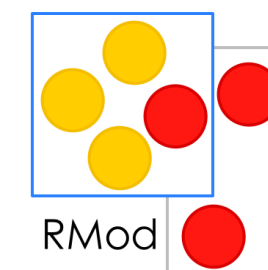




Structure of Machine Code Methods

An introduction

Pablo Tesone - 22/10/2020



Previously....

We have seen how we generate the code...

```
<76> push 1
```

```
<20> push 17
```

```
<B0> send #+
```

```
<7C> returnTop
```

bytecode

```
move r1 #1
```

```
move r2 #17
```

```
checkSmallInt r1
```

```
checkSmallInt r2
```

```
add r3 r1 r2
```

```
checkSmallInt r3
```

```
move r1 r3
```

```
ret
```

CogRTL - IR

```
move X5 #1
```

```
move X3 #17
```

```
test X5 #0x1
```

```
je 0x40
```

```
test X3 #0x1
```

```
je 0x32
```

```
add X0 X3 X5
```

```
test X0 #0x1
```

```
je 0x24
```

```
move X5 X0
```

```
ret
```

Machine Code

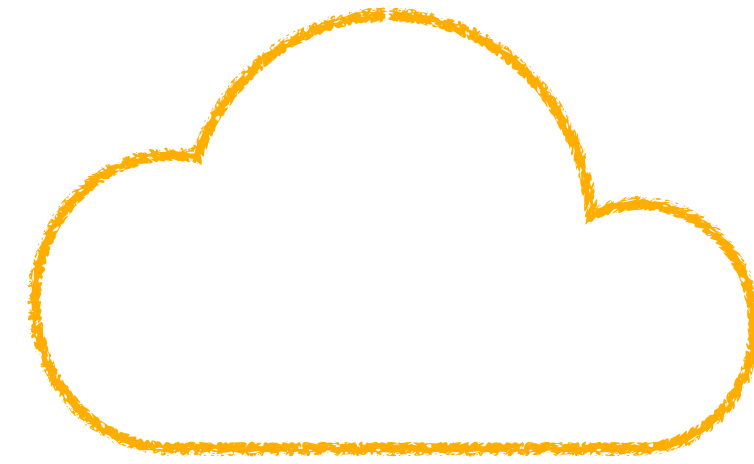
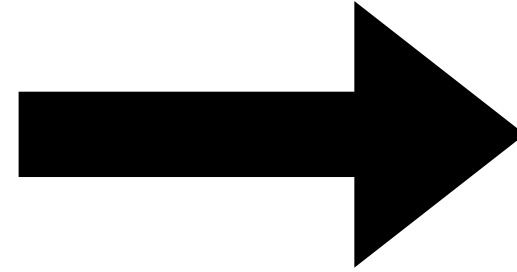
IR Generation
"gen*"

Machine Code Generation
"Concretize"

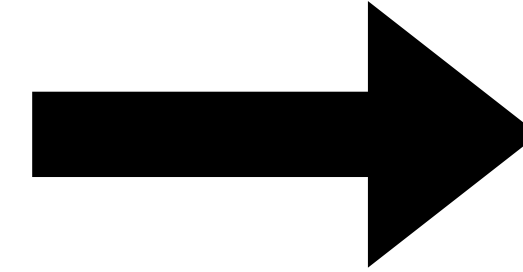
Machine Code



Compiled
Method



Cog Compiler



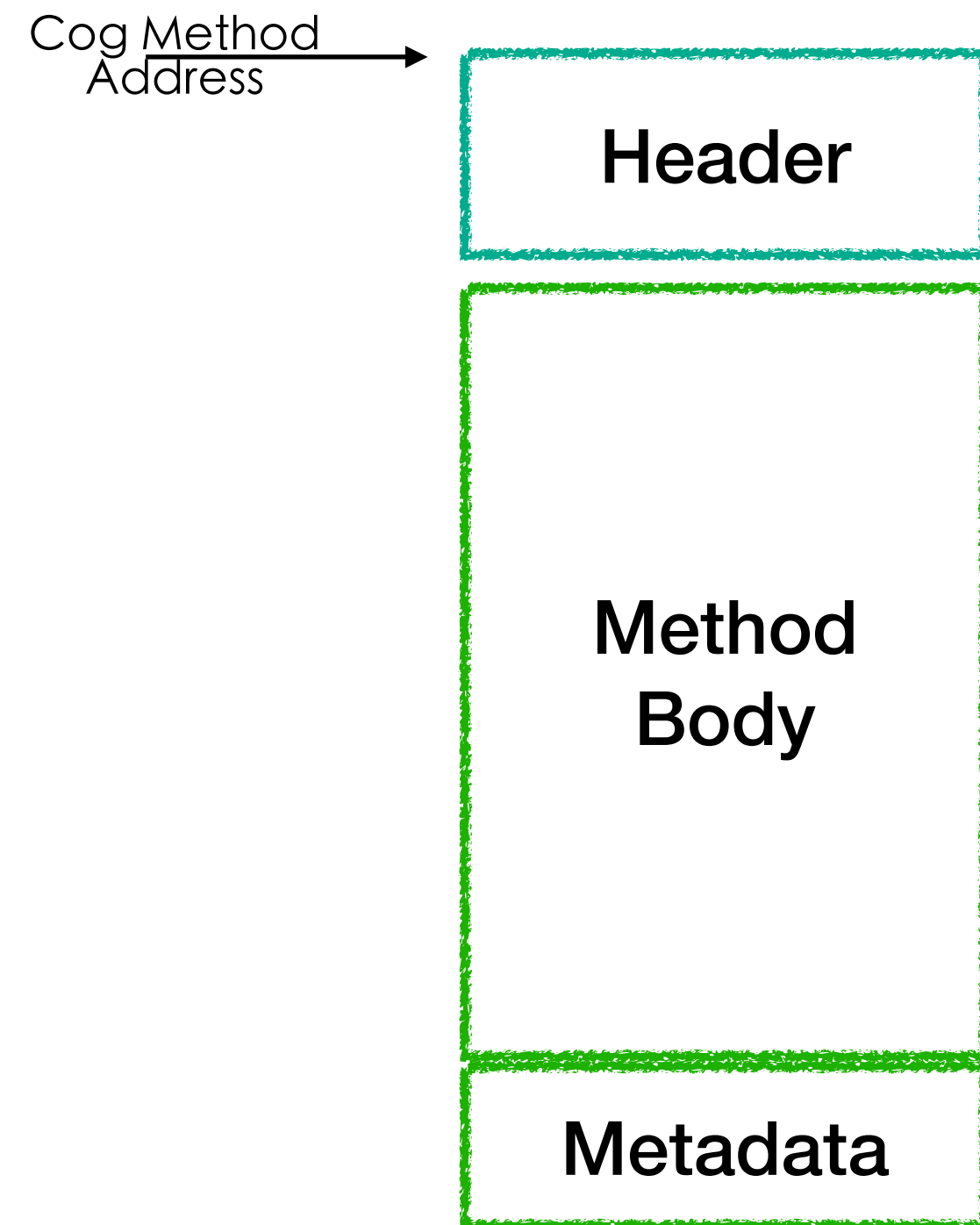
Machine
Code Method
(Cog Methods)



Method Zone (Cog Zone)

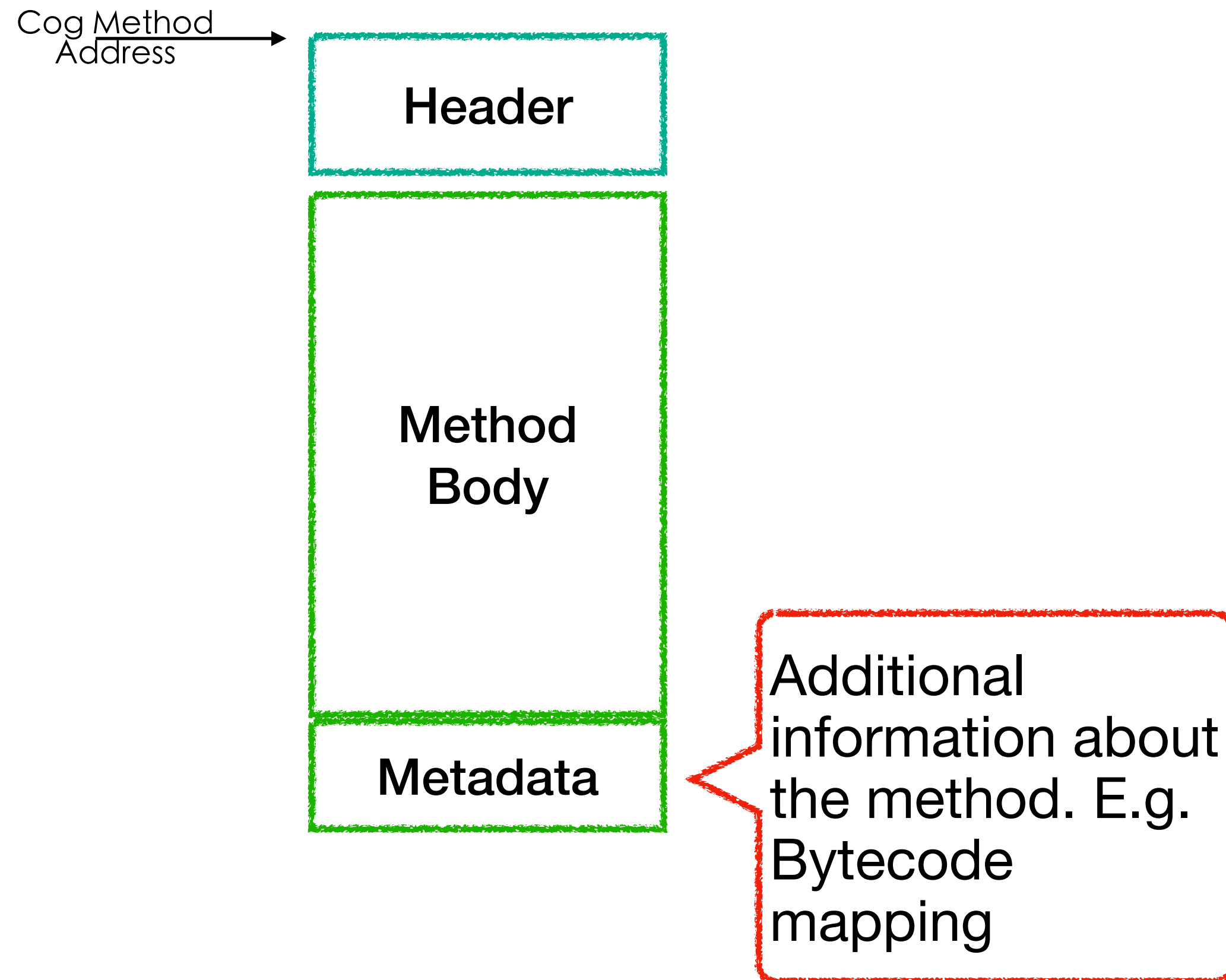
- Cog Methods are created in a particular region of memory.
- ~ 1.4Mb in a Pharo VM
- Used to store all machine code methods
- Garbage collected and reused

Cog Method Structure



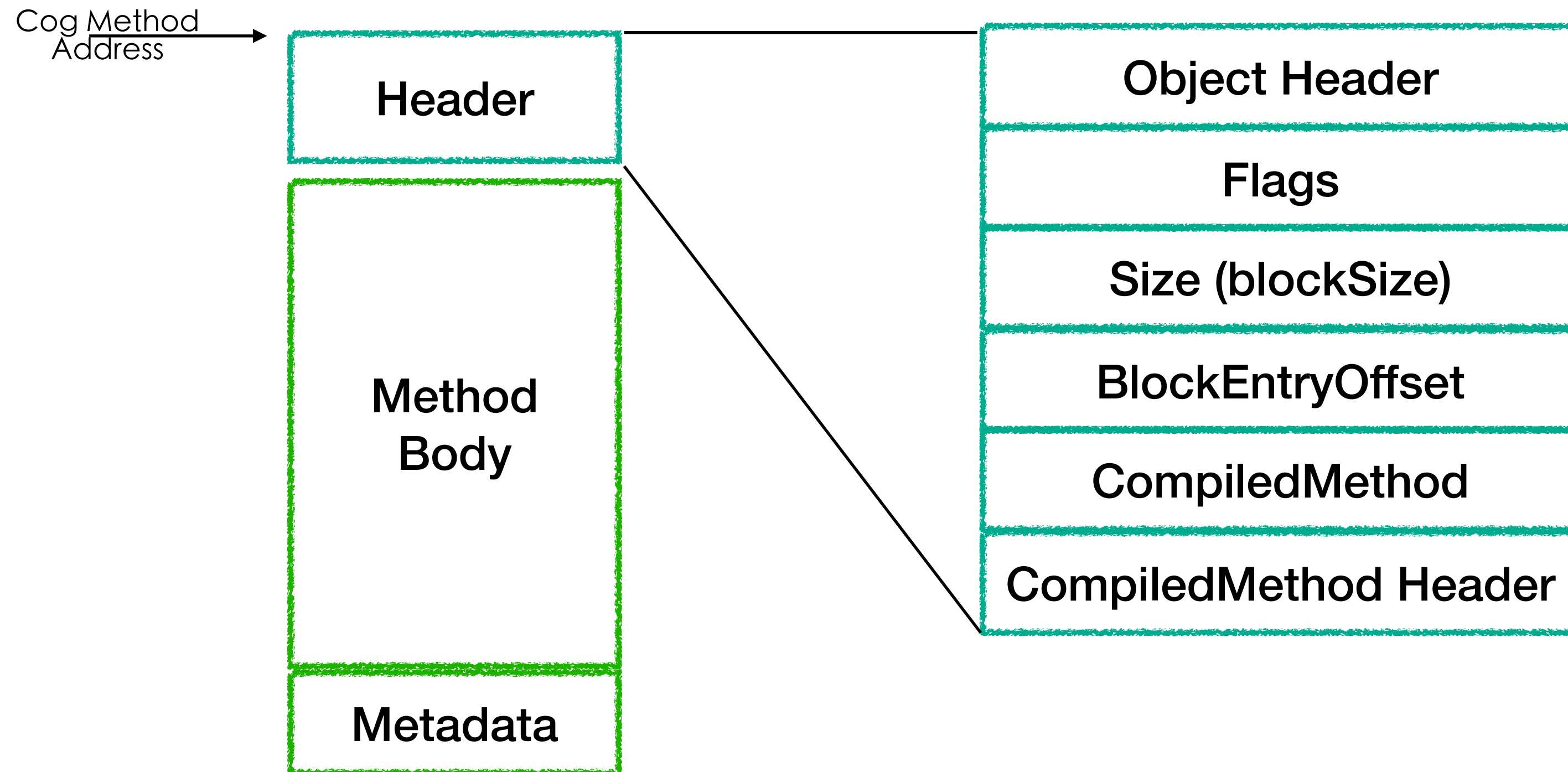
Cog Method Structure

Metadata



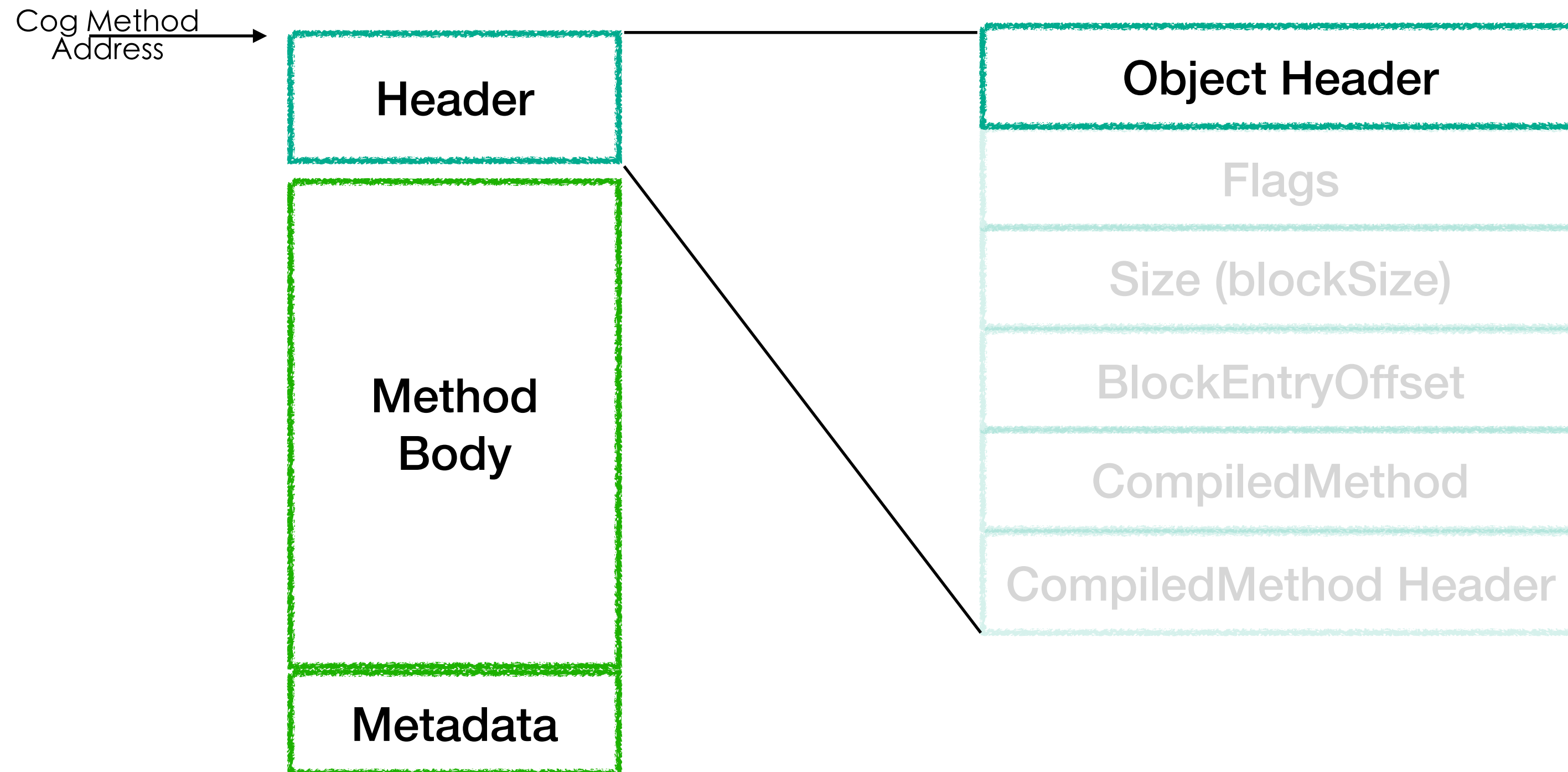
Cog Method Structure

Header



Cog Method Structure

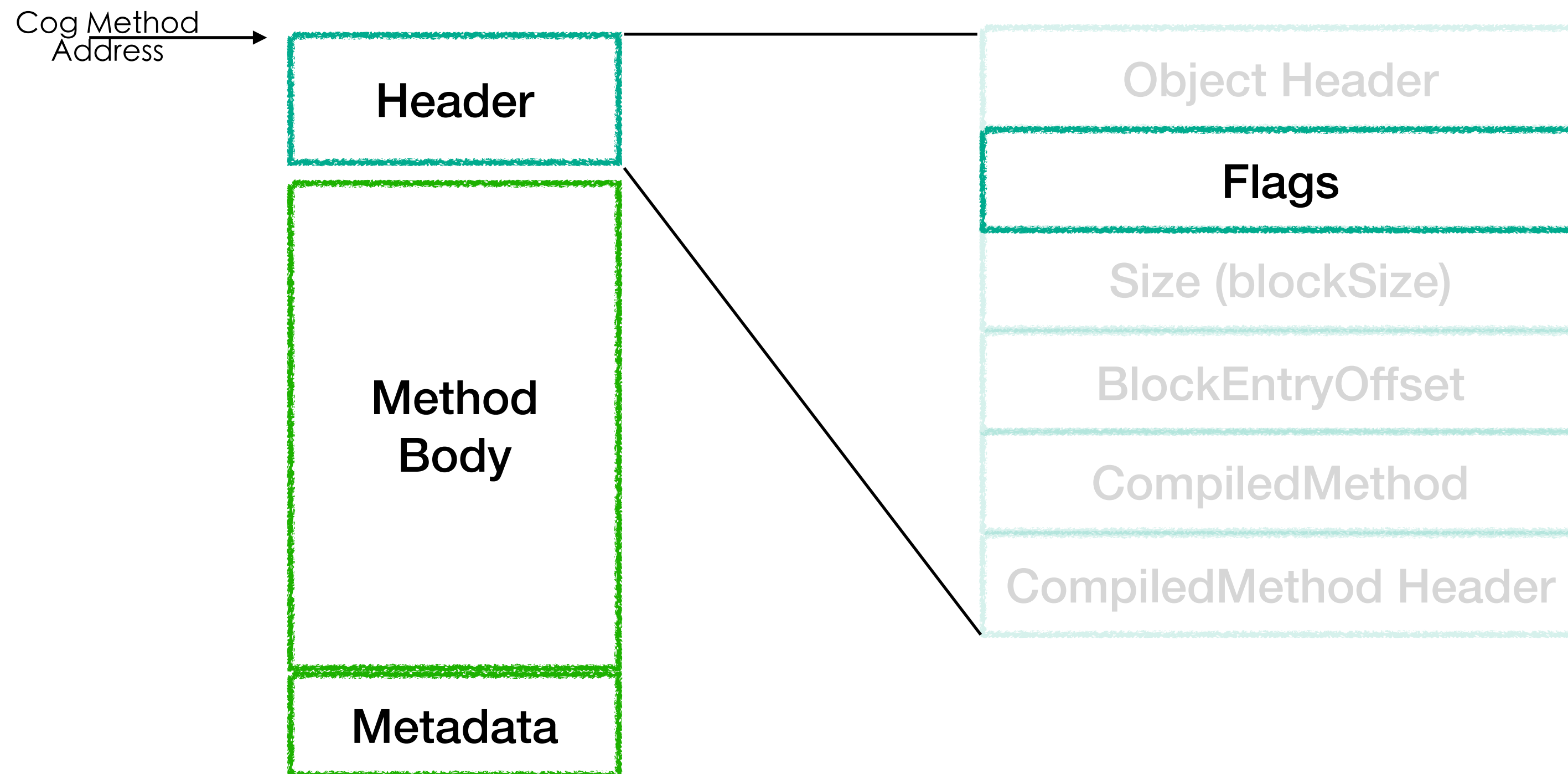
Header



Fake Object Header to be "similar" to objects

Cog Method Structure

Header



The Cog logo is a stylized, vertical graphic on the left side of the slide. It features a central vertical axis with a sunburst at the top. The sunburst is composed of several rays emanating from a central point, with colors transitioning from blue at the top to orange and red. Below the sunburst, the graphic consists of several overlapping, curved, ribbon-like shapes that spiral downwards. The colors of these shapes transition from dark red at the top to dark blue and finally to bright blue at the bottom. The overall effect is that of a cog or gear with a sunburst at its center.

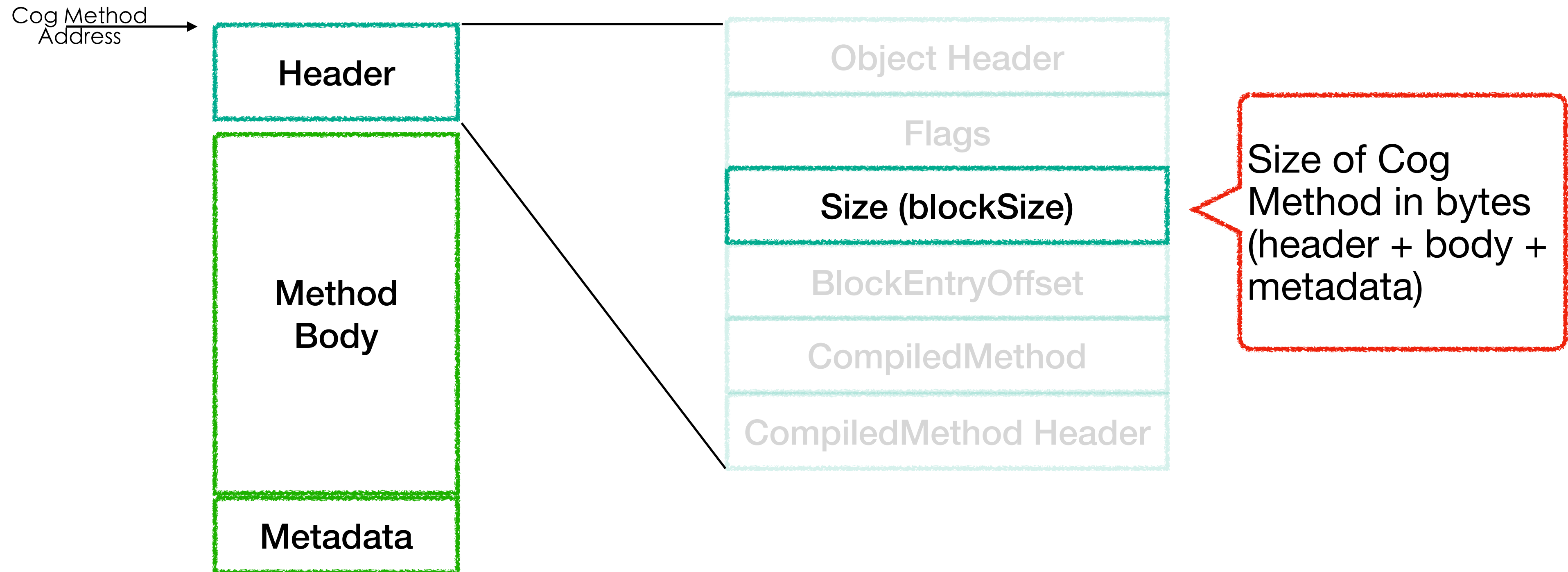
Cog Method Structure

Header - Flags

- Between other fields:
 - number of arguments
 - type (method, block, Polymorphic-Inline-Cache)
 - usage count

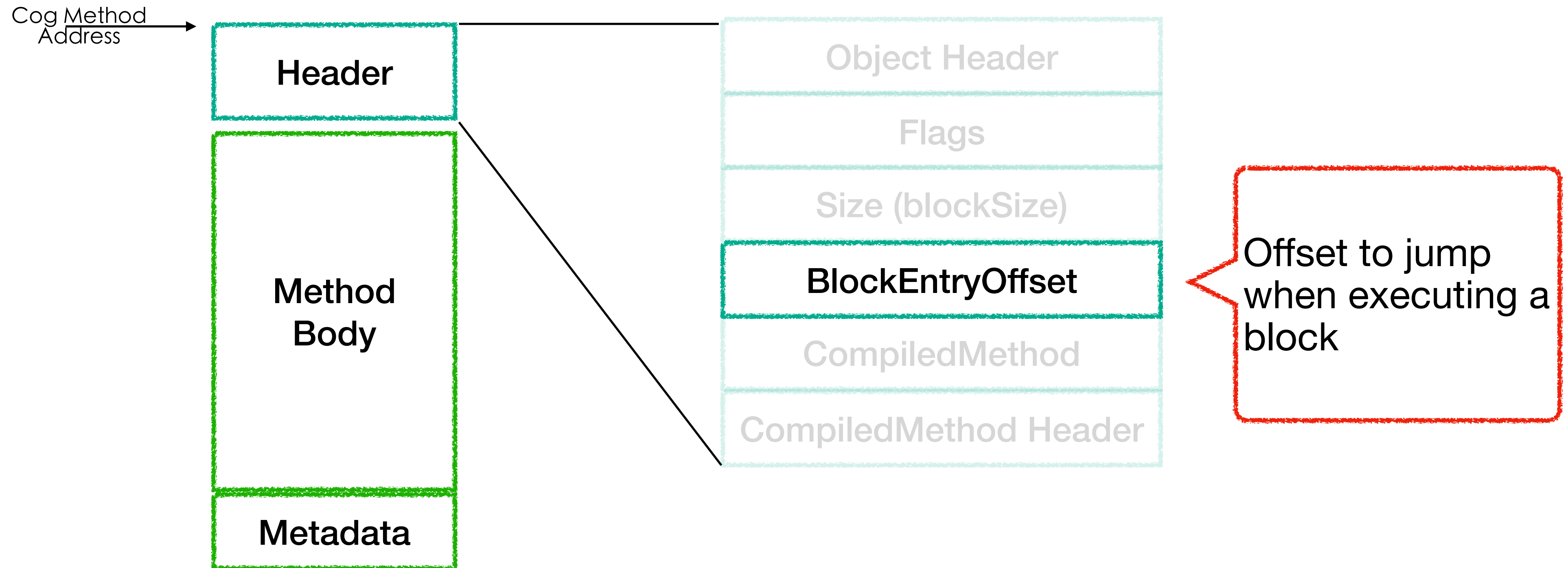
Cog Method Structure

Header



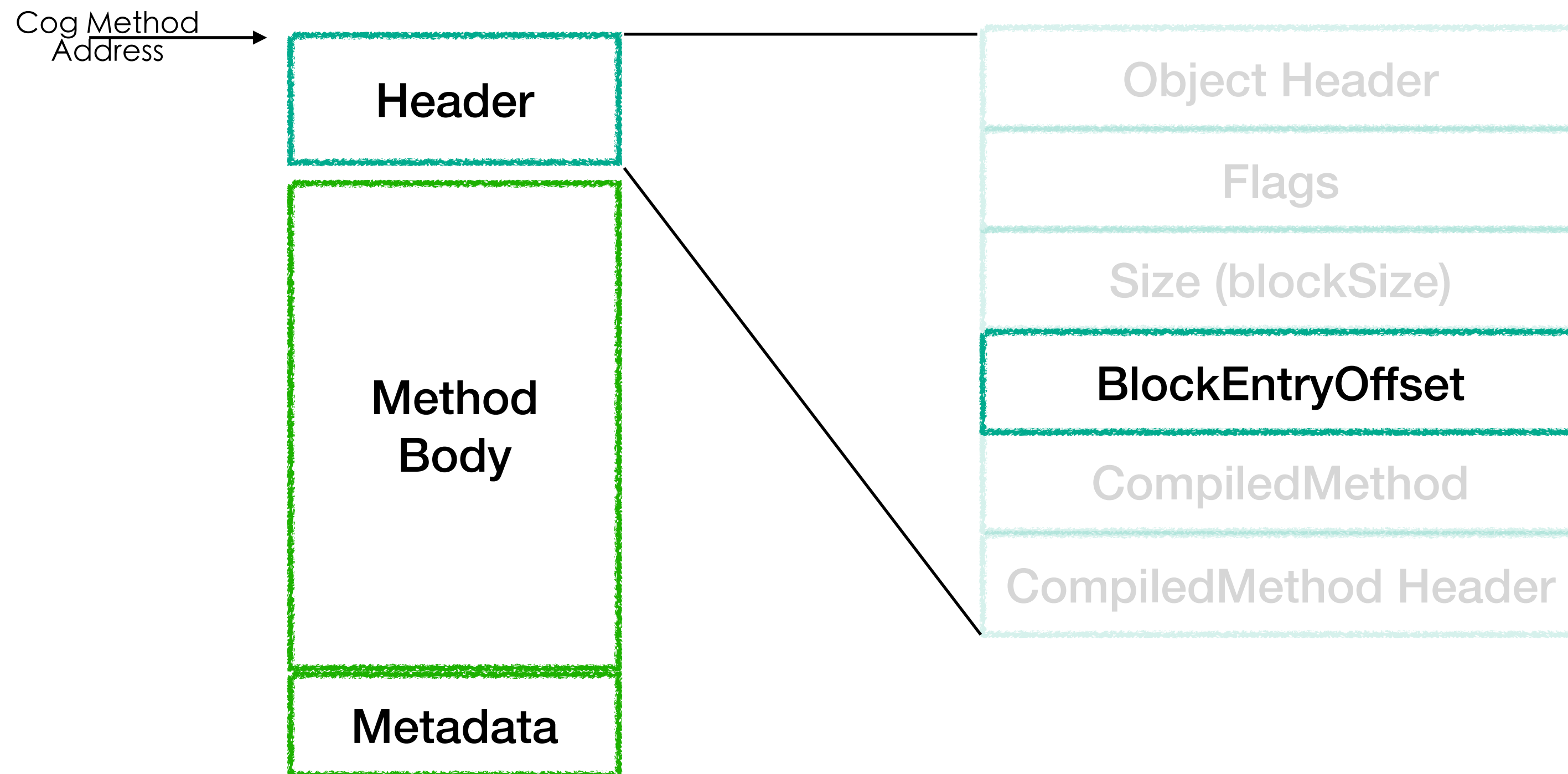
Cog Method Structure

Header



Cog Method Structure

Header

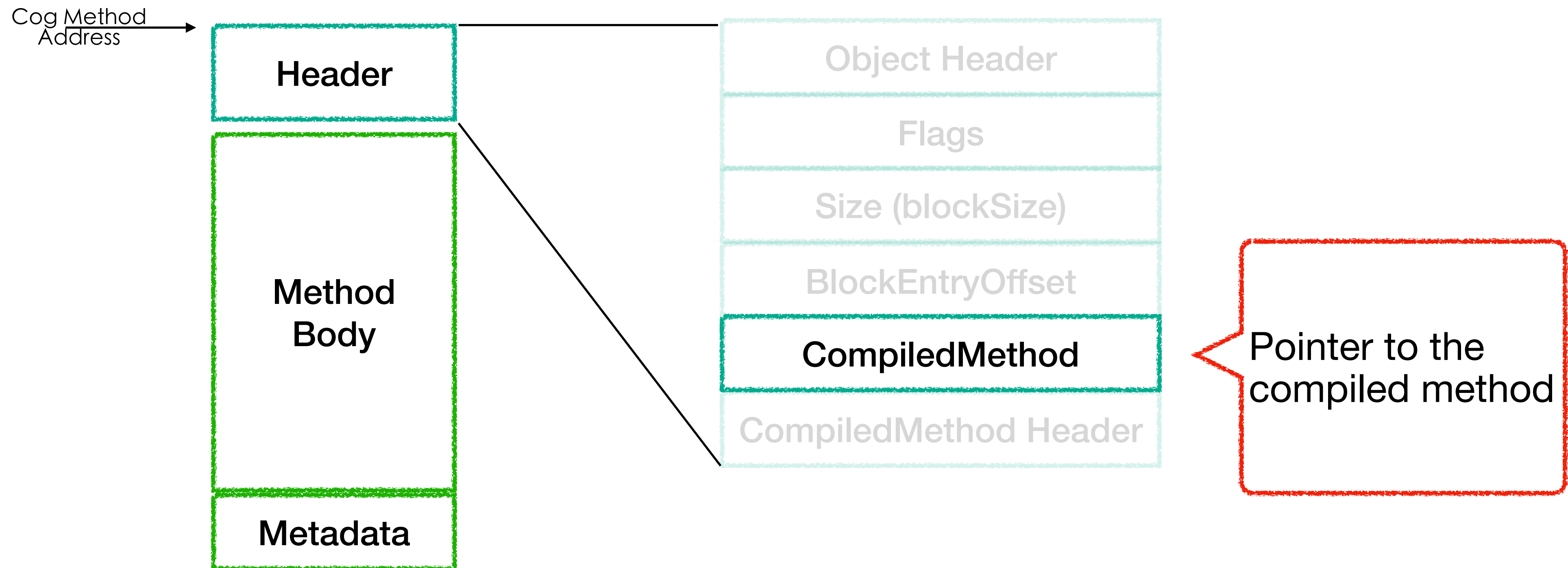


Offset to jump
when executing a
block

Outside the Scope of this
presentation... wait for the
block presentation :)

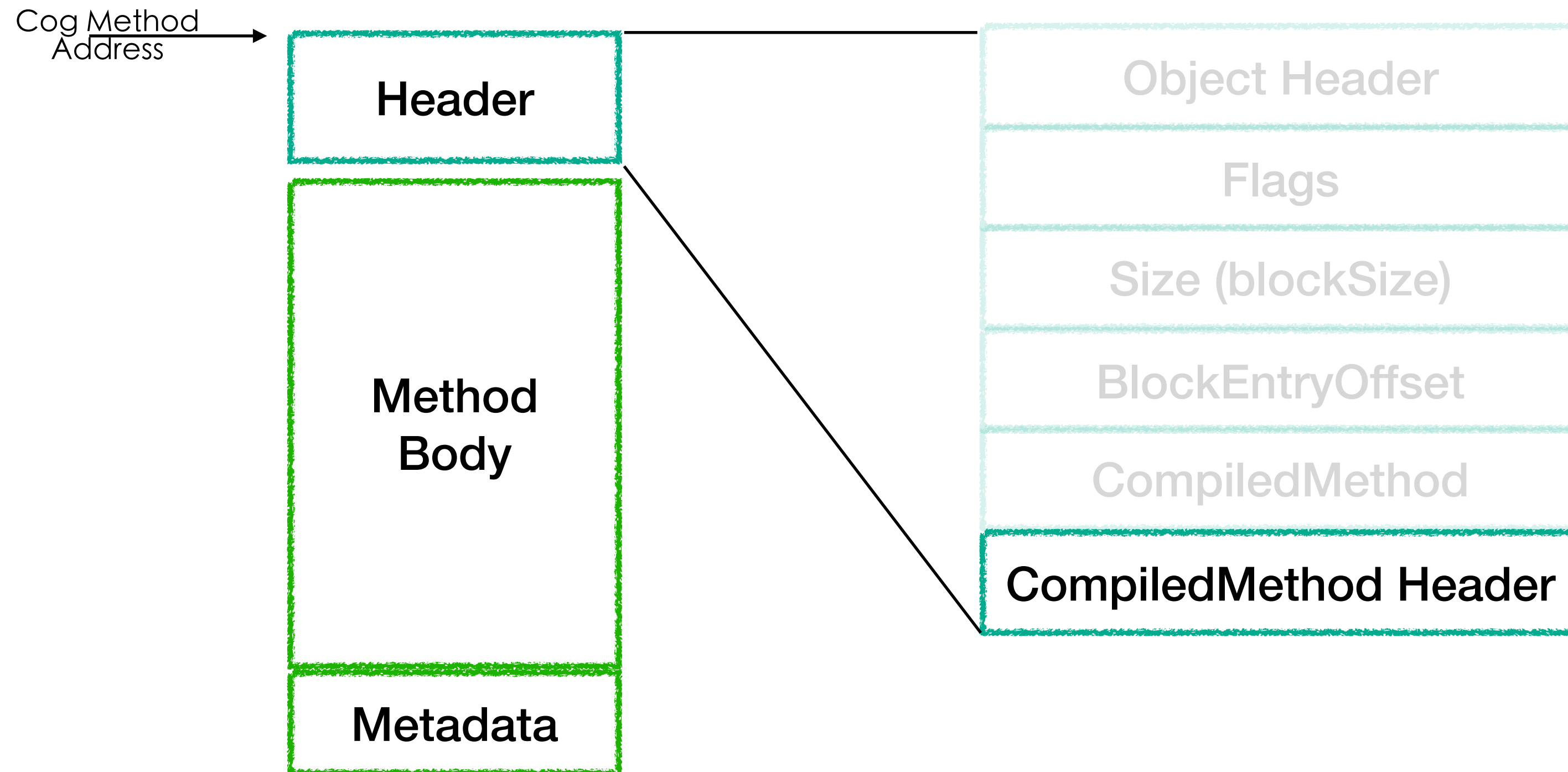
Cog Method Structure

Header



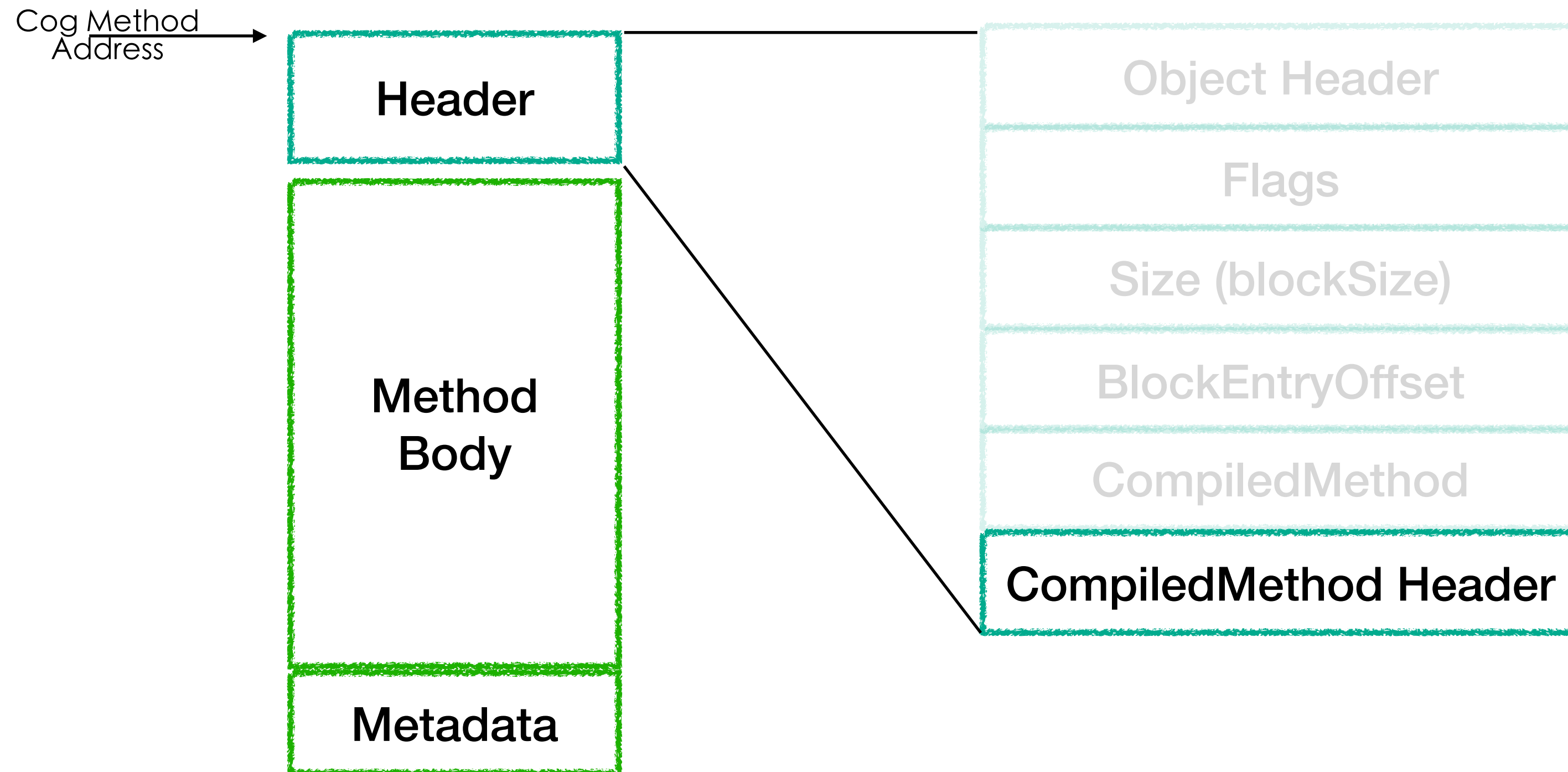
Cog Method Structure

Header



Cog Method Structure

Header



Why???

Original
Compiled
Method Header

Compiled Method and CogMethods

Without corresponding CogMethod

CompiledMethod



A number encoding:
#arguments,
#literals, encoder,
#temporaries, etc

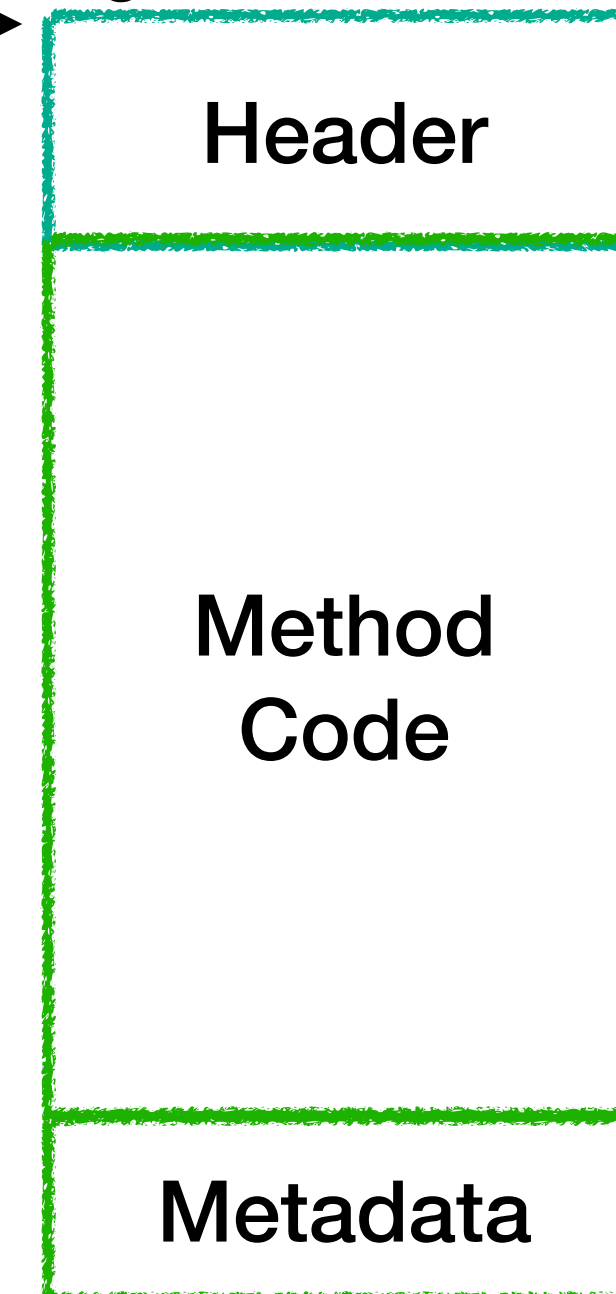
Compiled Method and CogMethods

With Corresponding CogMethod

CompiledMethod



CogMethod



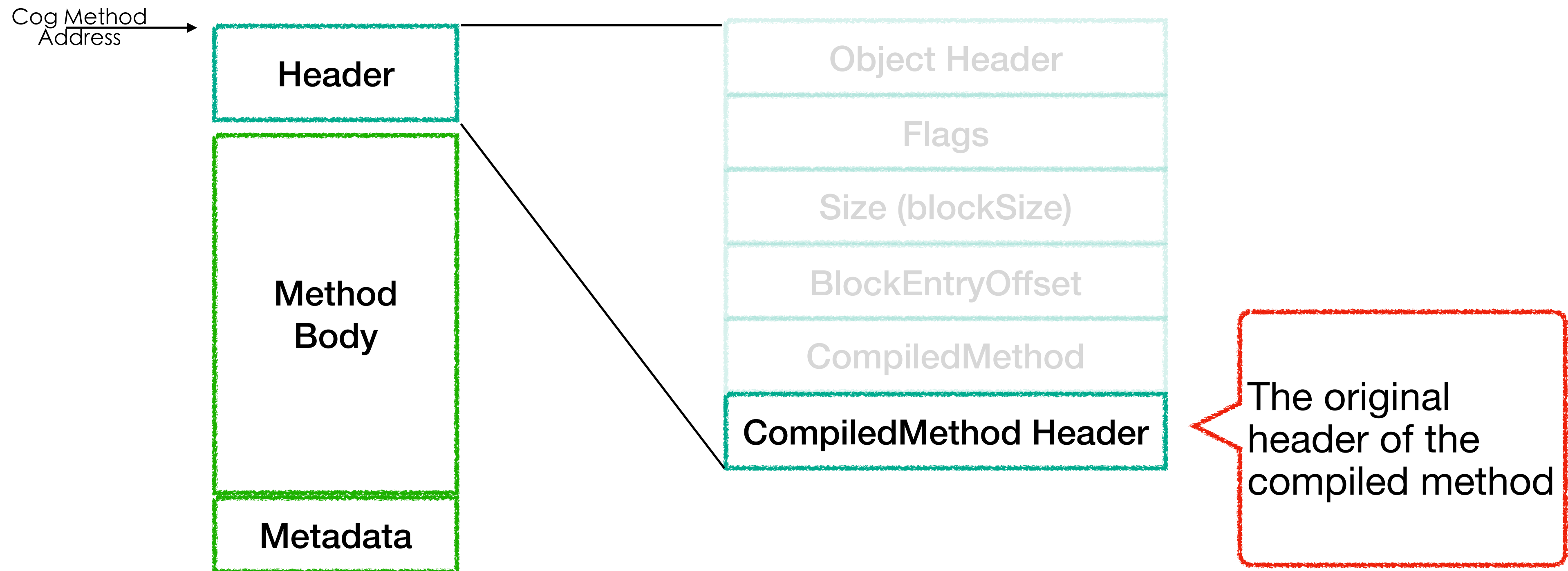
Replaced with a
Pointer to the
CogMethod



Compiled Method and CogMethods With Corresponding CogMethod

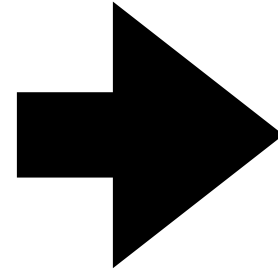
- So...
 - If a Compiled Method has a SmallInteger, it is the header and the method is not compiled in machine code.
 - If the Compiled Method has a Pointer as header, it has a corresponding compiled method in Machine code.
 - And the header??

Compiled Method and CogMethods With Corresponding CogMethod

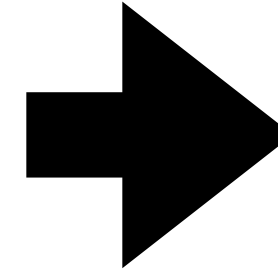


Compiling a Simple Method

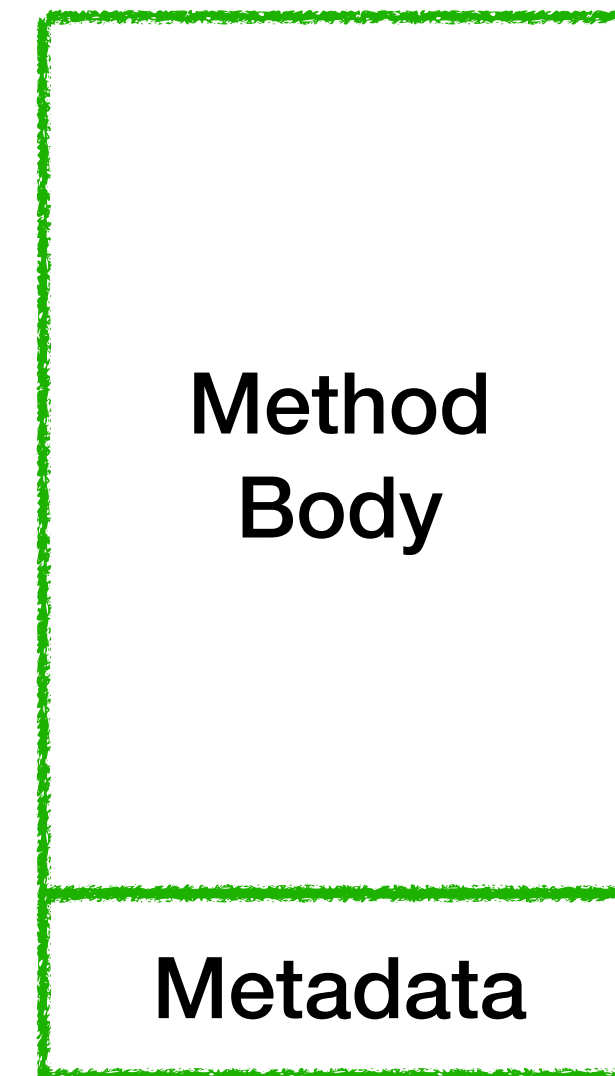
```
methodToCompile1  
  
  ^ 42
```



```
33 <20> pushConstant: 42  
34 <5C> returnTop
```



Object Header	16r8000000A000035
#Args / type	0 / 2 (Method)
Size	120
Block Entry	0
Method Object	16r10218C0
Method Header	16r19



Compiling a Simple Method

Body Content

```
33 <20> pushConstant: 42  
34 <5C> returnTop
```

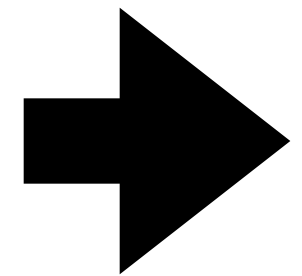


```
Label 0: MoveCqR 0 ReceiverResultReg  
PushR LinkReg  
Call 16777688/10001D8  
AlignmentNops 8  
Label 1: AndCqRR 7 ReceiverResultReg TempReg  
JumpNonZero Label 2  
MoveMwrR 0 ReceiverResultReg TempReg  
AndCqR 4194303/3FFFFFF TempReg  
Nop  
Nop  
Nop  
Label 2: CmpRR ClassReg TempReg  
JumpNonZero Label 0  
  
PushCq 337/151  
PopR ReceiverResultReg  
RetN 8
```

Compiling a Simple Method

Body Content

```
33 <20> pushConstant: 42  
34 <5C> returnTop
```



Method
Entry →

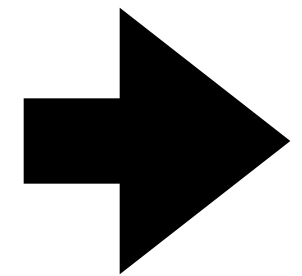
```
Label 0: MoveCqR 0 ReceiverResultReg  
PushR LinkReg  
Call 16777688/10001D8  
AlignmentNops 8  
Label 1: AndCqRR 7 ReceiverResultReg TempReg  
JumpNonZero Label 2  
MoveMwrR 0 ReceiverResultReg TempReg  
AndCqR 4194303/3FFFFFF TempReg  
Nop  
Nop  
Nop  
Label 2: CmpRR ClassReg TempReg  
JumpNonZero Label 0  
  
PushCq 337/151  
PopR ReceiverResultReg  
RetN 8
```

Type Guard &
Abort routine call

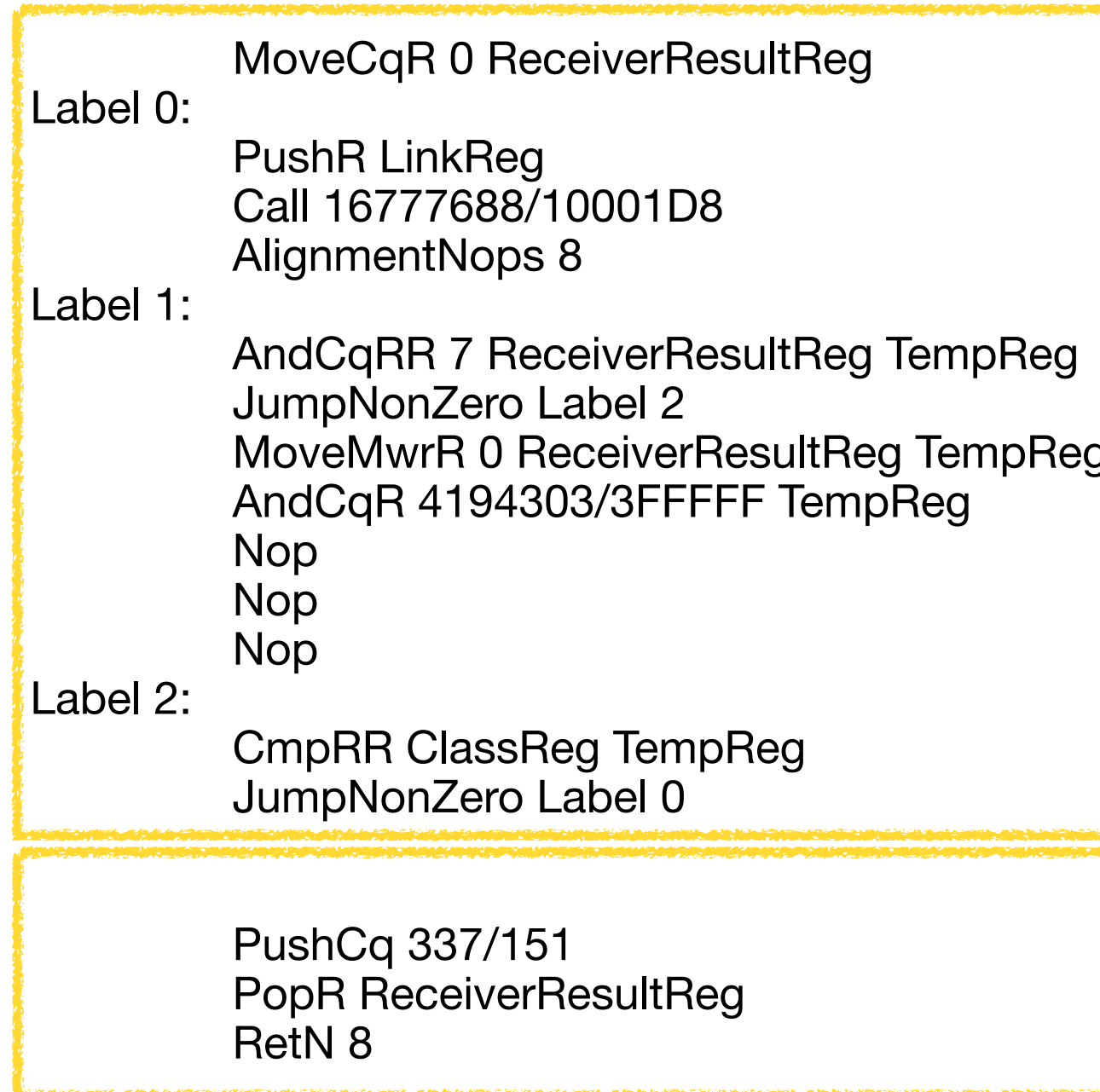
Compiling a Simple Method

Body Content

```
33 <20> pushConstant: 42  
34 <5C> returnTop
```



Method
Entry →



Method Body



Checking the Type on Method Activation

- When a Cog method is activated:
 - We need to check if the type is correct!
 - We need to compare the expected class with the class of the receiver.
 - In a simple scenario:
 - We have the class tag in a register (ClassReg)
 - We have the receiver in a Register (ReceiverResultReg)
 - If the method is not the expected, we need to abort and find the correct method (or MNU)



Checking the Type on Method Activation

- When a Cog method is activated:
 - We need to check if the type is correct!
 - We need to compare the expected class with the class of the receiver.
 - In a simple scenario:
 - We have the class tag in a register (ClassReg)
 - We have the receiver in a Register (ReceiverResultReg)
 - If the method is not the expected, we need to abort and find the correct method (or MNU)

More when talk about Message
Sends & Polymorphic Inline
Caches :)

Checking the Type on Method Activation



Method
Entry →

```
Label 0: MoveCqR 0 ReceiverResultReg
          PushR LinkReg
          Call 16777688/10001D8
          AlignmentNops 8
Label 1: AndCqRR 7 ReceiverResultReg TempReg
          JumpNonZero Label 2
          MoveMwrR 0 ReceiverResultReg TempReg
          AndCqR 4194303/3FFFFFF TempReg
          Nop
          Nop
          Nop
Label 2: CmpRR ClassReg TempReg
          JumpNonZero Label 0

          PushCq 337/151
          PopR ReceiverResultReg
          RetN 8
```

Call Abort
Routine

Checking the Type on Method Activation




Method
Entry →

```
Label 0: MoveCqR 0 ReceiverResultReg
          PushR LinkReg
          Call 16777688/10001D8
          AlignmentNops 8
Label 1:  AndCqRR 7 ReceiverResultReg TempReg
          JumpNonZero Label 2
          MoveMwrR 0 ReceiverResultReg TempReg
          AndCqR 4194303/3FFFFFF TempReg
          Nop
          Nop
          Nop
Label 2:  CmpRR ClassReg TempReg
          JumpNonZero Label 0

          PushCq 337/151
          PopR ReceiverResultReg
          RetN 8
```

Extracting Class from
receiver and
comparing



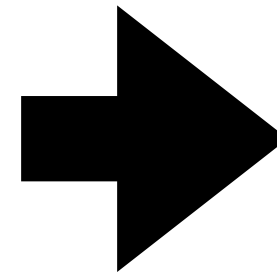
Frame Building

- The creation of the frame is done in the activated method.
 - We have two types of methods
 - Frameless
 - Frameful
 - Depends if they are Leaves (if they don't send messages)

Frame Building

A Frameless Example

```
33 <20> pushConstant: 42  
34 <5C> returnTop
```



Method
Entry →

Label 0: MoveCqR 0 ReceiverResultReg
PushR LinkReg
Call 16777688/10001D8
AlignmentNops 8

Label 1: AndCqRR 7 ReceiverResultReg TempReg
JumpNonZero Label 2
MoveMwrR 0 ReceiverResultReg TempReg
AndCqR 4194303/3FFFFFF TempReg
Nop
Nop
Nop

Label 2: CmpRR ClassReg TempReg
JumpNonZero Label 0

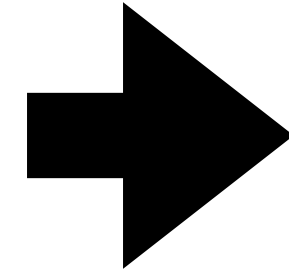
```
PushCq 337/151  
PopR ReceiverResultReg  
RetN 8
```

Method Body
without creating
frame

Frame Building

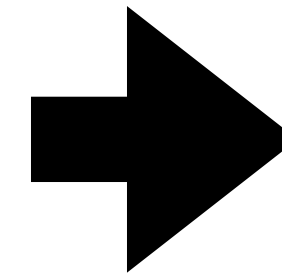
A Framefull Example

```
methodToCompile1
  ^ 42 bitInvert
```



```
41 <20> pushConstant: 42
42 <81> send: bitInvert
43 <5C> returnTop
```

Method
Entry →



Label 0:

```
MoveCqR 0 ReceiverResultReg
PushR LinkReg
Call 16777688/10001D8
AlignmentNops 8
```

Label 1:

```
AndCqRR 7 ReceiverResultReg TempReg
JumpNonZero Label 2
MoveMwrR 0 ReceiverResultReg TempReg
AndCqR 4194303/3FFFFFF TempReg
Nop
Nop
Nop
```

Label 2:

```
CmpRR ClassReg TempReg
JumpNonZero Label 0

PushR LinkReg
PushR FPReg
MoveRR SPReg FPReg
PushCw CogMethodAddress
MoveCqR (#bitInvert oop) SendNumArgsReg
PushR SendNumArgsReg
PushR ReceiverResultReg

PushCq 337/151
MoveMwrR 0 SPReg ReceiverResultReg
MovePatchableC32R 1 ClassReg
Call SendMessageRoutine

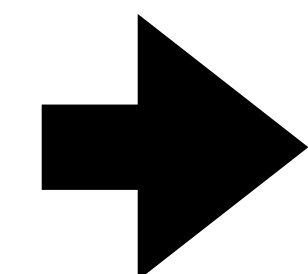
PushR ReceiverResultReg
PopR ReceiverResultReg
MoveRR FPReg SPReg@
PopR FPReg
PopR LinkReg
RetN 8
```



Frame Building

A Frameful Example

```
41 <20> pushConstant: 42  
42 <81> send: bitInvert  
43 <5C> returnTop
```



Method
Entry →

```
Label 0:  
MoveCqR 0 ReceiverResultReg  
PushR LinkReg  
Call 16777688/10001D8  
AlignmentNops 8  
Label 1:  
AndCqRR 7 ReceiverResultReg TempReg  
JumpNonZero Label 2  
MoveMwrR 0 ReceiverResultReg TempReg  
AndCqR 4194303/3FFFFFF TempReg  
Nop  
Nop  
Nop
```

```
Label 2:  
CmpRR ClassReg TempReg  
JumpNonZero Label 0  
  
PushR LinkReg  
PushR FReg  
MoveRR SPReg FReg  
PushCw CogMethodAddress  
MoveCqR (#bitInvert oop) SendNumArgsReg  
PushR SendNumArgsReg  
PushR ReceiverResultReg  
  
PushCq 337/151  
MoveMwrR 0 SPReg ReceiverResultReg  
MovePatchableC32R 1 ClassReg  
Call SendMessageRoutine  
  
PushR ReceiverResultReg  
PopR ReceiverResultReg  
MoveRR FReg SPReg@  
PopR FReg  
PopR LinkReg  
RetN 8
```

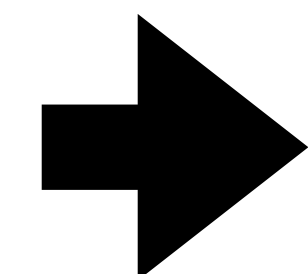
Type Guard &
Abort routine call



Frame Building

A Framefull Example

```
41 <20> pushConstant: 42  
42 <81> send: bitInvert  
43 <5C> returnTop
```



Method
Entry →

Label 0:

```
MoveCqR 0 ReceiverResultReg  
PushR LinkReg  
Call 16777688/10001D8  
AlignmentNops 8
```

Label 1:

```
AndCqRR 7 ReceiverResultReg TempReg  
JumpNonZero Label 2  
MoveMwrR 0 ReceiverResultReg TempReg  
AndCqR 4194303/3FFFFFF TempReg  
Nop  
Nop  
Nop
```

Label 2:

```
CmpRR ClassReg TempReg  
JumpNonZero Label 0
```

```
PushR LinkReg  
PushR FPReg  
MoveRR SPReg FPReg  
PushCw CogMethodAddress  
MoveCqR (#bitInvert oop) SendNumArgsReg  
PushR SendNumArgsReg  
PushR ReceiverResultReg
```

```
PushCq 337/151  
MoveMwrR 0 SPReg ReceiverResultReg  
MovePatchableC32R 1 ClassReg  
Call SendMessageRoutine
```

```
PushR ReceiverResultReg  
PopR ReceiverResultReg  
MoveRR FPReg SPReg  
PopR FPReg  
PopR LinkReg  
RetN 8
```

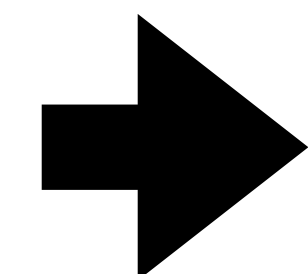
Frame Creation



Frame Building

A Framefull Example

```
41 <20> pushConstant: 42  
42 <81> send: bitInvert  
43 <5C> returnTop
```



Method
Entry →

Label 0:
MoveCqR 0 ReceiverResultReg
PushR LinkReg
Call 16777688/10001D8
AlignmentNops 8

Label 1:
AndCqRR 7 ReceiverResultReg TempReg
JumpNonZero Label 2
MoveMwrR 0 ReceiverResultReg TempReg
AndCqR 4194303/3FFFFFF TempReg
Nop
Nop
Nop

Label 2:
CmpRR ClassReg TempReg
JumpNonZero Label 0

PushR LinkReg
PushR FPCReg
MoveRR SPReg FPCReg
PushCw CogMethodAddress
MoveCqR CogMethodFlags SendNumArgsReg
PushR SendNumArgsReg
PushR ReceiverResultReg

PushCq 337/151
MoveMwrR 0 SPReg ReceiverResultReg
MovePatcheableC32R 1 ClassReg
Call SendMessageRoutine

PushR ReceiverResultReg
PopR ReceiverResultReg
MoveRR FPCReg SPReg
PopR FPCReg
PopR LinkReg
RetN 8

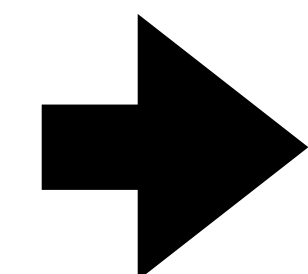
Message Send



Frame Building

A Framefull Example

```
41 <20> pushConstant: 42  
42 <81> send: bitInvert  
43 <5C> returnTop
```



Method
Entry →

Label 0:
MoveCqR 0 ReceiverResultReg
PushR LinkReg
Call 16777688/10001D8
AlignmentNops 8

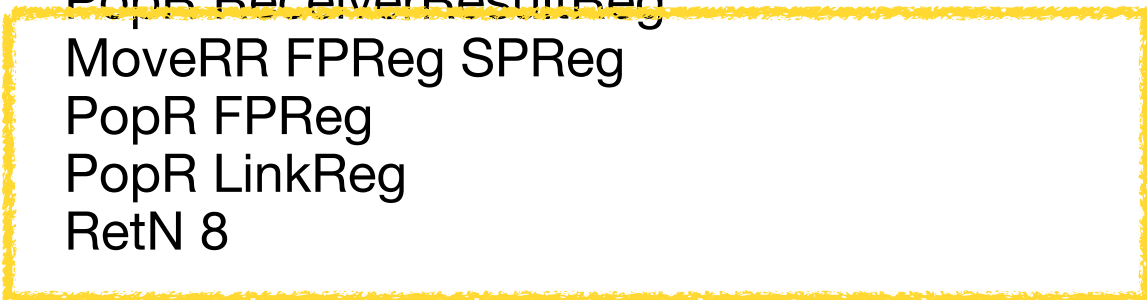
Label 1:
AndCqRR 7 ReceiverResultReg TempReg
JumpNonZero Label 2
MoveMwrR 0 ReceiverResultReg TempReg
AndCqR 4194303/3FFFFFF TempReg
Nop
Nop
Nop

Label 2:
CmpRR ClassReg TempReg
JumpNonZero Label 0

PushR LinkReg
PushR FPCReg
MoveRR SPReg FPCReg
PushCw CogMethodAddress
MoveCqR NilOop SendNumArgsReg
PushR SendNumArgsReg
PushR ReceiverResultReg

PushCq 337/151
MoveMwrR 0 SPReg ReceiverResultReg
MovePatcheableC32R 1 ClassReg
Call SendMessageRoutine

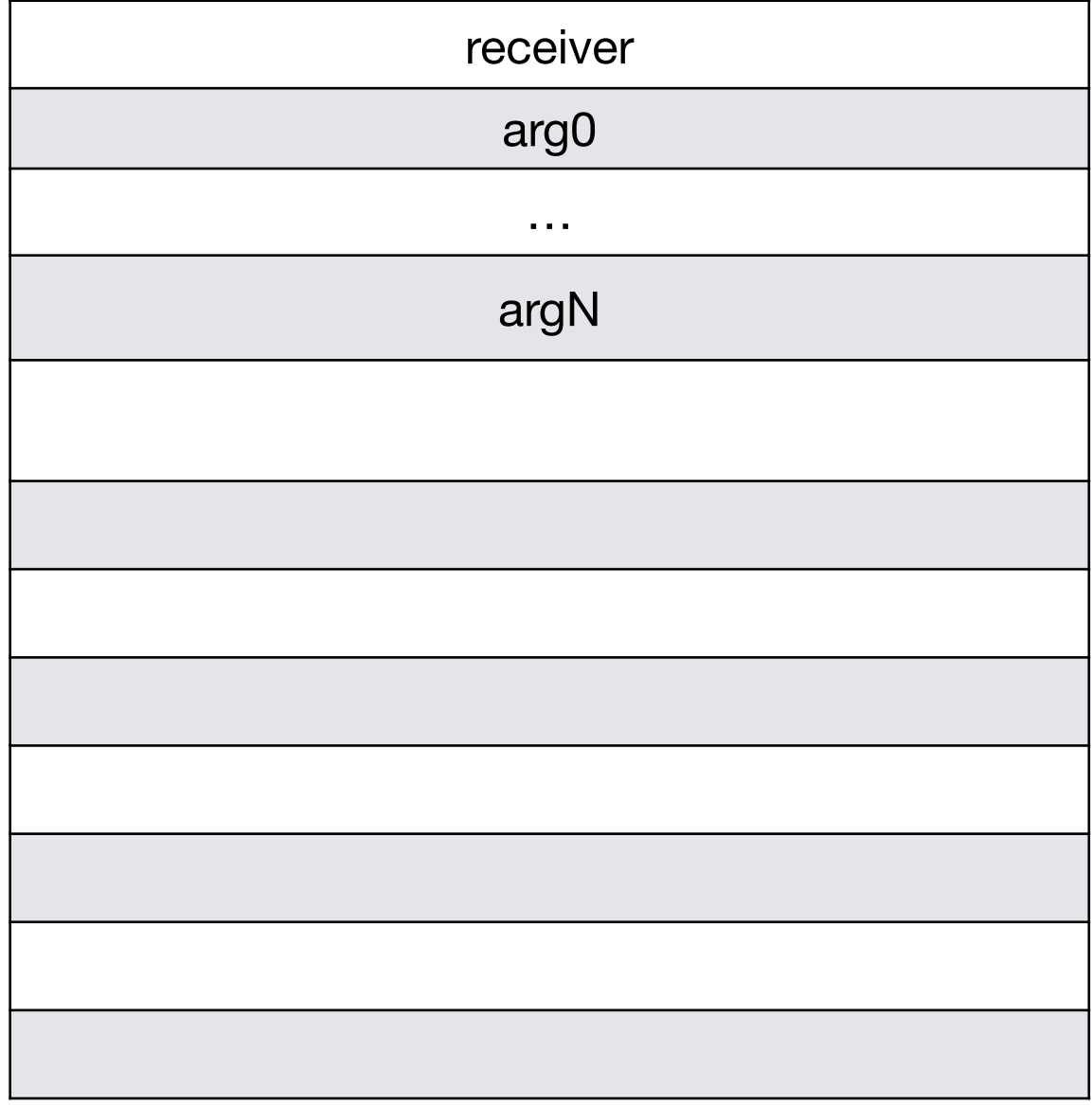
PushR ReceiverResultReg
PopR ReceiverResultReg
MoveRR FPCReg SPReg
PopR FPCReg
PopR LinkReg
RetN 8



Frame Pop &
Return

Frame Building

Machine Code Frame Structure



Pushed by Caller

```

Label 0:
MoveCqR 0 ReceiverResultReg
PushR LinkReg
Call 16777688/10001D8
AlignmentNops 8
AndCqRR 7 ReceiverResultReg TempReg
JumpNonZero Label 2
MoveMwrR 0 ReceiverResultReg TempReg
AndCqR 4194303/3FFFFFF TempReg
Nop
Nop
Nop
  
```

```

Label 2:
CmpRR ClassReg TempReg
JumpNonZero Label 0
PushR LinkReg
PushR FPreG
MoveRR SPReg FPreG
PushCw CogMethodAddress
MoveCqR (#bitInvert oop) SendNumArgsReg
PushR SendNumArgsReg
PushR ReceiverResultReg
PushCq 337/151
MoveMwrR 0 SPReg ReceiverResultReg
MovePatchableC32R 1 ClassReg
Call SendMessageRoutine
PushR ReceiverResultReg
PopR ReceiverResultReg
MoveRR FPreG SPReg
PopR FPreG
PopR LinkReg
RetN 8
  
```

Frame Creation

Frame Building

Machine Code Frame Structure



receiver
arg0
...
argN
caller InstructionPointer

Where to return

```
Label 0:
MoveCqR 0 ReceiverResultReg
PushR LinkReg
Call 16777688/10001D8
AlignmentNops 8
Label 1:
AndCqRR 7 ReceiverResultReg TempReg
JumpNonZero Label 2
MoveMwrR 0 ReceiverResultReg TempReg
AndCqR 4194303/3FFFFFF TempReg
Nop
Nop
Nop
```

```
Label 2:
CmpRR ClassReg TempReg
JumpNonZero Label 0
PushR LinkReg
PushR FPReg
MoveRR SPReg FPReg
PushCw CogMethodAddress
MoveCqR (#bitInvert oop) SendNumArgsReg
PushR SendNumArgsReg
PushR ReceiverResultReg
PushCq 337/151
MoveMwrR 0 SPReg ReceiverResultReg
MovePatchearableC32R 1 ClassReg
Call SendMessageRoutine
PushR ReceiverResultReg
PopR ReceiverResultReg
MoveRR FPReg SPReg
PopR FPReg
PopR LinkReg
RetN 8
```

PushR LinkReg
 PushR FPReg
 MoveRR SPReg FPReg
 PushCw CogMethodAddress
 MoveCqR (#bitInvert oop) SendNumArgsReg
 PushR SendNumArgsReg
 PushR ReceiverResultReg

Frame Creation

Frame Building

Machine Code Frame Structure



receiver
arg0
...
argN
caller InstructionPointer
caller FramePointer

We keep a pointer to previous frame

```
Label 0:
MoveCqR 0 ReceiverResultReg
PushR LinkReg
Call 16777688/10001D8
AlignmentNops 8
Label 1:
AndCqRR 7 ReceiverResultReg TempReg
JumpNonZero Label 2
MoveMwrR 0 ReceiverResultReg TempReg
AndCqR 4194303/3FFFFFF TempReg
Nop
Nop
Nop
```

```
Label 2:
CmpRR ClassReg TempReg
JumpNonZero Label 0
```

```
PushR LinkReg
PushR FPReg
MoveRR SPReg FPReg
PushCw CogMethodAddress
MoveCqR (#bitInvert oop) SendNumArgsReg
PushR SendNumArgsReg
PushR ReceiverResultReg
```

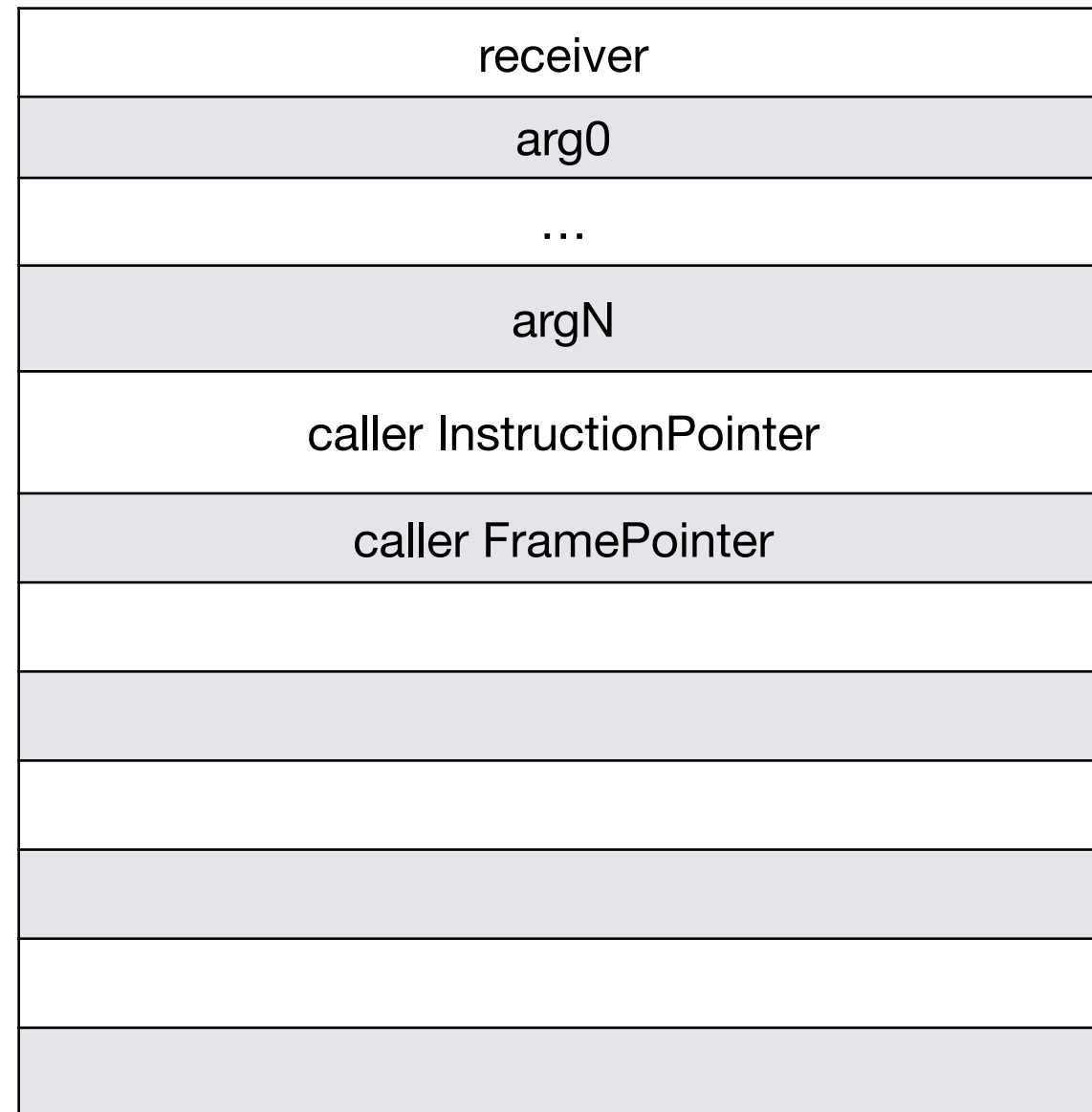
Frame Creation

```
PushCq 337/151
MoveMwrR 0 SPReg ReceiverResultReg
MovePatchableC32R 1 ClassReg
Call SendMessageRoutine
```

```
PushR ReceiverResultReg
PopR ReceiverResultReg
MoveRR FPReg SPReg
PopR FPReg
PopR LinkReg
RetN 8
```

Frame Building

Machine Code Frame Structure



Change the register to our frame

← Frame Pointer Register

```

Label 0:
MoveCqR 0 ReceiverResultReg
PushR LinkReg
Call 16777688/10001D8
AlignmentNops 8
Label 1:
AndCqRR 7 ReceiverResultReg TempReg
JumpNonZero Label 2
MoveMwrR 0 ReceiverResultReg TempReg
AndCqR 4194303/3FFFFFF TempReg
Nop
Nop
Nop
  
```

```

Label 2:
CmpRR ClassReg TempReg
JumpNonZero Label 0
  
```

```

PushR LinkReg
PushR FPReg
MoveRR SPReg FPReg
PushCw CogMethodAddress
MoveCqR (#bitInvert oop) SendNumArgsReg
PushR SendNumArgsReg
PushR ReceiverResultReg
  
```

Frame Creation

```

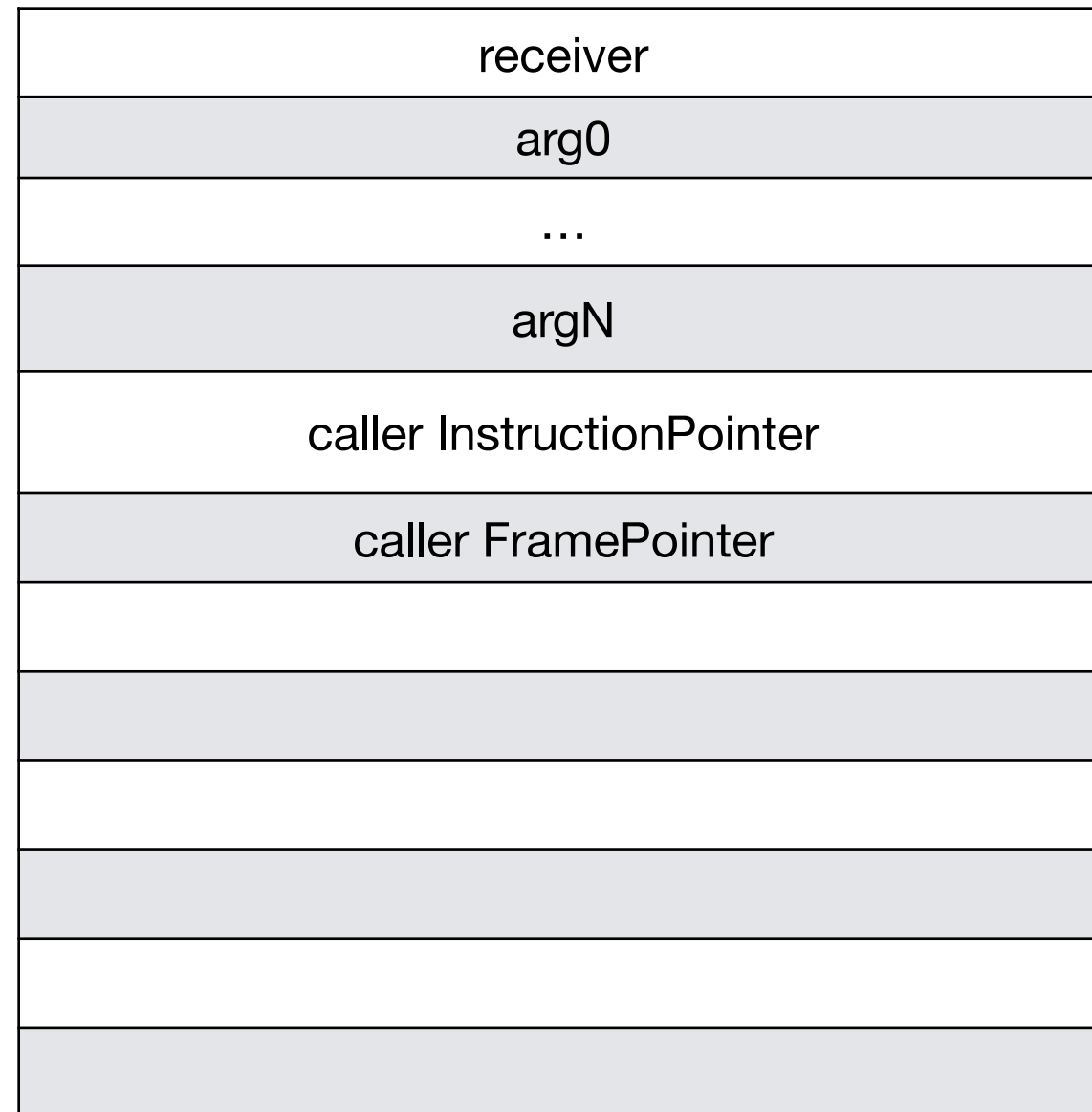
PushCq 337/151
MoveMwrR 0 SPReg ReceiverResultReg
MovePatchableC32R 1 ClassReg
Call SendMessageRoutine
  
```

```

PushR ReceiverResultReg
PopR ReceiverResultReg
MoveRR FPReg SPReg
PopR FPReg
PopR LinkReg
RetN 8
  
```

Frame Building

Machine Code Frame Structure



← Frame Pointer Register

```

Label 0:
MoveCqR 0 ReceiverResultReg
PushR LinkReg
Call 16777688/10001D8
AlignmentNops 8
Label 1:
AndCqRR 7 ReceiverResultReg TempReg
JumpNonZero Label 2
MoveMwrR 0 ReceiverResultReg TempReg
AndCqR 4194303/3FFFFFF TempReg
Nop
Nop
Nop
  
```

```

Label 2:
CmpRR ClassReg TempReg
JumpNonZero Label 0
  
```

```

PushR LinkReg
PushR FPReg
MoveRR SPReg FPReg
PushCw CogMethodAddress
MoveCqR (#bitInvert oop) SendNumArgsReg
PushR SendNumArgsReg
PushR ReceiverResultReg
  
```

Frame Creation

```

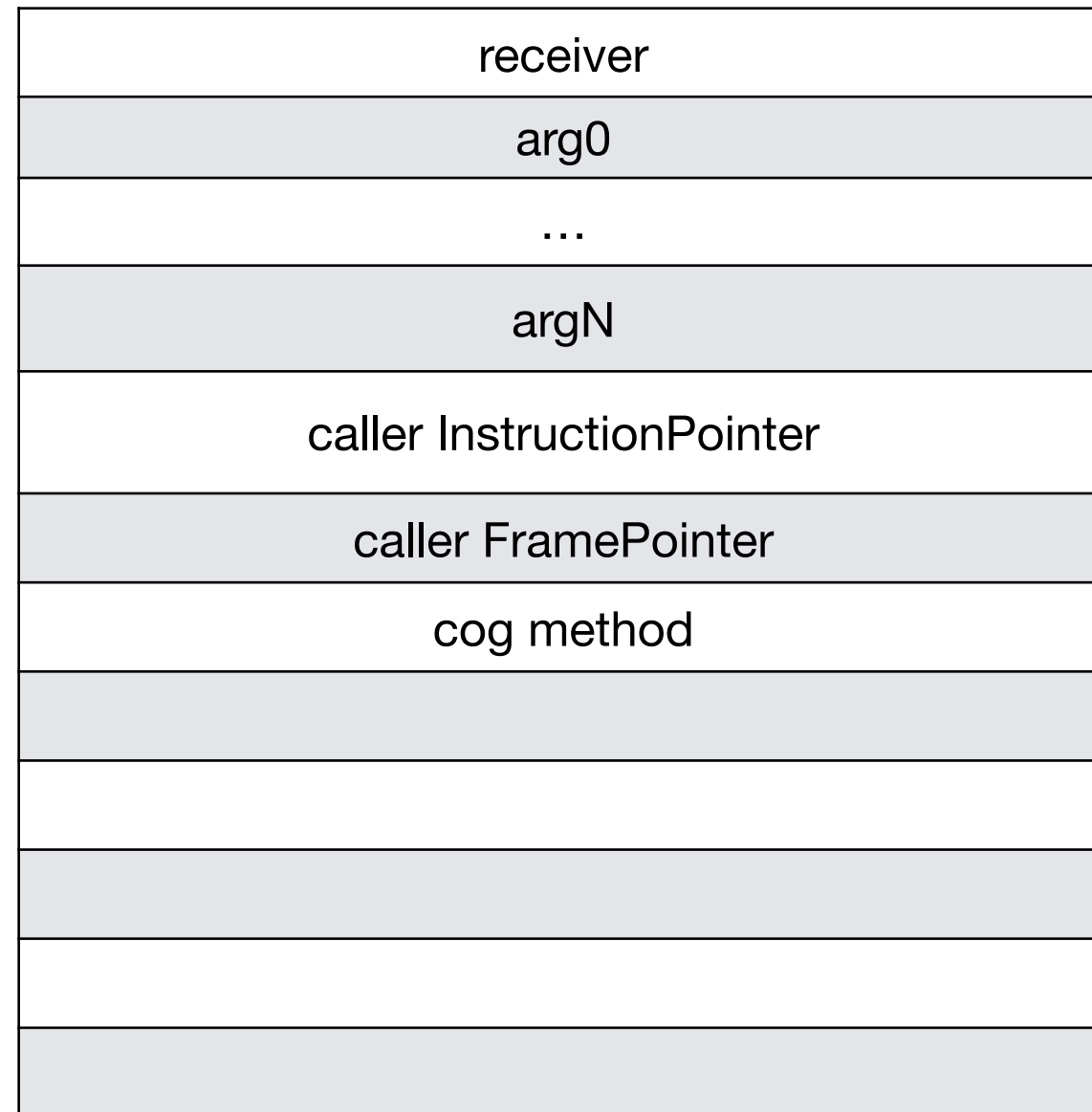
PushCq 337/151
MoveMwrR 0 SPReg ReceiverResultReg
MovePatchableC32R 1 ClassReg
Call SendMessageRoutine
  
```

```

PushR ReceiverResultReg
PopR ReceiverResultReg
MoveRR FPReg SPReg
PopR FPReg
PopR LinkReg
RetN 8
  
```


Frame Building

Machine Code Frame Structure



← Frame Pointer Register

This method

```

Label 0:
MoveCqR 0 ReceiverResultReg
PushR LinkReg
Call 16777688/10001D8
AlignmentNops 8

Label 1:
AndCqRR 7 ReceiverResultReg TempReg
JumpNonZero Label 2
MoveMwrR 0 ReceiverResultReg TempReg
AndCqR 4194303/3FFFFFF TempReg
Nop
Nop
Nop
  
```

```

Label 2:
CmpRR ClassReg TempReg
JumpNonZero Label 0

PushR LinkReg
PushR FPReg
MoveRR SPReg FPReg
PushCw CogMethodAddress
MoveCqR (#bitInvert oop) SendNumArgsReg
PushR SendNumArgsReg
PushR ReceiverResultReg

PushCq 337/151
MoveMwrR 0 SPReg ReceiverResultReg
MovePatchableC32R 1 ClassReg
Call SendMessageRoutine

PushR ReceiverResultReg
PopR ReceiverResultReg
MoveRR FPReg SPReg
PopR FPReg
PopR LinkReg
RetN 8
  
```

Frame Building

Machine Code Frame Structure



receiver
arg0
...
argN
caller InstructionPointer
caller FramePointer
cog method
context (initially nil)

← Frame Pointer Register

Context oop.

```

Label 0:
MoveCqR 0 ReceiverResultReg
PushR LinkReg
Call 16777688/10001D8
AlignmentNops 8
Label 1:
AndCqRR 7 ReceiverResultReg TempReg
JumpNonZero Label 2
MoveMwrR 0 ReceiverResultReg TempReg
AndCqR 4194303/3FFFFFF TempReg
Nop
Nop
Nop
  
```

```

Label 2:
CmpRR ClassReg TempReg
JumpNonZero Label 0
  
```

```

PushR LinkReg
PushR FPReg
MoveRR SPReg FPReg
PushCw CogMethodAddress
MoveCqR (#bitInvert oop) SendNumArgsReg
PushR SendNumArgsReg
PushR ReceiverResultReg
  
```

```

PushCq 337/151
MoveMwrR 0 SPReg ReceiverResultReg
MovePatchableC32R 1 ClassReg
Call SendMessageRoutine
  
```

```

PushR ReceiverResultReg
PopR ReceiverResultReg
MoveRR FPReg SPReg
PopR FPReg
PopR LinkReg
RetN 8
  
```

Frame Building

Machine Code Frame Structure



receiver
arg0
...
argN
caller InstructionPointer
caller FramePointer
cog method
context (initially nil)
receiver

← Frame Pointer Register

The receiver

```

Label 0:
MoveCqR 0 ReceiverResultReg
PushR LinkReg
Call 16777688/10001D8
AlignmentNops 8
Label 1:
AndCqRR 7 ReceiverResultReg TempReg
JumpNonZero Label 2
MoveMwrR 0 ReceiverResultReg TempReg
AndCqR 4194303/3FFFFFF TempReg
Nop
Nop
Nop
  
```

```

Label 2:
CmpRR ClassReg TempReg
JumpNonZero Label 0
  
```

```

PushR LinkReg
PushR FPReg
MoveRR SPReg FPReg
PushCw CogMethodAddress
MoveCqR (#bitInvert oop) SendNumArgsReg
PushR SendNumArgsReg
PushR ReceiverResultReg
  
```

```

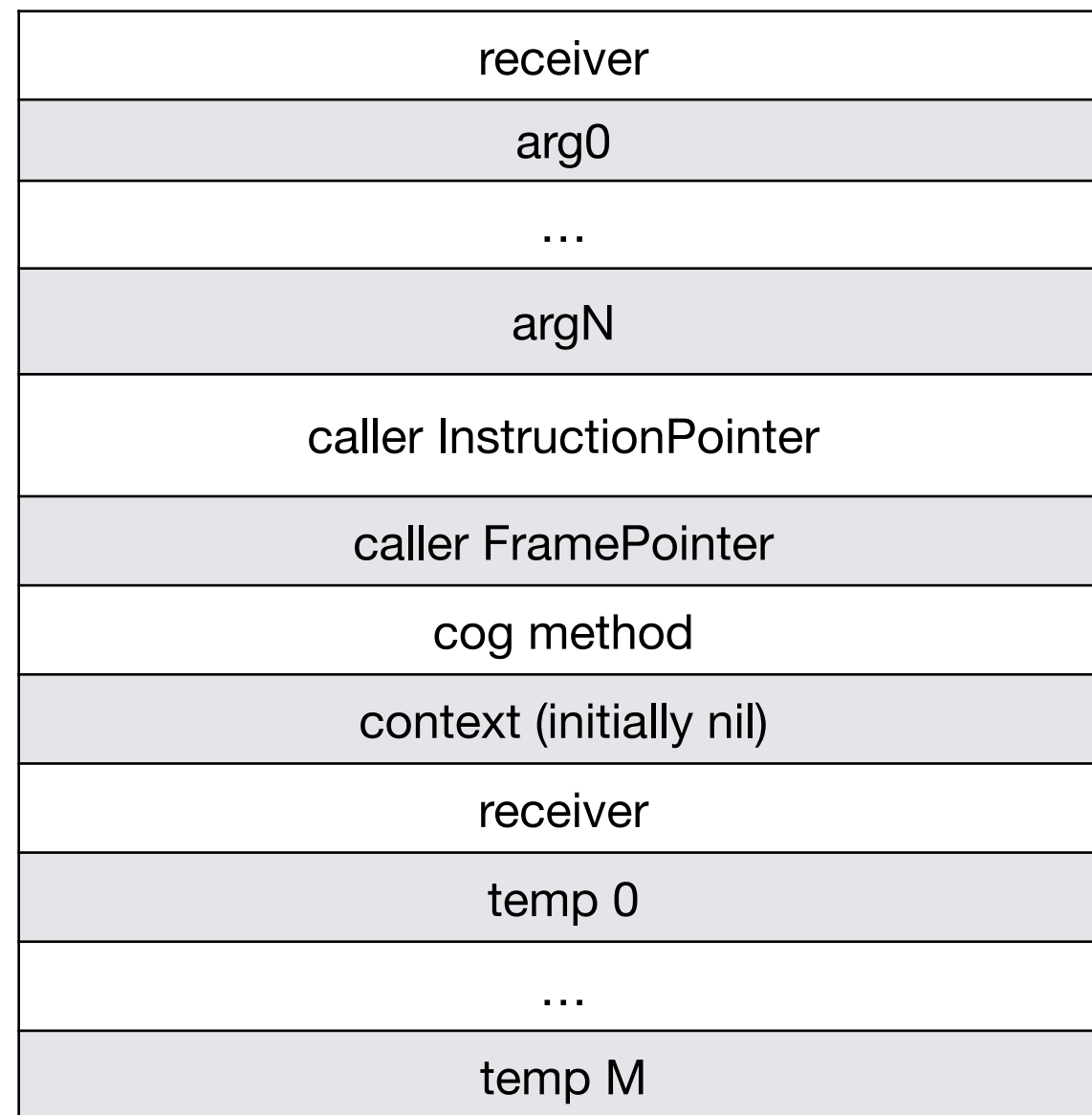
PushCq 337/151
MoveMwrR 0 SPReg ReceiverResultReg
MovePatchableC32R 1 ClassReg
Call SendMessageRoutine
  
```

```

PushR ReceiverResultReg
PopR ReceiverResultReg
MoveRR FPReg SPReg
PopR FPReg
PopR LinkReg
RetN 8
  
```

Frame Building

Machine Code Frame Structure



← Frame Pointer Register

Temps if used

```

Label 0:
MoveCqR 0 ReceiverResultReg
PushR LinkReg
Call 16777688/10001D8
AlignmentNops 8
AndCqRR 7 ReceiverResultReg TempReg
JumpNonZero Label 2
MoveMwrR 0 ReceiverResultReg TempReg
AndCqR 4194303/3FFFFFF TempReg
Nop
Nop
Nop
  
```

```

Label 2:
CmpRR ClassReg TempReg
JumpNonZero Label 0
  
```

```

PushR LinkReg
PushR FPReg
MoveRR SPReg FPReg
PushCw CogMethodAddress
MoveCqR (#bitInvert oop) SendNumArgsReg
PushR SendNumArgsReg
PushR ReceiverResultReg
  
```

```

PushCq 337/151
MoveMwrR 0 SPReg ReceiverResultReg
MovePatchableC32R 1 ClassReg
Call SendMessageRoutine
  
```

```

PushR ReceiverResultReg
PopR ReceiverResultReg
MoveRR FPReg SPReg
PopR FPReg
PopR LinkReg
RetN 8
  
```

Frame Building

Machine Code Frame Structure



receiver
arg0
...
argN
caller InstructionPointer
caller FramePointer
cog method
context (initially nil)
receiver
temp 0
...
temp M

← Frame Pointer Register

```
Label 0:
MoveCqR 0 ReceiverResultReg
PushR LinkReg
Call 16777688/10001D8
AlignmentNops 8
Label 1:
AndCqRR 7 ReceiverResultReg TempReg
JumpNonZero Label 2
MoveMwrR 0 ReceiverResultReg TempReg
AndCqR 4194303/3FFFFFF TempReg
Nop
Nop
Nop
```

```
Label 2:
CmpRR ClassReg TempReg
JumpNonZero Label 0
```

```
PushR LinkReg
PushR FPReg
MoveRR SPReg FPReg
PushCw CogMethodAddress
MoveCqR (#bitInvert oop) SendNumArgsReg
PushR SendNumArgsReg
PushR ReceiverResultReg
```

Frame Creation

```
PushCq 337/151
MoveMwrR 0 SPReg ReceiverResultReg
MovePatchableC32R 1 ClassReg
Call SendMessageRoutine
```

```
PushR ReceiverResultReg
PopR ReceiverResultReg
MoveRR FPReg SPReg
PopR FPReg
PopR LinkReg
RetN 8
```

Attention, it is not the same in the interpreter

Frame Building

Machine Code Frame Structure



receiver
arg0
...
argN
caller InstructionPointer
caller FramePointer
cog method
context (initially nil)
receiver
temp 0
...
temp M

← Frame Pointer Register

```

Label 0:
MoveCqR 0 ReceiverResultReg
PushR LinkReg
Call 16777688/10001D8
AlignmentNops 8
Label 1:
AndCqRR 7 ReceiverResultReg TempReg
JumpNonZero Label 2
MoveMwrR 0 ReceiverResultReg TempReg
AndCqR 4194303/3FFFFFF TempReg
Nop
Nop
Nop
  
```

```

Label 2:
CmpRR ClassReg TempReg
JumpNonZero Label 0
  
```

```

PushR LinkReg
PushR FPReg
MoveRR SPReg FPReg
PushCw CogMethodAddress
MoveCqR (#bitInvert oop) SendNumArgsReg
PushR SendNumArgsReg
PushR ReceiverResultReg
  
```

Frame Creation

```

PushCq 337/151
MoveMwrR 0 SPReg ReceiverResultReg
MovePatchableC32R 1 ClassReg
Call SendMessageRoutine
  
```

```

PushR ReceiverResultReg
PopR ReceiverResultReg
MoveRR FPReg SPReg
PopR FPReg
PopR LinkReg
RetN 8
  
```

Attention, it is not the same in the interpreter

Frame Pop

Machine Code Frame Structure



receiver
arg0
...
argN
caller InstructionPointer
caller FramePointer
cog method
context (initially nil)
receiver
temp 0
...
temp M

← Frame Pointer Register

```

Label 0:
MoveCqR 0 ReceiverResultReg
PushR LinkReg
Call 16777688/10001D8
AlignmentNops 8

Label 1:
AndCqRR 7 ReceiverResultReg TempReg
JumpNonZero Label 2
MoveMwrR 0 ReceiverResultReg TempReg
AndCqR 4194303/3FFFFFF TempReg
Nop
Nop
Nop
  
```

```

Label 2:
CmpRR ClassReg TempReg
JumpNonZero Label 0

PushR LinkReg
PushR FPRreg
MoveRR SPReg FPRreg
PushCw CogMethodAddress
MoveCqR (#bitInvert oop) SendNumArgsReg
PushR SendNumArgsReg
PushR ReceiverResultReg

PushCq 337/151
MoveMwrR 0 SPReg ReceiverResultReg
MovePatchableC32R 1 ClassReg
Call SendMessageRoutine
  
```

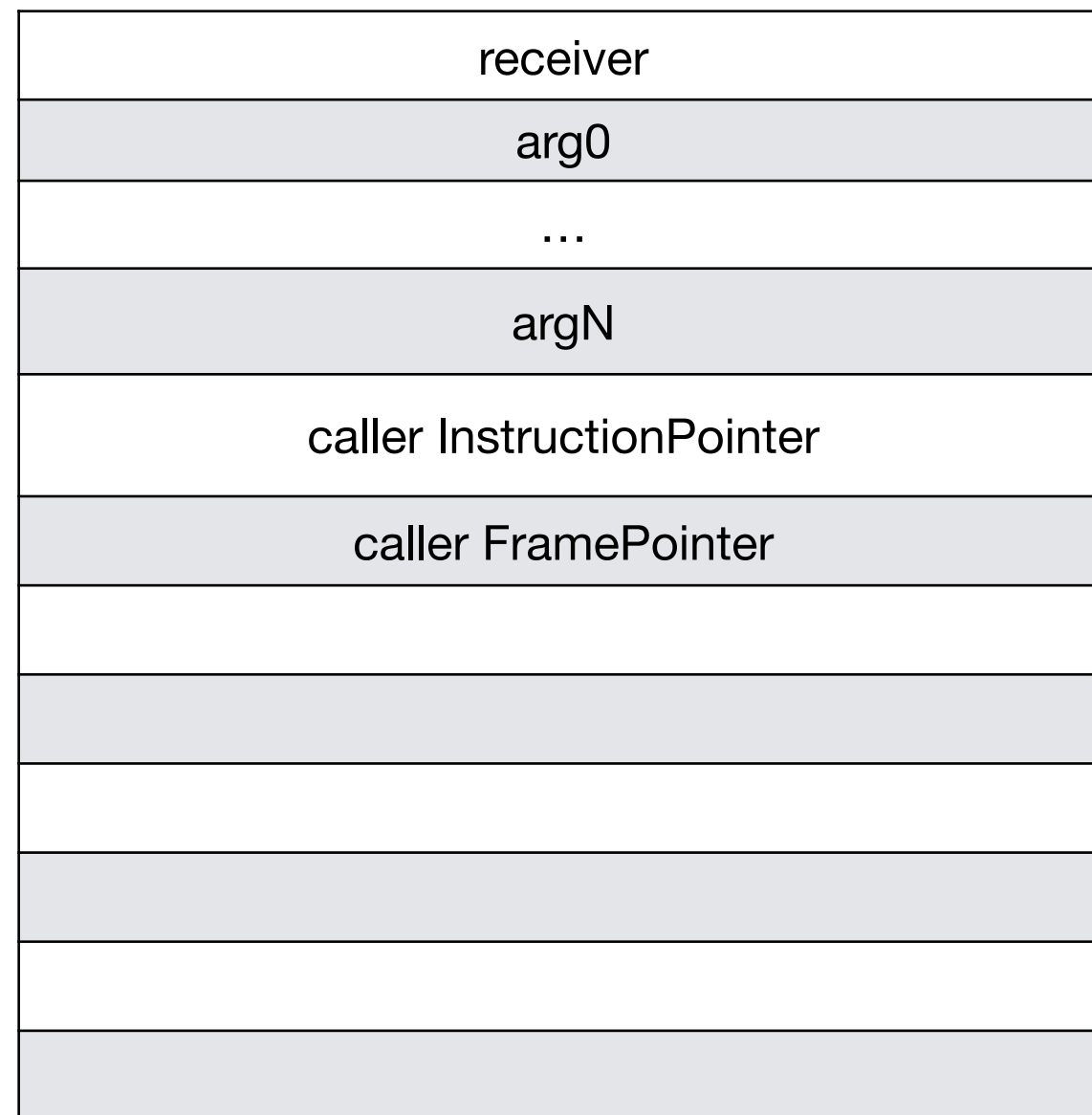
```

PushR ReceiverResultReg
PopR ReceiverResultReg
MoveRR FPRreg SPReg
PopR FPRreg
PopR LinkReg
RetN 8
  
```

Frame Pop & Return

Frame Pop

Machine Code Frame Structure



← Frame Pointer Register

```

Label 0:
MoveCqR 0 ReceiverResultReg
PushR LinkReg
Call 16777688/10001D8
AlignmentNops 8

Label 1:
AndCqRR 7 ReceiverResultReg TempReg
JumpNonZero Label 2
MoveMwrR 0 ReceiverResultReg TempReg
AndCqR 4194303/3FFFFFF TempReg
Nop
Nop
Nop
  
```

```

Label 2:
CmpRR ClassReg TempReg
JumpNonZero Label 0

PushR LinkReg
PushR FPReg
MoveRR SPReg FPReg
PushCw CogMethodAddress
MoveCqR (#bitInvert oop) SendNumArgsReg
PushR SendNumArgsReg
PushR ReceiverResultReg

PushCq 337/151
MoveMwrR 0 SPReg ReceiverResultReg
MovePatchableC32R 1 ClassReg
Call SendMessageRoutine
  
```

```

PushR ReceiverResultReg
PopR ReceiverResultReg
MoveRR FPReg SPReg
PopR FPReg
PopR LinkReg
RetN 8
  
```

Frame pop & Return

Frame Pop

Machine Code Frame Structure



receiver
arg0
...
argN

```
Label 0:  
MoveCqR 0 ReceiverResultReg  
PushR LinkReg  
Call 16777688/10001D8  
AlignmentNops 8  
  
AndCqRR 7 ReceiverResultReg TempReg  
JumpNonZero Label 2  
MoveMwrR 0 ReceiverResultReg TempReg  
AndCqR 4194303/3FFFFFF TempReg  
Nop  
Nop  
Nop
```

```
Label 2:  
CmpRR ClassReg TempReg  
JumpNonZero Label 0  
  
PushR LinkReg  
PushR FPREg  
MoveRR SPReg FPREg  
PushCw CogMethodAddress  
MoveCqR (#bitInvert oop) SendNumArgsReg  
PushR SendNumArgsReg  
PushR ReceiverResultReg  
  
PushCq 337/151  
MoveMwrR 0 SPReg ReceiverResultReg  
MovePatchableC32R 1 ClassReg  
Call SendMessageRoutine
```

```
PushR ReceiverResultReg  
PopR ReceiverResultReg  
MoveRR FPREg SPReg  
PopR FPREg  
PopR LinkReg  
RetN 8
```

Frame pop & Return

Frame Pop

Machine Code Frame Structure



```

Label 0:
MoveCqR 0 ReceiverResultReg
PushR LinkReg
Call 16777688/10001D8
AlignmentNops 8

AndCqRR 7 ReceiverResultReg TempReg
JumpNonZero Label 2
MoveMwrR 0 ReceiverResultReg TempReg
AndCqR 4194303/3FFFFFF TempReg
Nop
Nop
Nop
  
```

```

Label 2:
CmpRR ClassReg TempReg
JumpNonZero Label 0

PushR LinkReg
PushR FPReg
MoveRR SPReg FPReg
PushCw CogMethodAddress
MoveCqR (#bitInvert oop) SendNumArgsReg
PushR SendNumArgsReg
PushR ReceiverResultReg

PushCq 337/151
MoveMwrR 0 SPReg ReceiverResultReg
MovePatchableC32R 1 ClassReg
Call SendMessageRoutine
  
```

```

PushR ReceiverResultReg
PopR ReceiverResultReg
MoveRR FPReg SPReg
PopR FPReg
PopR LinkReg
RetN 8
  
```

Frame Pop & Return



Primitives

Basics

- A primitive is a special kind of method
- They have:
 - an implementation in C or Native Code
 - fallback code (in case the previous implementation) fails
- We need the same behaviour in a Cog Method



Primitives

Many Kinds

- Depending in the native part implementation
 - A Function written in C (interpreter / plugin)
 - A Complete Machine Code implementation
 - A partial Machine Code + delegating in C Function

Primitives

Only C Function

Method
Body

Type Guard

Call to C
Function

Frame Build

Fallback
Code

Frame Pop

Bytecode
Map

Primitives

Only C Function

Method
Body

Type Guard

Call to C
Function

Frame Build

Fallback
Code

Frame Pop

Bytecode
Map

If the primitive is
successful,
return.

Primitives

Only C Function

Method
Body

Type Guard

Call to C
Function

Frame Build

Fallback
Code

Frame Pop

Bytecode
Map

If the primitive
fails it continue
executing the
rest of the
method

Primitives

Only Machine Code Implementation

Method
Body

Type Guard

Primitive
Machine
Code
Implementation

Frame Build

Fallback
Code

Frame Pop

Bytecode
Map

Primitives

Only Machine Code Implementation

Method
Body

Type Guard

Primitive
Machine
Code
Implementation

Frame Build

Fallback
Code

Frame Pop

Bytecode
Map

If the primitive is
successful,
return.

Primitives

Only Machine Code Implementation

Method
Body

Type Guard

Primitive
Machine
Code
Implementation

Frame Build

Fallback
Code

Frame Pop

Bytecode
Map

If the primitive
fails it continue
executing the
rest of the
method

Primitives

Only Machine Code Implementation

Method
Body

Type Guard

Primitive
Machine
Code
Implementation

Frame Build

Fallback
Code

Frame Pop

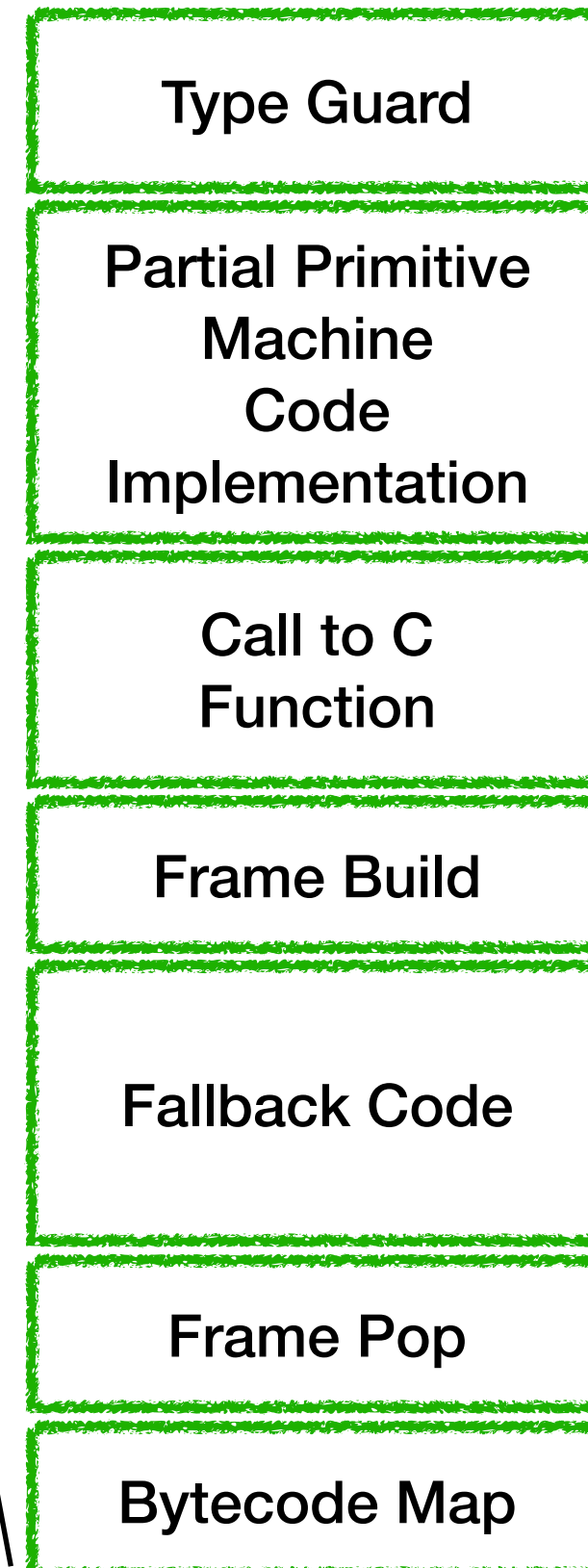
Bytecode
Map

They are called
“Complete Primitives”

Primitives

A partial Machine Code + delegating in C Function

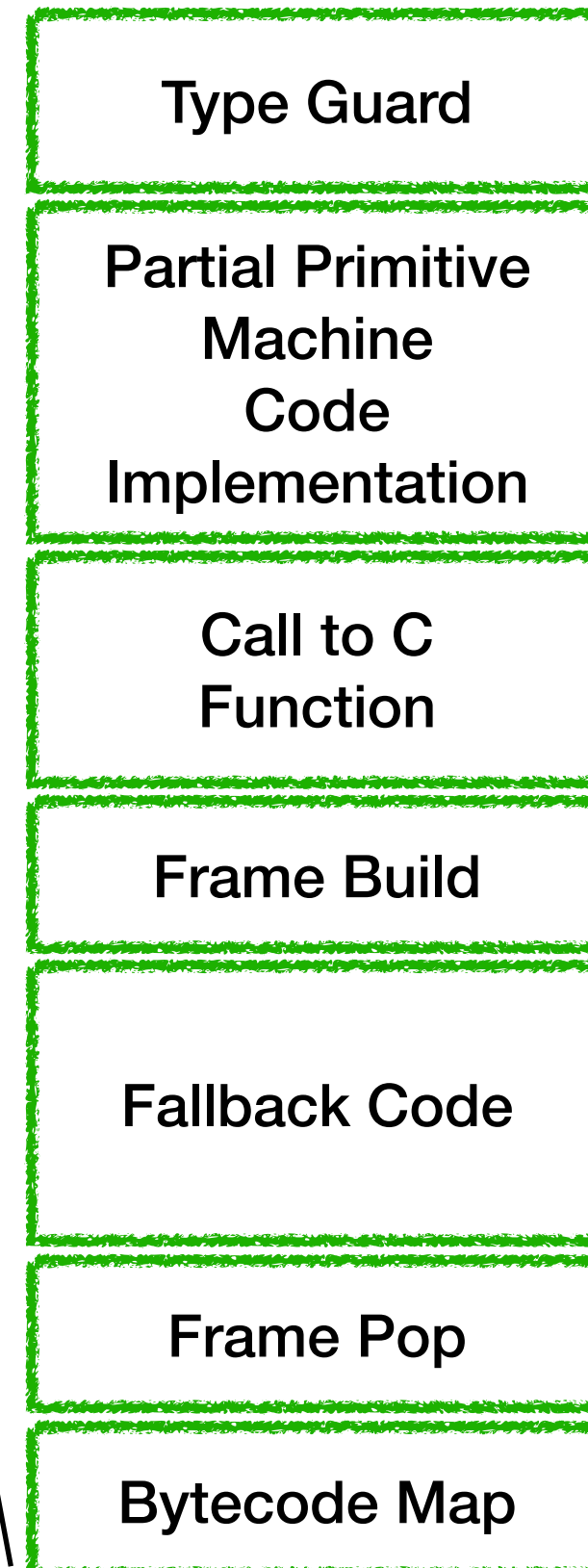
Method
Body



Primitives

A partial Machine Code + delegating in C Function

Method
Body

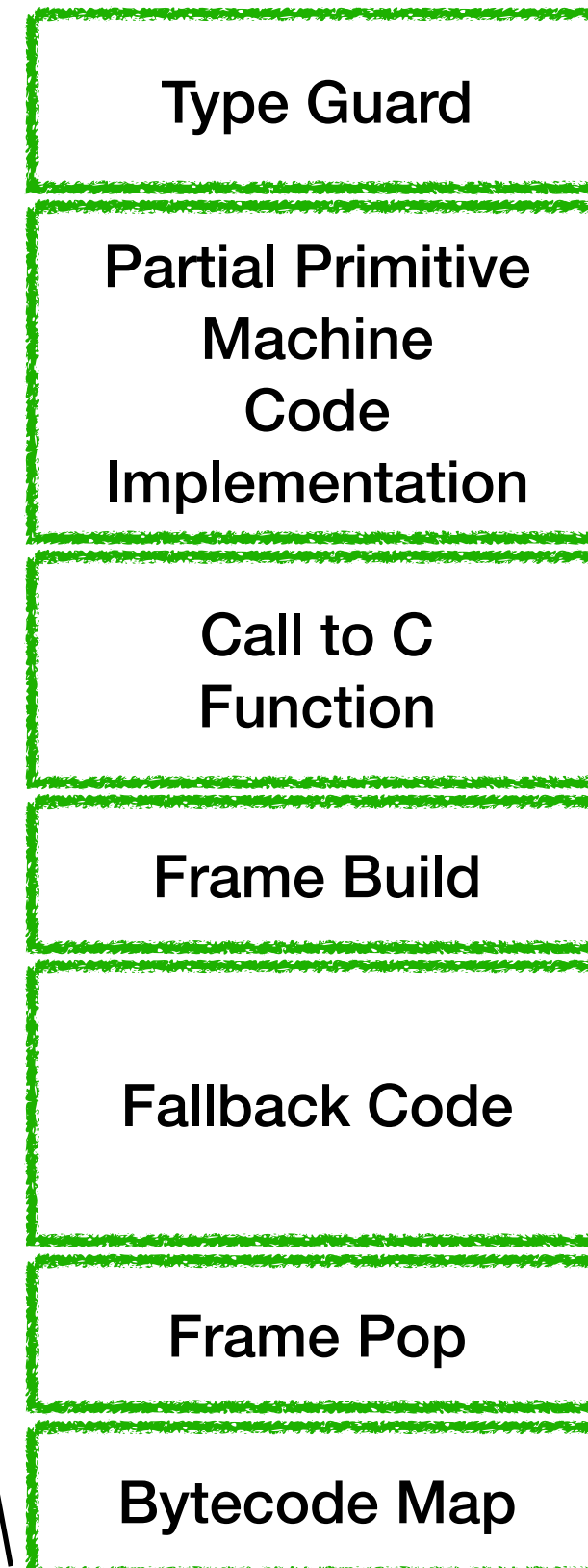


First, handle
some fast paths
in the MC. If
successful,
returns

Primitives

A partial Machine Code + delegating in C Function

Method
Body

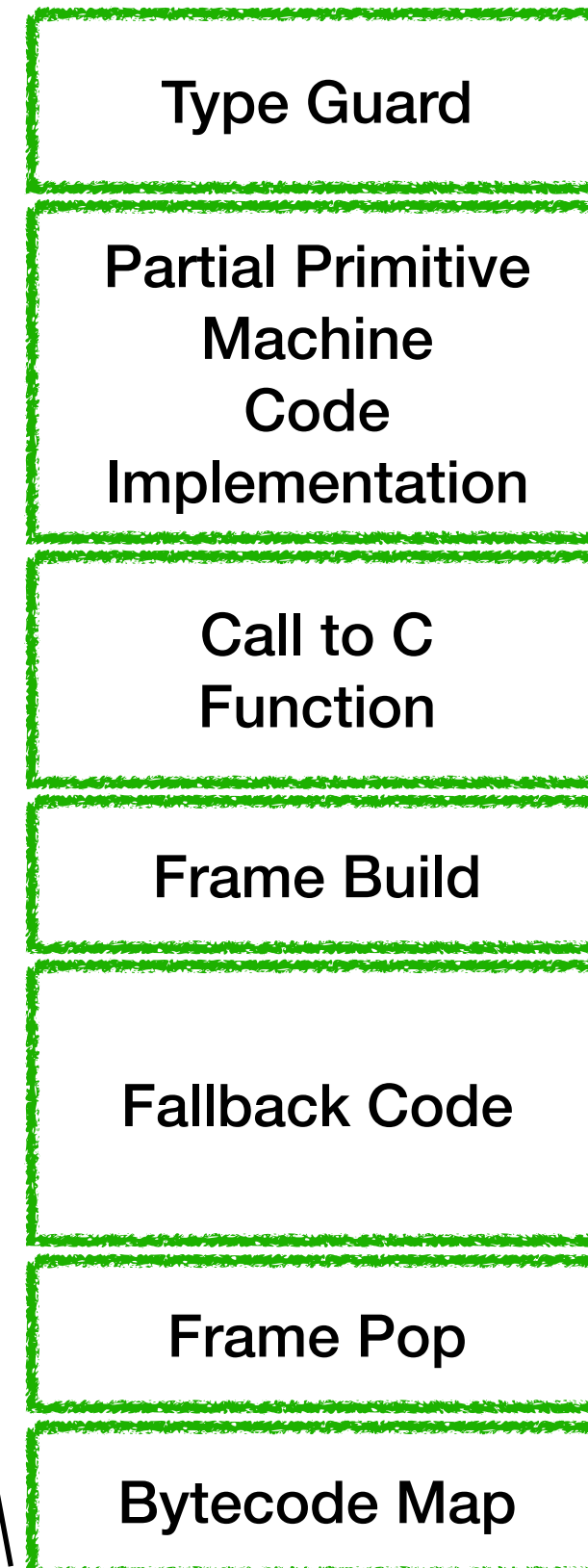


If the case is not covered as a “fast path”, it delegates to the C function

Primitives

A partial Machine Code + delegating in C Function

Method
Body

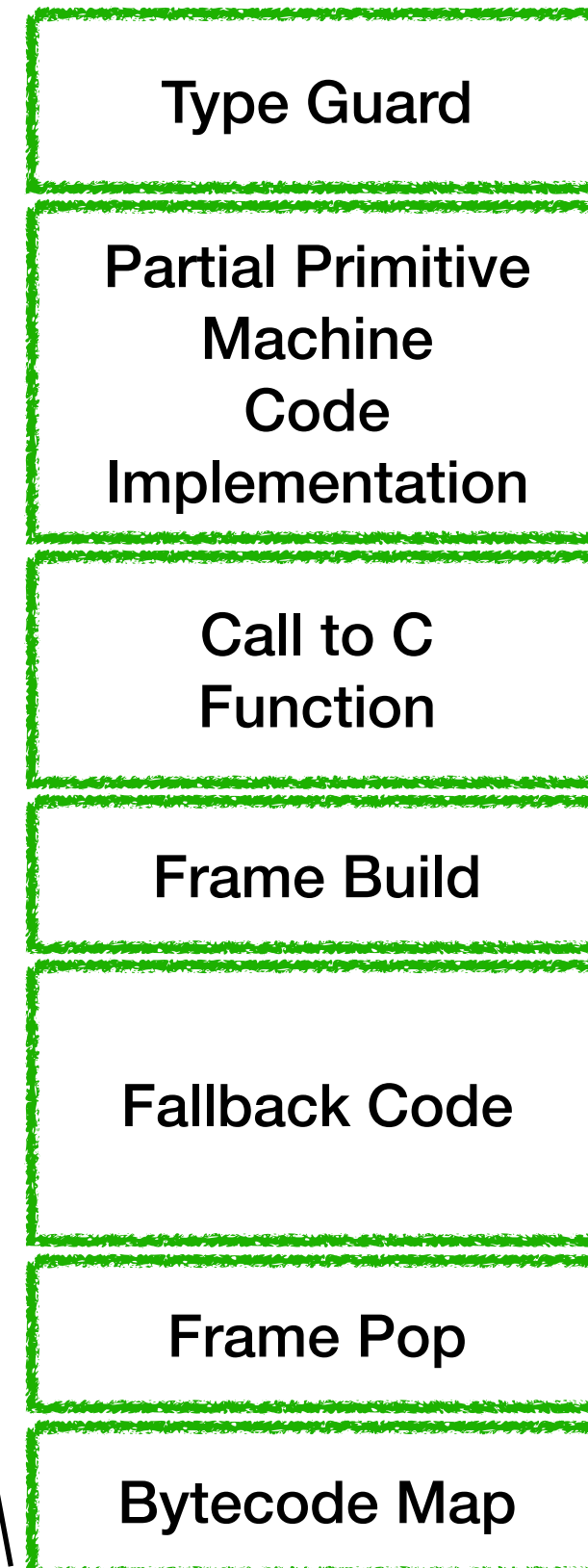


Executes the C
function and if
the primitive is
successful,
returns

Primitives

A partial Machine Code + delegating in C Function

Method
Body



If the C primitive also fails it continue executing the rest of the method



Cog Methods

Remaining Subjects

- Blocks (3 flavours)
- Message Sends & PICs
- Bytecode Mapping
- Smalltalk-Stack Primitives
-



Cog Methods

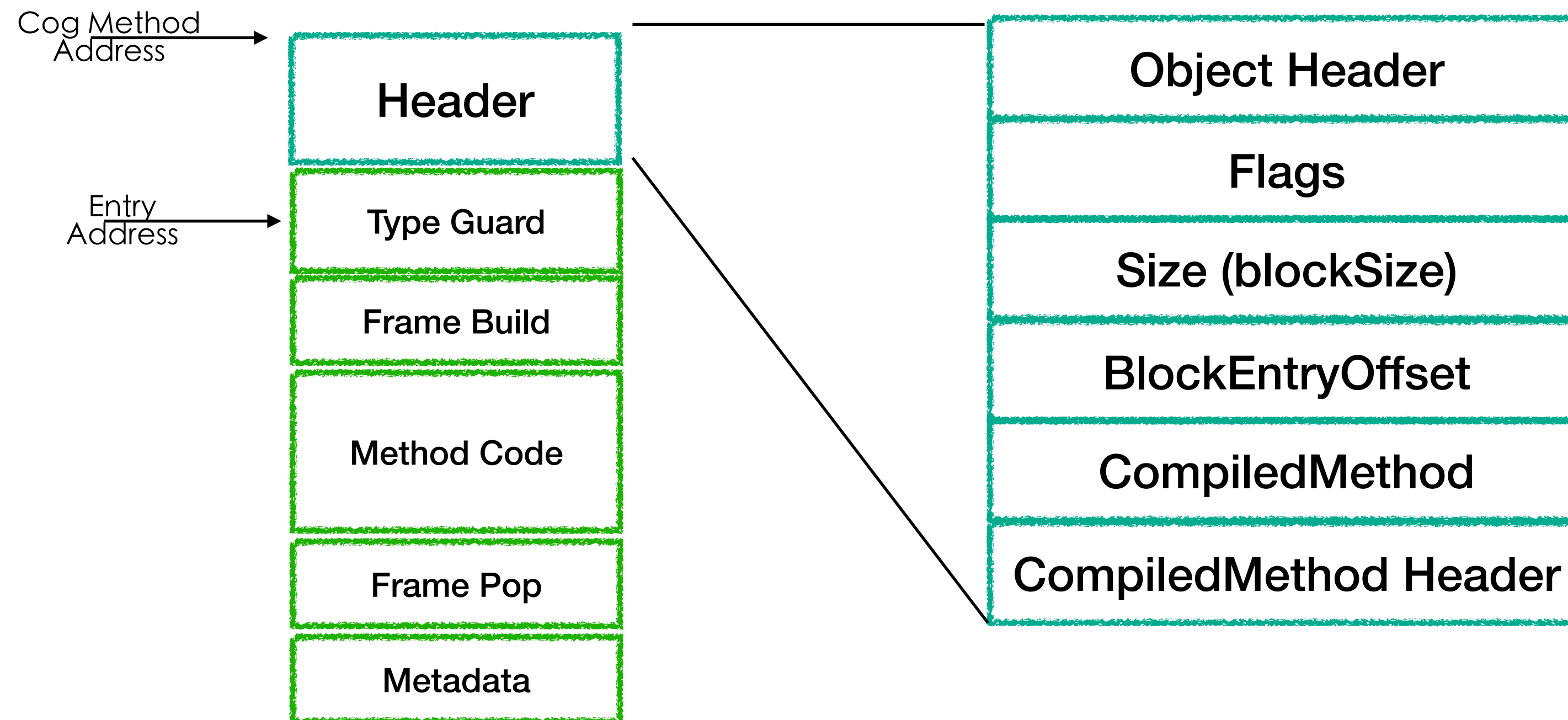
Remaining Subjects

- Blocks (3 flavours)
- Message Sends & PICs
- Bytecode Mapping
- Smalltalk-Stack Primitives
- Code Zone compaction
-

More to continue
learning

Structure of Machine Code Methods

An introduction



Thanks!!!!