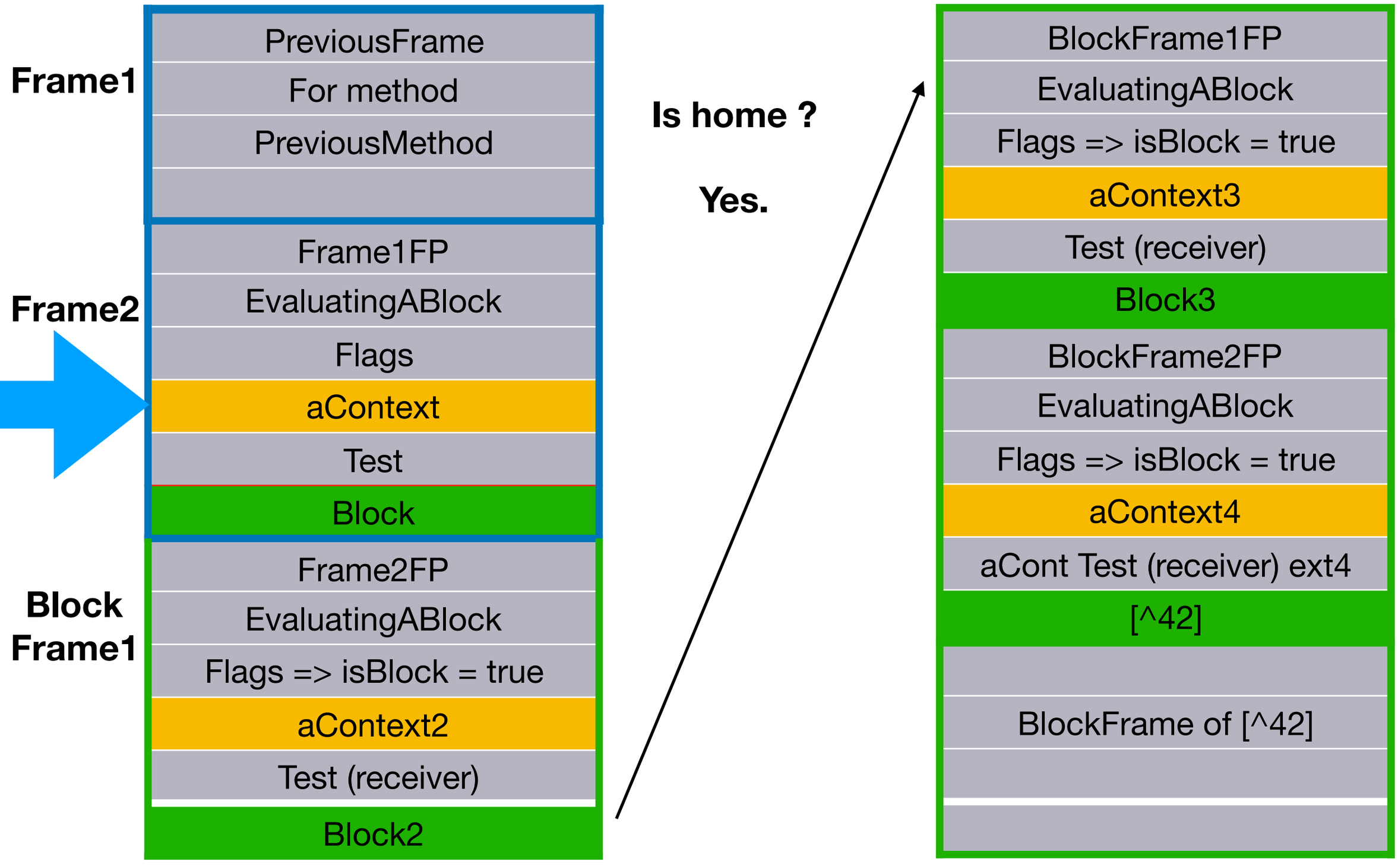
Rince and repeat



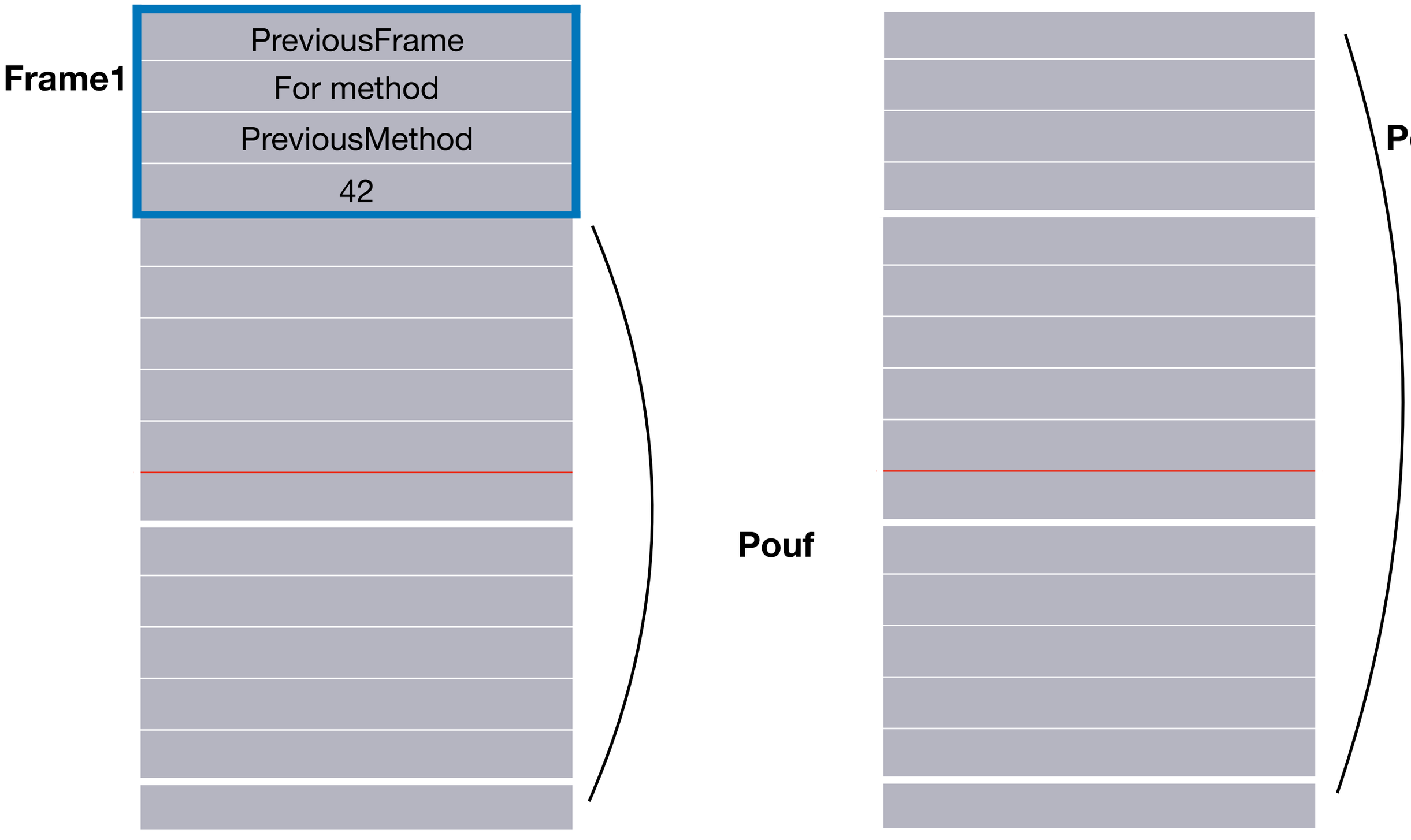
Rince and repeat



Rince and repeat



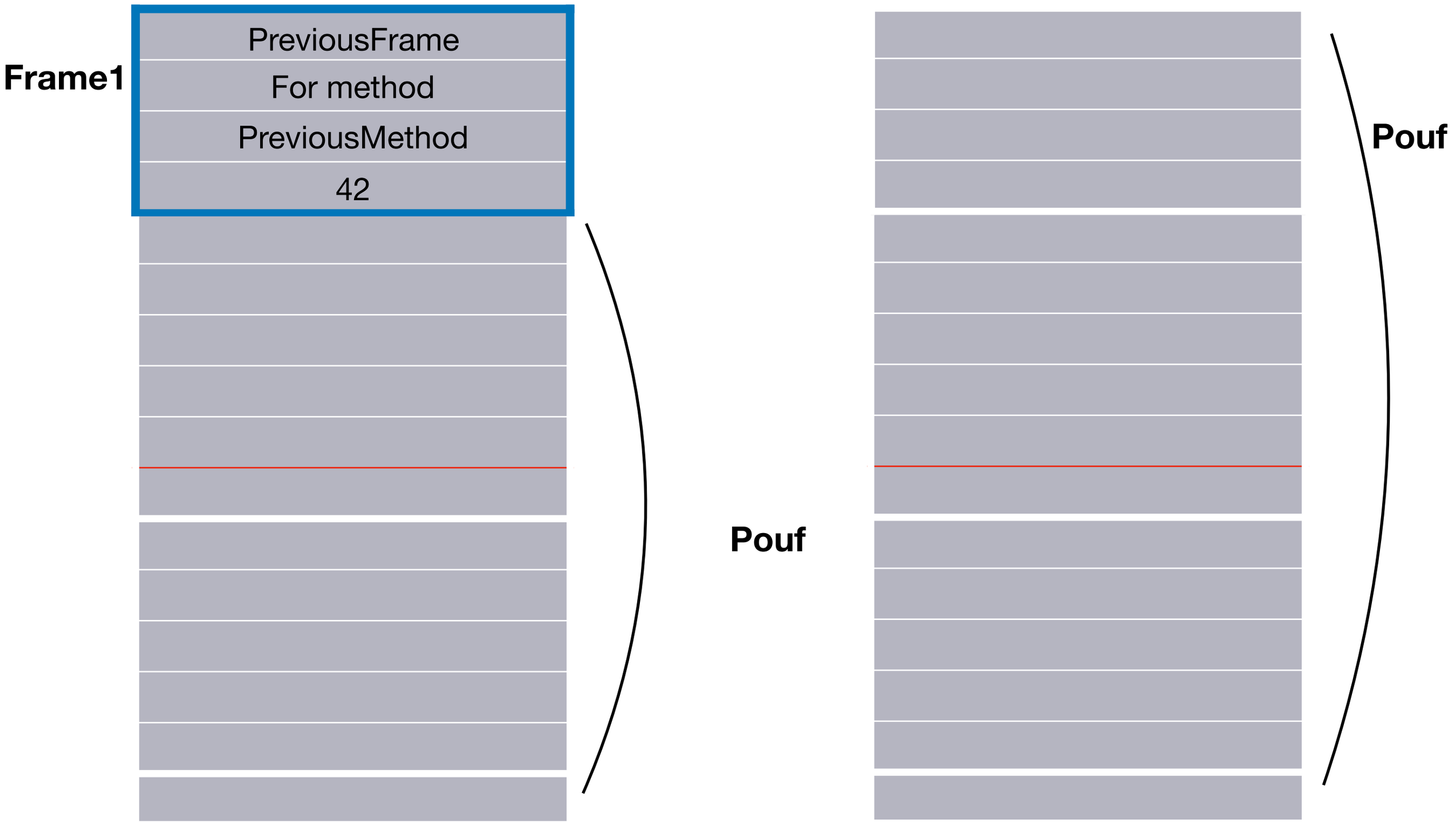
Non local return



Plan

- BlockClosure creation
- The value message
- Return and non local return
- Unwind protect

What happens if one of the frames has an ensure: ?



What is ensure ?

runCase

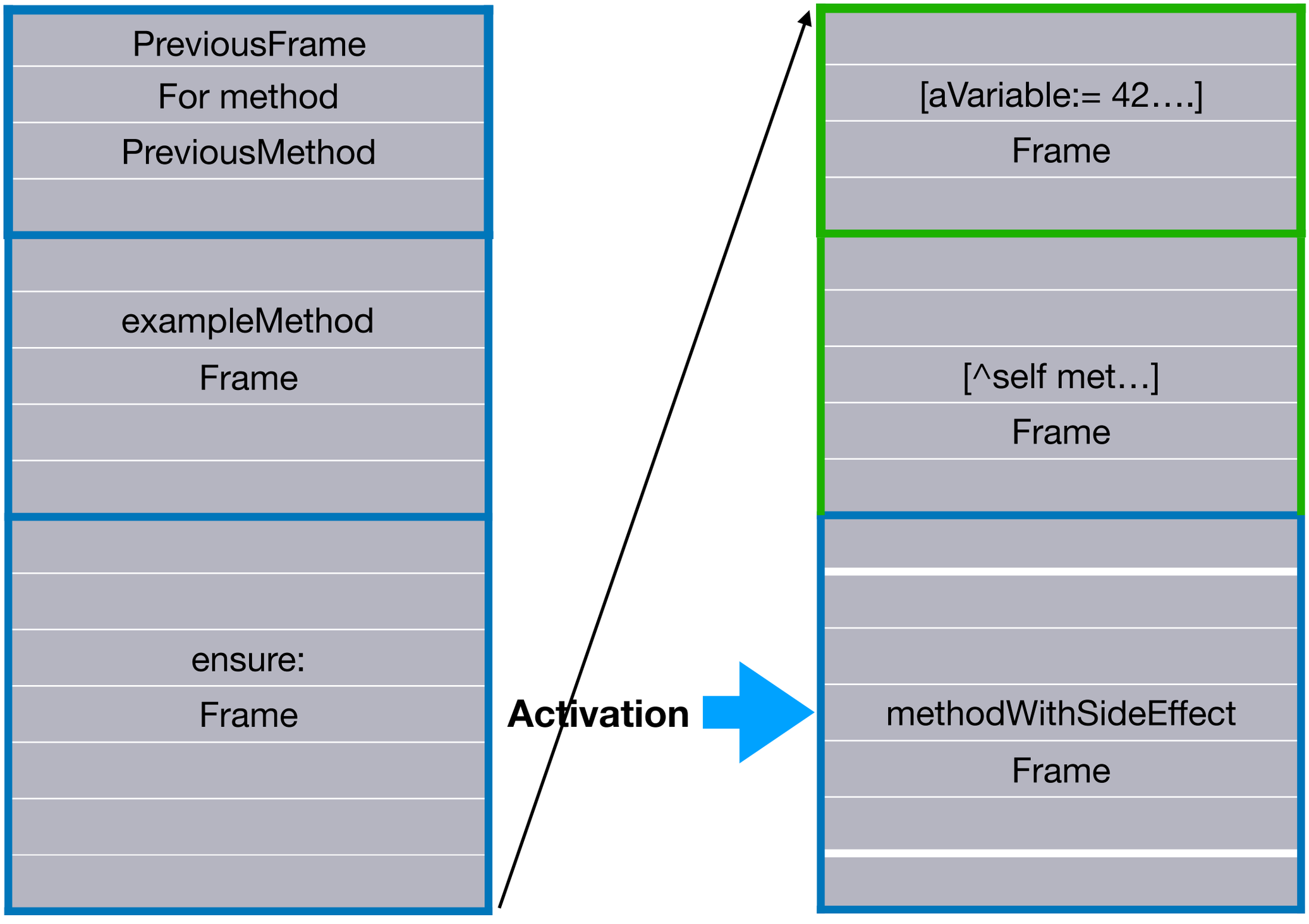
```
runCase
    self resources do: [:each | each availableFor: self].
    [self setUp.
    self performTest] ensure: [
        self tearDown.
        self cleanUpInstanceVariables]
```

You want to « ensure » something is executed whenever a block returns.

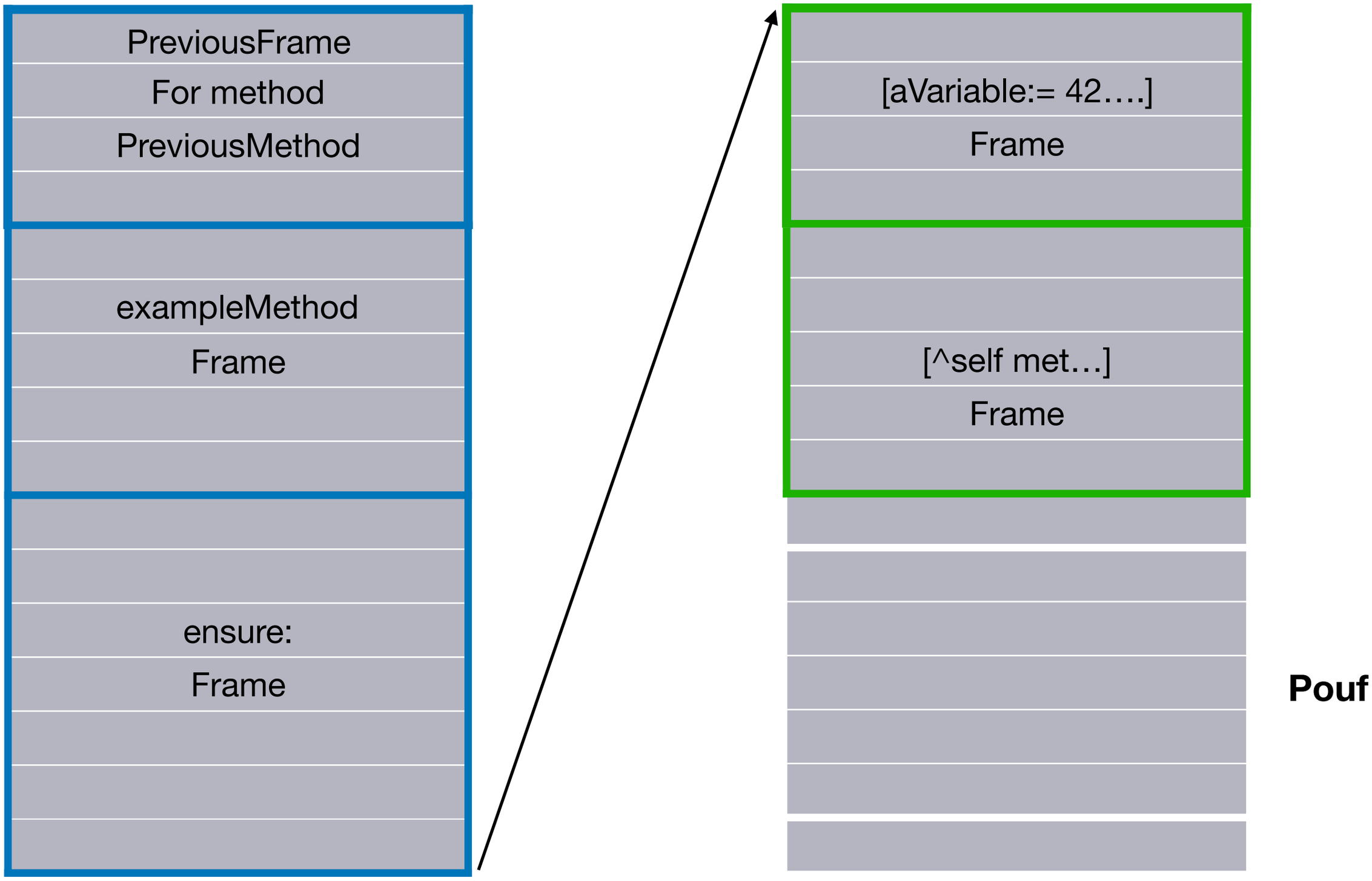
An example

```
exampleMethod

  | aVariable |
  [aVariable := 42.
  [^self methodWithSideEffectsOn: aVariable ]]
ensure: [ self cleanUp: aVariable ]
```

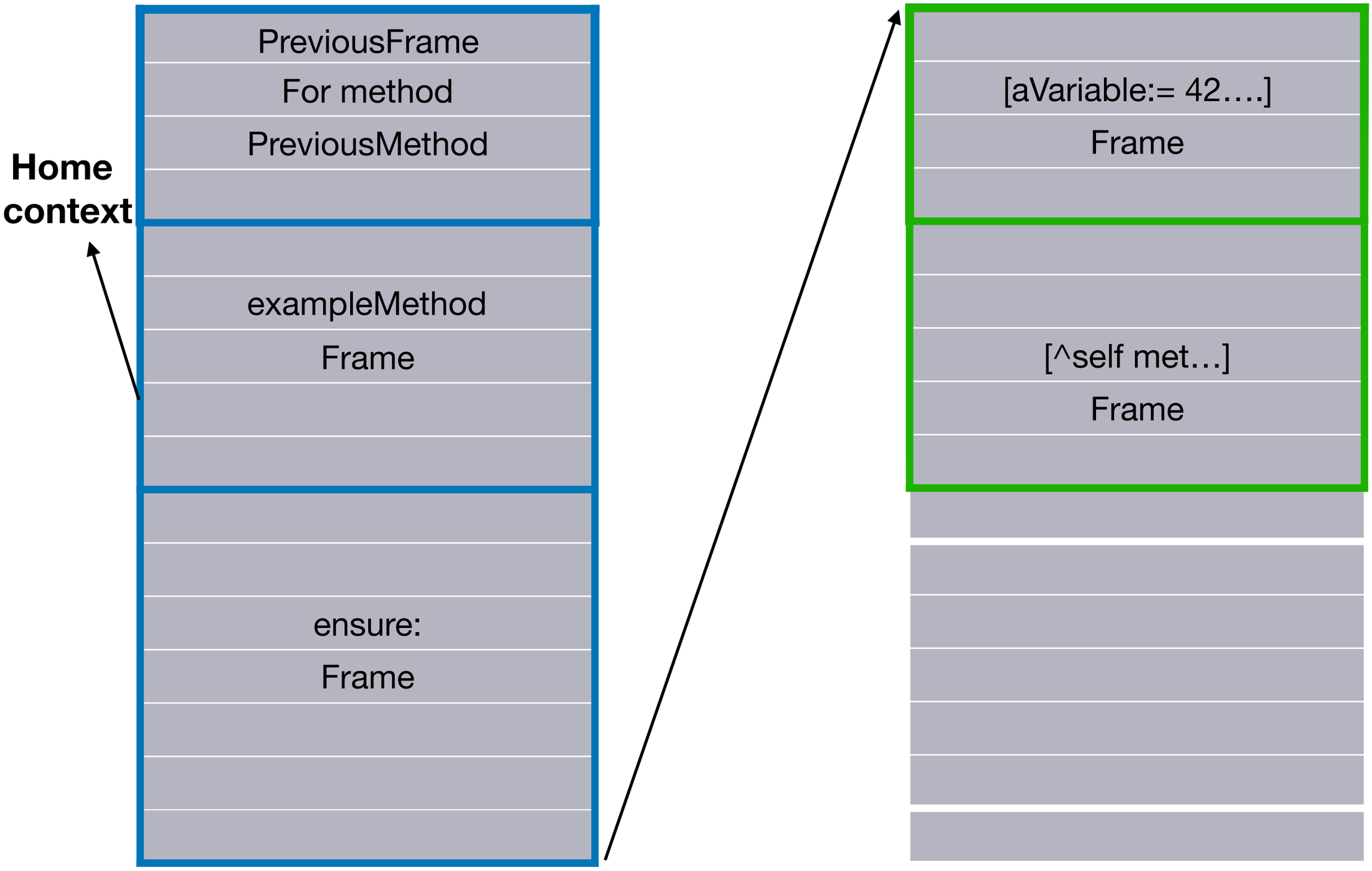
Stack point of view



Method return



Block non local return

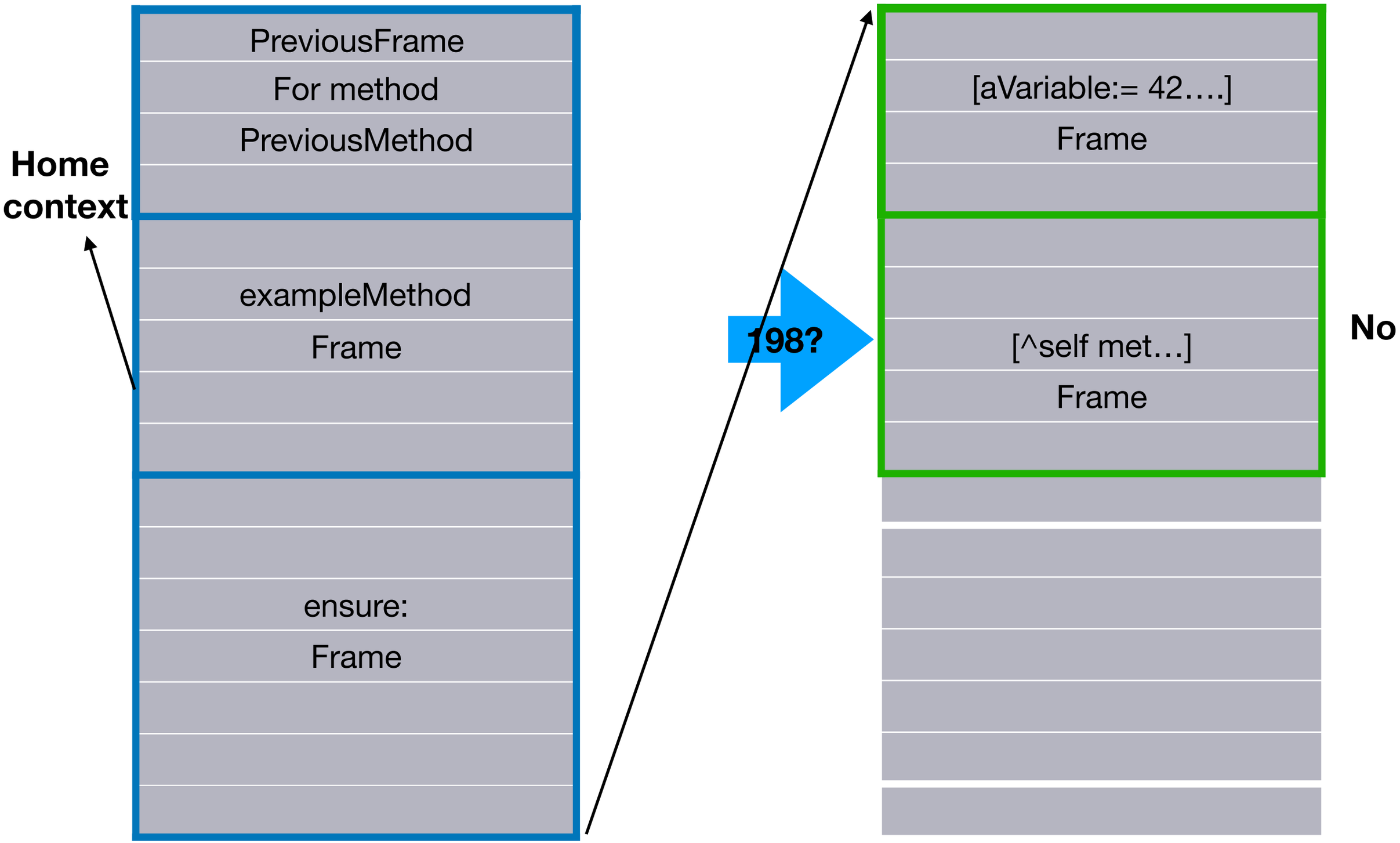


How to detect unwind?

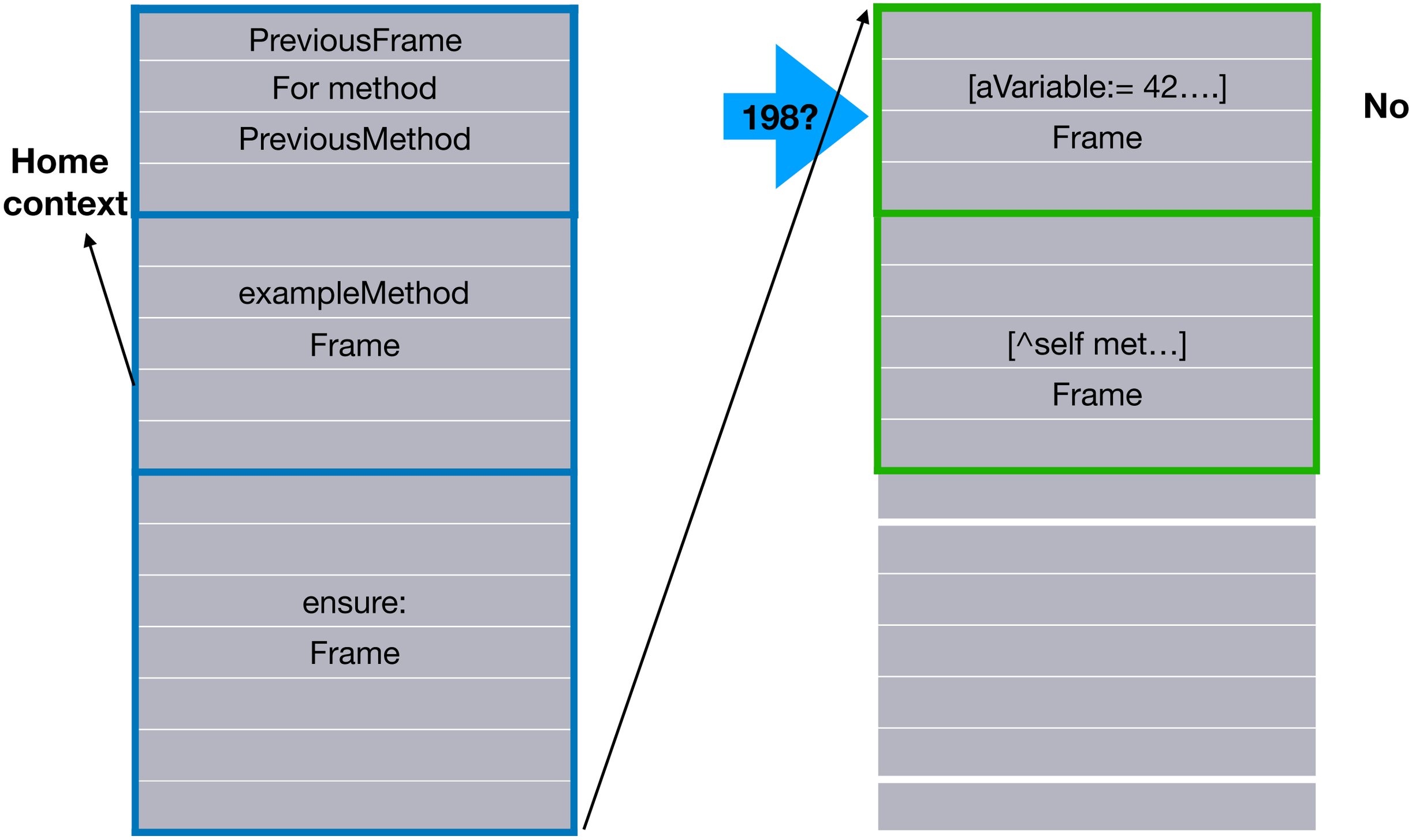
When you encounter primitive 198 in the stack:

```
ensure: aBlock
   "Evaluate a termination block after evaluating the receiver, regardless of
    whether the receiver's evaluation completes.  N.B.  This method is *not*
    implemented as a primitive.  Primitive 198 always fails.  The VM uses prim
    198 in a context's method as the mark for an ensure:/ifCurtailed: activation."

   | complete returnValue |
   <primitive: 198>
   returnValue := self valueNoContextSwitch.
   complete ifNil:[
      complete := true.
      aBlock value.
   ].
   ^ returnValue
```

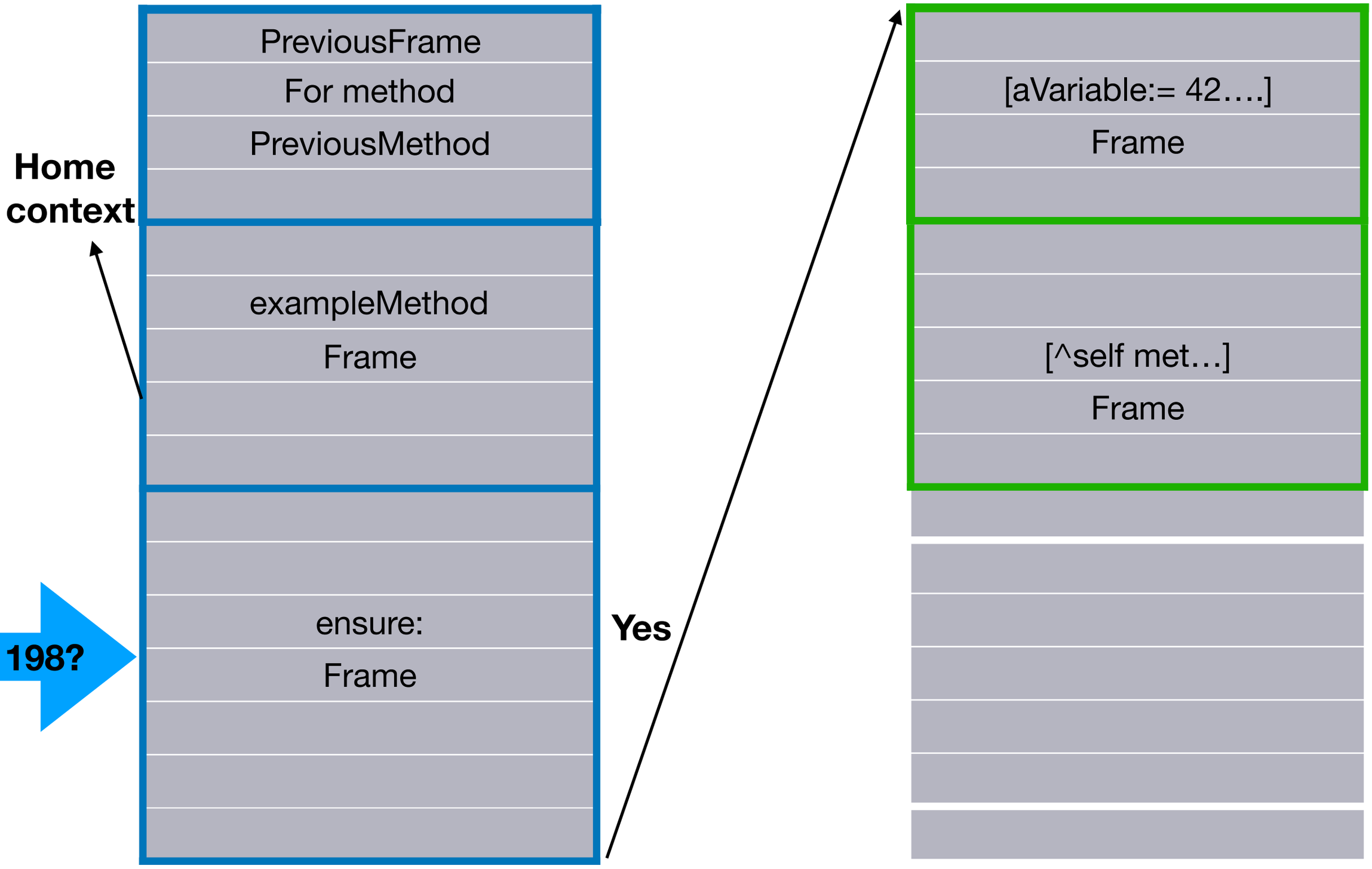
Before return, look for unwind from current frame



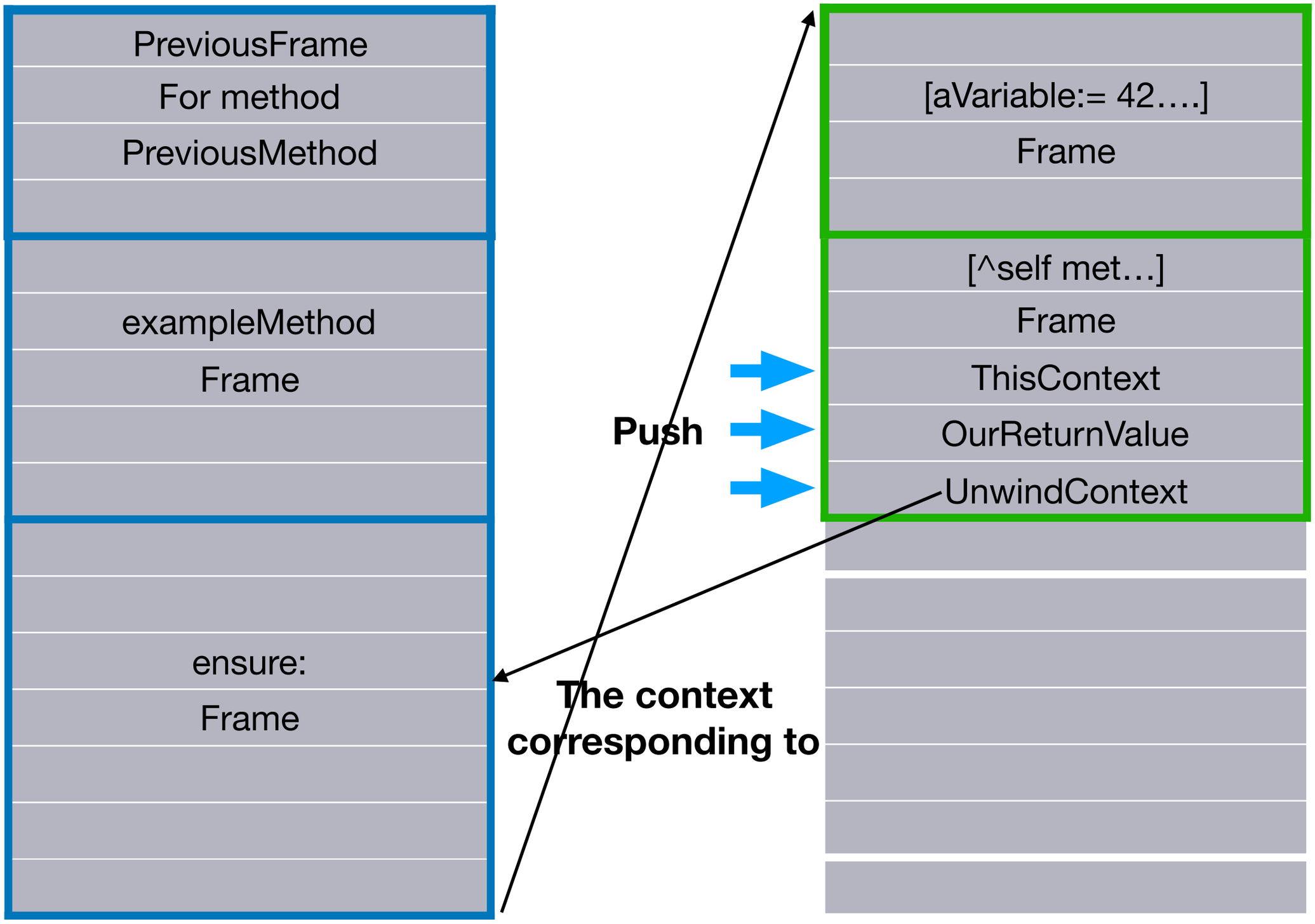
To home context



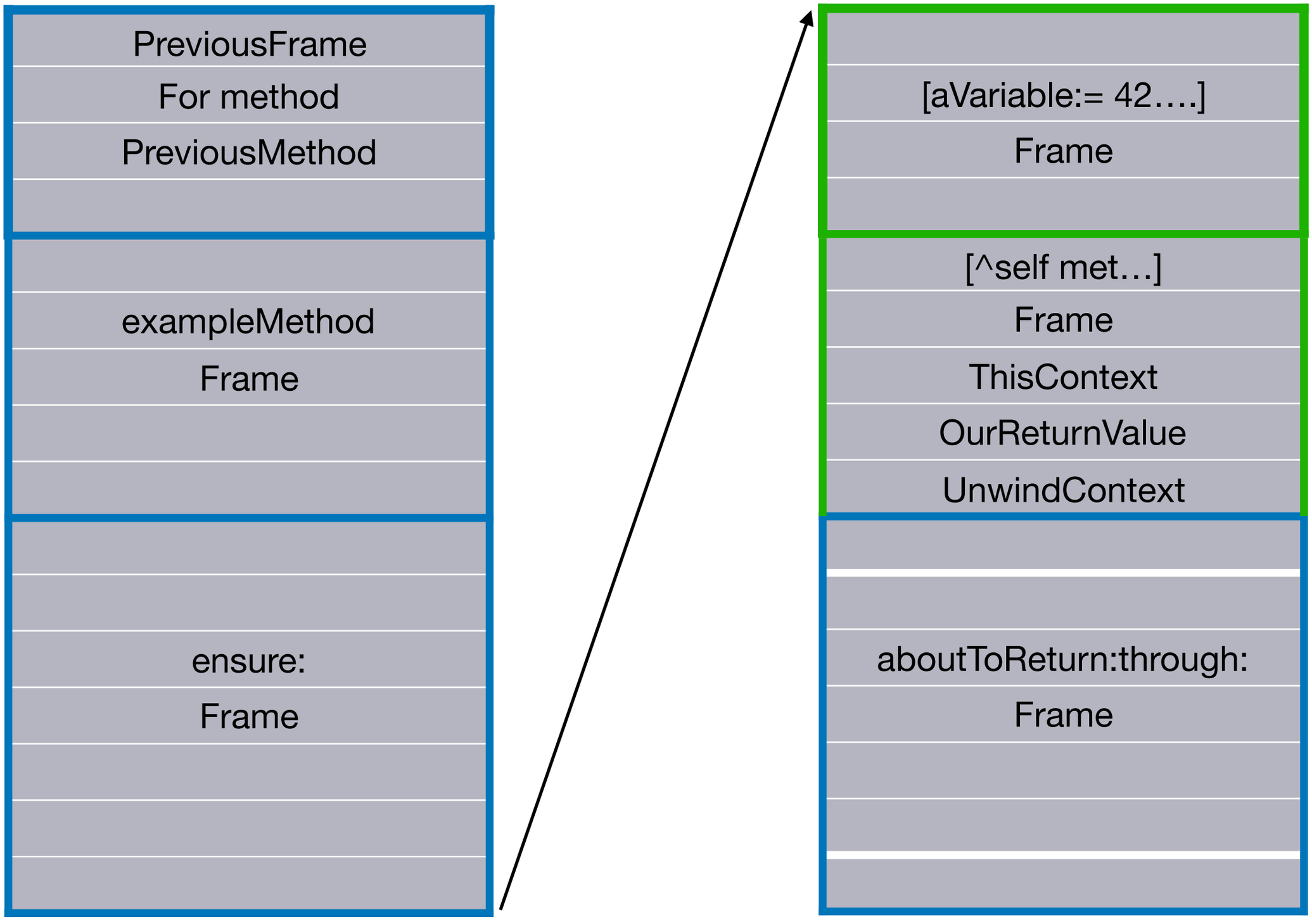
To home context



Prepare message send



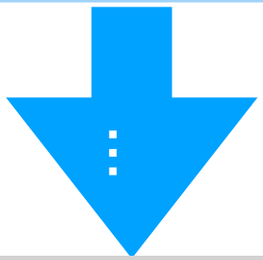
Sends message SelectorAboutToReturn



The image handles the rest

aboutToReturn: ×

```
aboutToReturn: result through: firstUnwindContext
    "Called from VM when an unwindBlock is found between self and its home.
     Return to home's sender, executing unwind blocks on the way."

    self methodReturnContext return: result through: firstUnwindContext
```



resume:through: ×

```
resume: value through: firstUnwindContext
    "Unwind thisContext to self and resume with value as result of last send.
     Execute any unwind blocks while unwinding.
     ASSUMES self is a sender of thisContext."

    | context unwindBlock |
    self isDead
        ifTrue: [ self cannotReturn: value to: self ].
    context := firstUnwindContext.
    [ context isNil ] whileFalse: [
        context unwindComplete ifNil:[
            context unwindComplete: true.
            unwindBlock := context unwindBlock.
            thisContext terminateTo: context.
            unwindBlock value].
        context := context findNextUnwindContextUpTo: self].
    thisContext terminateTo: self.
    ^value
```

Find the unwind block from the unwind context.
Value the unwind block.
Find next unwind.
Etc..

Disclaimer

Oversimplified examples but almost all of the cases work like this.

Only in the code relies the truth.

https://github.com/pharo-project/opensmalltalk-vm.git

StackFrames and BlockClosures with JIT will have their own presentations.

Plan

- BlockClosure creation => special bytecode
- The value message => create a frame with Block
- Return and non local return => find the correct frame (home context) to return to
- Unwind protect => before returning, search between block frame and home context for unwind if found send SelectorAboutToReturn