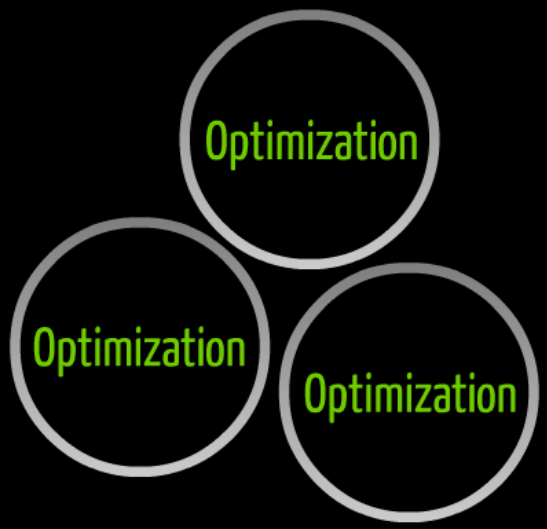
Bytecode Optimization

guillermo amaral - caesar systems

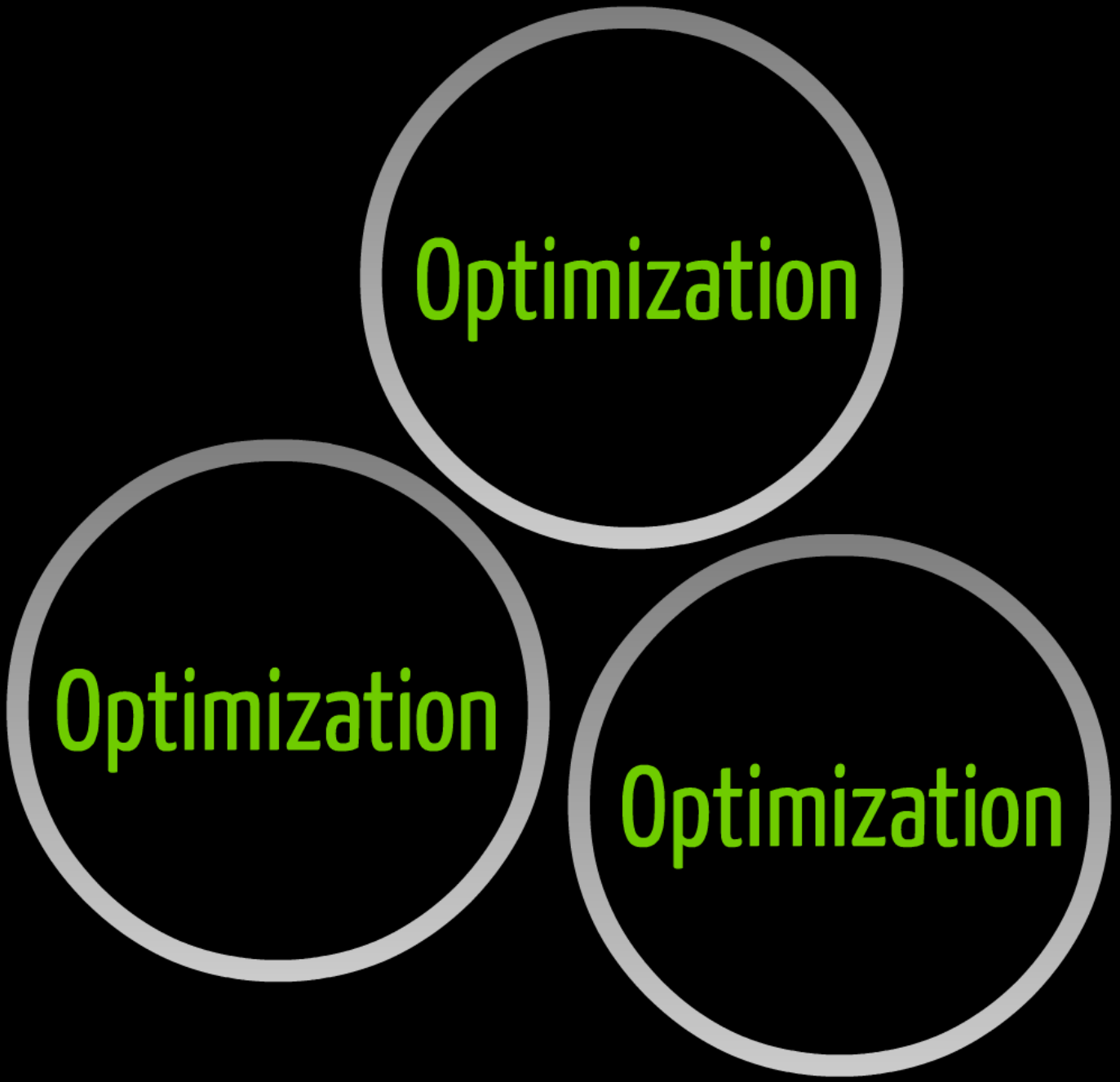
Bytecode Optimization

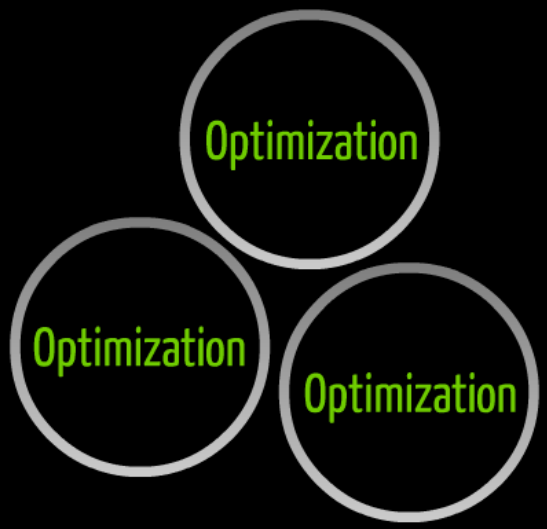
guillermo amaral - caesar systems



vs

Understand
&
Document





vs

Understand
&
Document

Smalltalk code

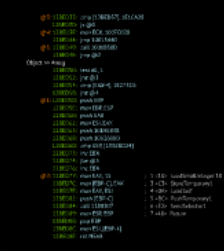
```
msg
    | t |
    t := 10.
    ^self msg: t
```

Bytecodes

#[18 14 C3 0A BC E2 48]

```
18  LoadSmallInteger 10
C3  StoreTemporary1 t
0A  LoadSelf
BC  PushTemporary1 t
E2  SendSelector1 #msg:
48  Return
```

Machine code



Smalltalk code

```
msg
    | t |
    t := 10.
    ^self msg: t
```

Bytecodes

#[18 14 C3 0A BC E2 48]

```
18  LoadSmallInteger 10
C3  StoreTemporary1 t
0A  LoadSelf
BC  PushTemporary1 t
E2  SendSelector1 #msg:
48  Return
```

Bytecodes

#[18 14 C3 0A BC E2 48]

```
18  LoadSmallInteger 10
C3  StoreTemporary1 t
0A  LoadSelf
BC  PushTemporary1 t
```

```
18  LoadSmallInteger 10
C3  StoreTemporary1 t
0A  LoadSelf
BC  PushTemporary1 t
E2  SendSelector1 #msg:
48  Return
```

Bytecodes

#[18 14 C3 0A BC E2 48]

```
18  LoadSmallInteger 10
C3  StoreTemporary1 t
0A  LoadSelf
BC  PushTemporary1 t
E2  SendSelector1 #msg:
48  Return
```

Machine code

```
    @3: 12BED33:  cmp [12BED57], 1E1CA28
        12BED3D:  jz @1
    @4: 12BED3F:  mov ECX, 100FC028
        12BED44:  jmp 10015440
    @5: 12BED49:  call 1001B50D
        12BED4E:  jmp @2
Object >> #msg
        12BED50:  test AL, 1
        12BED52:  jnz @3
        12BED54:  cmp [EAX-4], 1E273C0
        12BED5B:  jnz @4
    @1: 12BED5D:  push EBP
        12BED5E:  mov EBP, ESP
        12BED60:  push EAX
        12BED61:  mov ESI, EAX
        12BED63:  push 10E9E838
        12BED68:  push 10026060
        12BED6D:  cmp ESP, [10028CD4]
        12BED73:  inc EBX
        12BED74:  jbe @5
        12BED76:  inc EBX
    @2: 12BED77:  mov EAX, 15           ;  1 <18>  LoadSmallInteger 10
        12BED7C:  mov [EBP-C], EAX      ;  3 <C3>  StoreTemporary1
        12BED7F:  mov EAX, ESI          ;  4 <0A>  LoadSelf
        12BED81:  push [EBP-C]          ;  5 <BC>  PushTemporary1
        12BED84:  call 1280207          ;  6 <E2>  SendSelector1
        12BED89:  mov ESP, EBP          ;  7 <48>  Return
        12BED8B:  pop EBP
        12BED8C:  mov ESI, [EBP-4]
        12BED8F:  ret NEAR
```

```
        @3: 12BED33:  cmp [12BED57], 1E1CA28
            12BED3D:  jz @1
        @4: 12BED3F:  mov ECX, 100FC028
            12BED44:  jmp 10015440
        @5: 12BED49:  call 1001B50D
            12BED4E:  jmp @2
Object >> #msg
            12BED50:  test AL, 1
            12BED52:  jnz @3
            12BED54:  cmp [EAX-4], 1E273C0
            12BED5B:  jnz @4
        @1: 12BED5D:  push EBP
            12BED5E:  mov EBP, ESP
            12BED60:  push EAX
            12BED61:  mov ESI, EAX
            12BED63:  push 10E9E838
            12BED68:  push 10026060
            12BED6D:  cmp ESP, [10028CD4]
            12BED73:  inc EBX
            12BED74:  jbe @5
            12BED76:  inc EBX
        @2: 12BED77:  mov EAX, 15              ;  1 <18>  LoadSmallInteger 10
            12BED7C:  mov [EBP-C], EAX         ;  3 <C3>  StoreTemporary1
            12BED7F:  mov EAX, ESI             ;  4 <0A>  LoadSelf
            12BED81:  push [EBP-C]             ;  5 <BC>  PushTemporary1
            12BED84:  call 1280207             ;  6 <E2>  SendSelector1
            12BED89:  mov ESP, EBP             ;  7 <48>  Return
            12BED8B:  pop EBP
            12BED8C:  mov ESI, [EBP-4]
            12BED8F:  ret NEAR
```

```
      12BED63:  push 10E9E838
      12BED68:  push 10026060
      12BED6D:  cmp ESP, [10028CD4]
      12BED73:  inc EBX
      12BED74:  jbe @5
      12BED76:  inc EBX
@2:   12BED77:  mov EAX, 15                ;  1 <18>  LoadSmallInteger 10
      12BED7C:  mov [EBP-C], EAX           ;  3 <C3>  StoreTemporary1
      12BED7F:  mov EAX, ESI               ;  4 <0A>  LoadSelf
      12BED81:  push [EBP-C]               ;  5 <BC>  PushTemporary1
      12BED84:  call 1280207               ;  6 <E2>  SendSelector1
      12BED89:  mov ESP, EBP               ;  7 <48>  Return
      12BED8B:  pop EBP
      12BED8C:  mov ESI, [EBP-4]
      12BED8F:  ret NEAR
```

Digitalk's VM

VM architecture

VM architecture

Digitalk's VM

A Smalltalk Virtual Machine Architectural Model

Allen Wirfs-Brock
Pat Caudill
Instantiations, Inc.

Allen_Wirfs-Brock@Instantiations.com
Pat_Caudill@Instantiations.com

This document describes a bytecode architecture for an experimental Smalltalk virtual machine. The architecture was originally developed by the authors at Digitalk Inc. in 1993-1994. This document is published with the permission of ObjectShare Inc, the successor company to Digitalk.

This architecture is intended to be used for the representation of compiled methods in the context of a dynamic translating ("JIT") virtual machine. As such, its main purpose is to provide information that can be easily and efficiently processed by such a translator. An important design goal was the minimization of the space needed to represent compiled methods. Because it was designed to be the input to a translator, minimization of decoding time was not a consideration of the design. It was not intended for direct interpretation and certain features, such as the "Label" construct would be inefficient if used by an interpreter.

Architectural influences include the Smalltalk-80 virtual machine, Tektronix Smalltalk implementations, and various Digitalk ("Smalltalk/V") virtual machines. The basic architecture is that of an accumulator+stack machine. Unlike the Smalltalk-80 virtual machine but similar to the Digitalk vm's, the architecture uses an explicit stack of activation records instead of a linked-list of context objects. Most arguments and local variables are allocated in an activation record but an activation record may also reference a heap allocated "context" object (in retrospect, "environments" would have been a better name for these objects) that contain any local variables that have non-FIFO lifetimes. The bytecode compiler is responsible for determining which methods require context objects and for determining which variables need to be allocated in contexts.

The register+stack architecture was driven by these observations about Smalltalk programs:

- *Deep call chains*
- *Frequent adjacent calls*
- *Most computation happens in primitives and leaf methods*
- *Most methods have 0 or 1 arguments (plus receiver)*

The registers ensure that the receiver and arguments arrive at primitives or leaf methods in machine registers. Such methods typically do not need full activation records and do not need to store the arguments to the stack. Non-leaf methods save their arguments to the stack, where they stay across sequences of adjacent calls.

The X (index) register is used for indirect address and is primarily used for up-level addressing of lexically nested blocks.

special purpose registers

+

stack (frames)

JIT compiler

A Smalltalk Virtual Machine Architectural Model

Allen Wirfs-Brock
Pat Caudill
Instantiations, Inc.

Allen_Wirfs-Brock@Instantiations.com
Pat_Caudill@Instantiations.com

This document describes a bytecode architecture for an experimental Smalltalk virtual machine. The architecture was originally developed by the authors at Digitalk Inc. in 1993-1994. This document is published with the permission of ObjectShare Inc, the successor company to Digitalk.

This architecture is intended to be used for the representation of compiled methods in the context of a dynamic translating ("JIT") virtual machine. As such, its main purpose is to provide information that can be easily and efficiently processed by such a translator. An important design goal was the minimization of the space needed to represent compiled methods. Because it was designed to be the input to a translator, minimization of decoding time was not a consideration of the design. It was not intended for direct interpretation and certain features, such as the "Label" construct would be inefficient if used by

special purpose registers

+

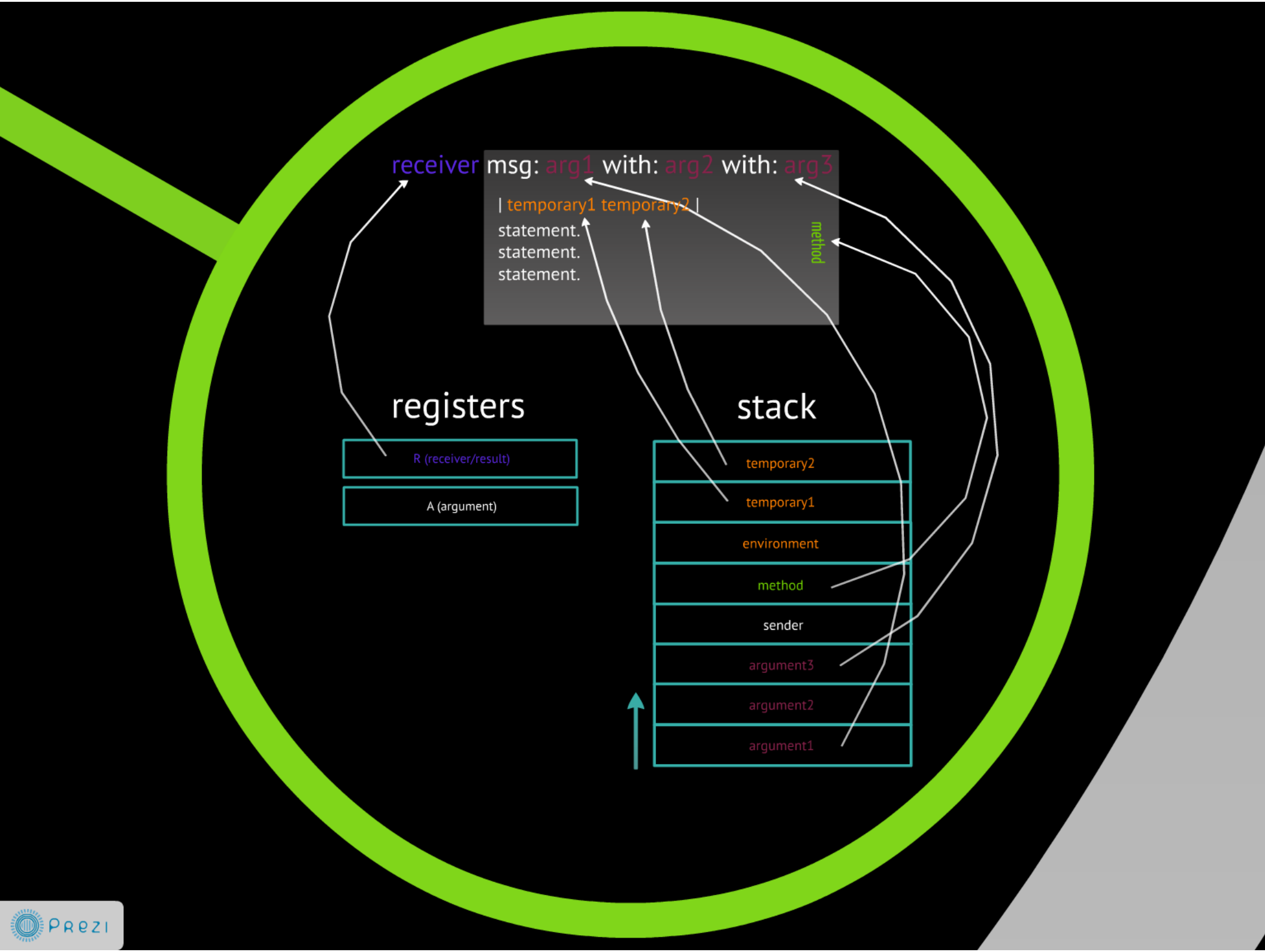
stack (frames)

special purpose registers

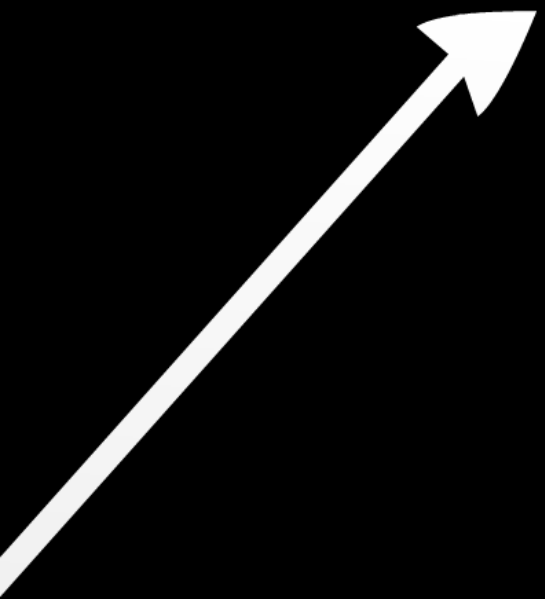
+

stack (frames)

JIT compiler


receiver msg: arg1 with: arg2 with: arg3
| temporary1 temporary2 |
statement.
statement.
statement.
method
registers
R (receiver/result)
A (argument)
stack
temporary2
temporary1
environment
method
sender
argument3
argument2
argument1

receiver msg



| tem

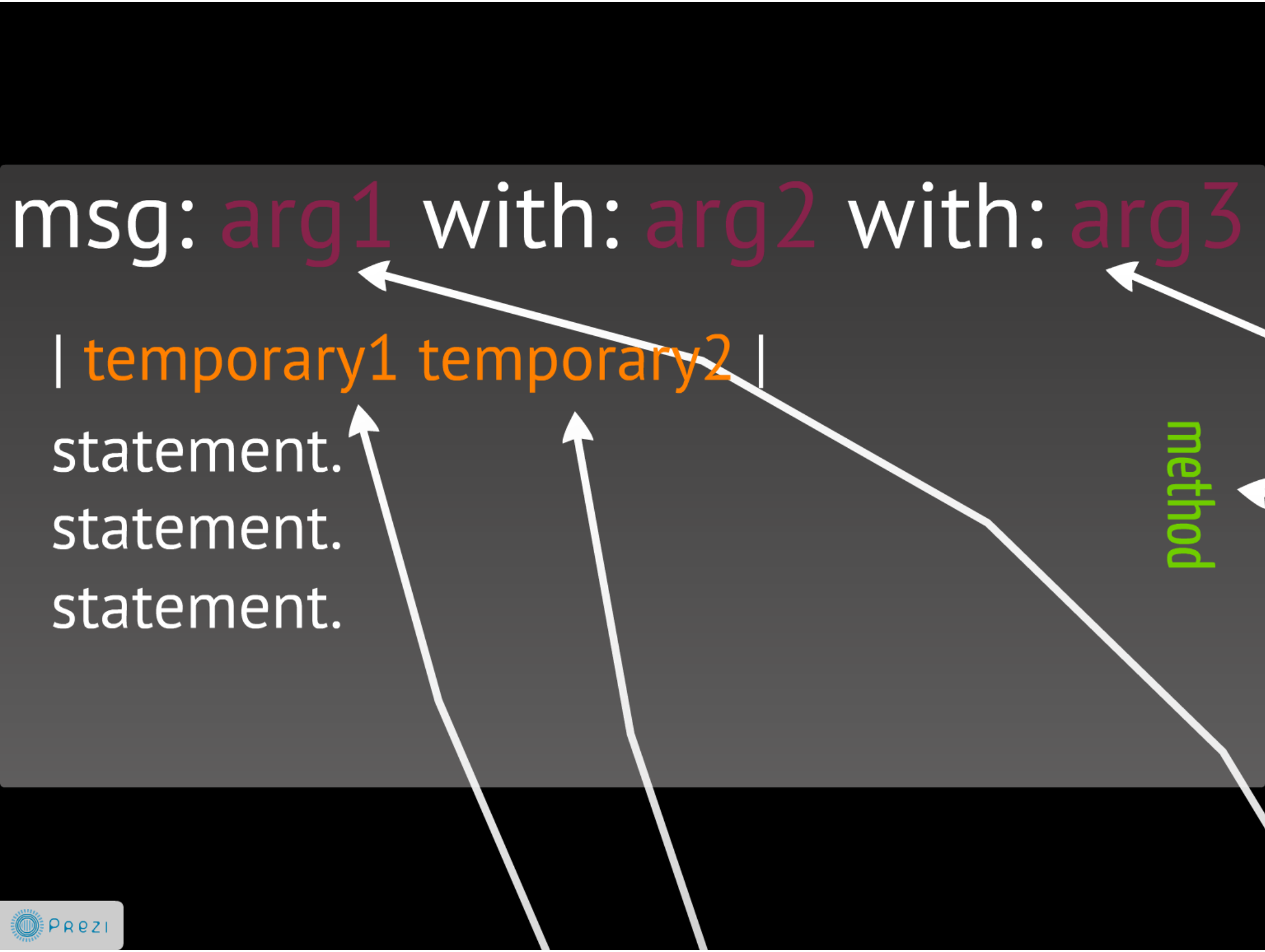
state

state

registers

R (receiver/result)

A (argument)

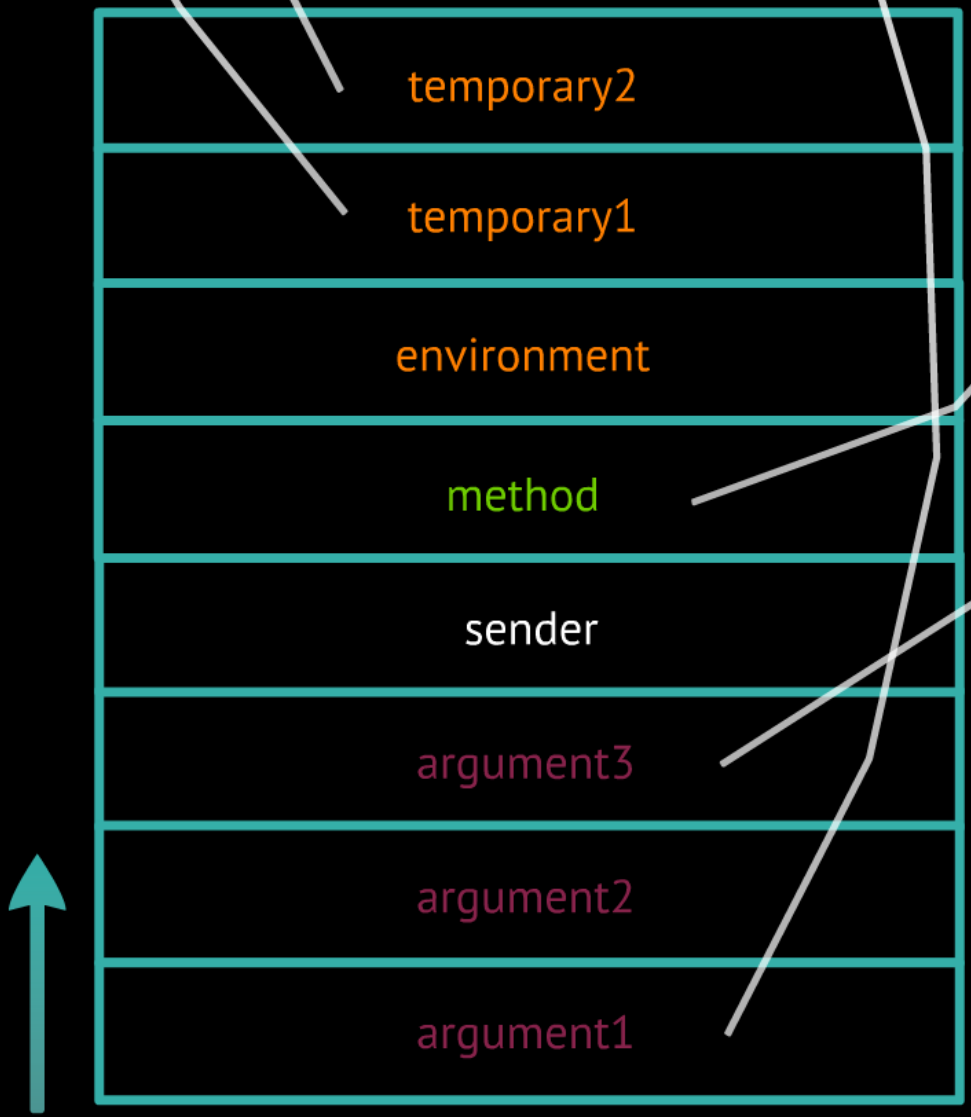

msg: arg1 with: arg2 with: arg3
| temporary1 temporary2 |
statement.
statement.
statement.
method

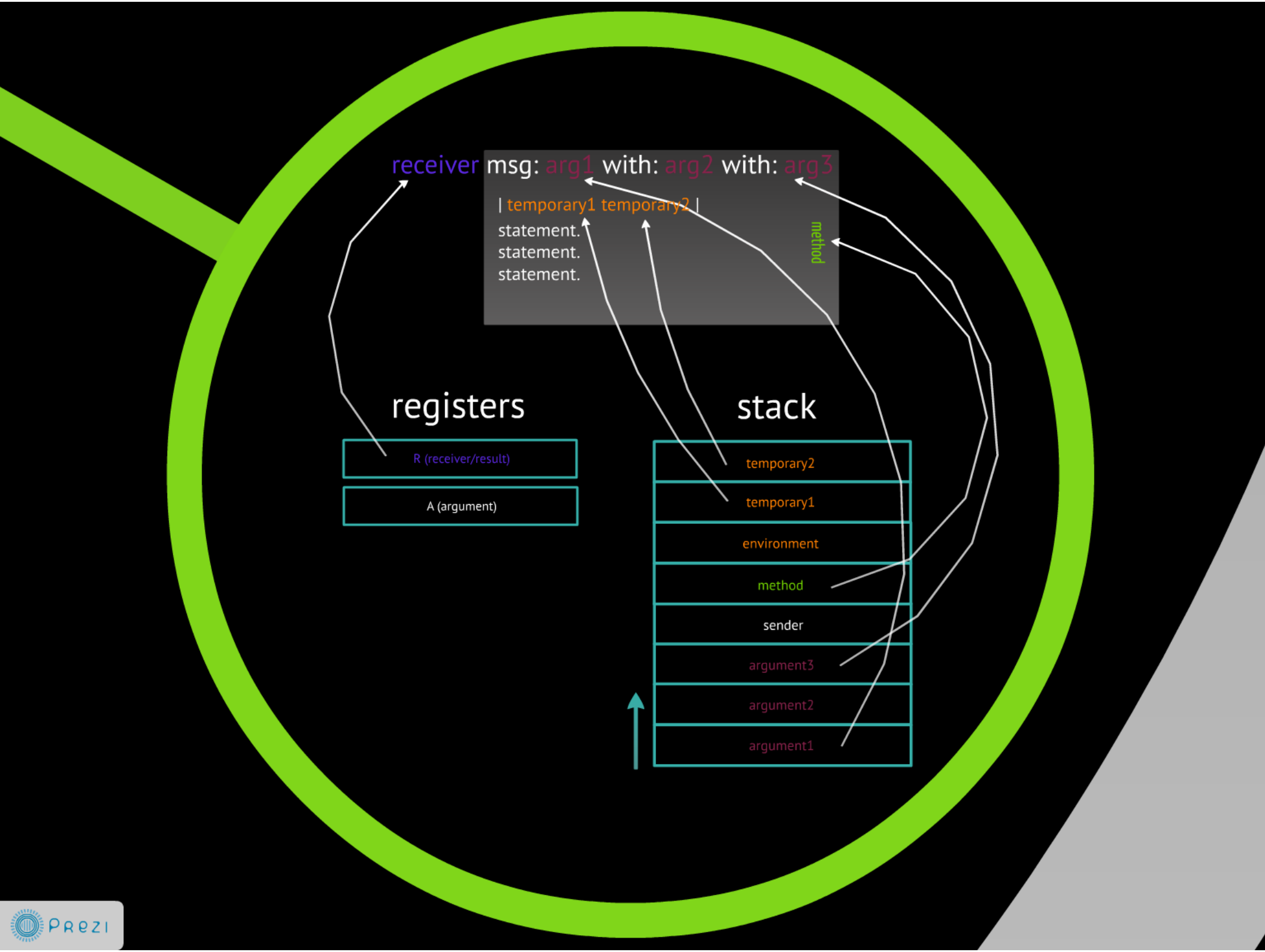
sters

er/result)

ument)

stack



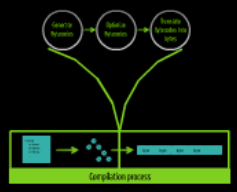

receiver msg: arg1 with: arg2 with: arg3
| temporary1 temporary2 |
statement.
statement.
statement.
method
registers
R (receiver/result)
A (argument)
stack
temporary2
temporary1
environment
method
sender
argument3
argument2
argument1

Optimize

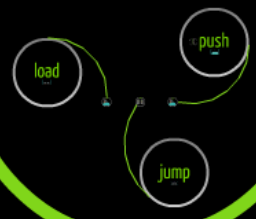
Why?

- Performance
- Simplicity
- Framework
- Accessibility

How?



What?




Optimize


Why?
Performance
Reduce processor cycles and unnecessary memory accesses
Simplicity
Reduce bytecodes complexity (then reducing native code complexity)
Framework
Provide a framework to study new optimizations
Accessibility
Make the compilation domain accessible to any smalltalker

Performance

Reduce processor cycles and unnecessary memory accesses

Simplicity

Reduce bytecodes complexity (then reducing native code complexity)

Framework

Provide a framework to study new optimizations

Accessibility

Make the compilation domain accessible to any smalltalker

smalltalker

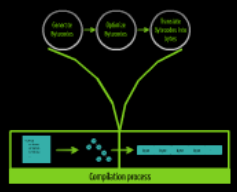
specialist

Optimize

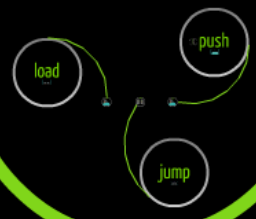
Why?

- Performance
- Simplicity
- Framework
- Accessibility

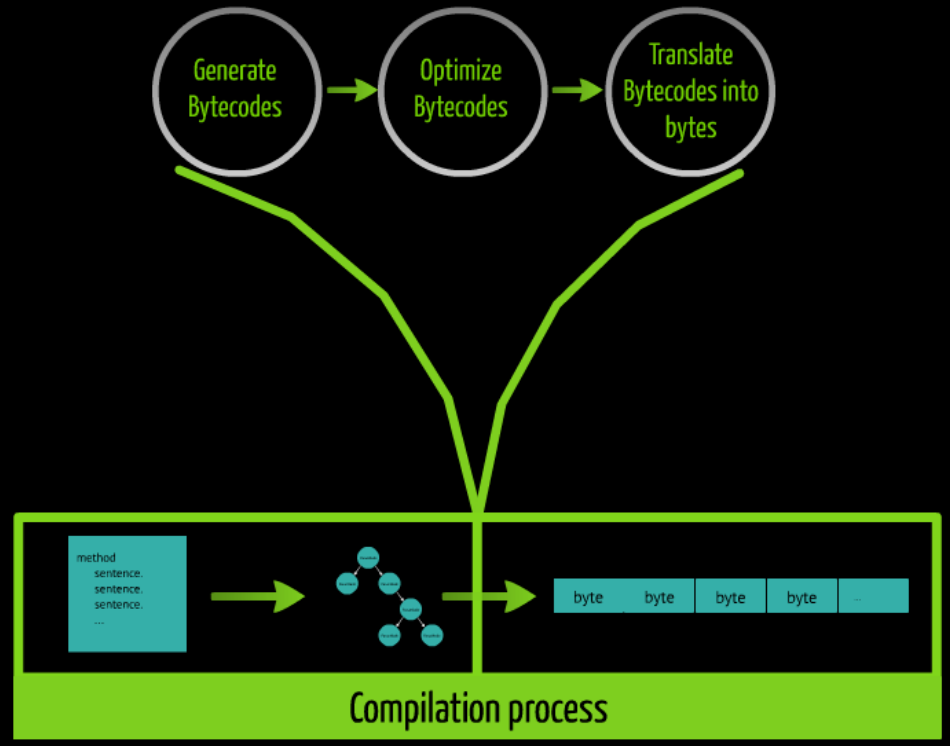
How?

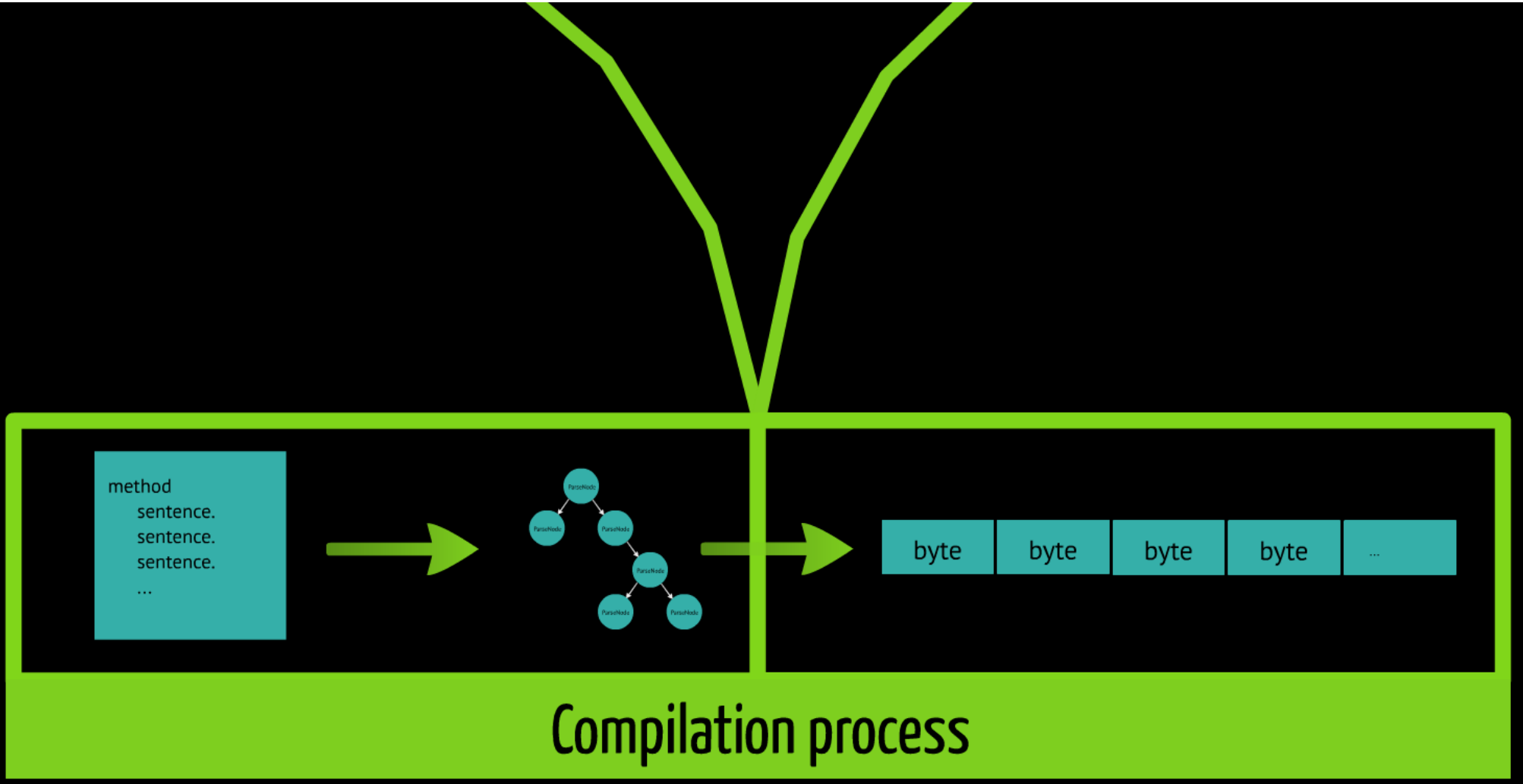


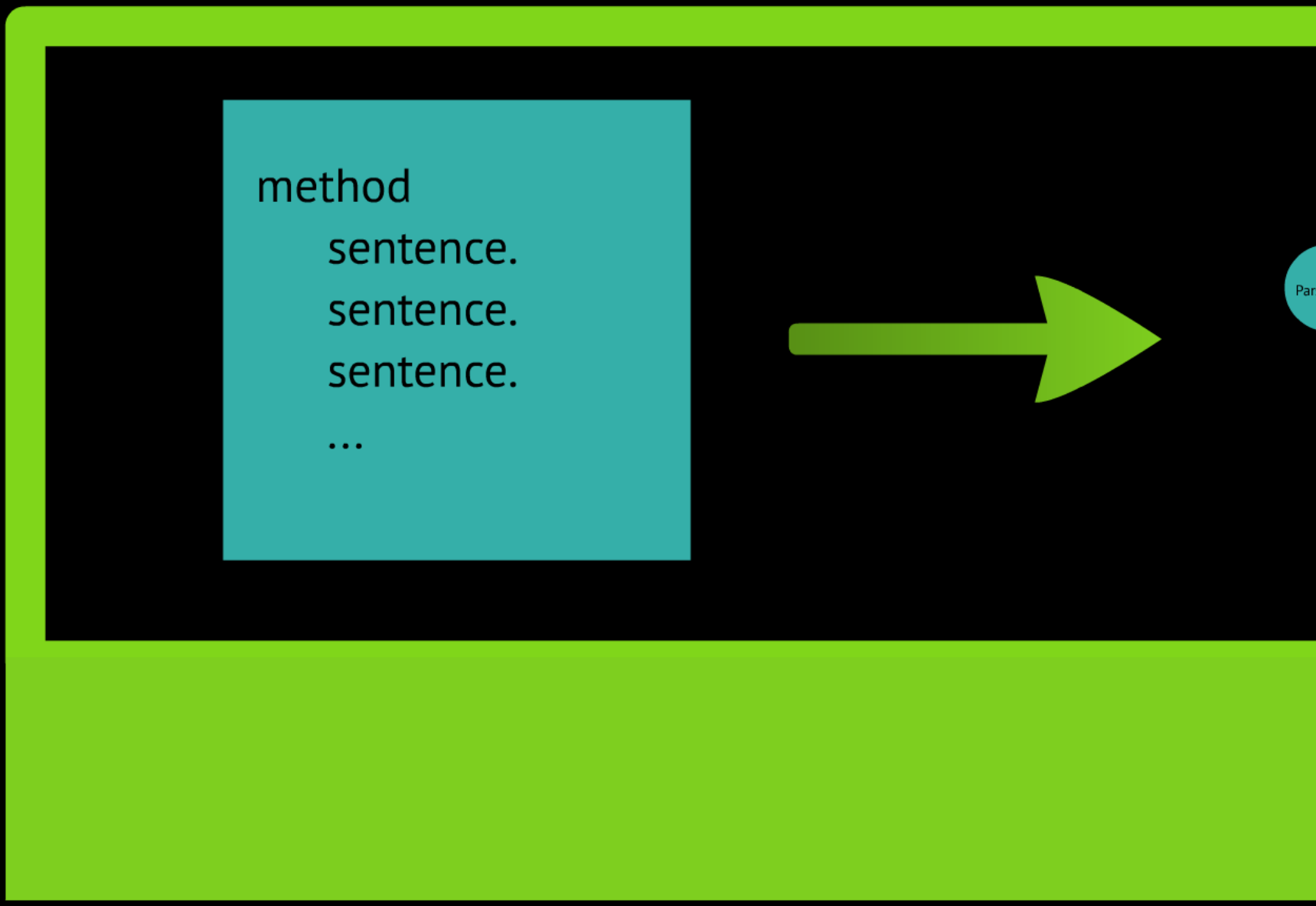
What?

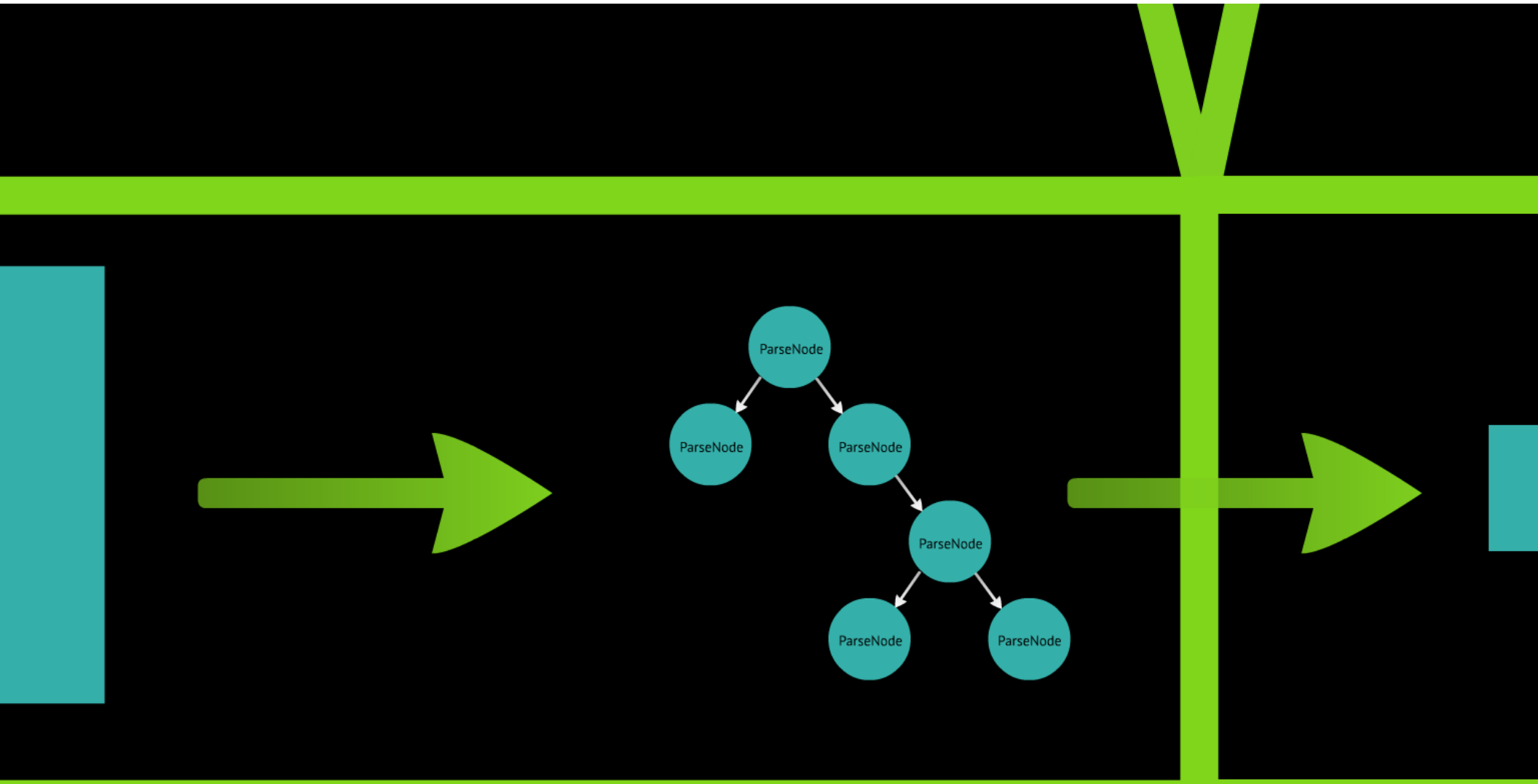


How?


Generate Bytecodes
Optimize Bytecodes
Translate Bytecodes into bytes
method
sentence.
sentence.
sentence.
...
byte
byte
byte
byte
...
Compilation process

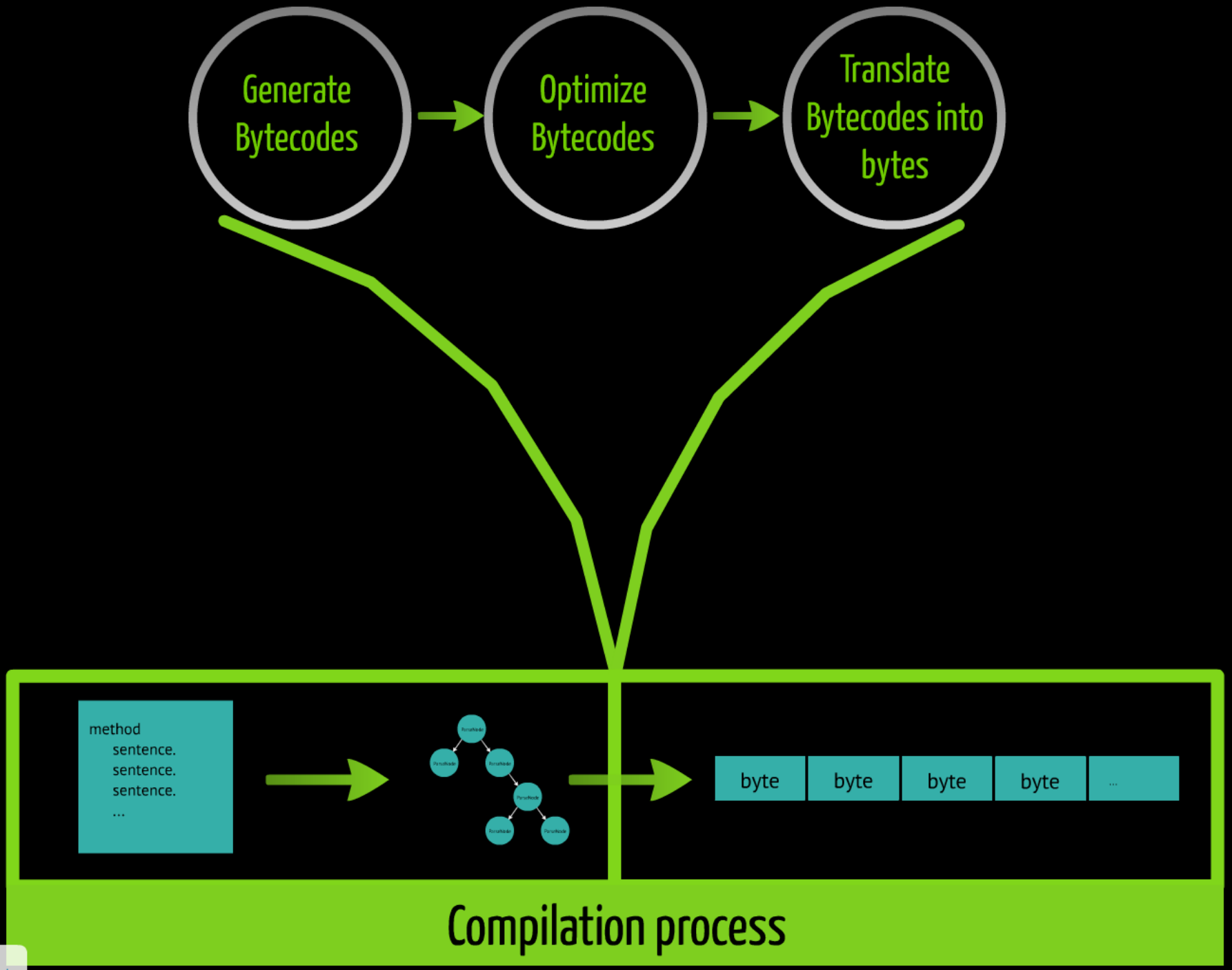

method
sentence.
sentence.
sentence.
...
ParseNode
ParseNode
ParseNode
ParseNode
ParseNode
ParseNode
byte
byte
byte
byte
...
Compilation process


method
sentence.
sentence.
sentence.
...

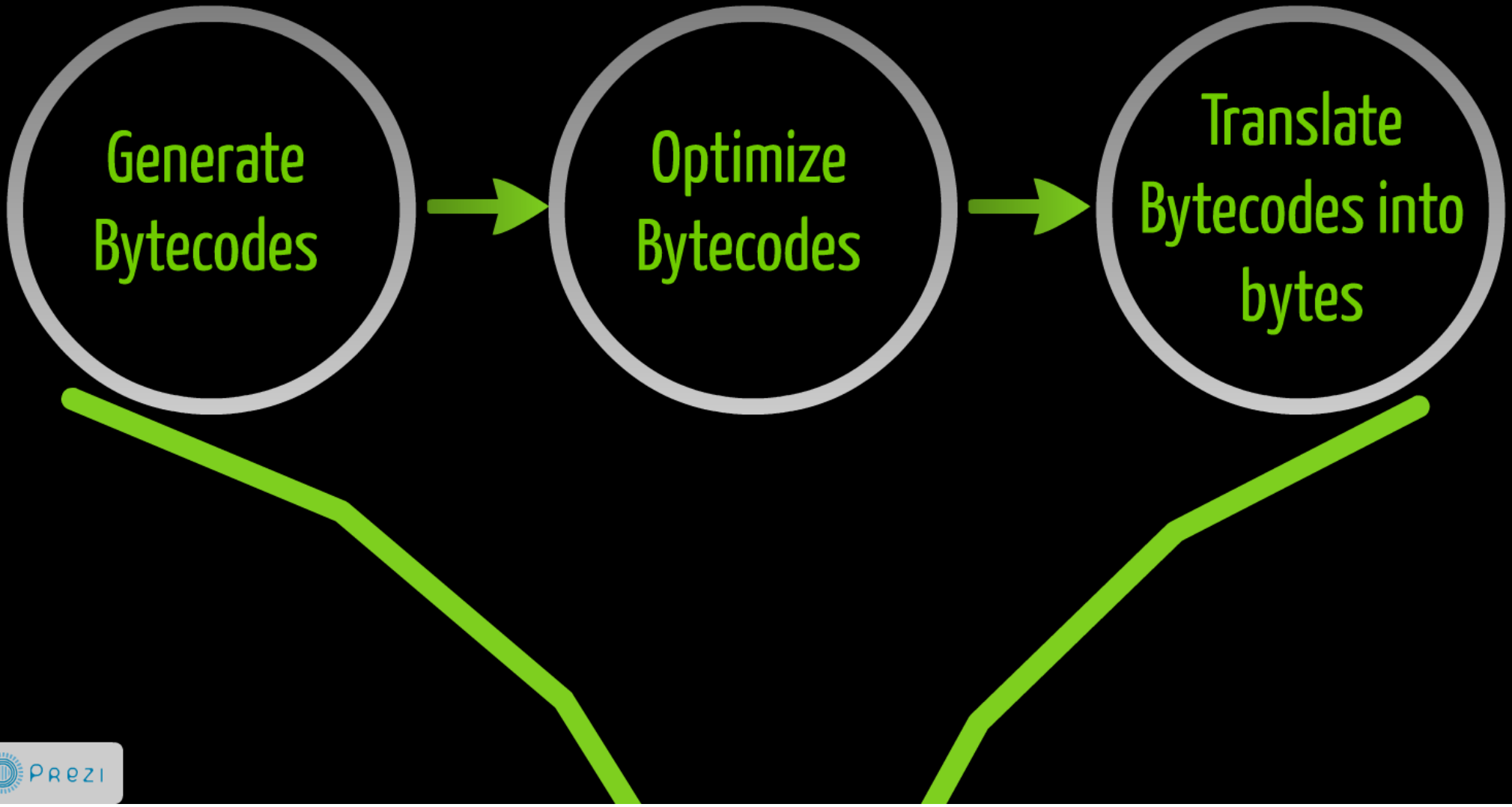


Compilation proc

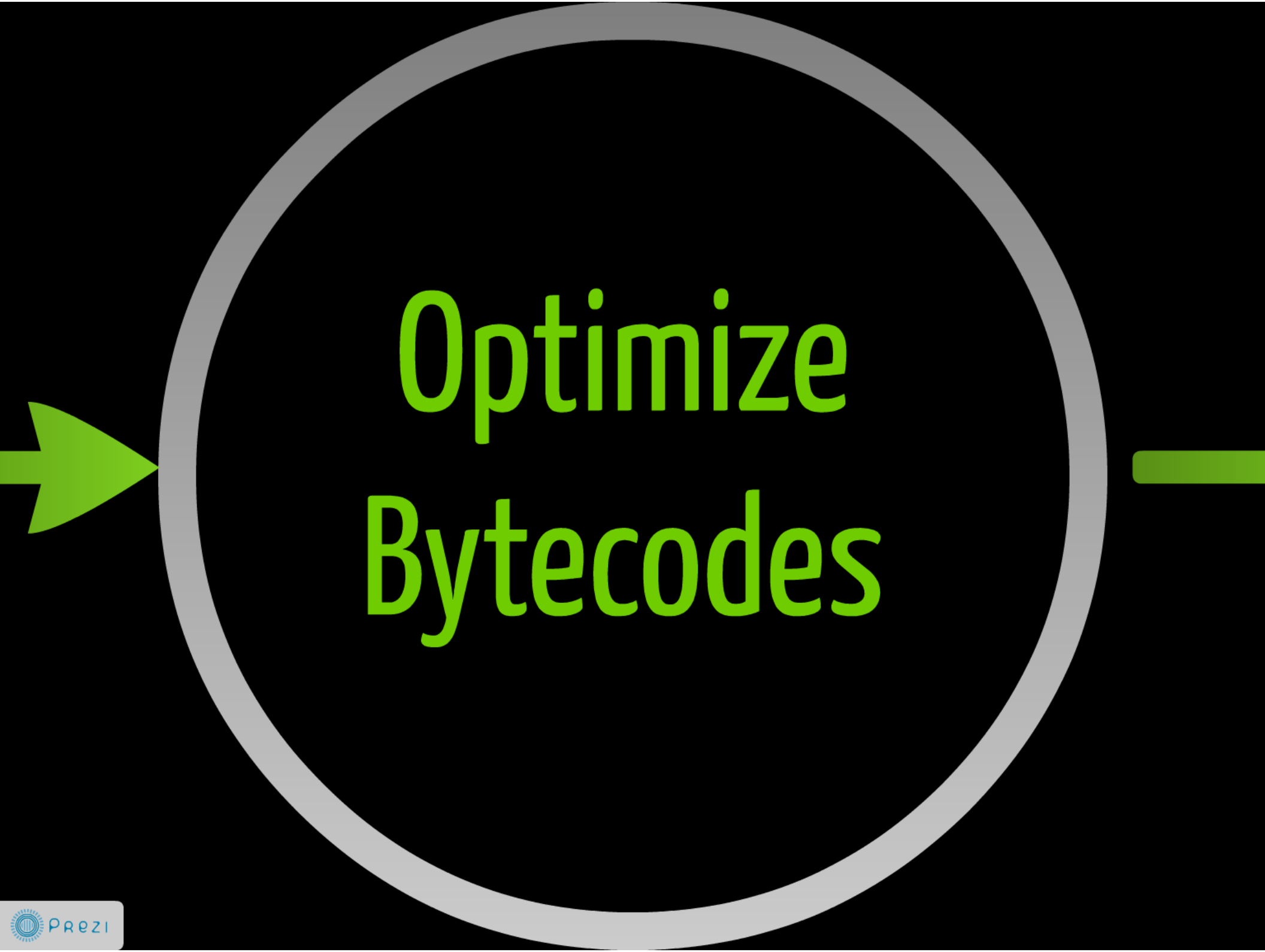
n process

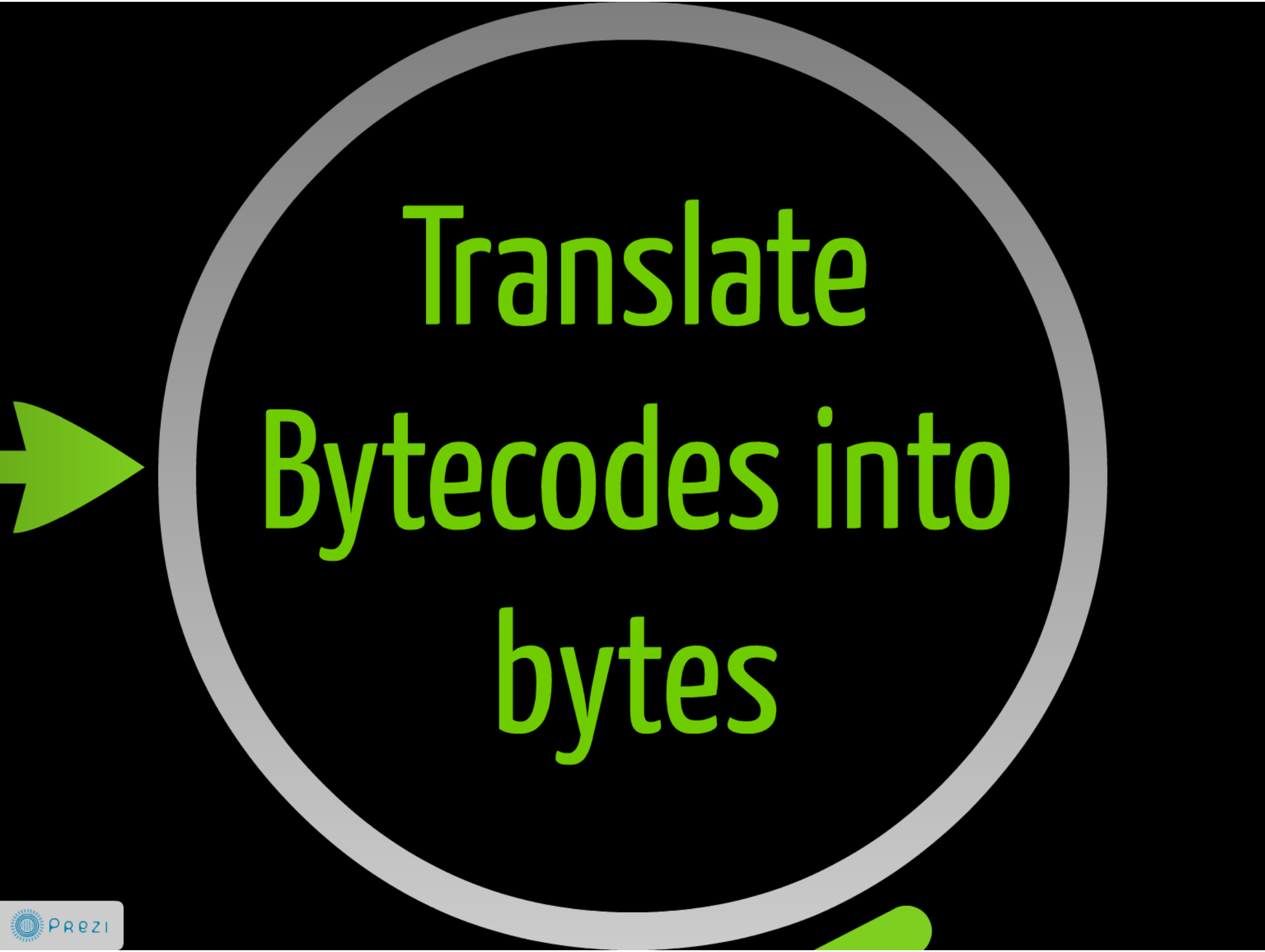

Generate Bytecodes
Optimize Bytecodes
Translate Bytecodes into bytes
method
sentence.
sentence.
sentence.
...
byte
byte
byte
byte
...
Compilation process

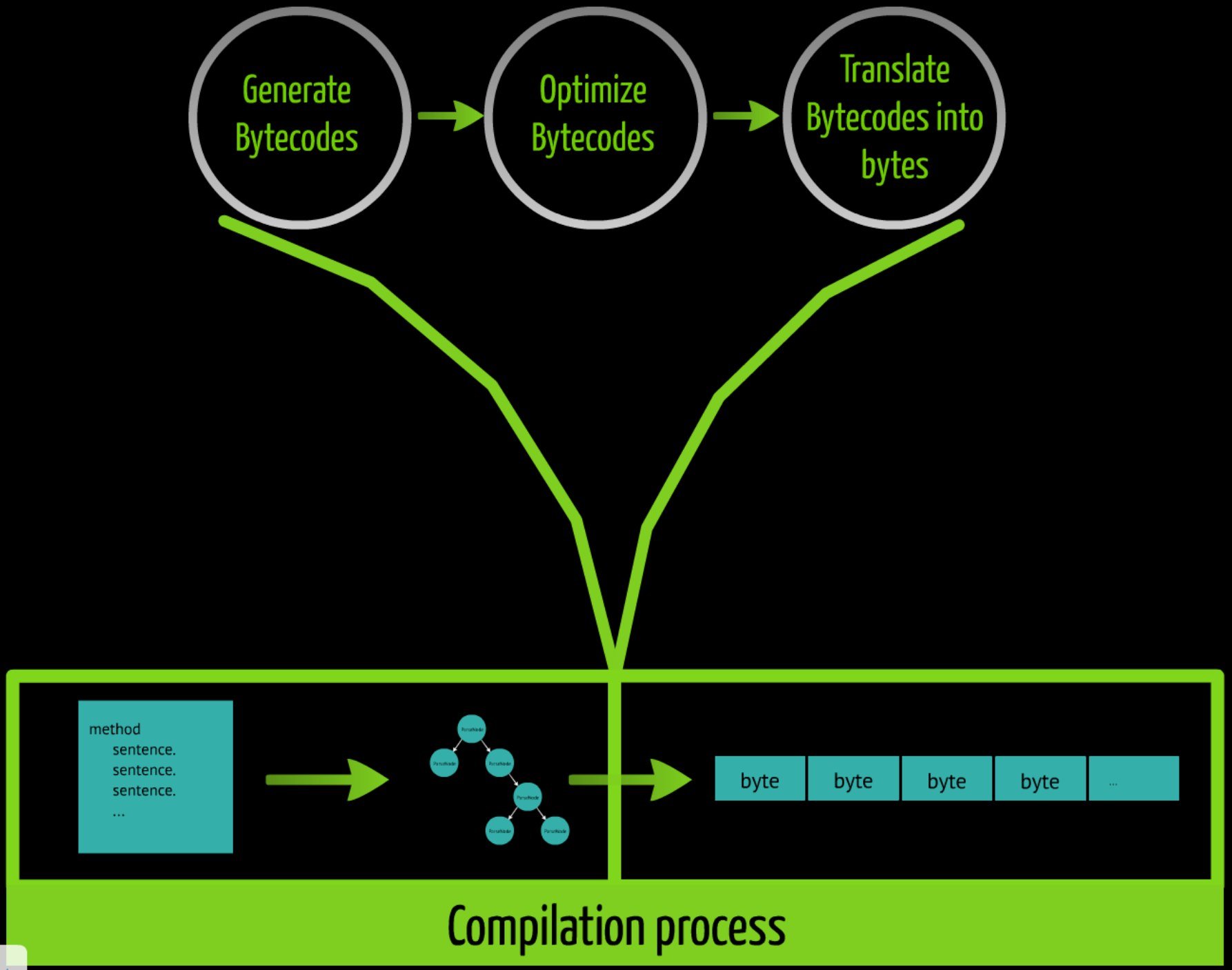
How?

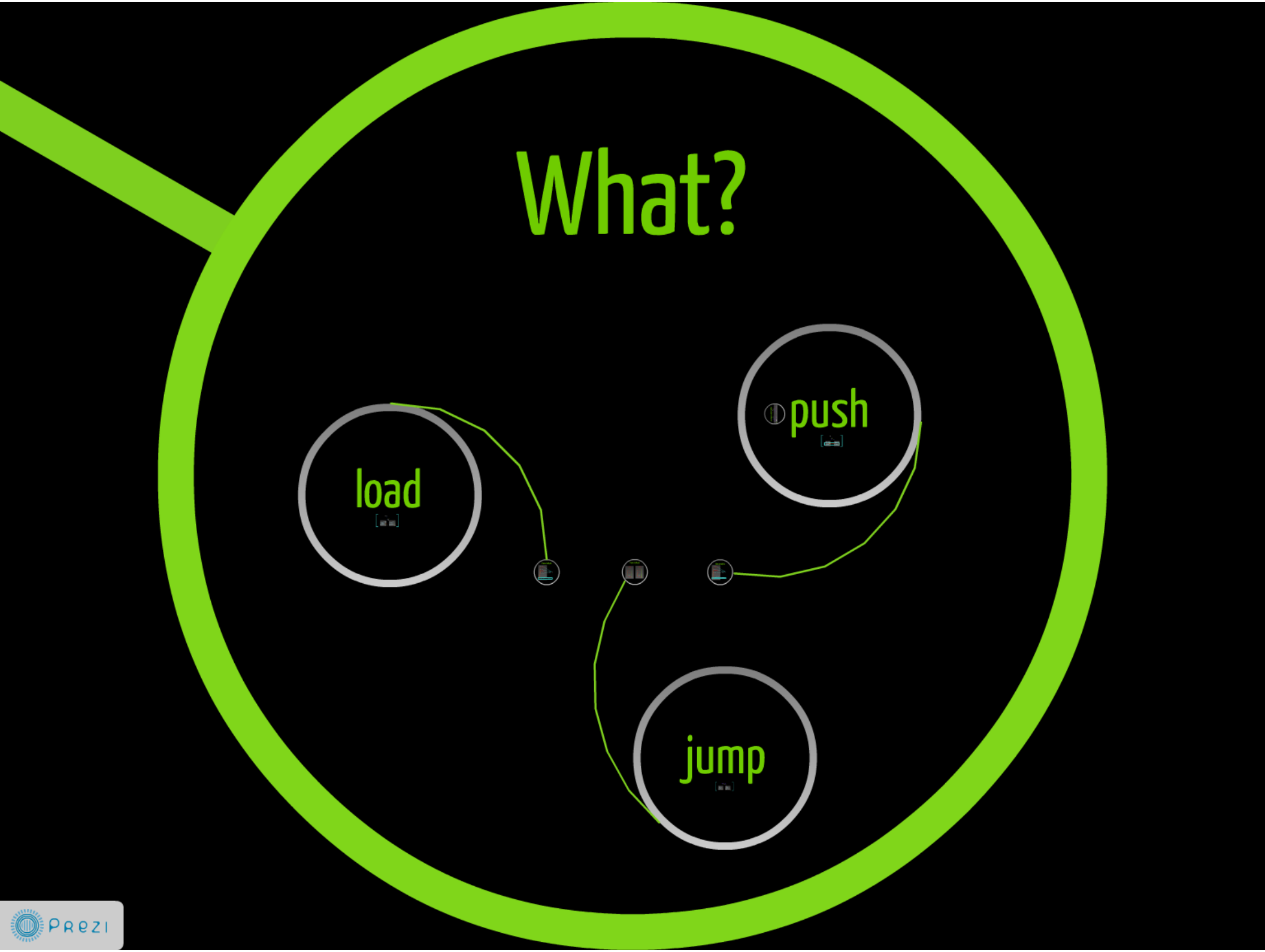

Generate Bytecodes
Optimize Bytecodes
Translate Bytecodes into bytes

Generate Bytecodes


Optimize
Bytecodes


Translate
Bytecodes into
bytes


Generate Bytecodes
Optimize Bytecodes
Translate Bytecodes into bytes
method
sentence.
sentence.
sentence.
...
byte
byte
byte
byte
...
Compilation process


What?
load
push
jump

load

```
msg
    | t |
    t := 10.
    t msg
```

```
LoadSmallInteger 10
StoreTemporary1 t
LoadTemporary1 t
SendSelector1 #msg
ReturnSelf
```

R -> t

```
LoadSmallInteger 10
StoreTemporary1 t
LoadTemporary1 t
SendSelector1 #msg
ReturnSelf
```

```
msg
    | t |
    t := 10.
    t msg
```

```
LoadSmallInteger 10
StoreTemporary1 t
LoadTemporary1 t
SendSelector1 #msg
ReturnSelf
```

R -> t

```
LoadSmallInteger 10
StoreTemporary1 t
LoadTemporary1 t
SendSelector1 #msg
ReturnSelf
```

R -> t

```
LoadSmallInteger 10
StoreTemporary1 t
LoadTemporary1 t
SendSelector1 #msg
ReturnSelf
```

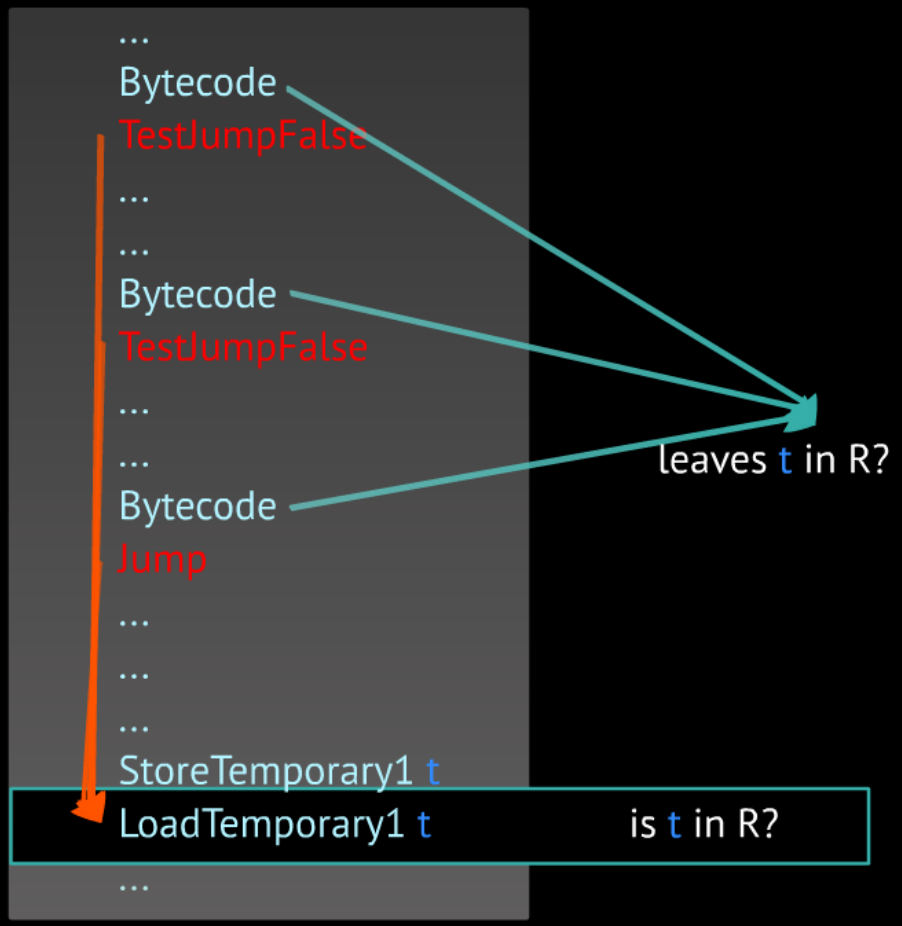
```
msg
    | t |
    t := 10.
    t msg
```

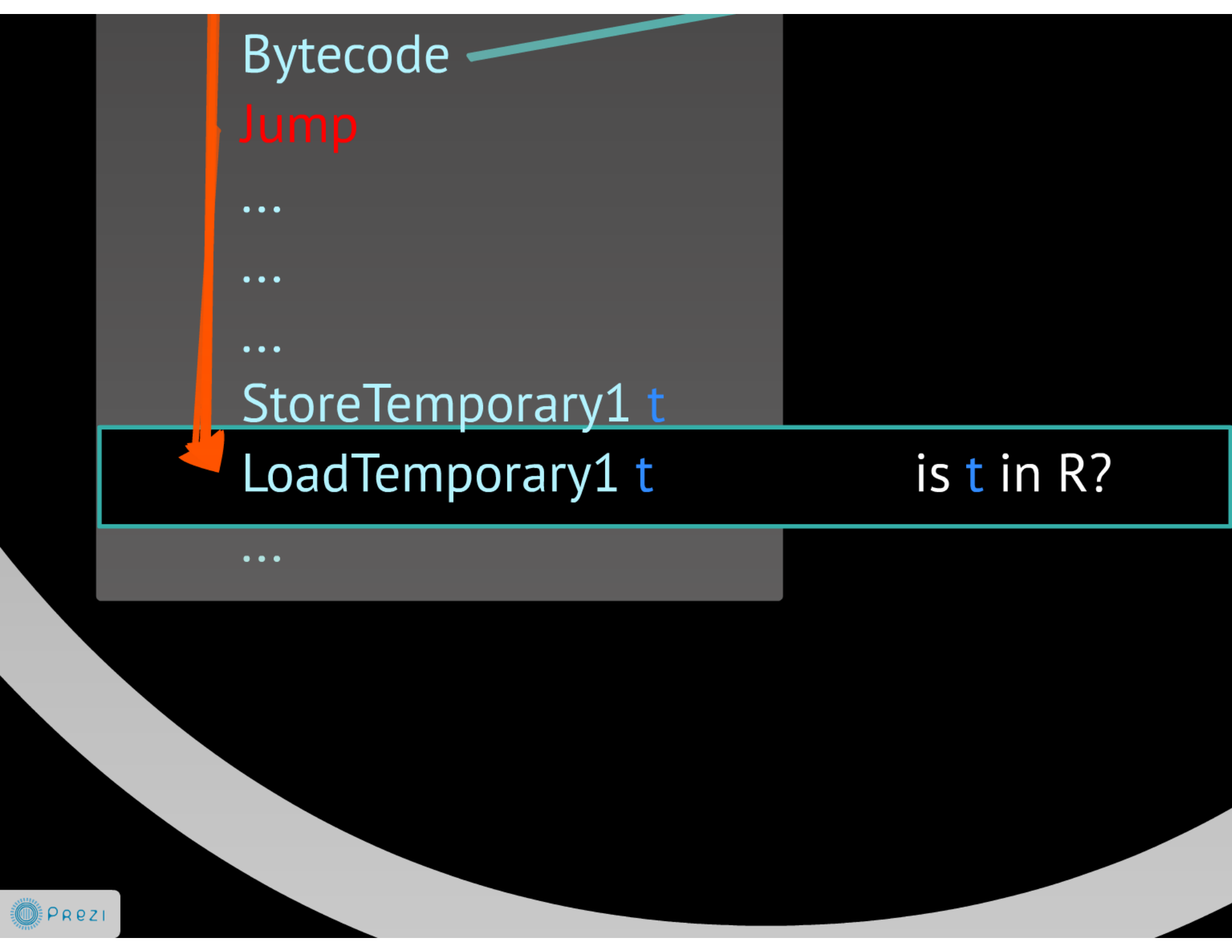
```
LoadSmallInteger 10
StoreTemporary1 t
LoadTemporary1 t
SendSelector1 #msg
ReturnSelf
```

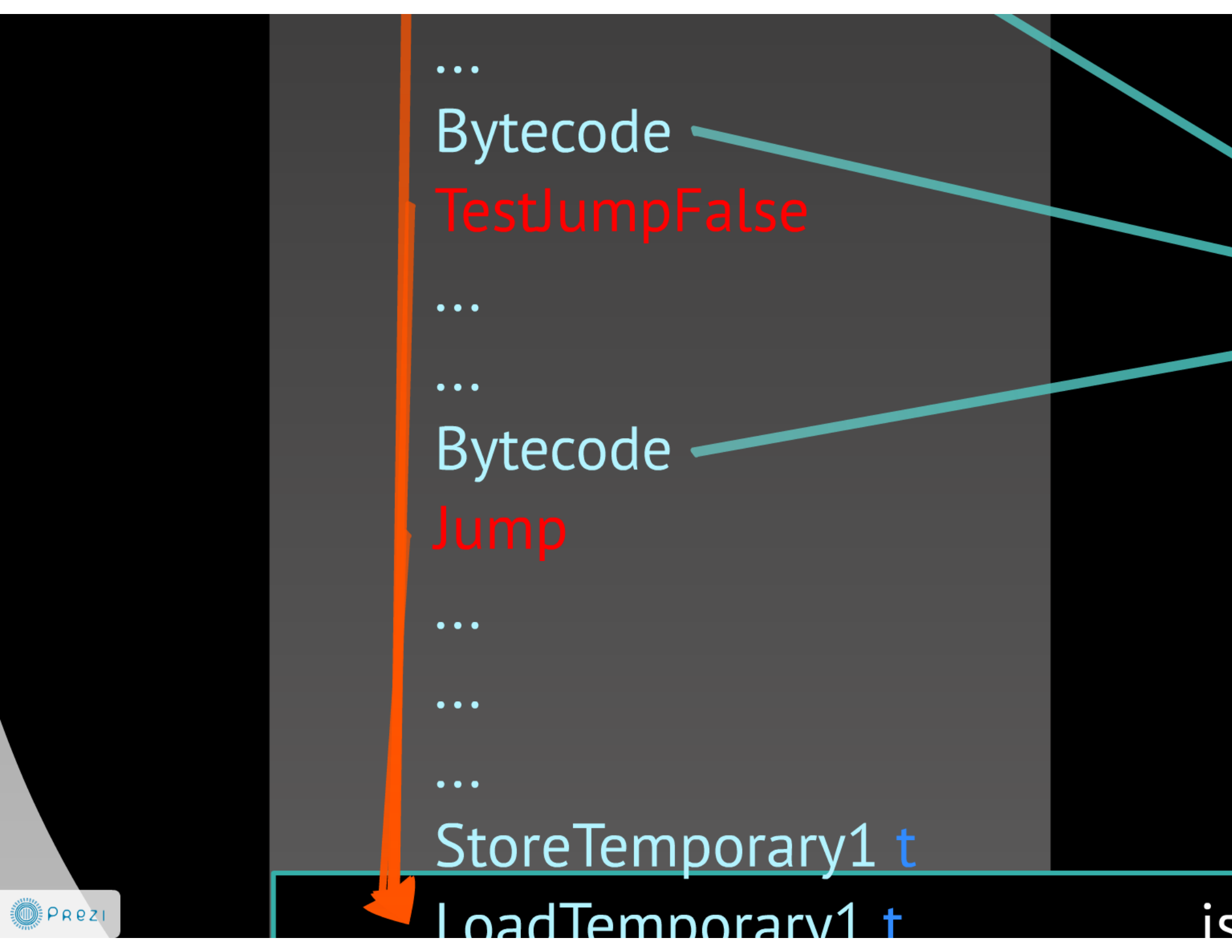
R -> t

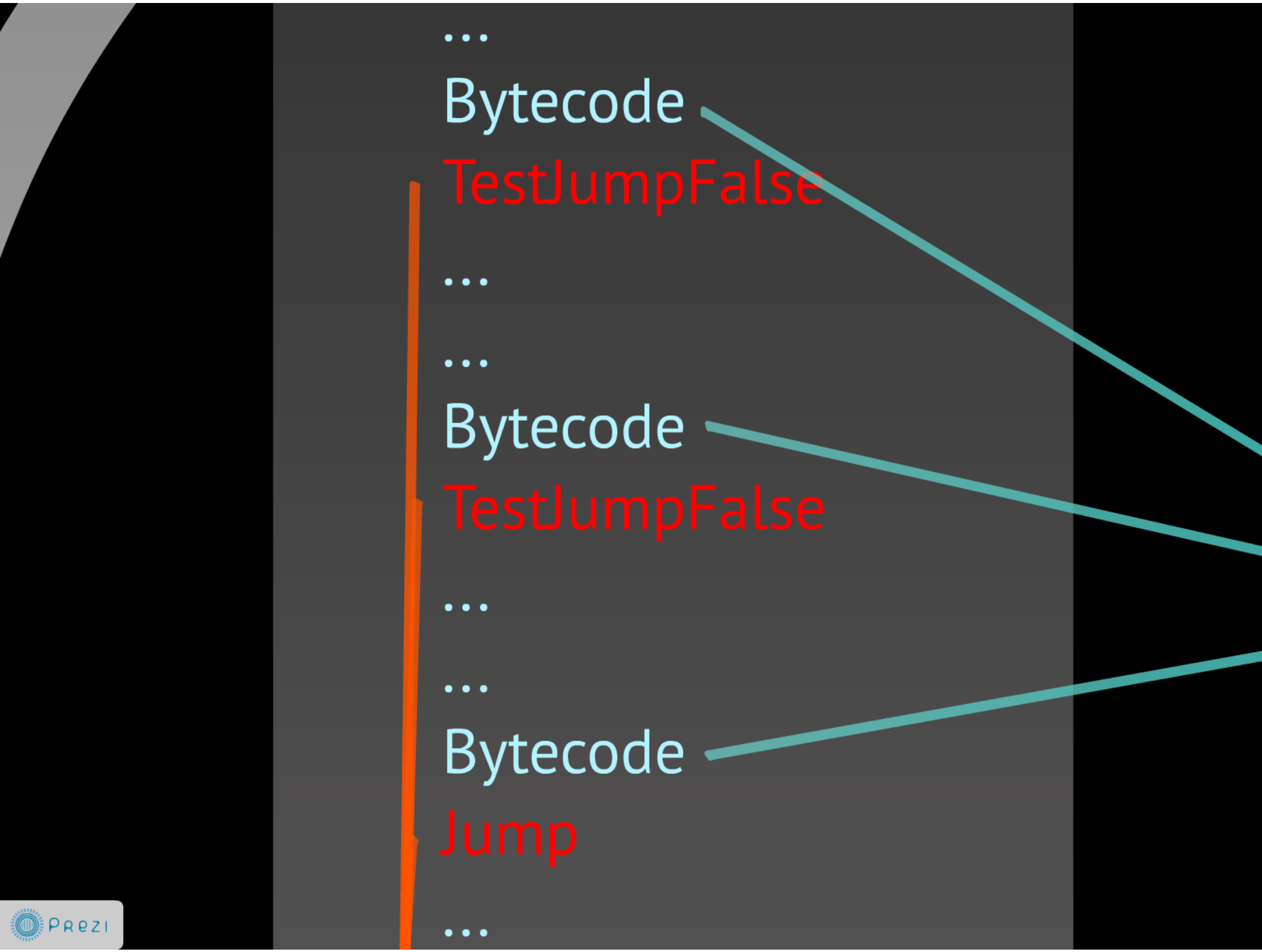
```
LoadSmallInteger 10
StoreTemporary1 t
LoadTemporary1 t
SendSelector1 #msg
ReturnSelf
```

Flow analysis

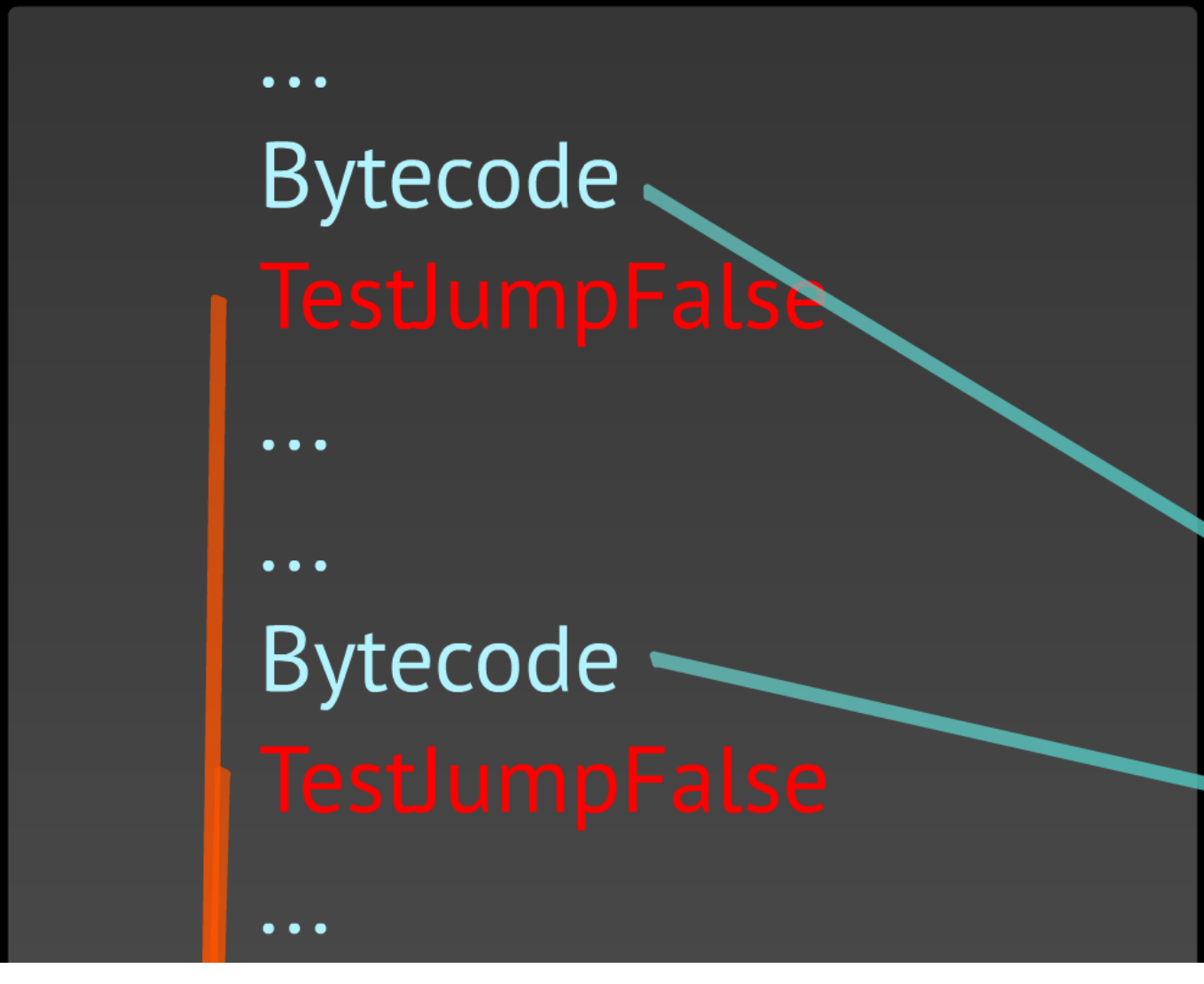


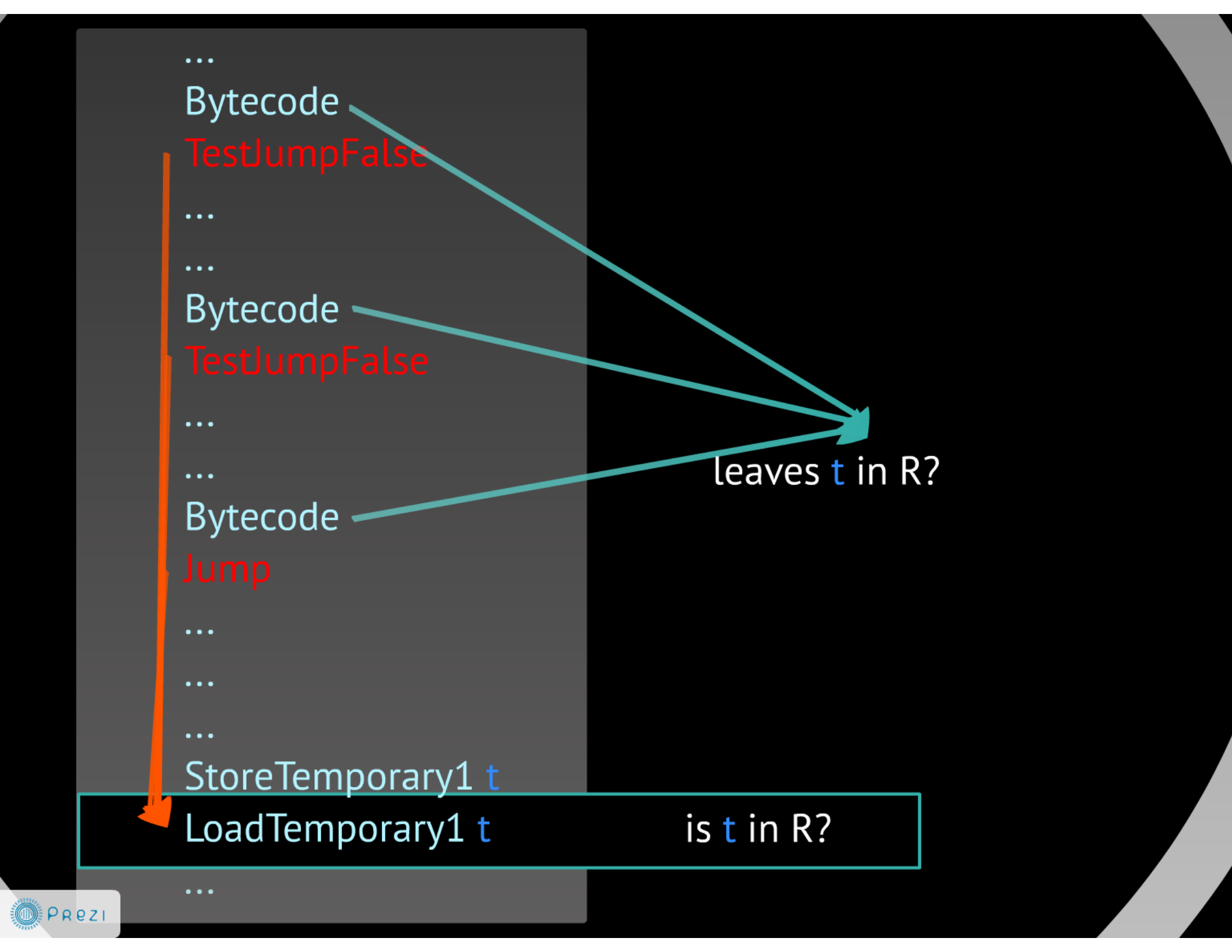

Bytecode
Jump
...
...
...
StoreTemporary1 t
LoadTemporary1 t
is t in R?
...

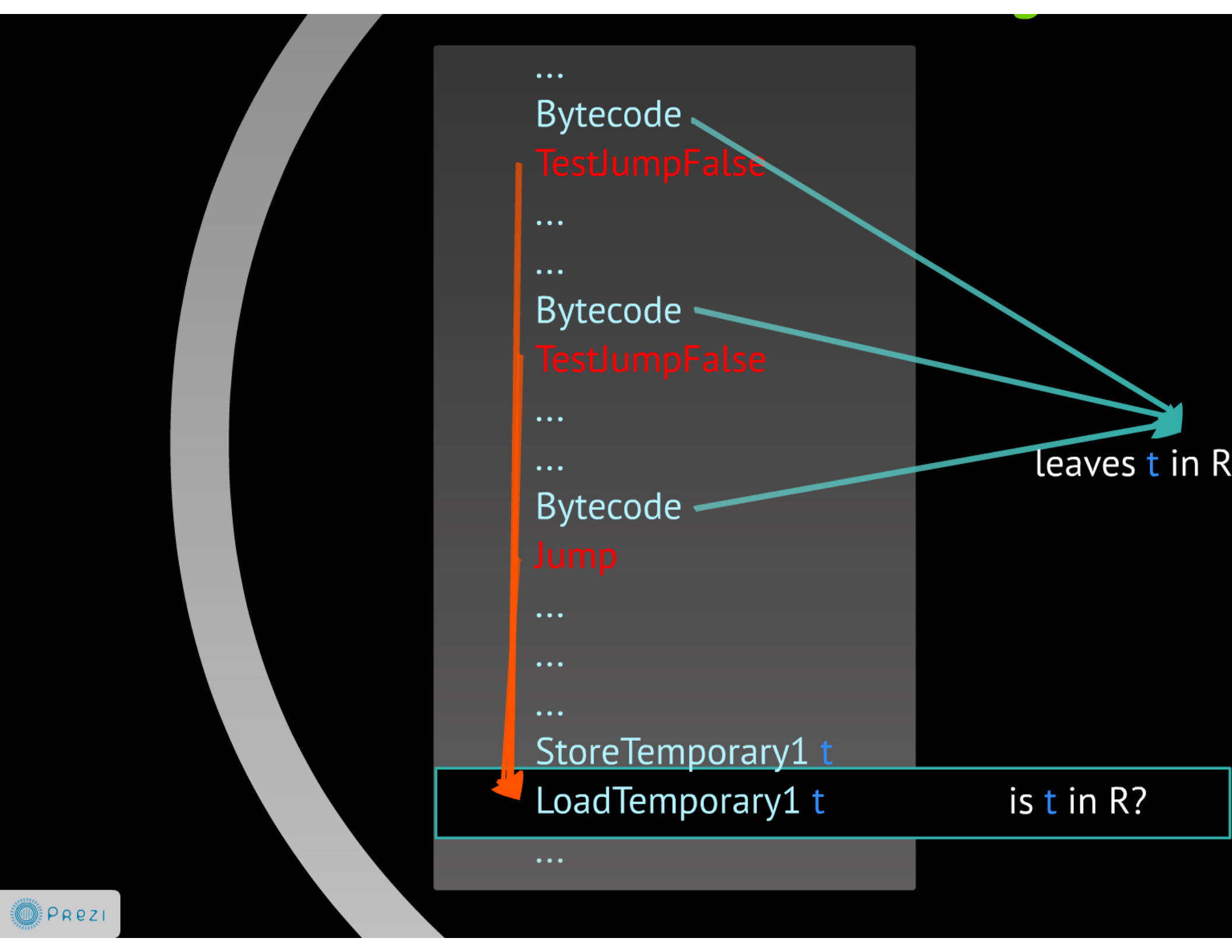

...
Bytecode
TestJumpFalse
...
...
Bytecode
Jump
...
...
...
StoreTemporary1 t
LoadTemporary1 t


...
Bytecode
TestJumpFalse
...
...
Bytecode
TestJumpFalse
...
...
Bytecode
Jump
...

Flow analy

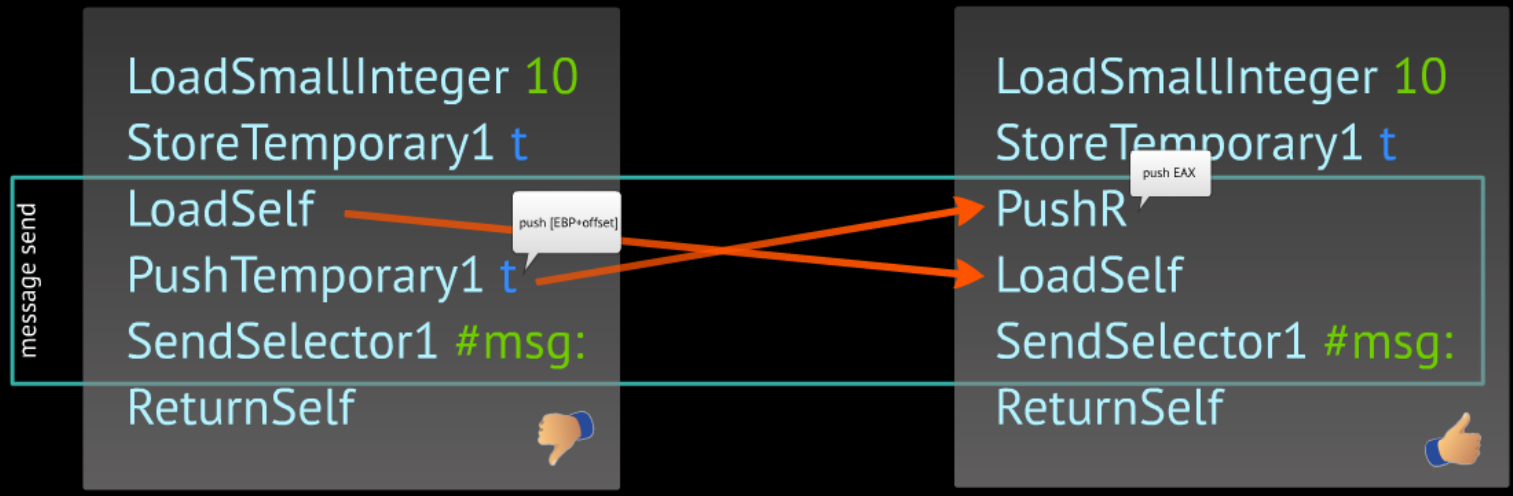



...
Bytecode
TestJumpFalse
...
...
Bytecode
TestJumpFalse
...
...
Bytecode
Jump
...
...
...
StoreTemporary1 t
LoadTemporary1 t
is t in R?
...
leaves t in R?


...
Bytecode
TestJumpFalse
...
...
Bytecode
TestJumpFalse
...
...
Bytecode
Jump
...
...
...
StoreTemporary1 t
LoadTemporary1 t
is t in R?
...
leaves t in R

push

```
msg
    | t |
    t := 10.
    self msg: t
```



```
msg
    | t |
    t := 10.
    self msg: t
```

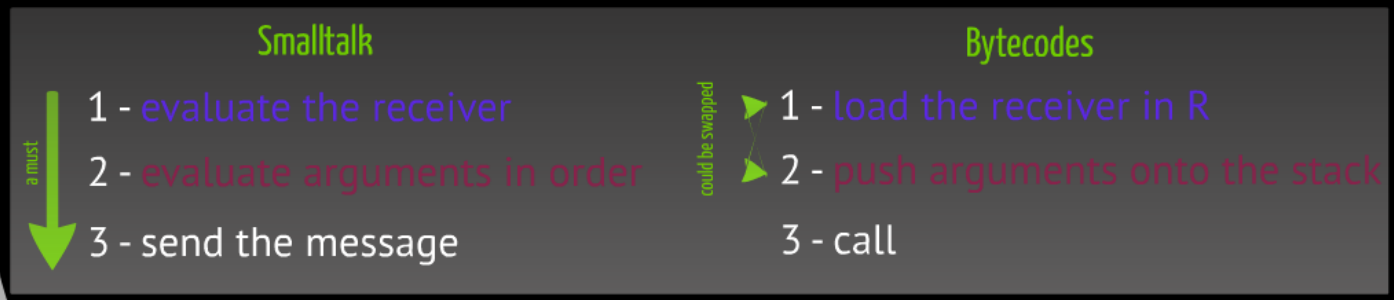
Message send

```
LoadSmallInteger 10
StoreTemporary1 t
LoadSelf
PushTemporary1 t
SendSelector1 #msg:
ReturnSelf
```

push [EBP+offset]

message send semantics

receiver msg: arg1 with: arg2 with: arg3



message send semantics

receiver msg: arg1 with: arg2 with: arg3

Smalltalk	Bytecodes
1 - evaluate the receiver	1 - load the receiver in R
2 - evaluate arguments in order	2 - push arguments onto the stack
3 - send the message	3 - call

a must

could be swapped

receiver msg: arg1

Smalltalk

1 - evaluate the receiver

2 - evaluate arguments in order

3 - send the message

with: arg2 with: arg3

Bytecodes

1 - load the receiver in R

2 - push arguments onto the stack

3 - call

Message send

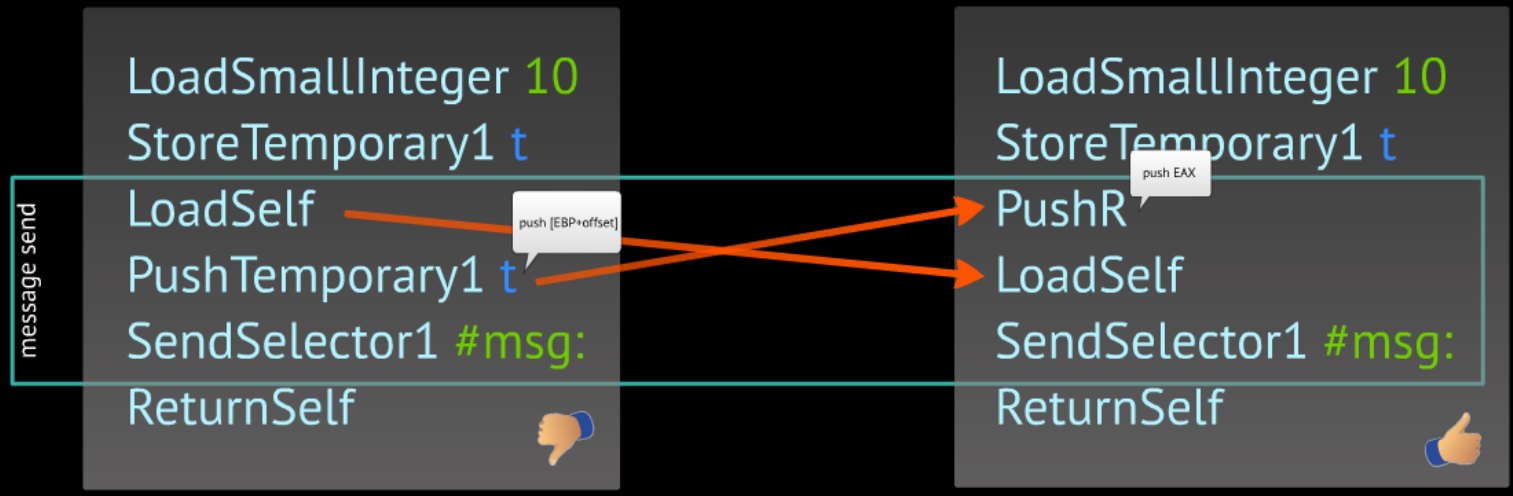
```
LoadSmallInteger 10
StoreTemporary1 t
LoadSelf
PushTemporary1 t
SendSelector1 #msg:
ReturnSelf
```

push [EBP+offset]

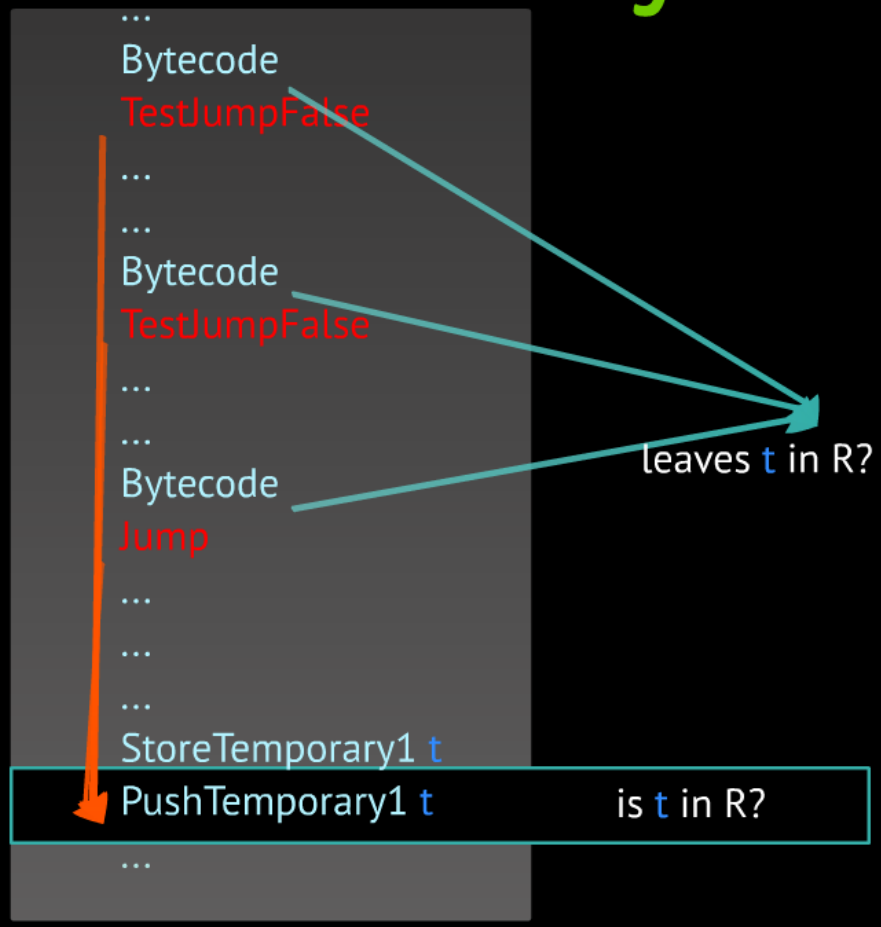
```
LoadSmallInteger 10
StoreTemporary1 t
PushR
LoadSelf
SendSelector1 #msg:
ReturnSelf
```

push EAX

```
msg
    | t |
    t := 10.
    self msg: t
```

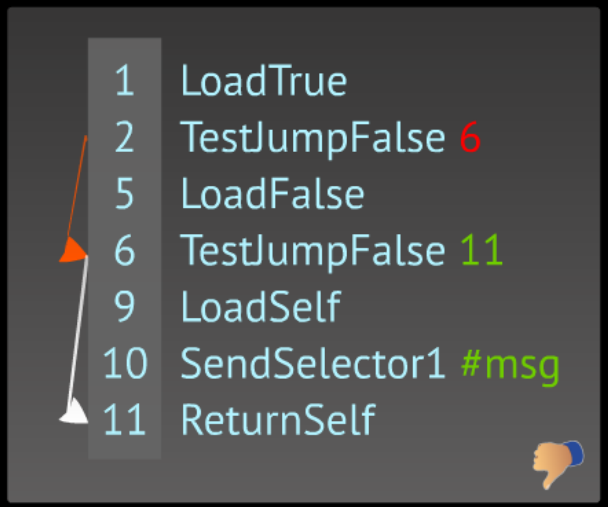


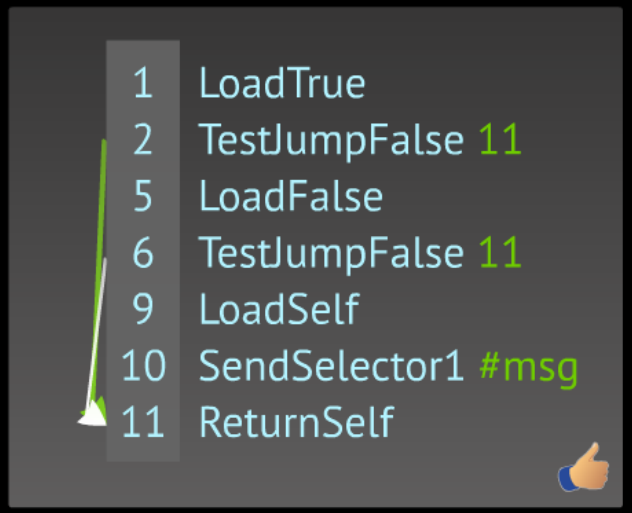
Flow analysis



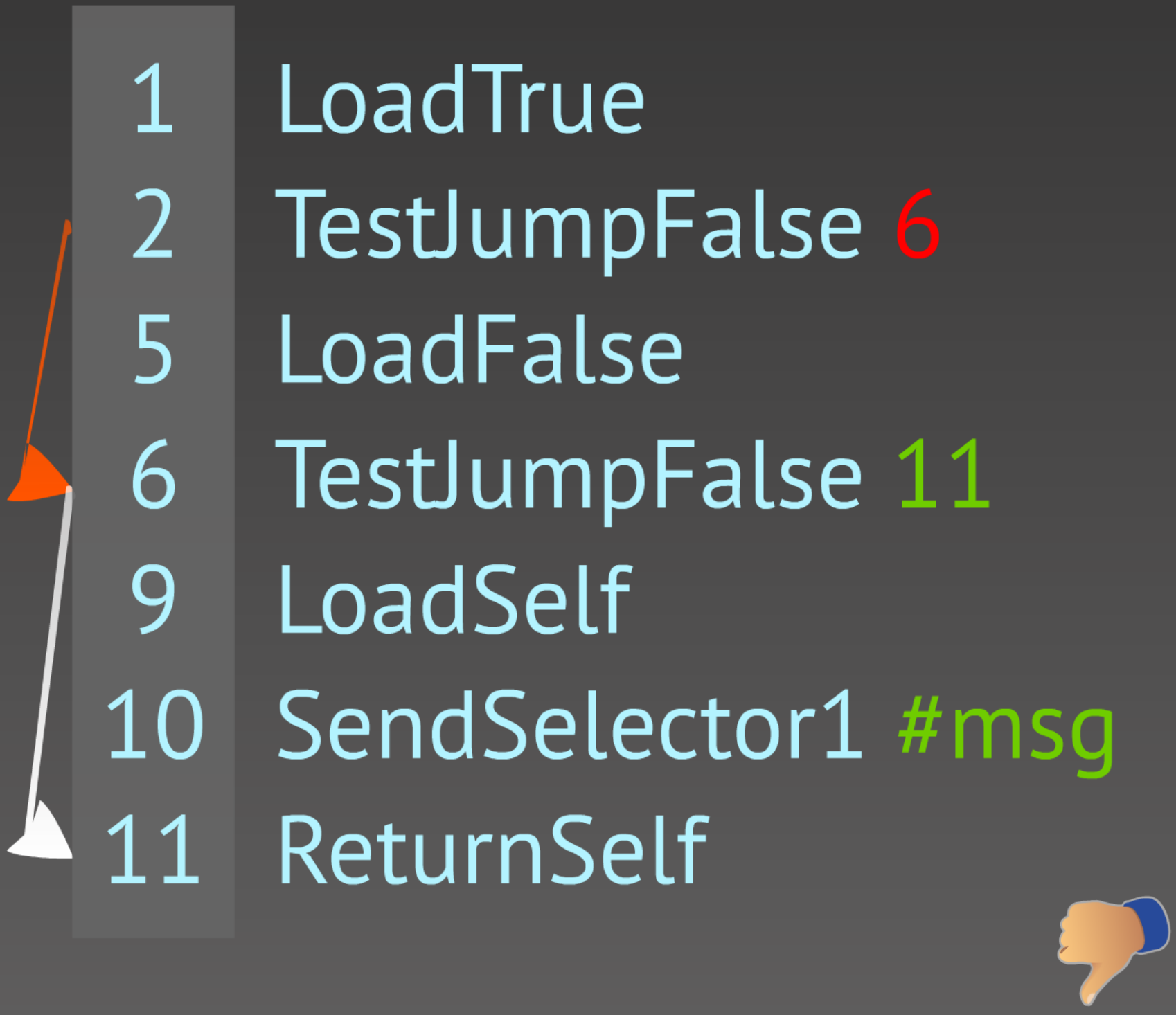
jump

```
msg
    (true and: [false])
        ifTrue: [self msg]
```



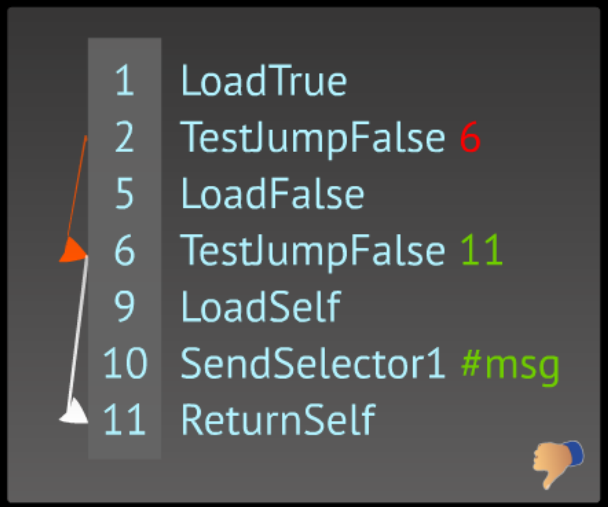


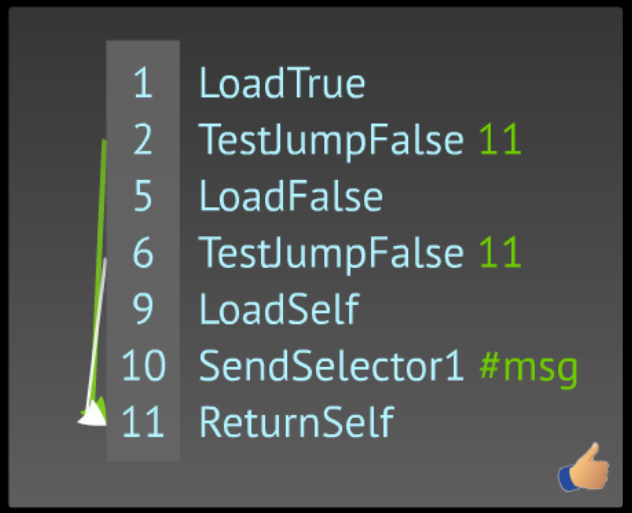
```
msg
    (true and: [false])
        ifTrue: [self msg]
```


1 LoadTrue
2 TestJumpFalse 6
5 LoadFalse
6 TestJumpFalse 11
9 LoadSelf
10 SendSelector1 #msg
11 ReturnSelf

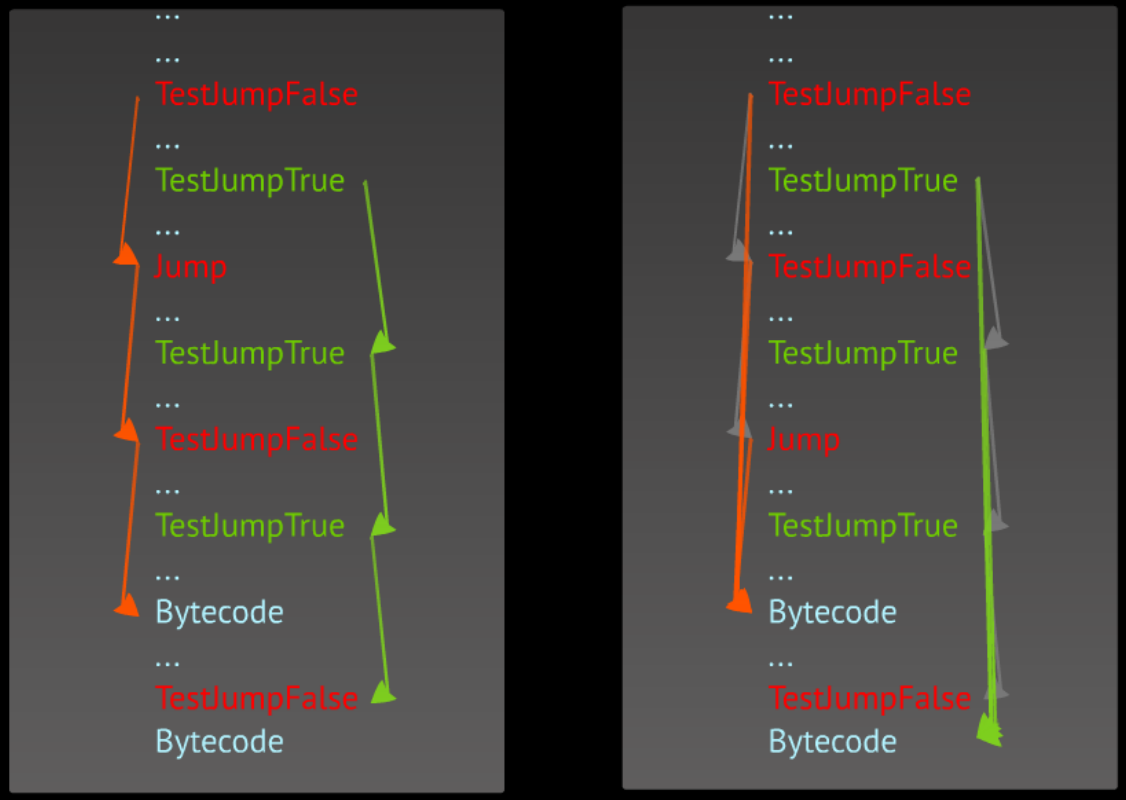
```
 1  LoadTrue
 2  TestJumpFalse 11
 5  LoadFalse
 6  TestJumpFalse 11
 9  LoadSelf
10  SendSelector1 #msg
11  ReturnSelf
```

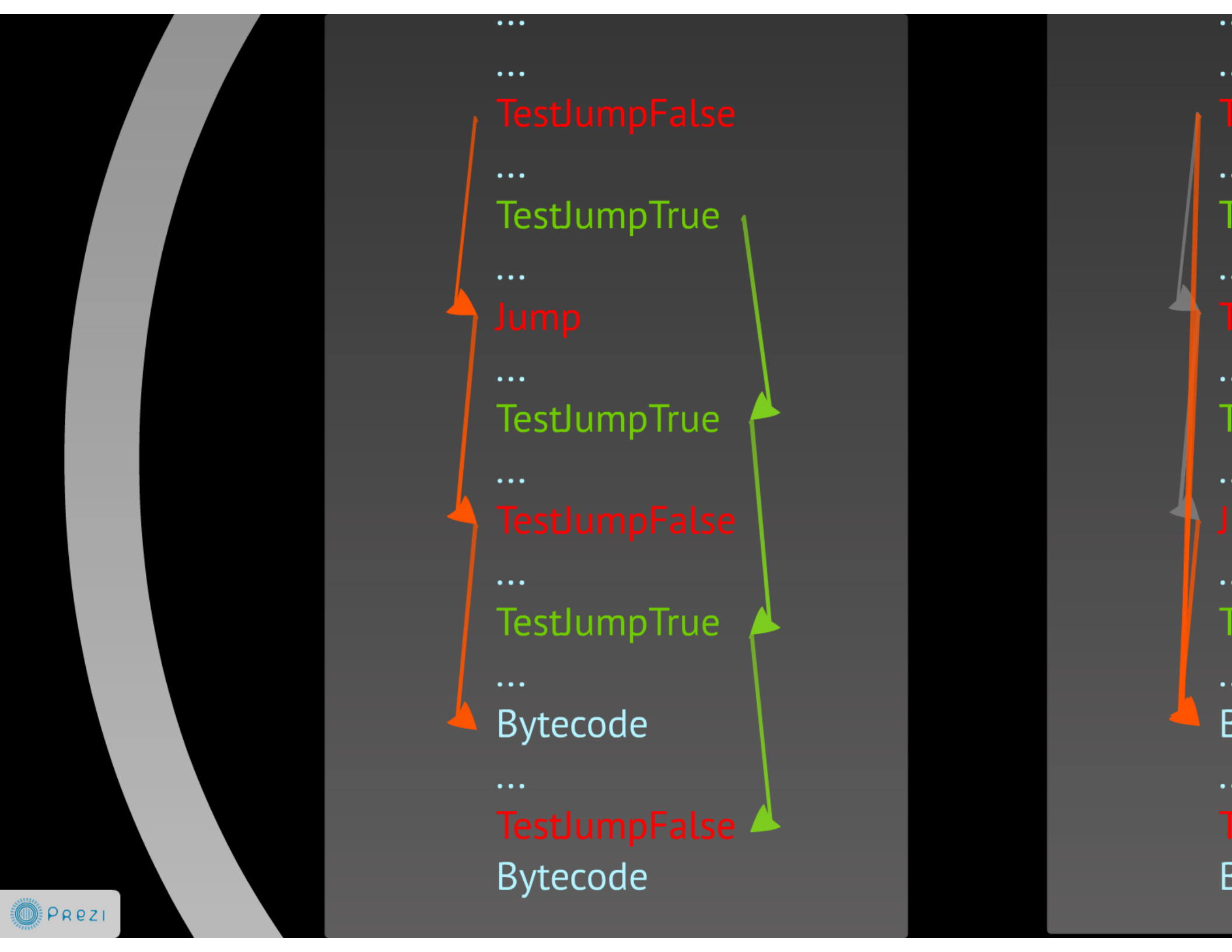
```
msg
    (true and: [false])
        ifTrue: [self msg]
```

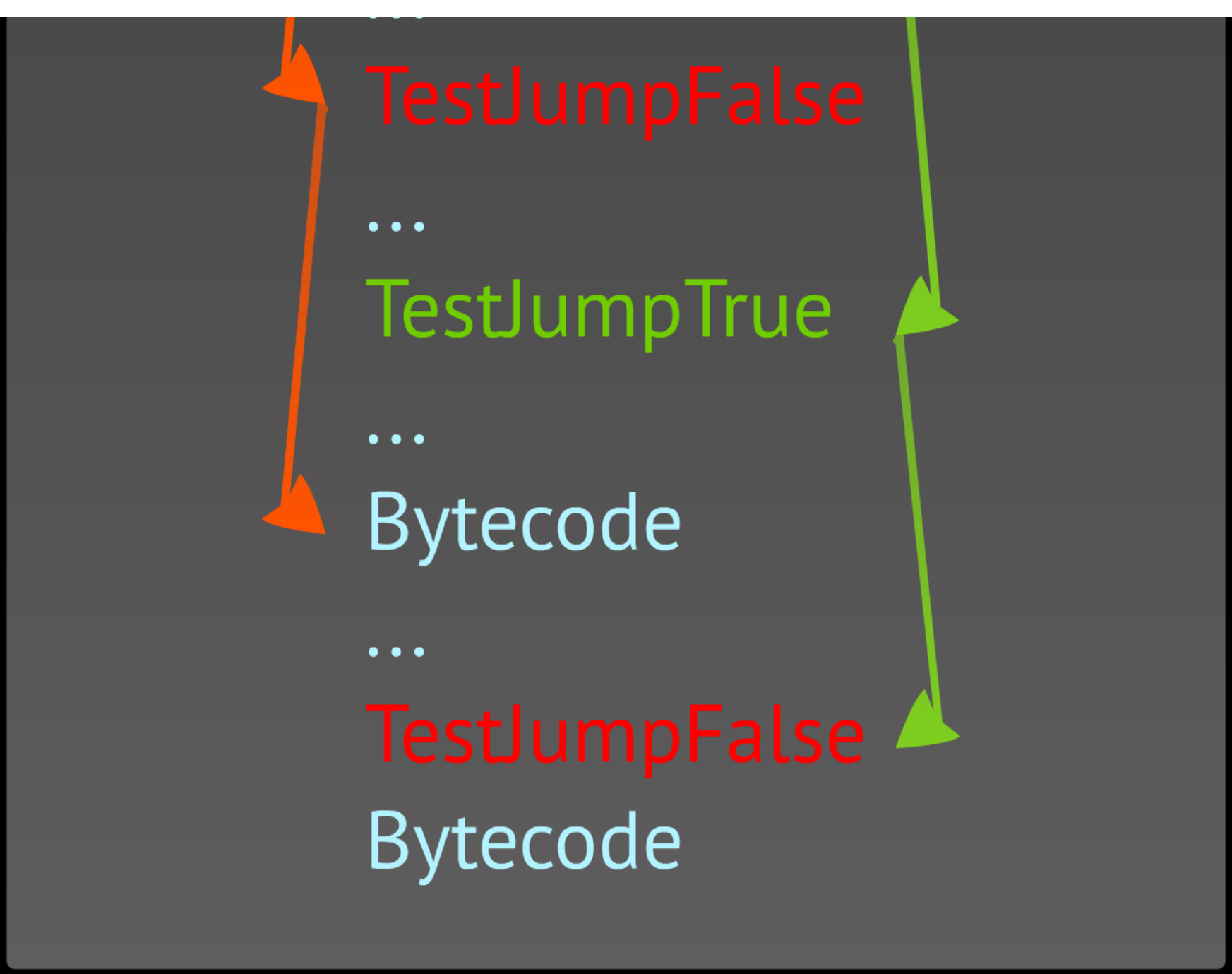


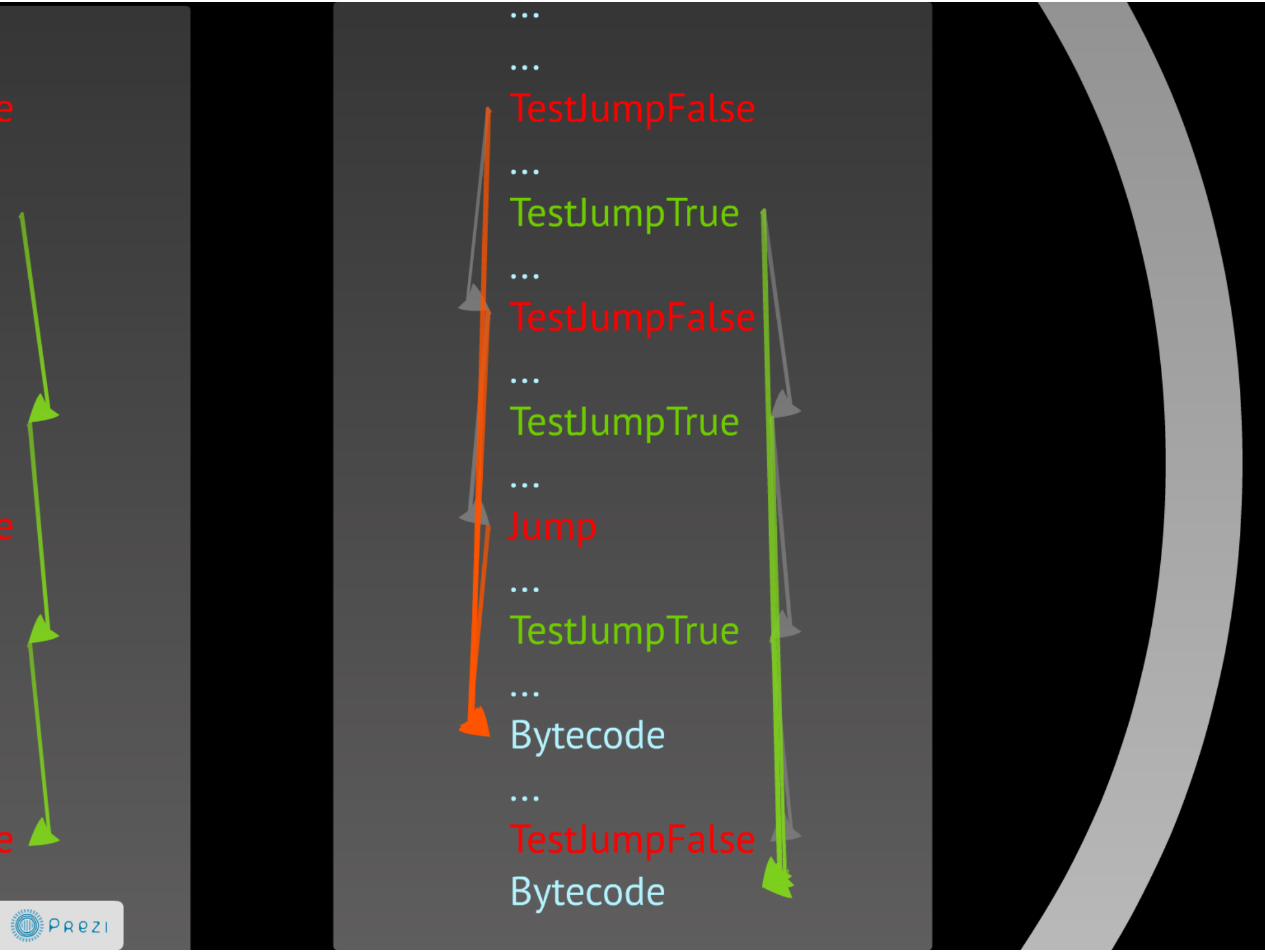


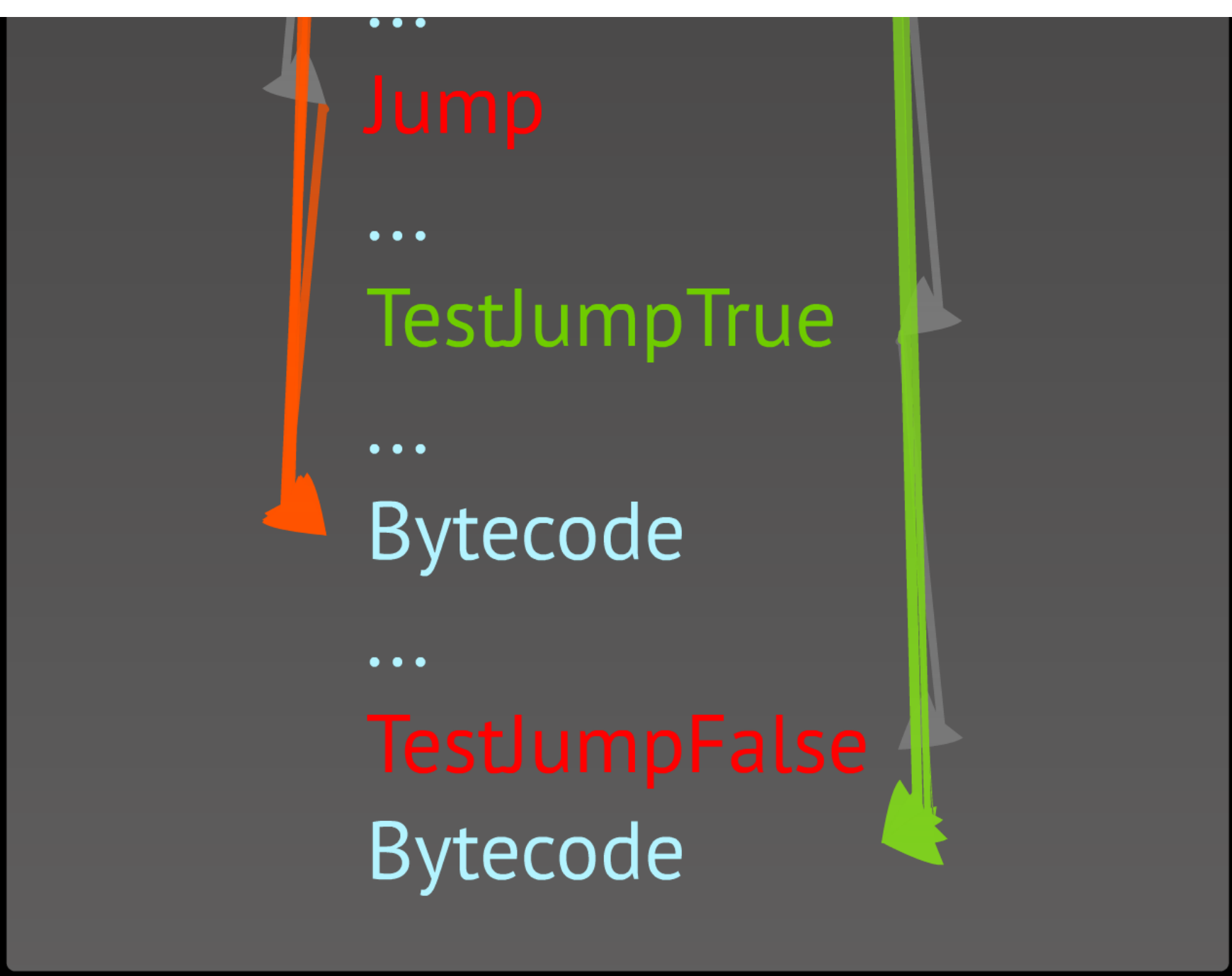
Flow analysis

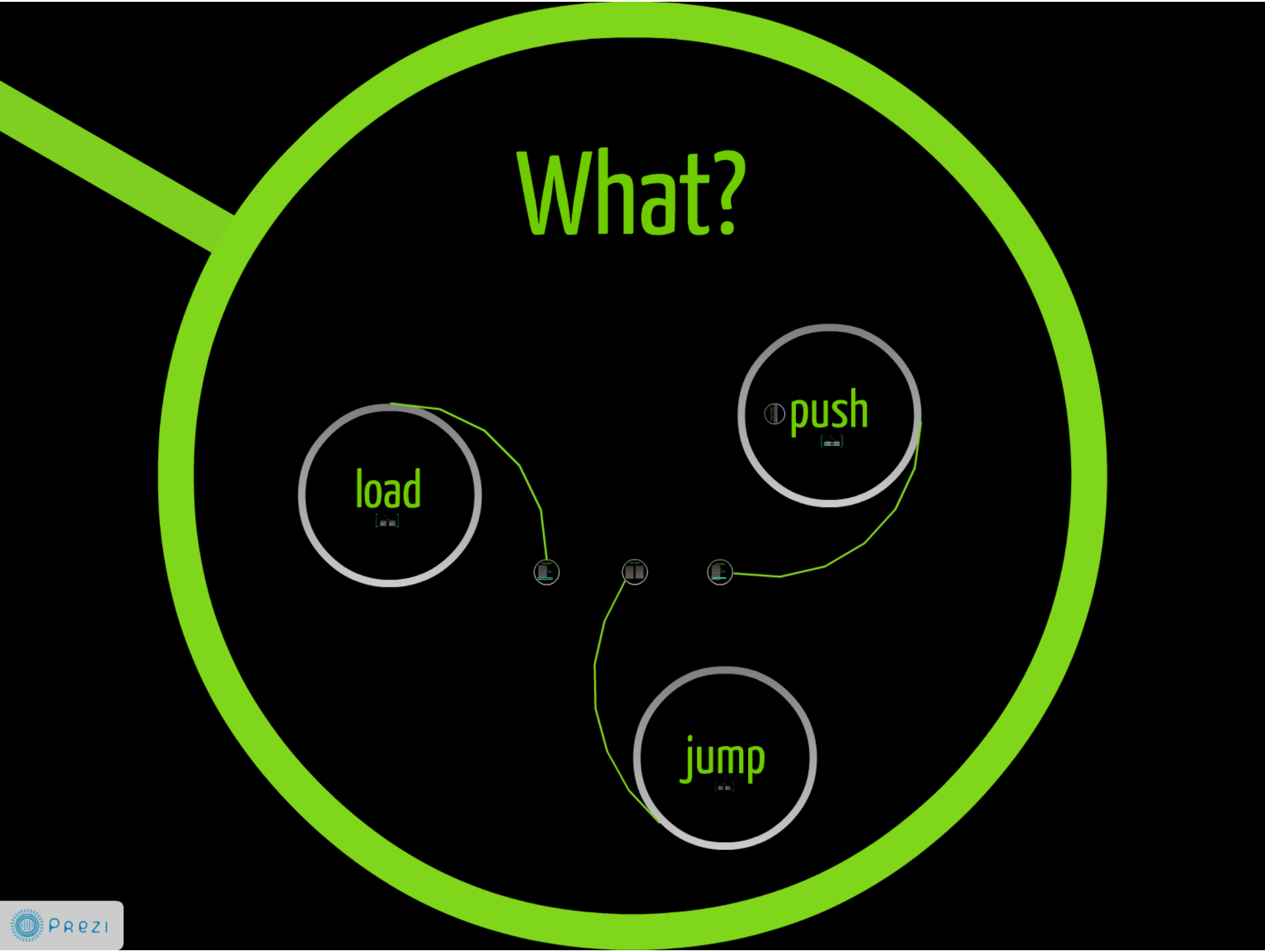



...
...
TestJumpFalse
...
TestJumpTrue
...
Jump
...
TestJumpTrue
...
TestJumpFalse
...
TestJumpTrue
...
Bytecode
...
TestJumpFalse
Bytecode

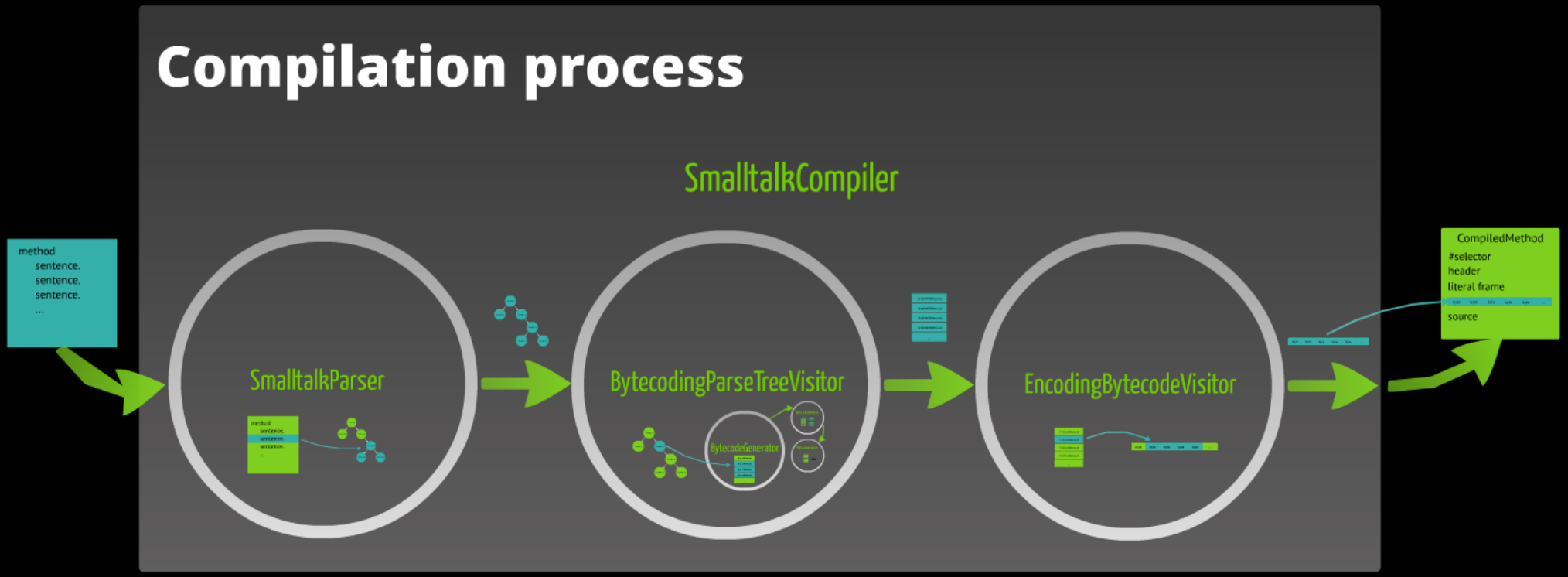

TestJumpFalse
...
TestJumpTrue
...
Bytecode
...
TestJumpFalse
Bytecode


...
...
TestJumpFalse
...
TestJumpTrue
...
TestJumpFalse
...
TestJumpTrue
...
Jump
...
TestJumpTrue
...
Bytecode
...
TestJumpFalse
Bytecode


...
Jump
...
TestJumpTrue
...
Bytecode
...
TestJumpFalse
Bytecode


What?
load
push
jump

Compilation process



method

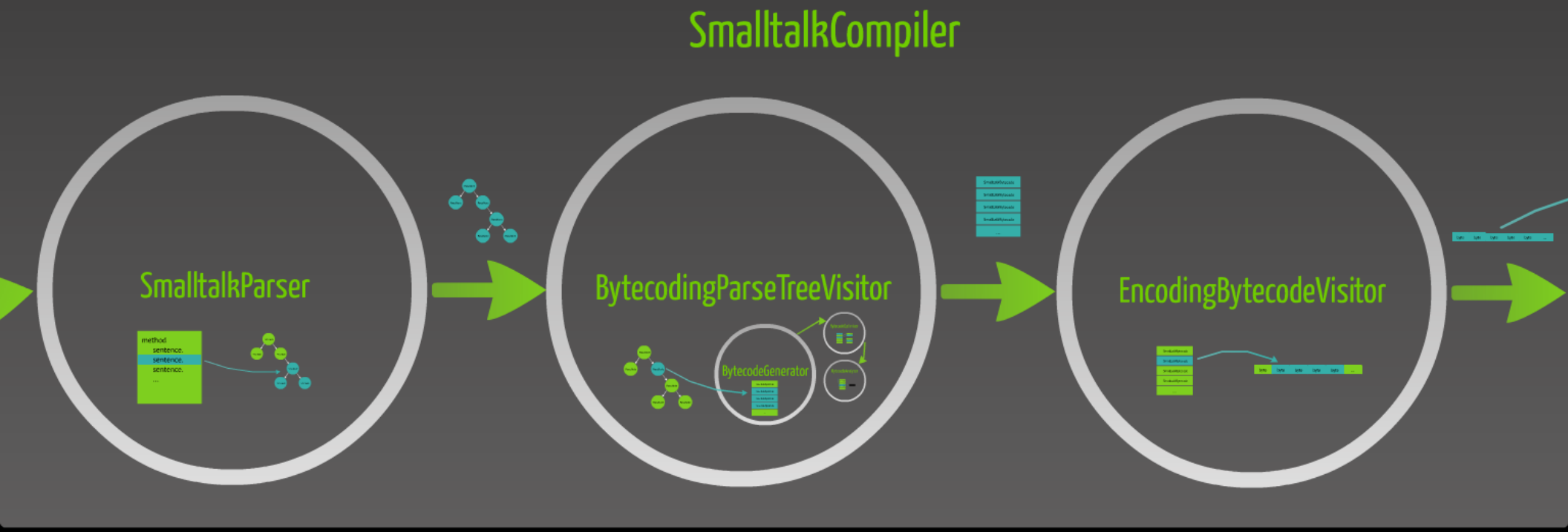
sentence.

sentence.

sentence.

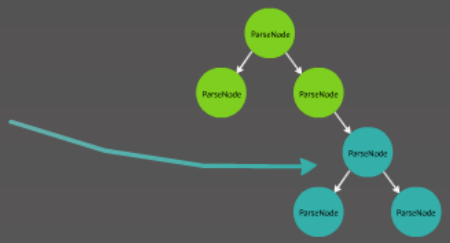
...

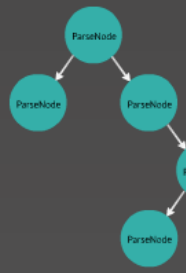
Compilation process



SmalltalkParser

```
method
    sentence.
    sentence.
    sentence.
    ...
```





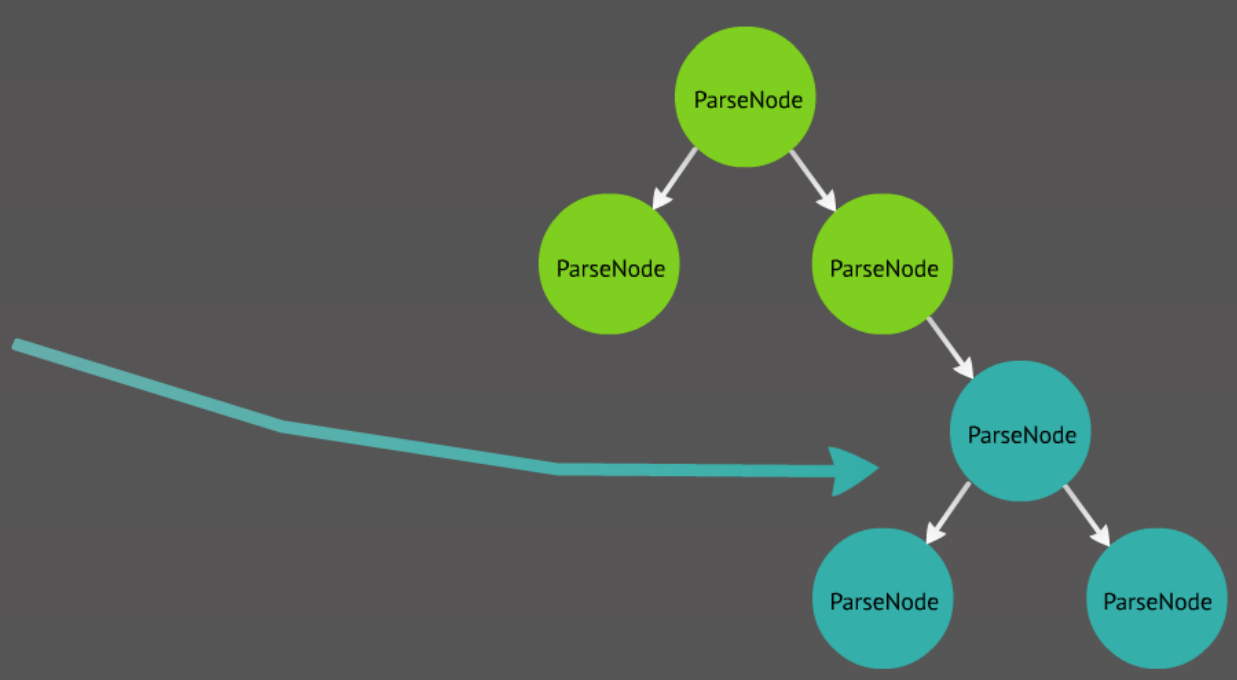
method

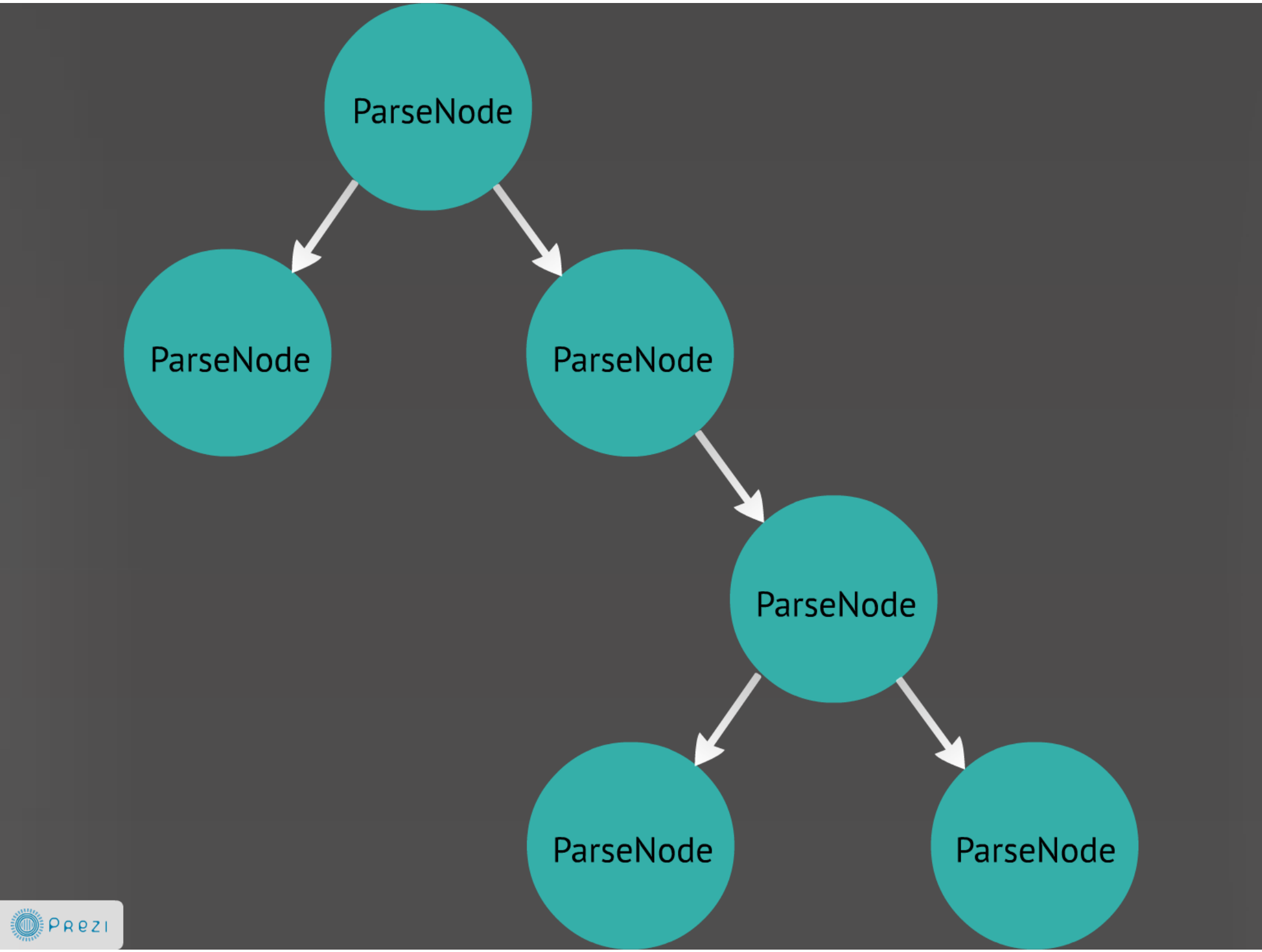
sentence.

sentence.

sentence.

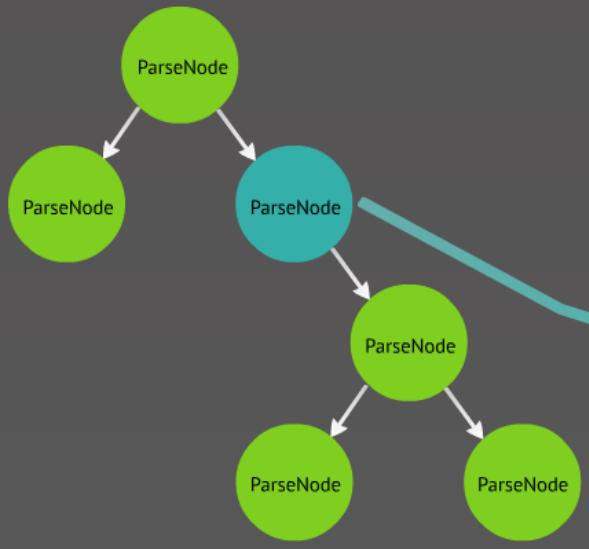
...




ParseNode
ParseNode
ParseNode
ParseNode
ParseNode
ParseNode


ParseNode
ParseNode
ParseNode
BytecodingParseTreeVisitor
ParseNode
ParseNode
ParseNode
ParseNode
ParseNode
BytecodeGenerator
SmalltalkBytecode
SmalltalkBytecode
SmalltalkBytecode
SmalltalkBytecode
...
BytecodeOptimizer
BytecodeAnalyzer
SmalltalkB
SmalltalkB
SmalltalkB
SmalltalkB
...

ytecodingParseTreeVisit

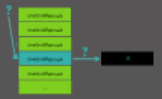


BytecodeGenerator

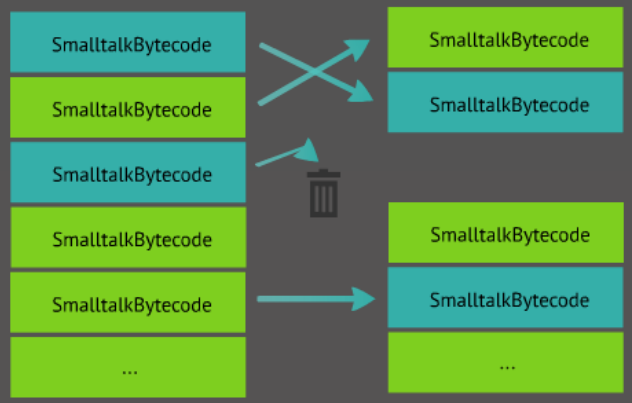
- SmalltalkBytecode
- SmalltalkBytecode
- SmalltalkBytecode
- SmalltalkBytecode
- ...

BytecodeOptimizer

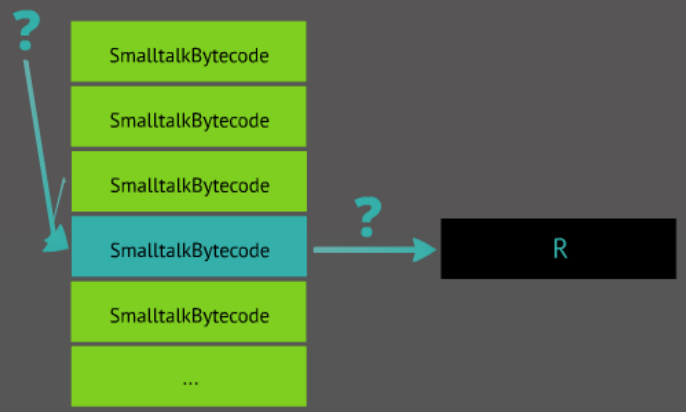
BytecodeAnalyzer

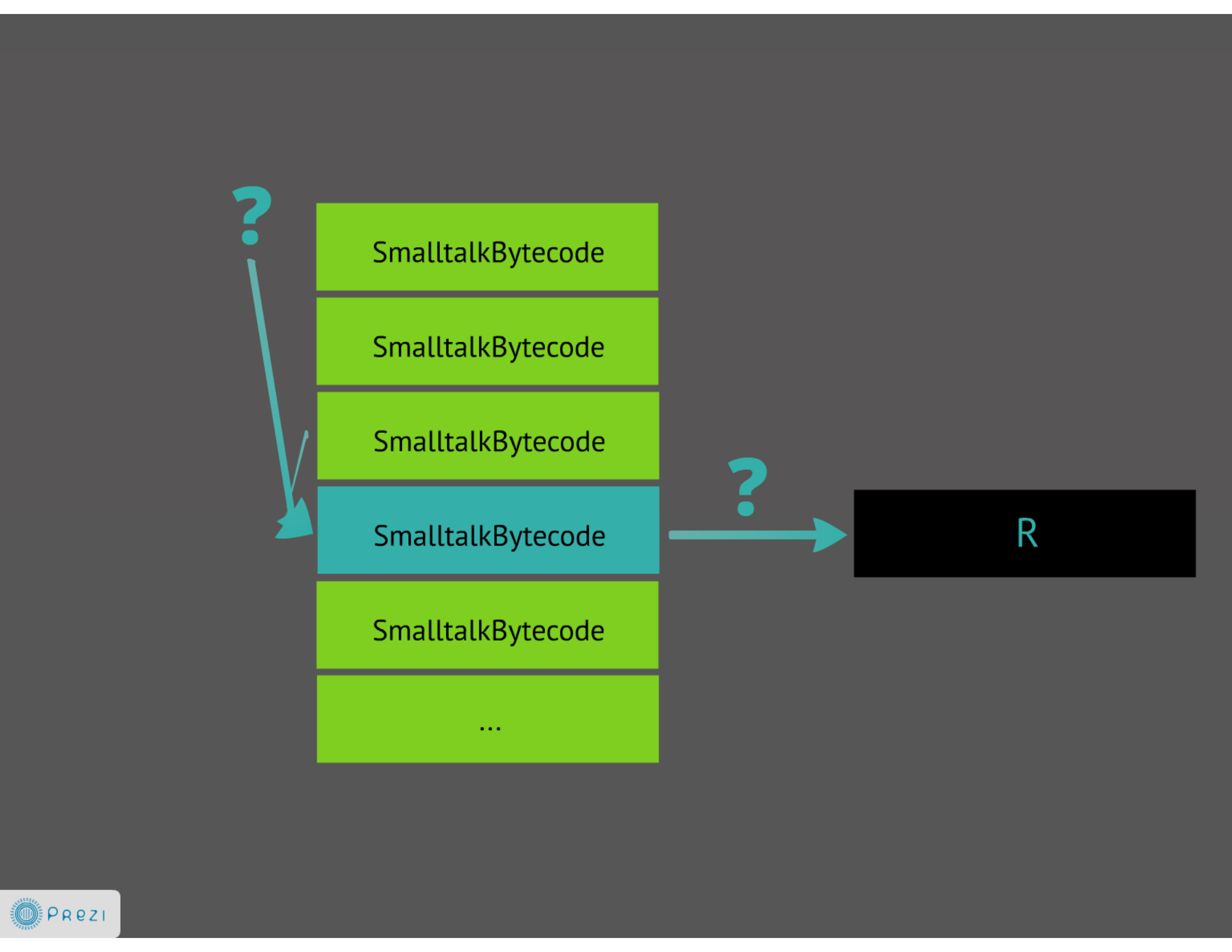


BytecodeOptimizer

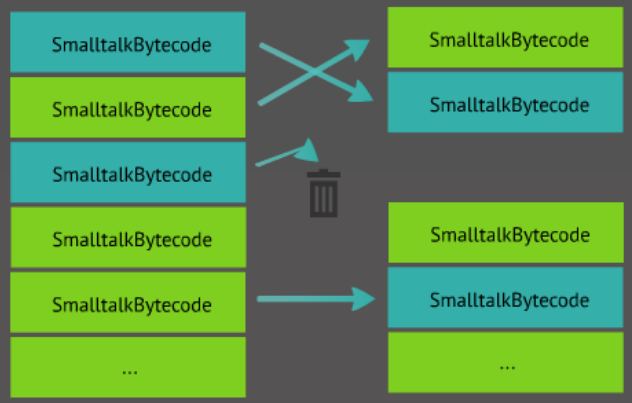


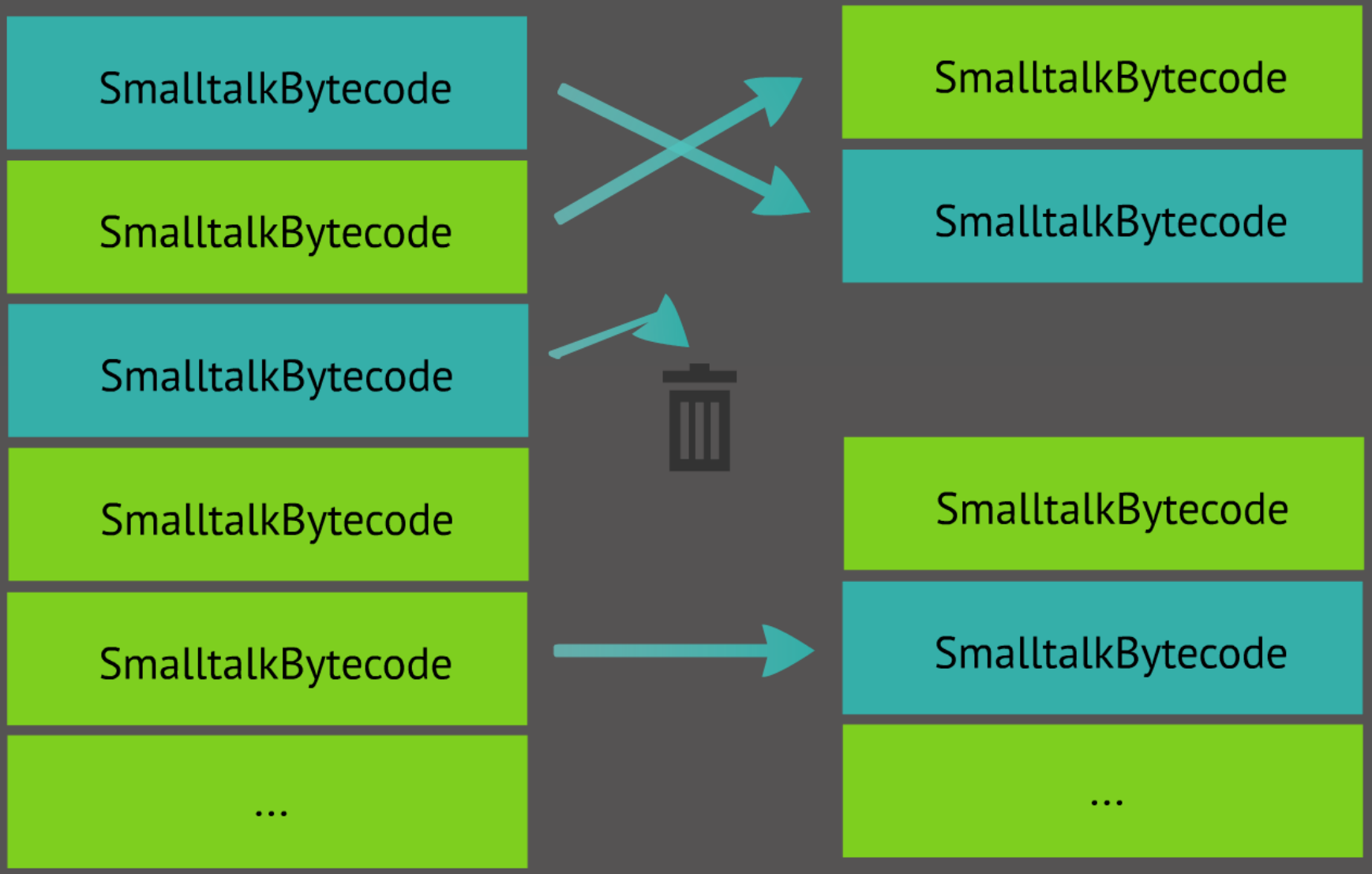
BytecodeAnalyzer



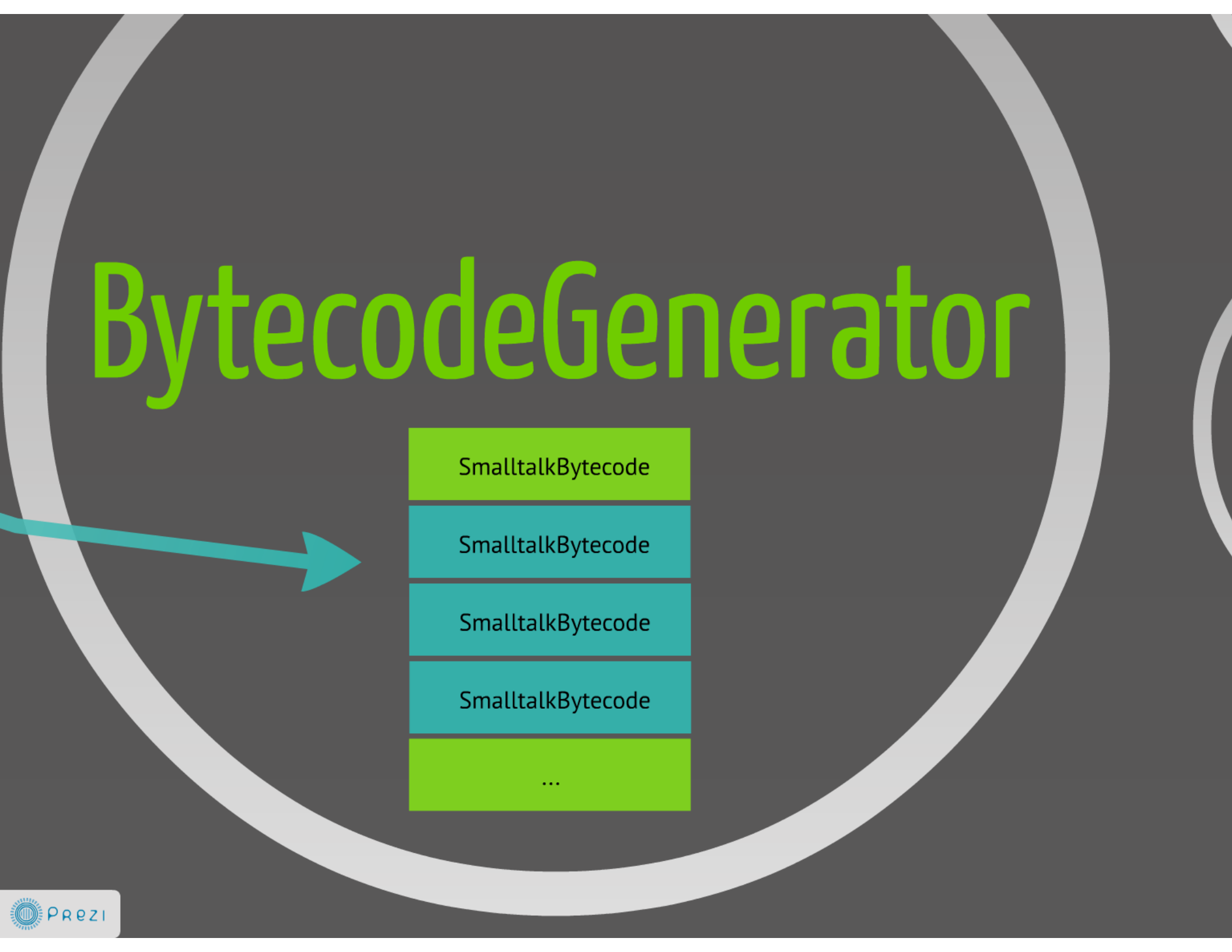

?
SmalltalkBytecode
SmalltalkBytecode
SmalltalkBytecode
SmalltalkBytecode
SmalltalkBytecode
...
?
R

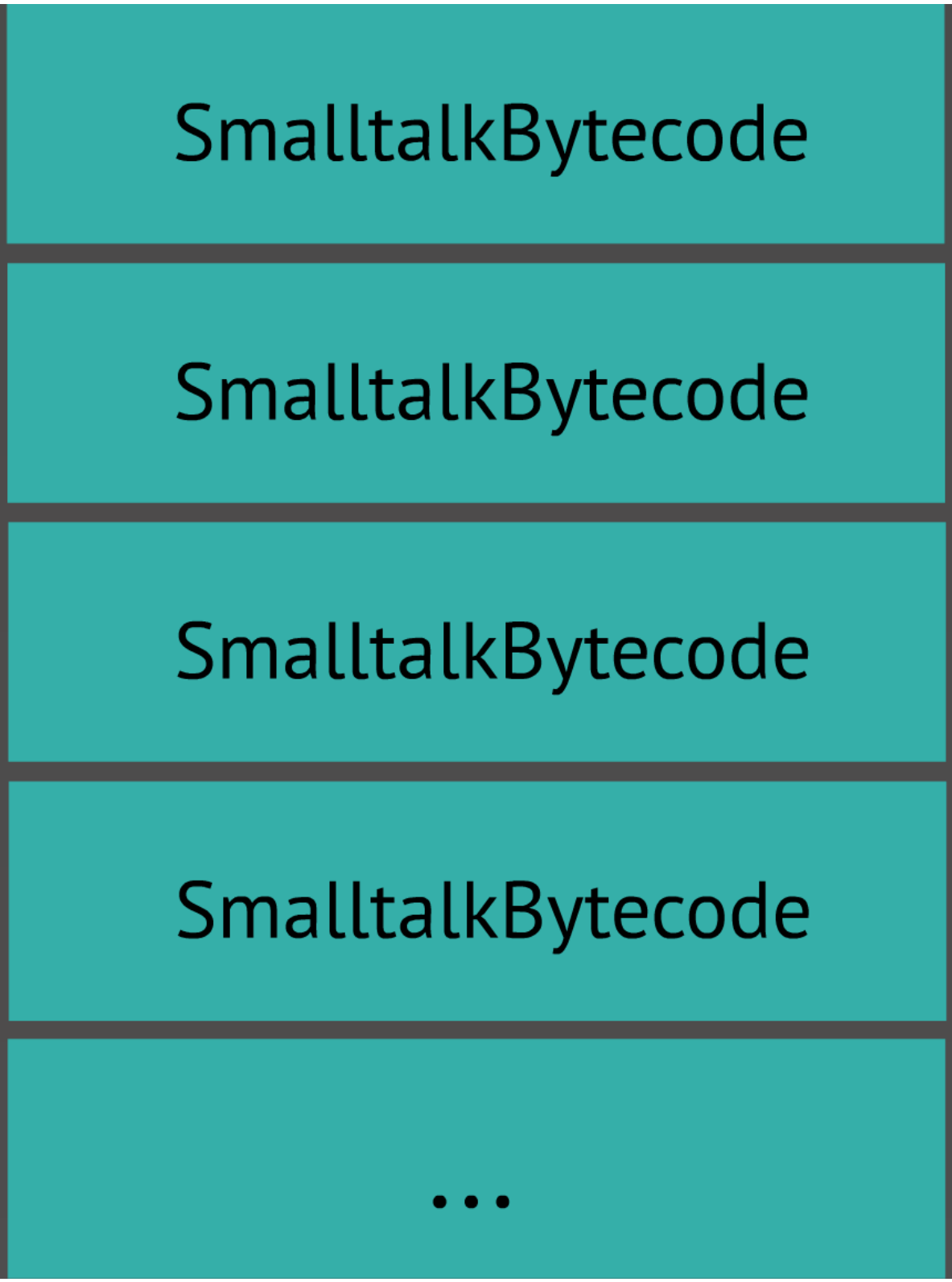
BytecodeOptimizer

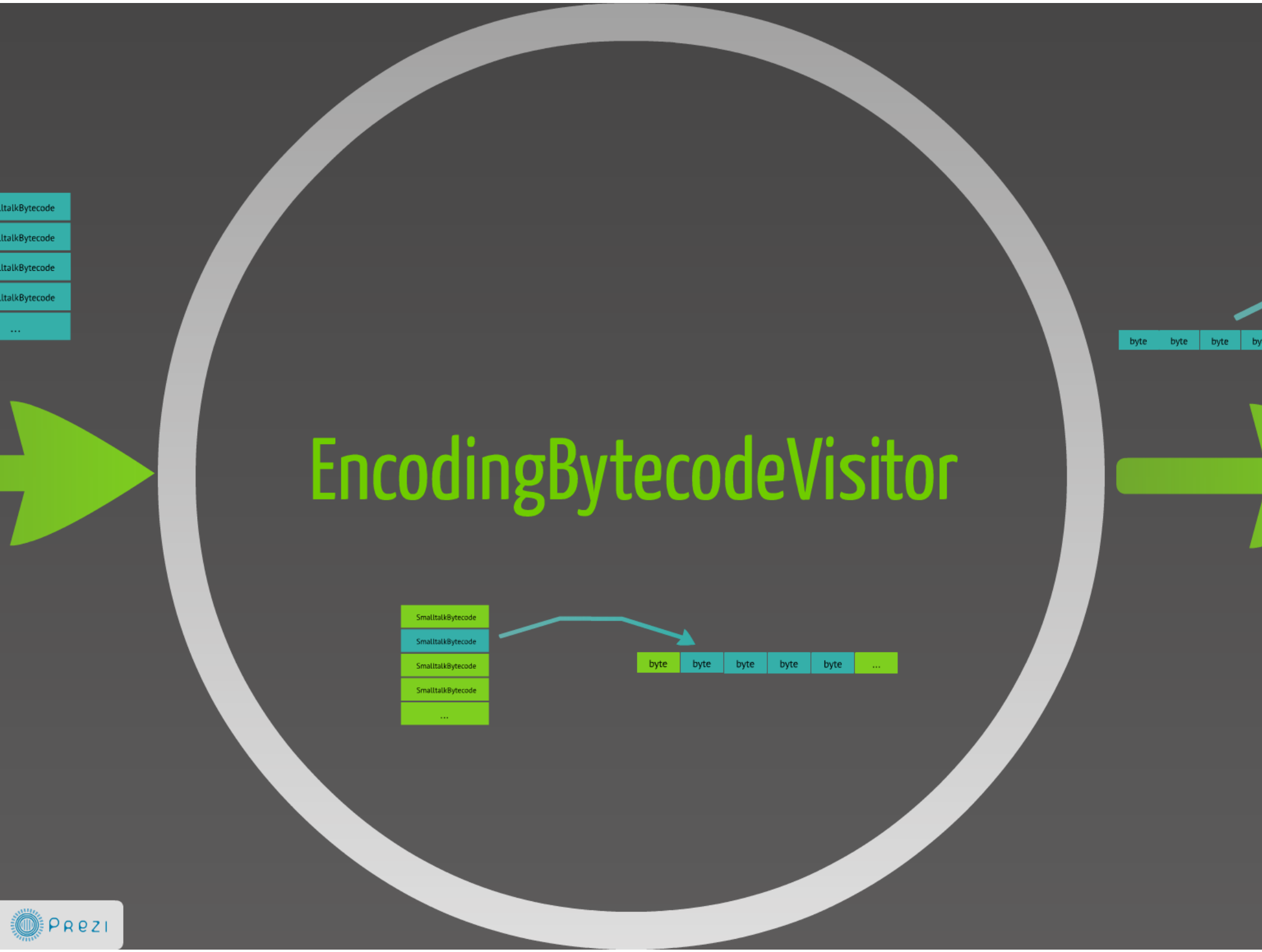



SmalltalkBytecode
SmalltalkBytecode
SmalltalkBytecode
SmalltalkBytecode
SmalltalkBytecode
...
SmalltalkBytecode
SmalltalkBytecode
SmalltalkBytecode
SmalltalkBytecode
...

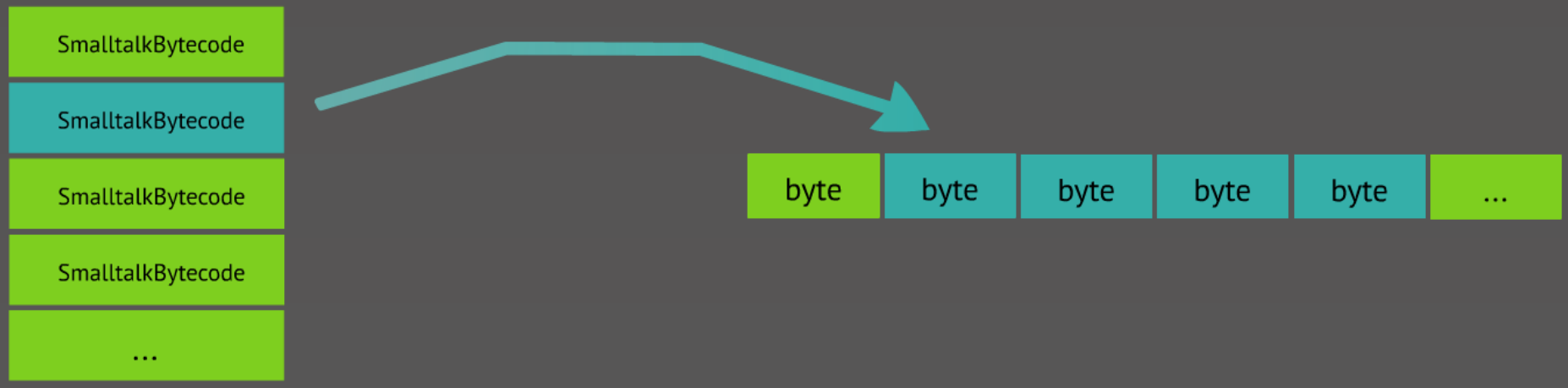
BytecodeGenerator

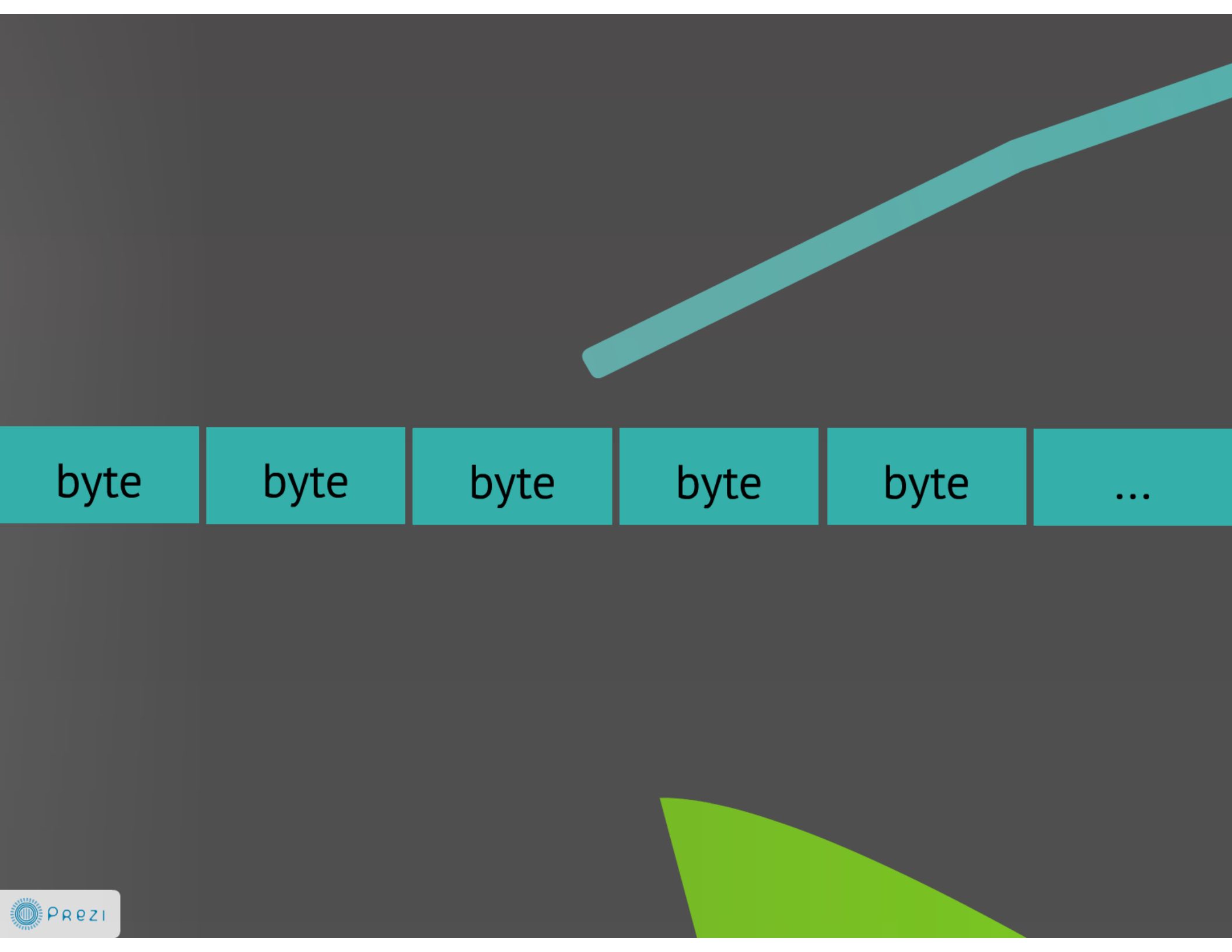



SmalltalkBytecode
SmalltalkBytecode
SmalltalkBytecode
SmalltalkBytecode
...

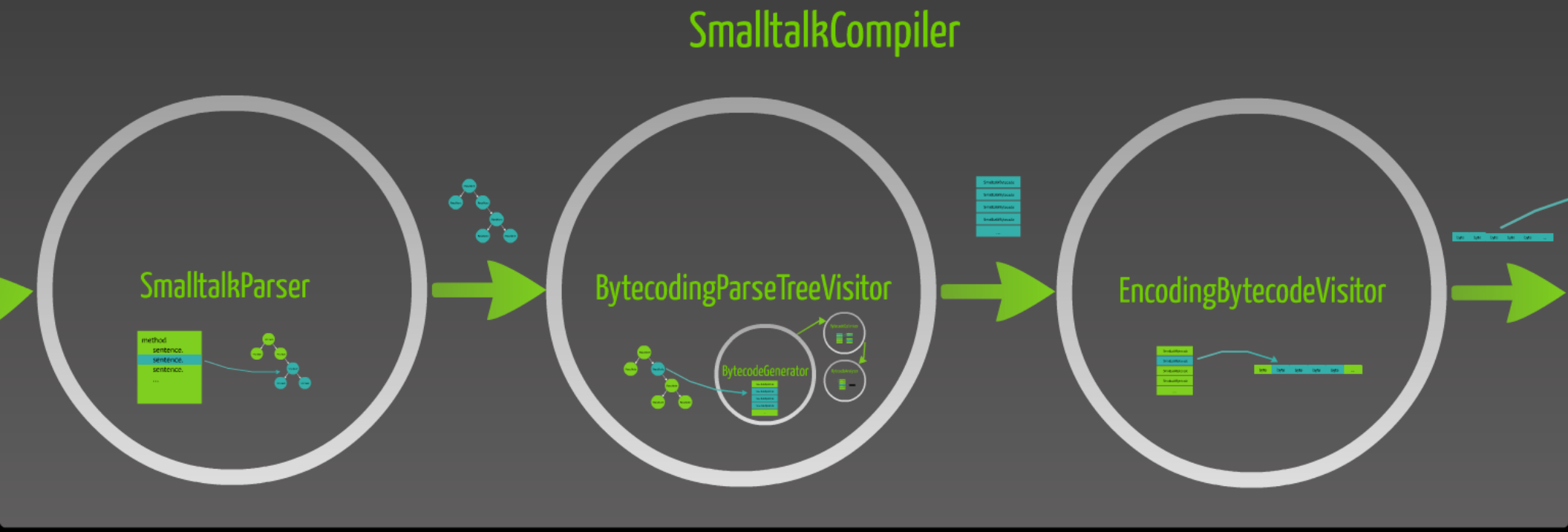

ltalkBytecode
ltalkBytecode
ltalkBytecode
ltalkBytecode
...
EncodingBytecodeVisitor
SmalltalkBytecode
SmalltalkBytecode
SmalltalkBytecode
SmalltalkBytecode
...
byte
byte
byte
byte
byte
...
byte
byte
byte

ncodingBytecodeVisito




byte
byte
byte
byte
byte
...

Compilation process



CompiledMethod

#selector

header

literal frame

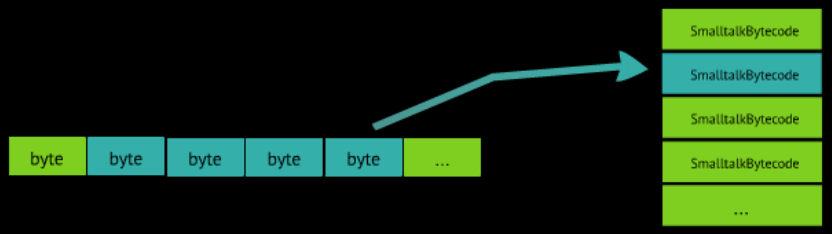
source

Framework objects

SmalltalkBytecode

BytecodeParser

(a PetitParser @ Lukas Renggli)

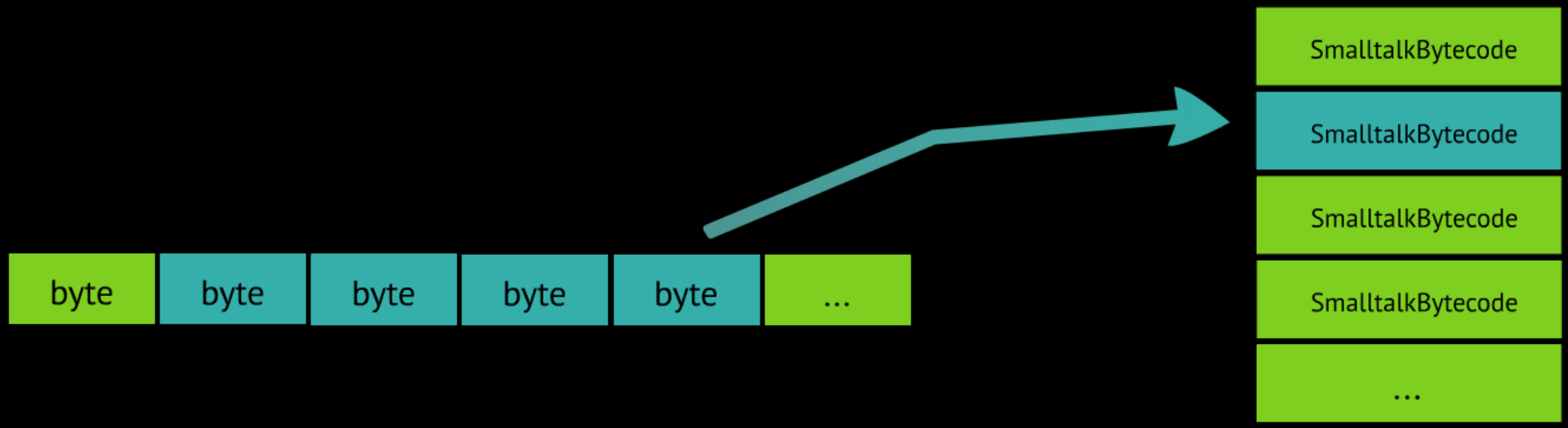


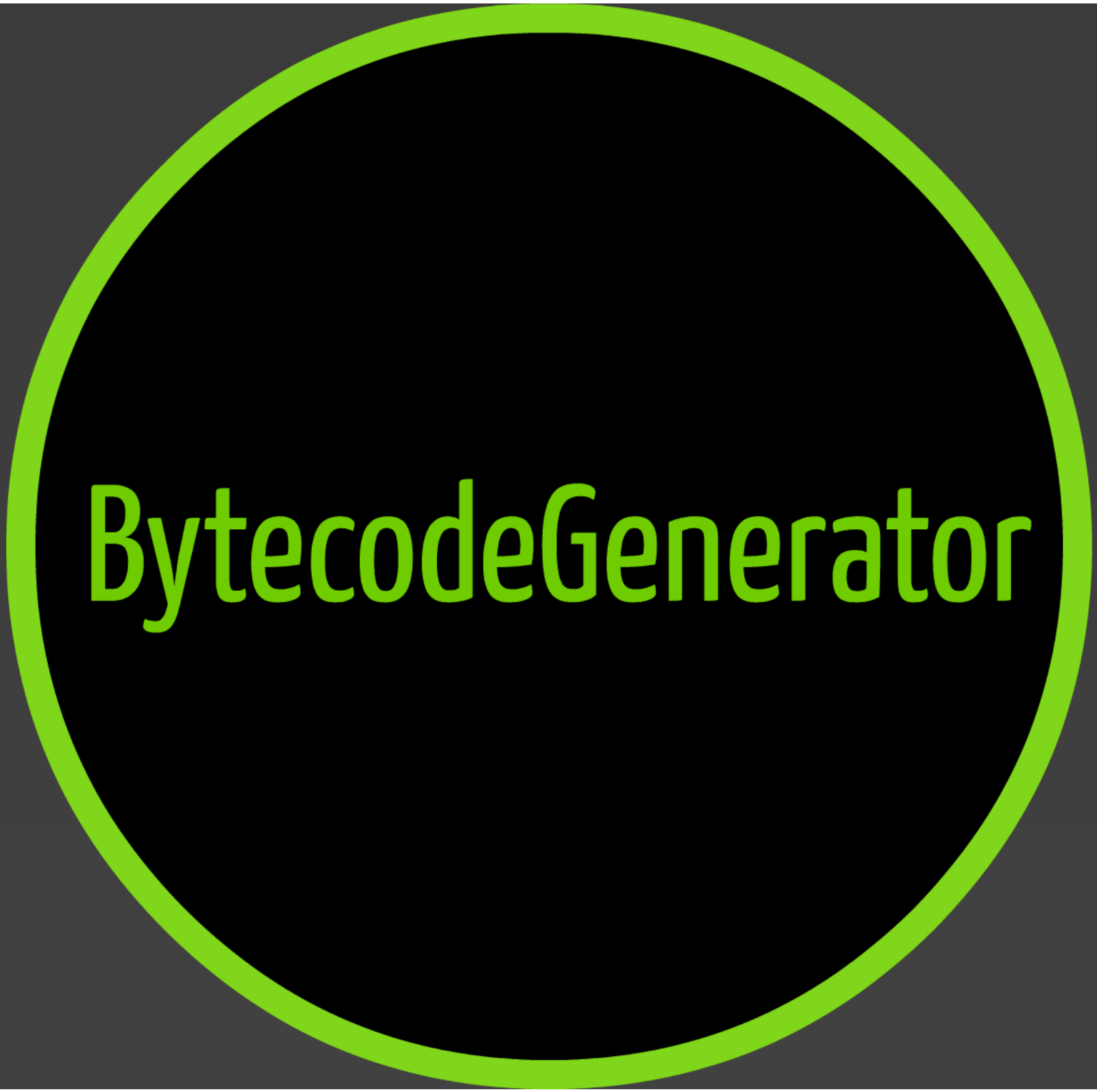
Byteco

(a PetitParser @ Lukas Renggli)

bytecodeParse

titParser @ Lukas Renggli)

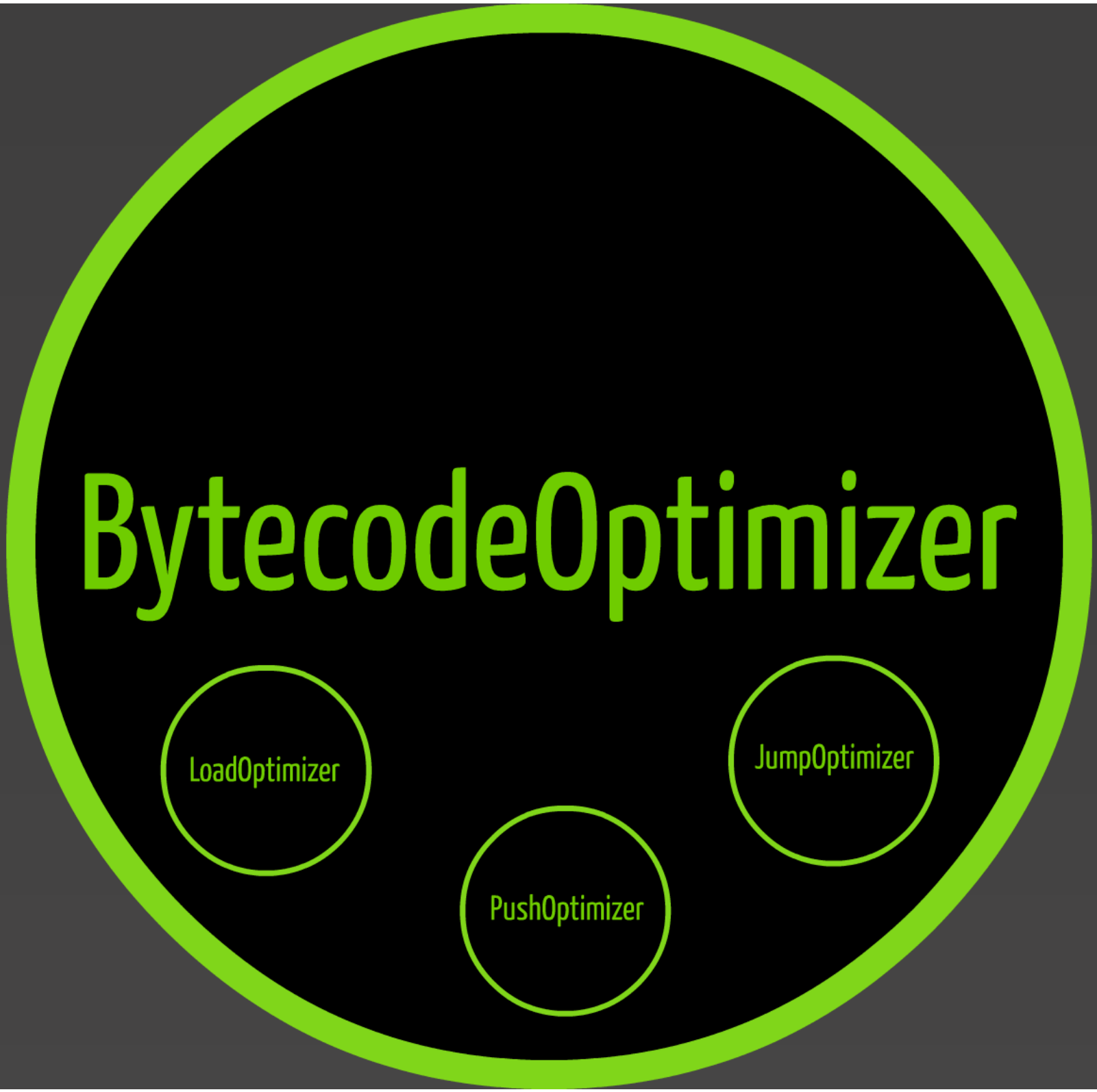





BytecodeVisitor
PrintingBytecodeVisitor
EncodingBytecodeVisitor

PrintingBytecodeVisitor

EncodingBytecodeVisitor

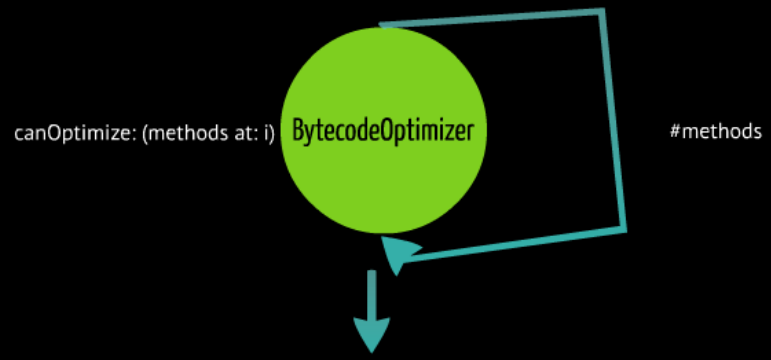

BytecodeOptimizer
LoadOptimizer
JumpOptimizer
PushOptimizer

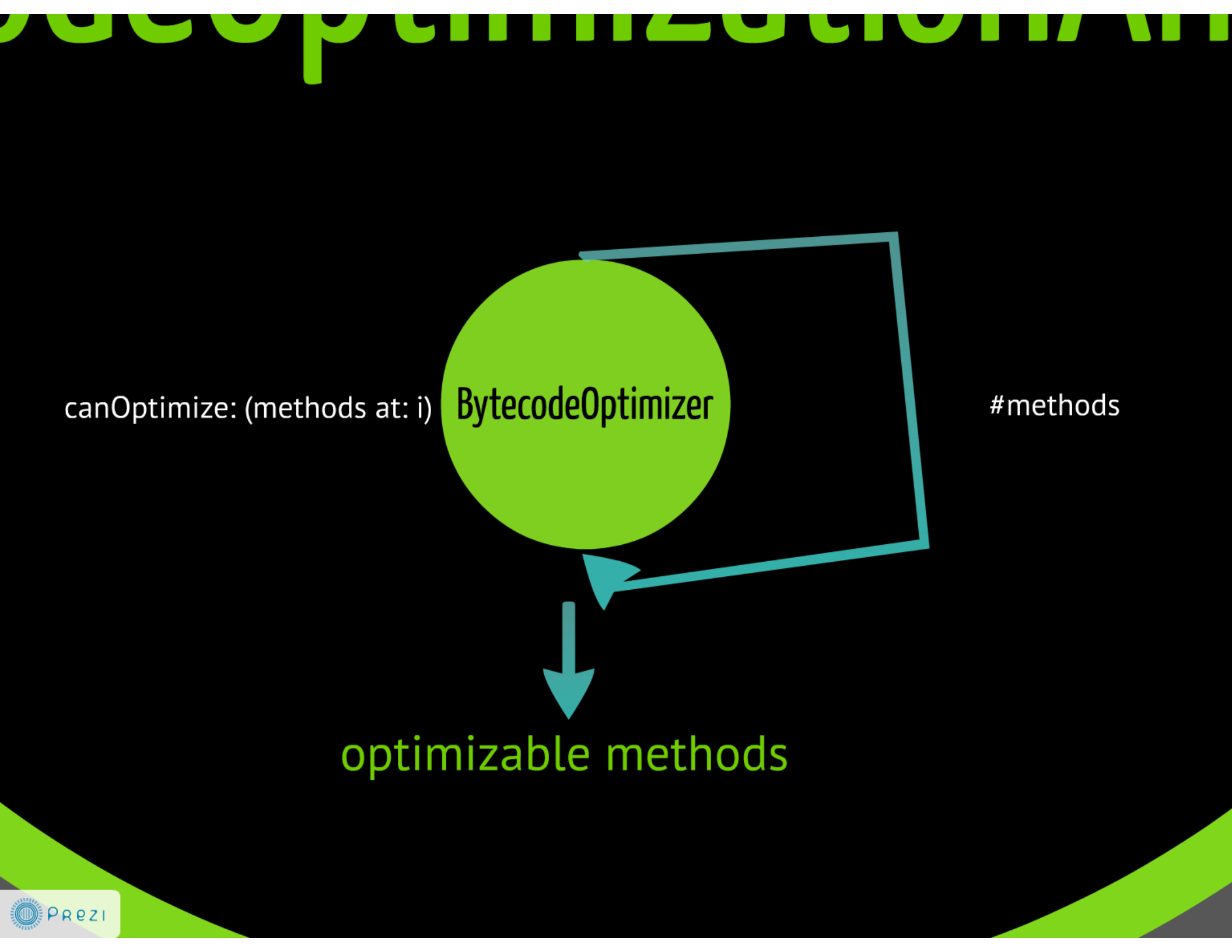
LoadOptimizer

JumpOptimizer

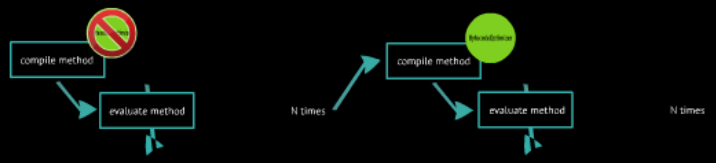
BytecodeAnalyzer

BytecodeOptimizationAnalyzer

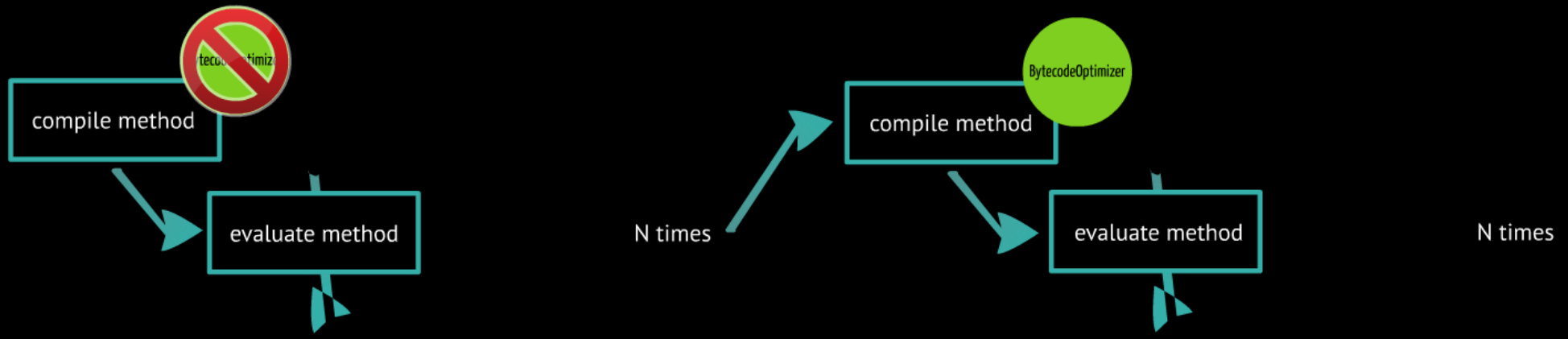



canOptimize: (methods at: i)
BytecodeOptimizer
#methods
optimizable methods

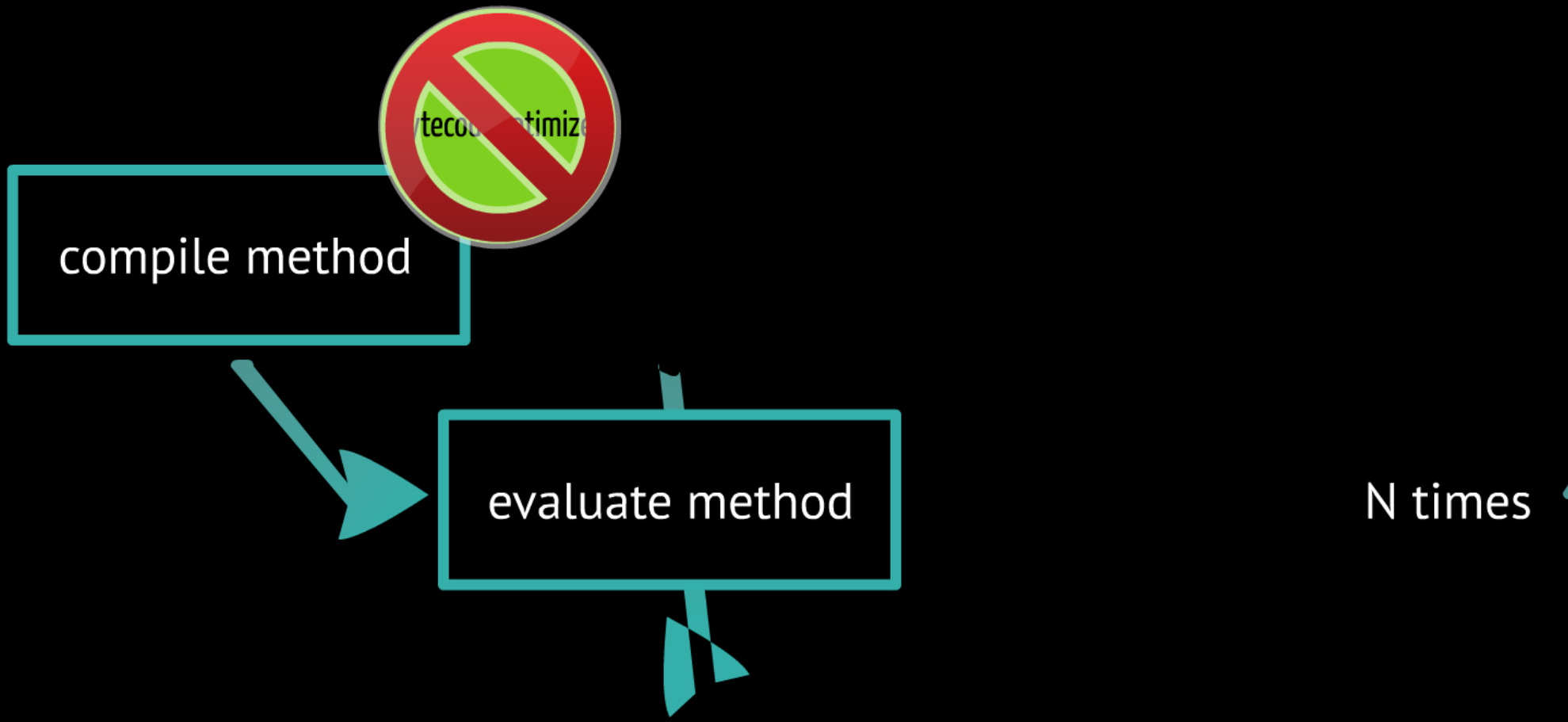
BytecodeOptimizationBenchmark

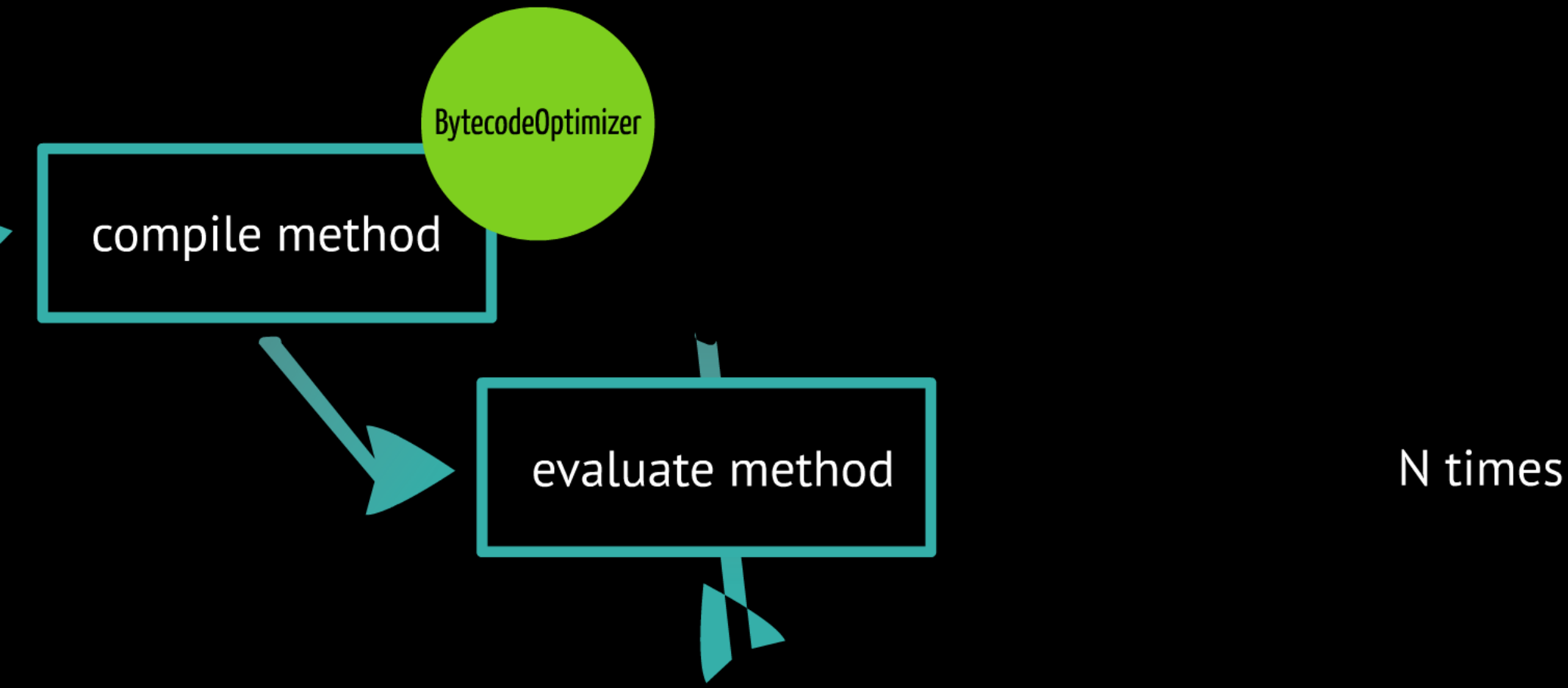


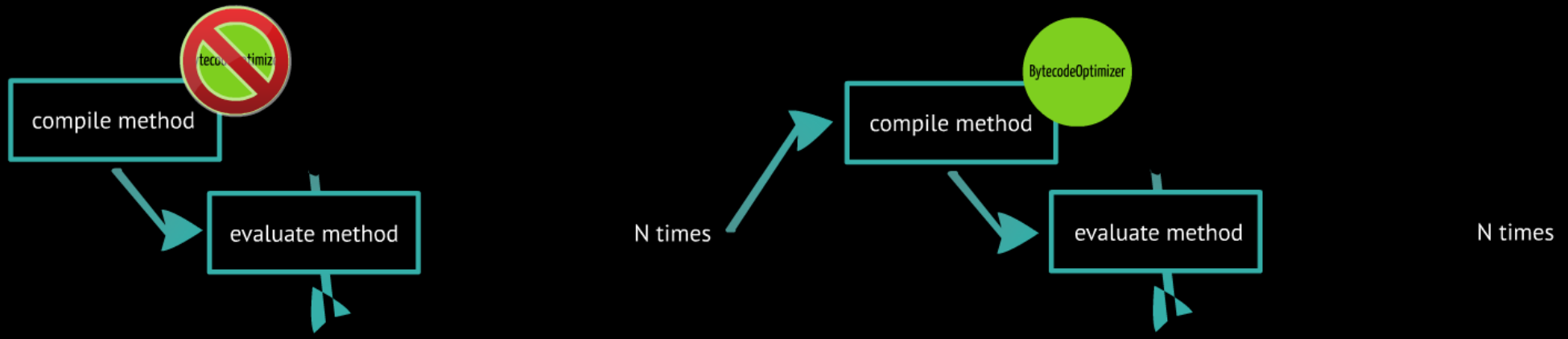
#optimization statistics



#optimization statistics


compile method
evaluate method
N times

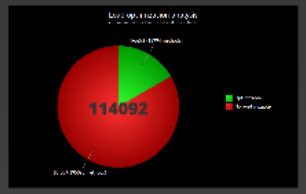

BytecodeOptimizer
compile method
evaluate method
N times

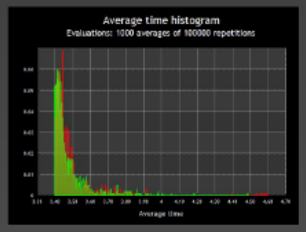


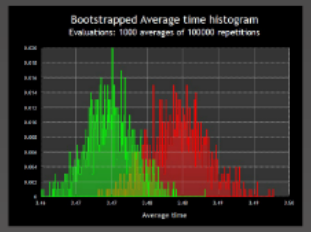
#optimization statistics

Statistics

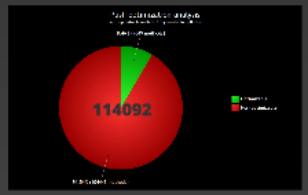
Load optimization

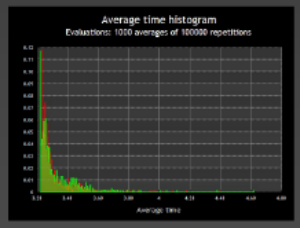


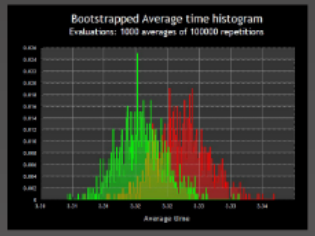




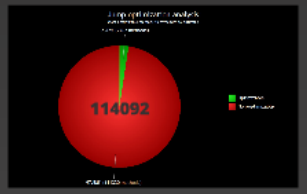
Push optimization

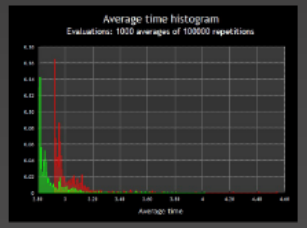


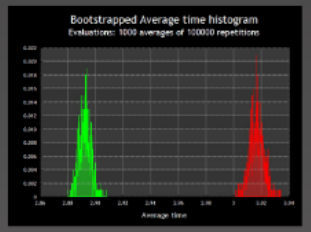




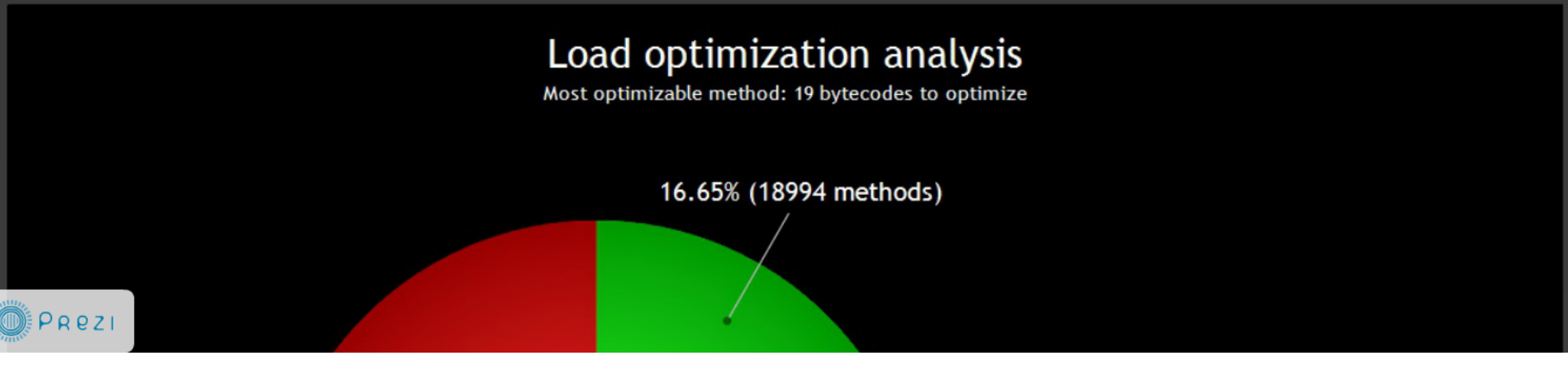
Jump optimization





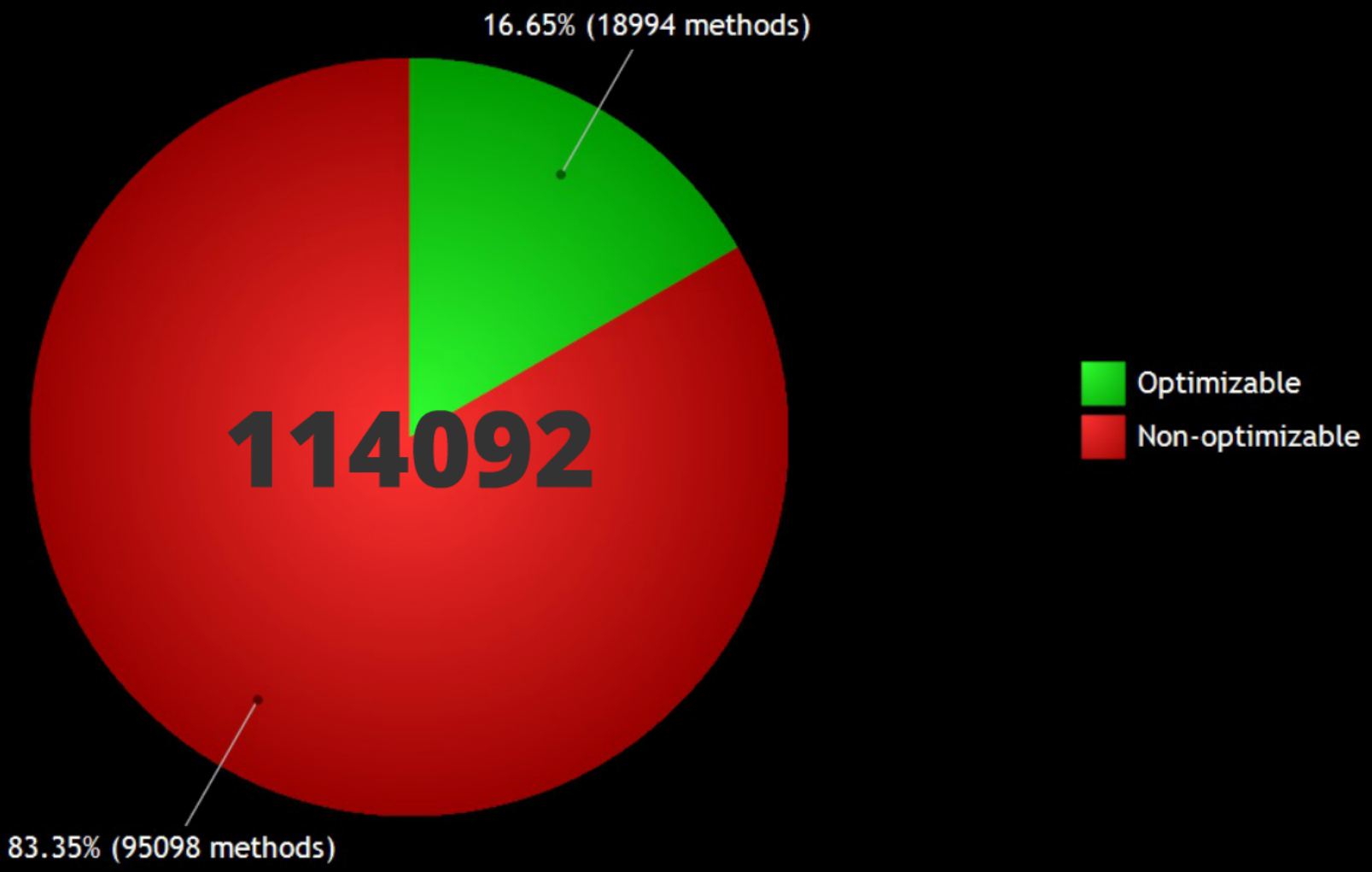


Load optimization



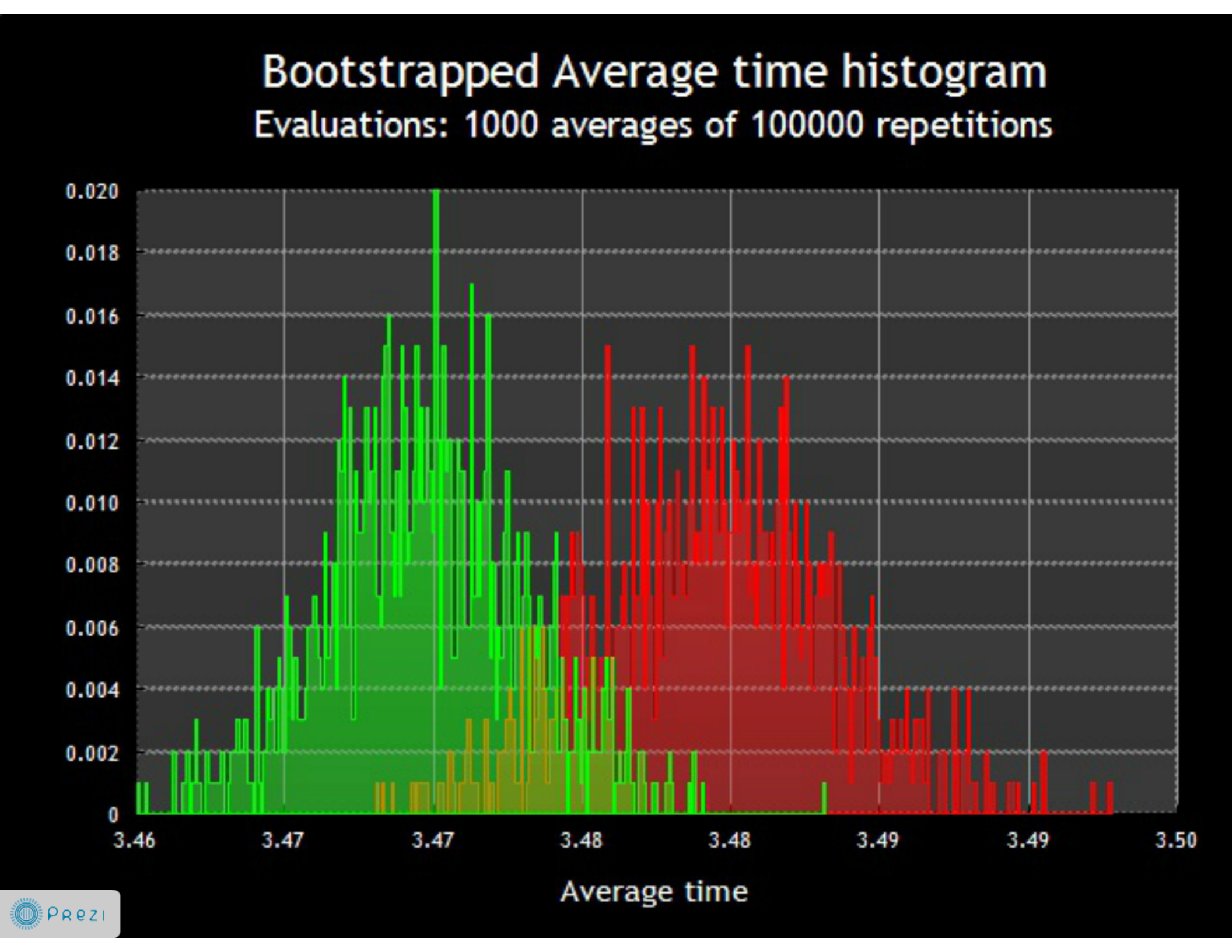
Load optimization analysis

Most optimizable method: 19 bytecodes to optimize

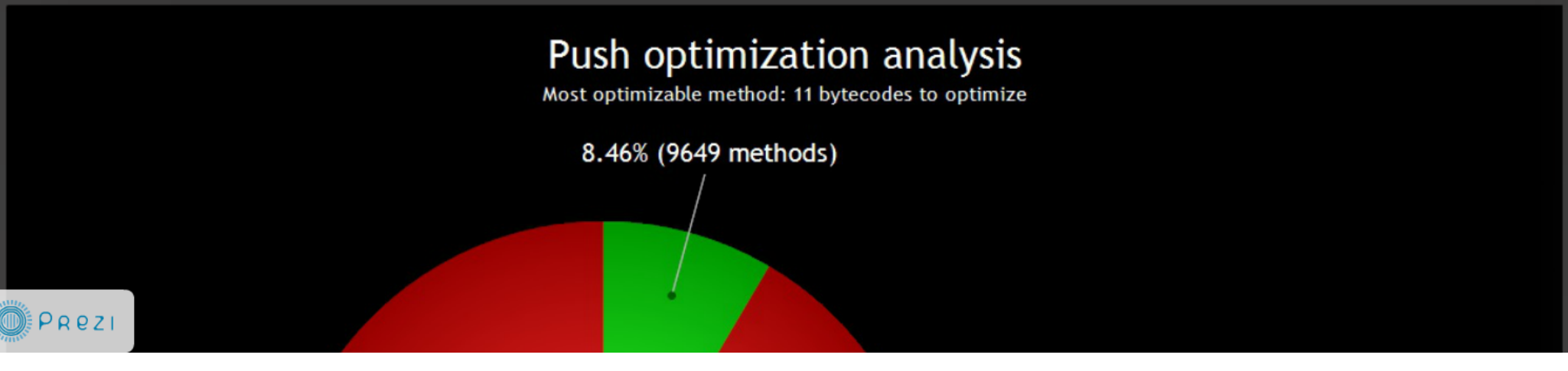


Average time histogram

Evaluations: 1000 averages of 100000 repetitions

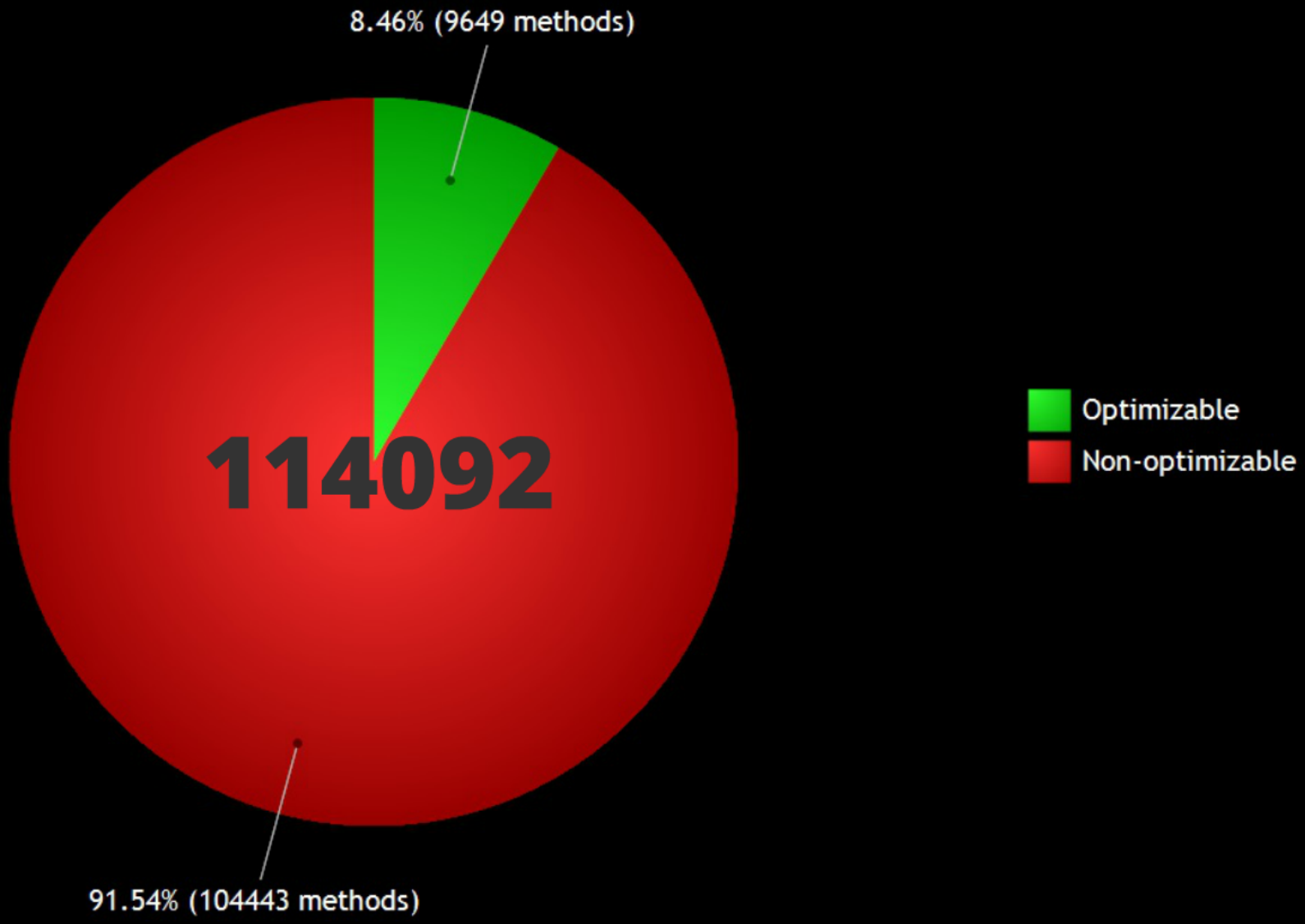

Bootstrapped Average time histogram
Evaluations: 1000 averages of 100000 repetitions
0.020
0.018
0.016
0.014
0.012
0.010
0.008
0.006
0.004
0.002
0
3.46
3.47
3.47
3.48
3.48
3.49
3.49
3.50
Average time
Prezi

Push optimization



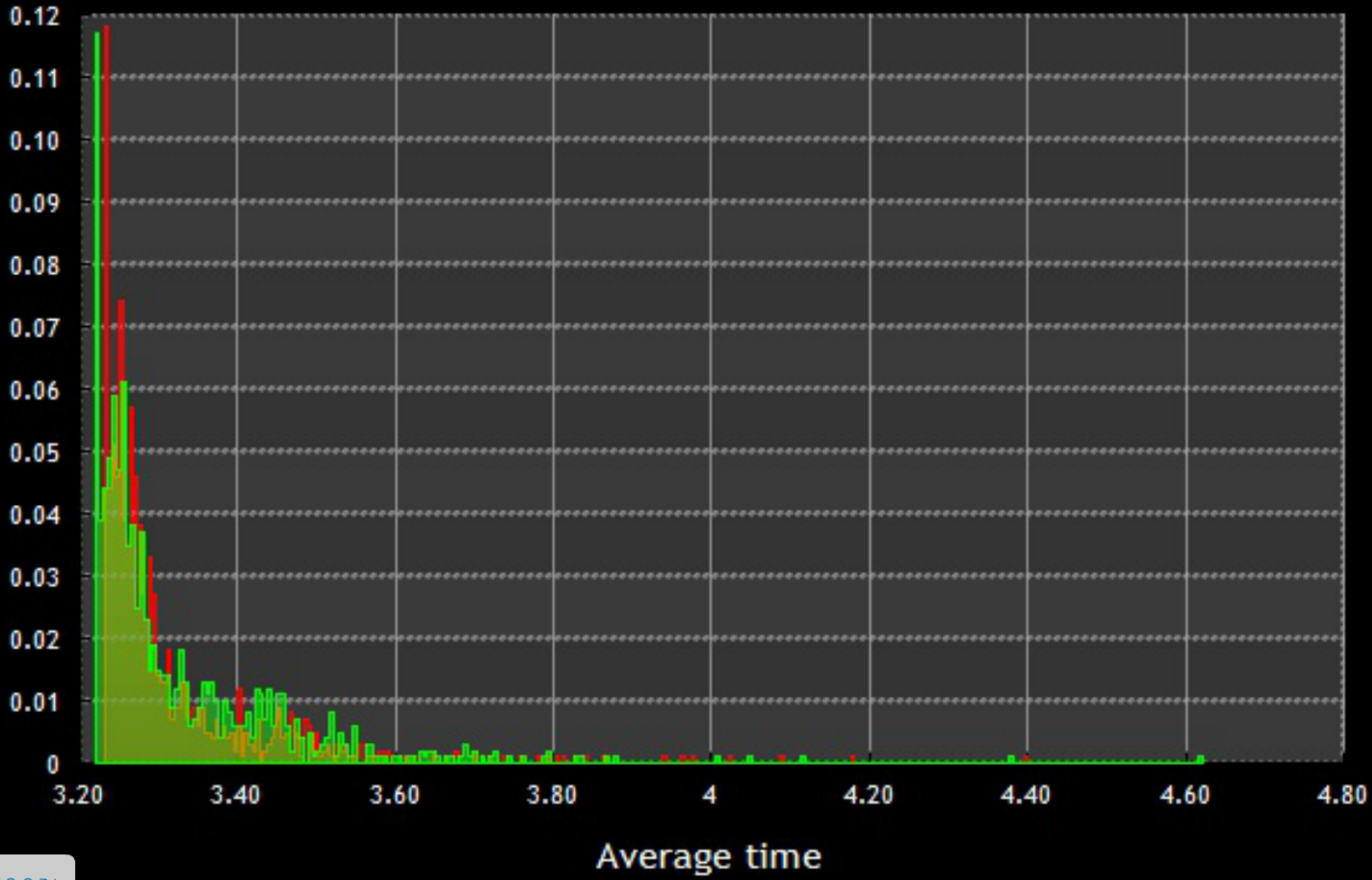
Push optimization analysis

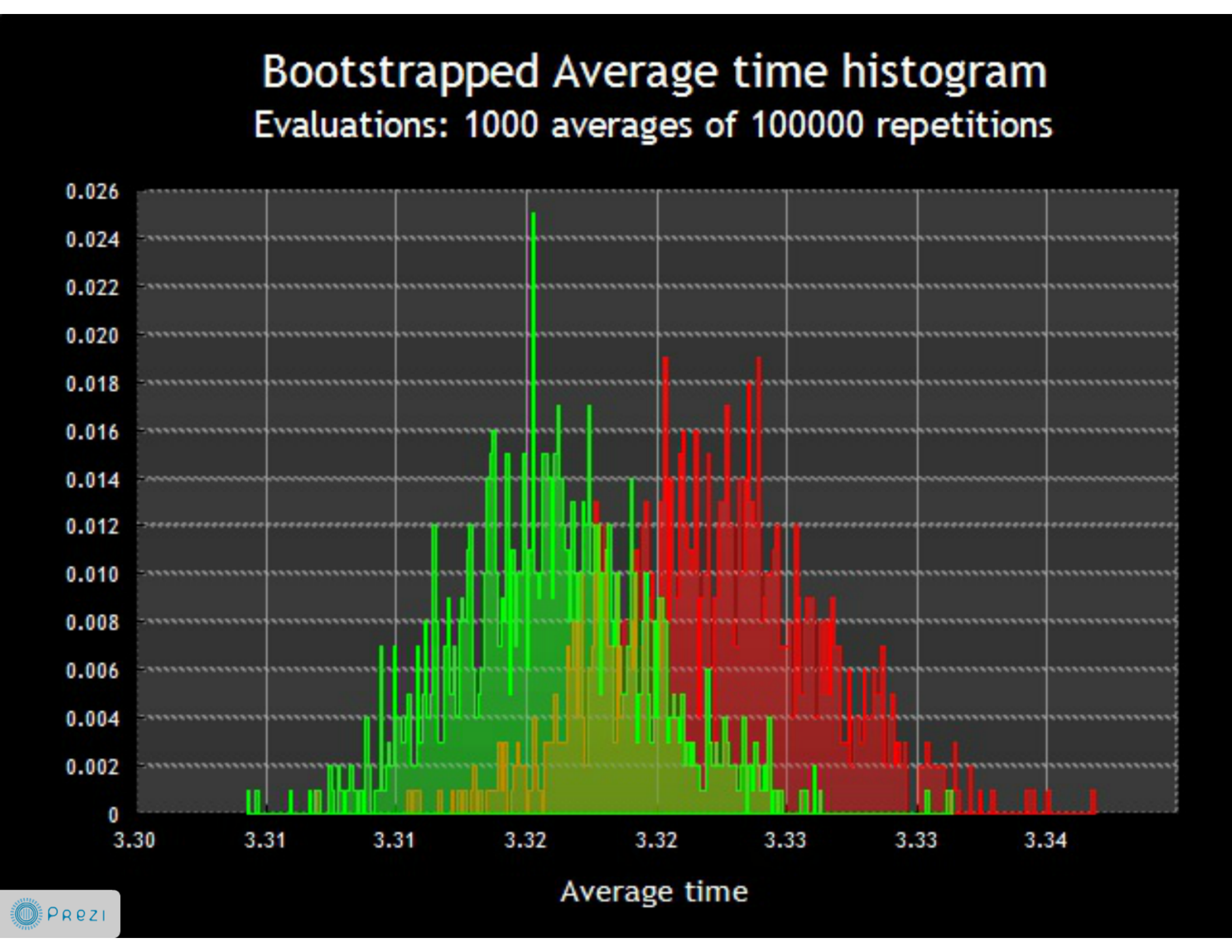
Most optimizable method: 11 bytecodes to optimize



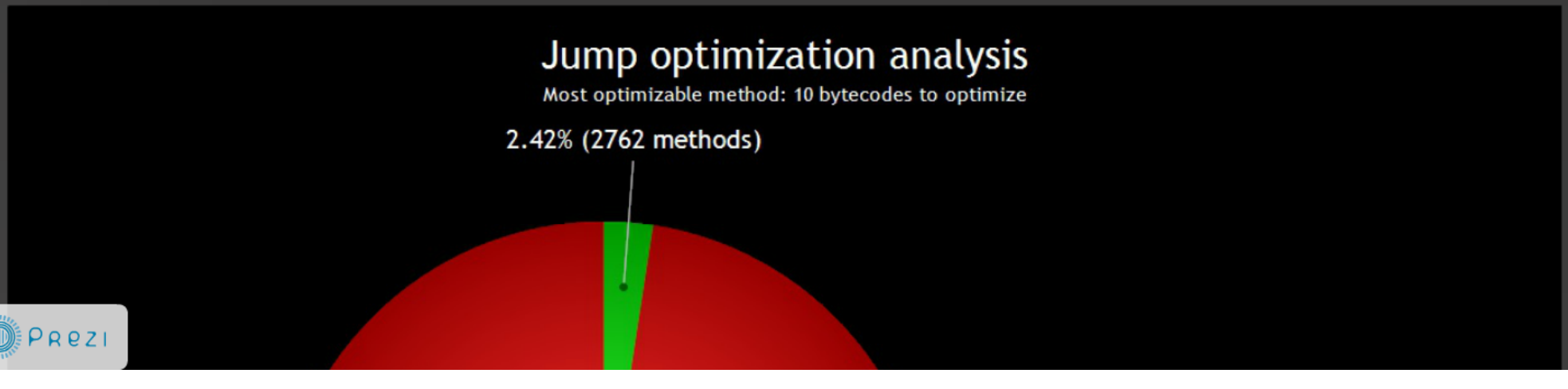
Average time histogram

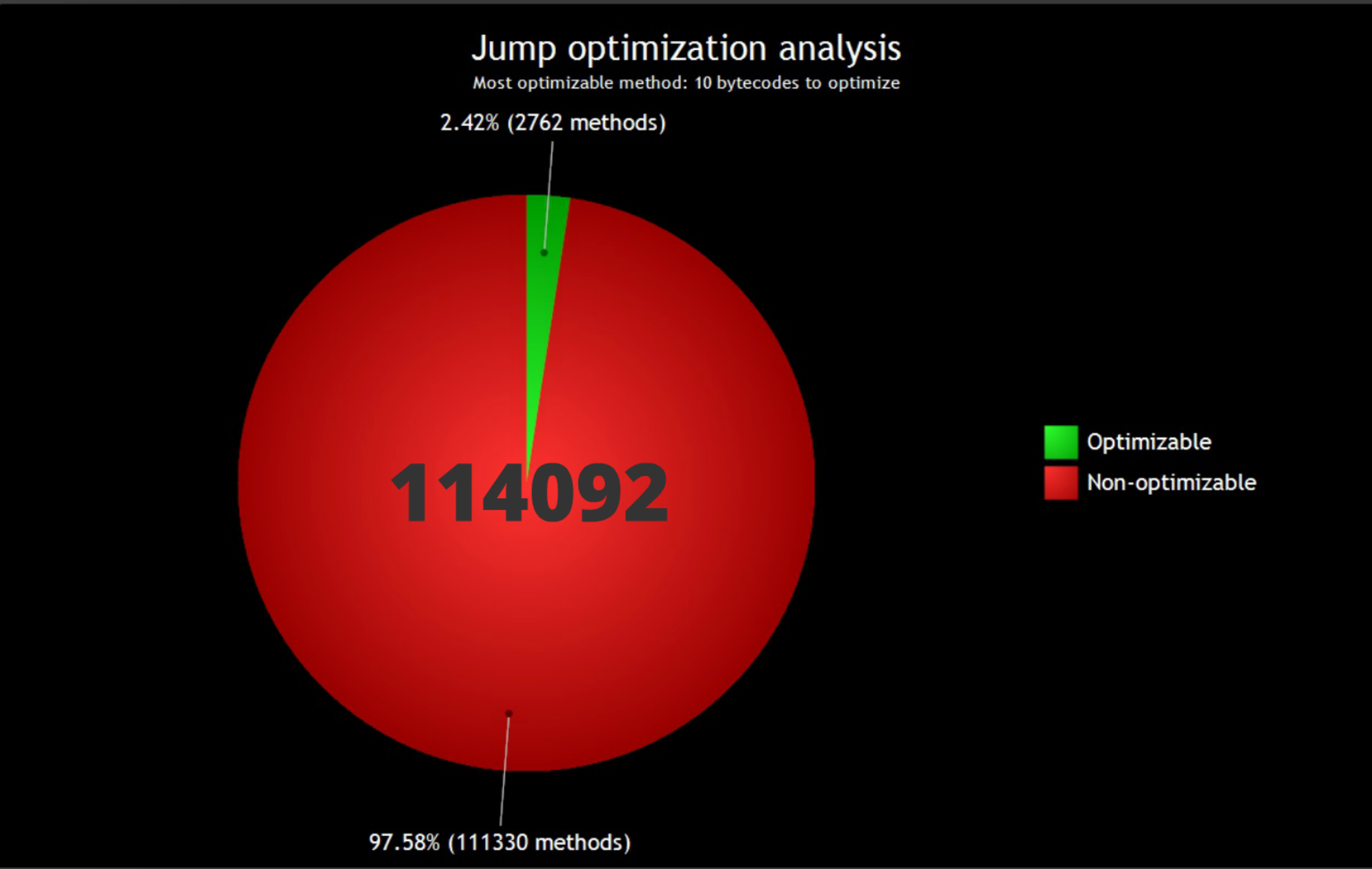
Evaluations: 1000 averages of 100000 repetitions




Bootstrapped Average time histogram
Evaluations: 1000 averages of 100000 repetitions
0.026
0.024
0.022
0.020
0.018
0.016
0.014
0.012
0.010
0.008
0.006
0.004
0.002
0
3.30
3.31
3.31
3.32
3.32
3.33
3.33
3.34
Average time
Prezi

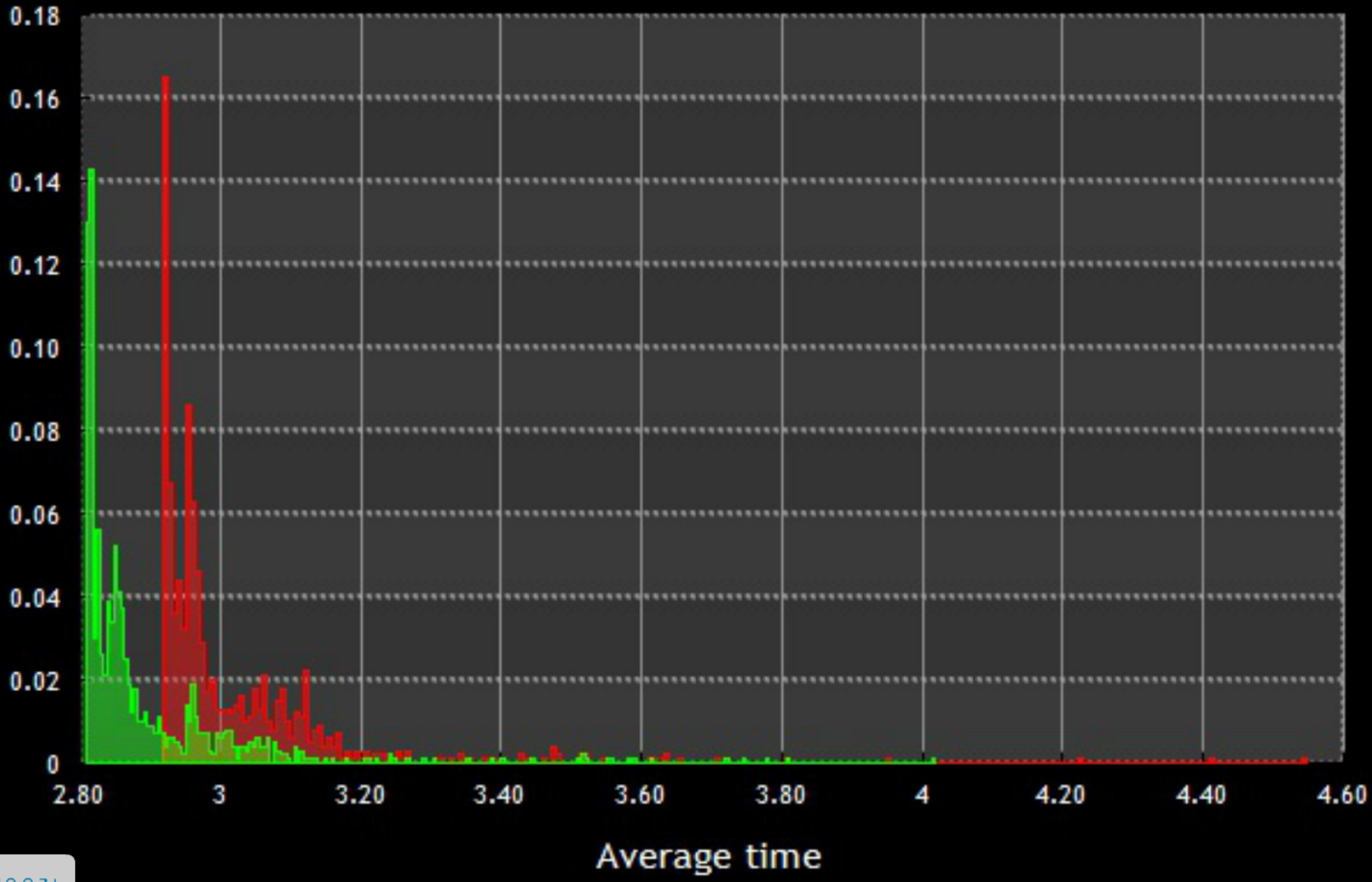
Jump optimization




Jump optimization analysis
Most optimizable method: 10 bytecodes to optimize
2.42% (2762 methods)
114092
97.58% (111330 methods)
Optimizable
Non-optimizable

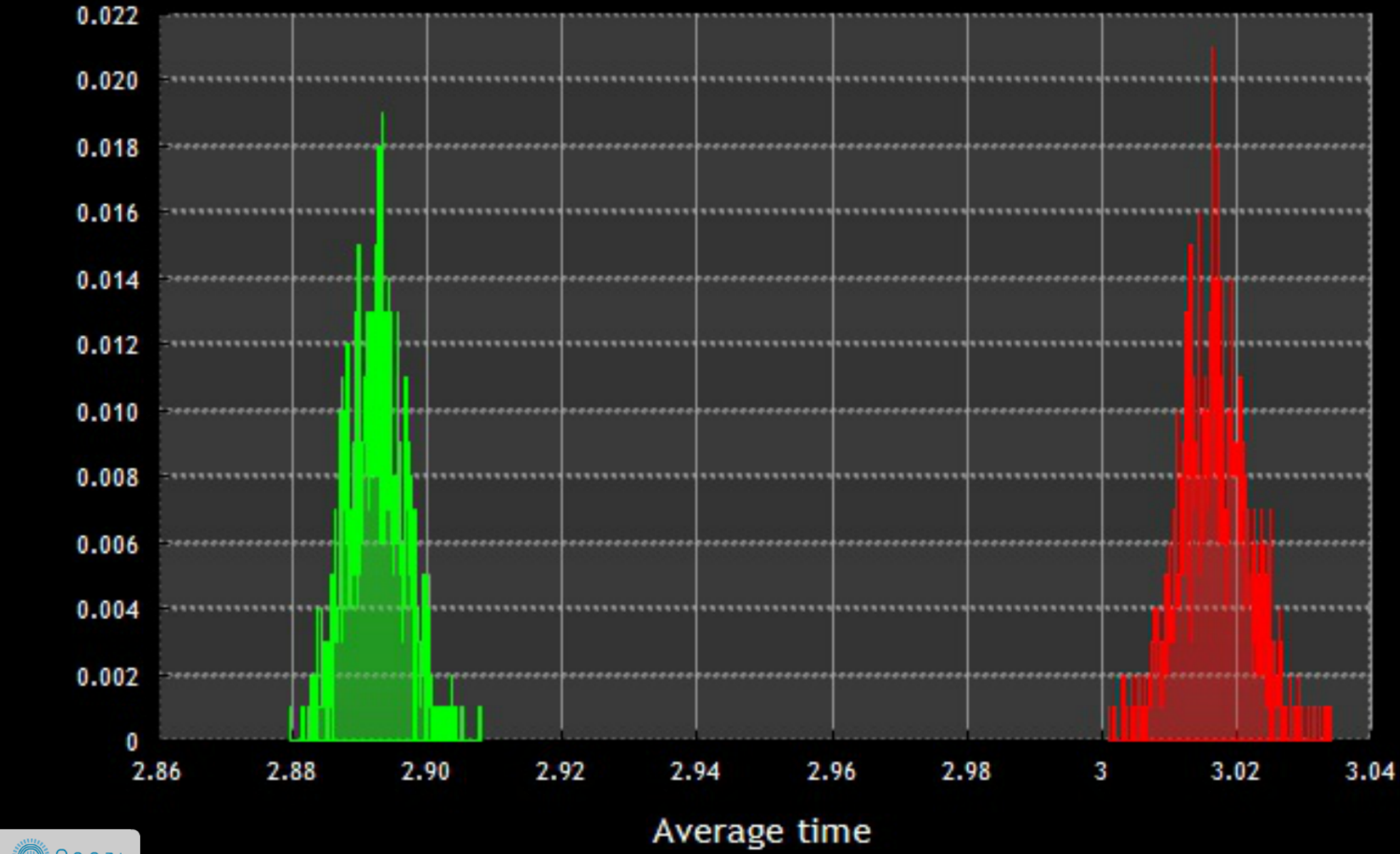
Average time histogram

Evaluations: 1000 averages of 100000 repetitions

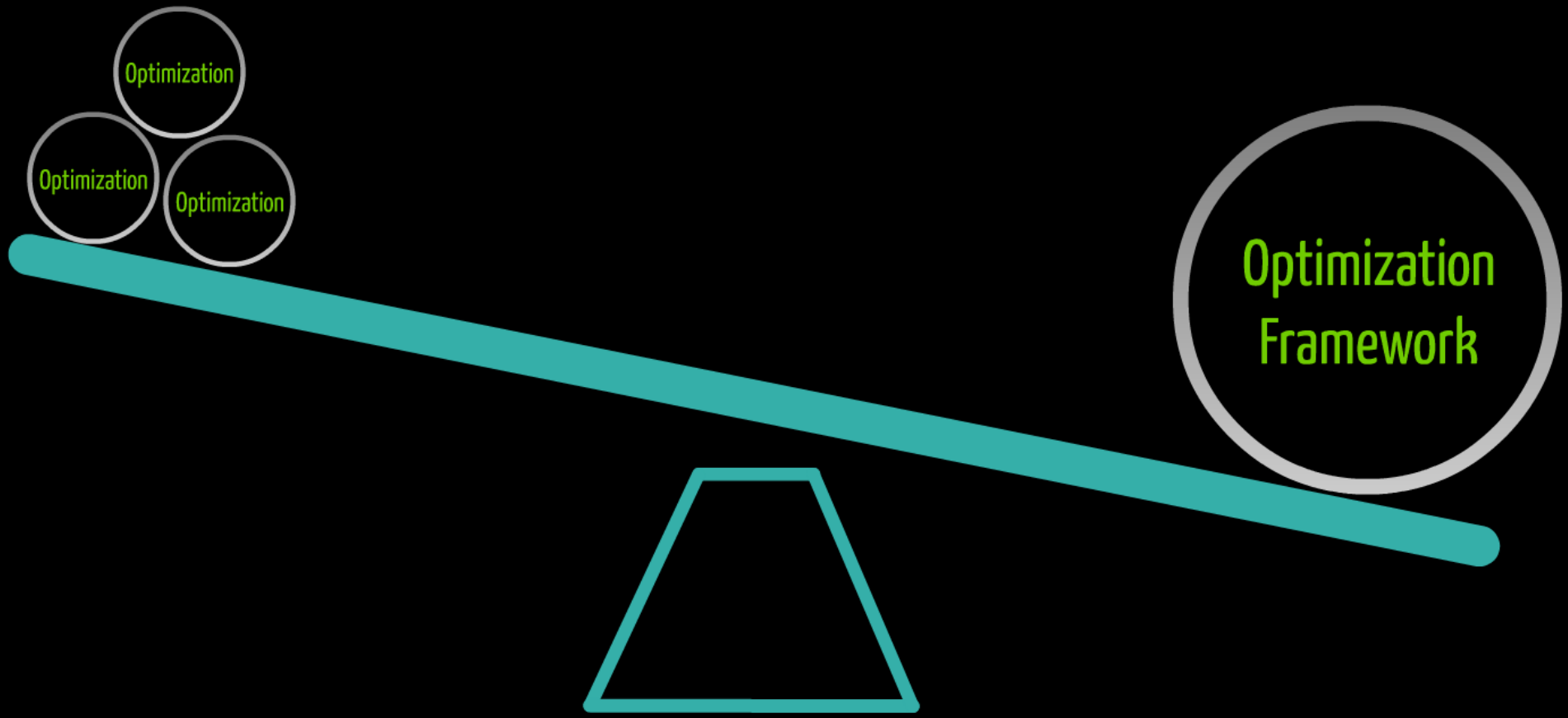


Bootstrapped Average time histogram

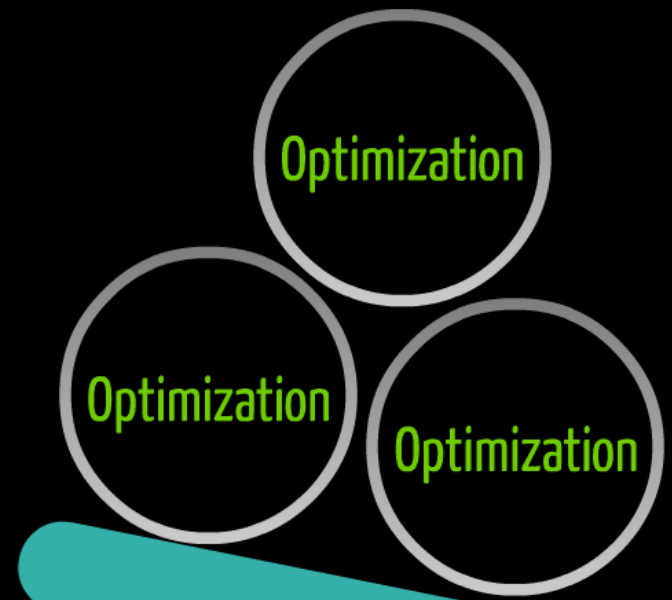
Evaluations: 1000 averages of 100000 repetitions



Conclusion



Conc



Optimization Framework

Questions?

Thanks

gamaral@caesarsystems.com