Sista: Improving Cog's JIT performance

Clément Béra

Main people involved in Sista

- Eliot Miranda
  - Over 30 years experience in Smalltalk VM

- Clément Béra
  - 2 years engineer in the Pharo team
  - Phd student starting from october

Cog ?

- Smalltalk virtual machine
- Runs Pharo, Squeak, Newspeak, ...

Plan

- Efficient VM architecture (Java, C#, ...)
- Sista goals
- Example: optimizing a method
- Our approach
- Status

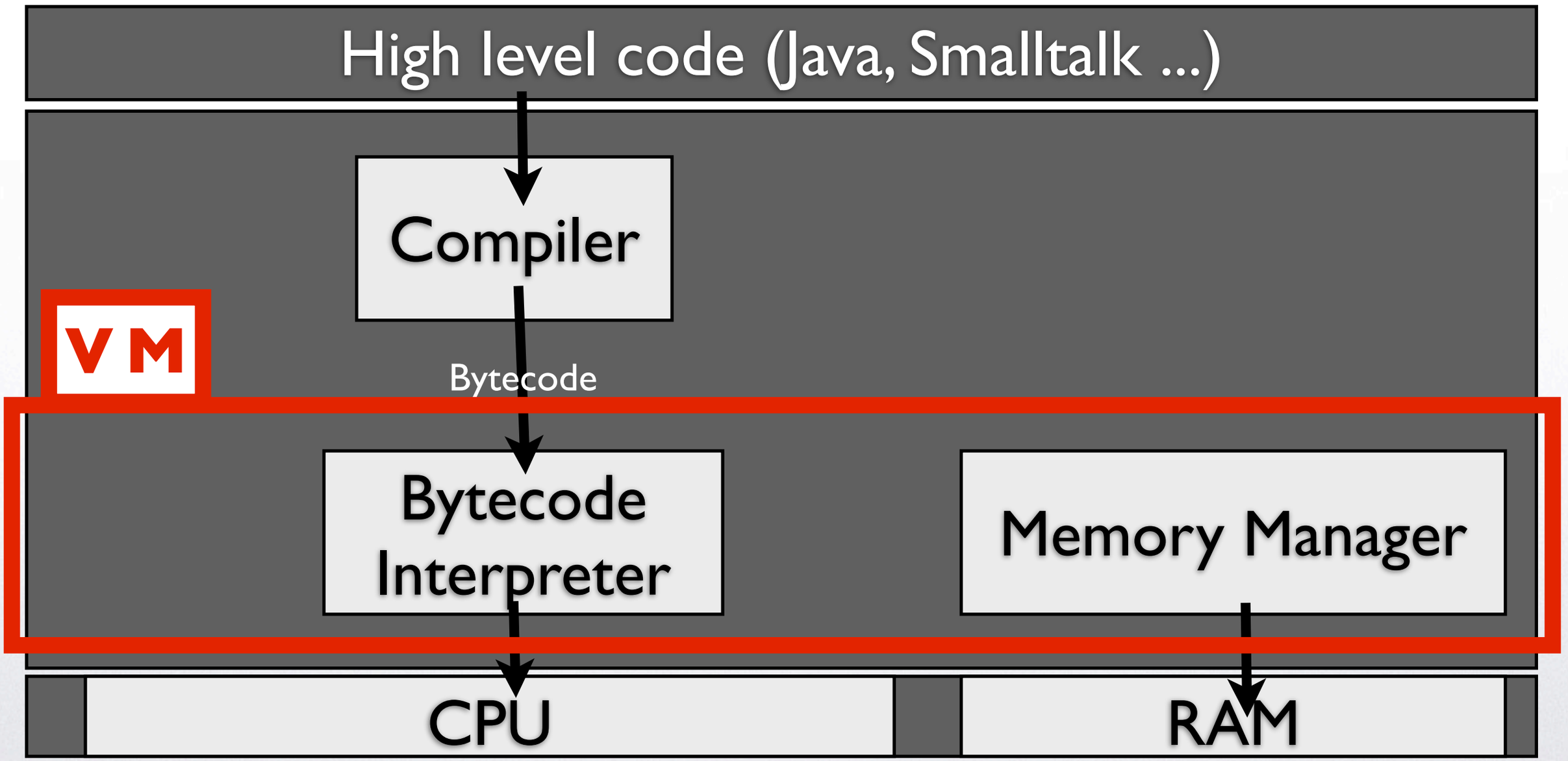
Virtual machine

High level code (Java, Smalltalk ...)

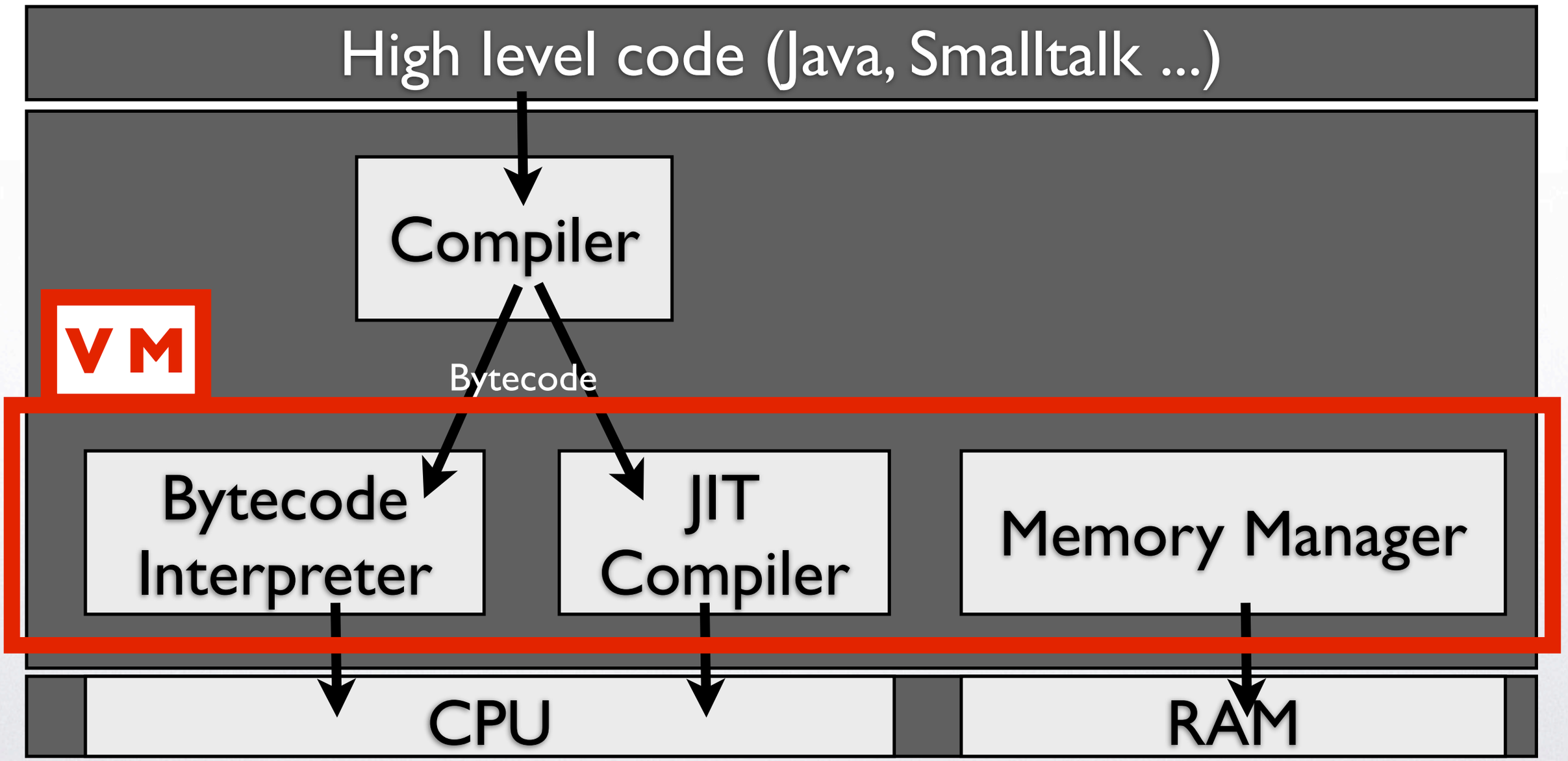
Runtime environment (JRE, JDK, Object engine...)

CPU: intel, ARM ... - OS: Windows, Android, Linux ..

Basic Virtual machine



Fast virtual machine



Efficient JIT compiler

```
display: listOfDrinks
   listOfDrinks do: [ :drink |
      self displayOnScreen: drink ]
```

Execution number	time to run (ms)	Comments
1	1	lookup of #displayOnScreen (cache) and byte code interpretation
2 to 6	0,5	byte code interpretation
7	2	generation of native code for displayOnScreen and native code run
8 to 999	0,01	native code run
1000	2	adaptive recompilation based on runtime type information, generation of native code and optimized native code run
1000 +	0,002	optimized native code run

In Cog

```
display: listOfDrinks
   listOfDrinks do: [ :drink |
      self displayOnScreen: drink ]
```

Execution number	time to run (ms)	Comments
1	1	lookup of #displayOnScreen (cache) and byte code interpretation
2 to 6	0,5	byte code interpretation
7	2	generation of native code for display and native code run
8 to 999	0,01	native code run
1000	2	adaptive recompilation based on runtime type information, generation of native code and optimized native code run
1000 +	0,002	optimized native code run

A look into webkit

- Webkit Javascript engine
  - LLInt = Low Level Interpreter
  - DFG JIT = Data Flow Graph JIT

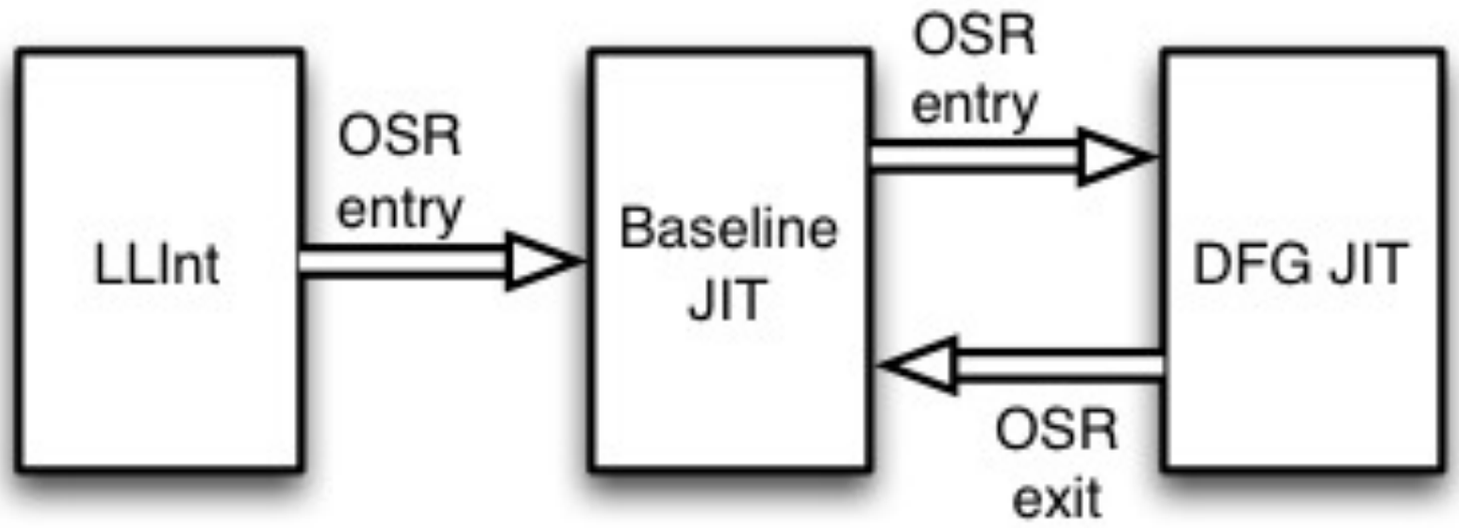


Figure 1. The WebKit three-tier architecture. Arrows indicate on-stack replacement, or OSR for short.

Webkit JS benchs

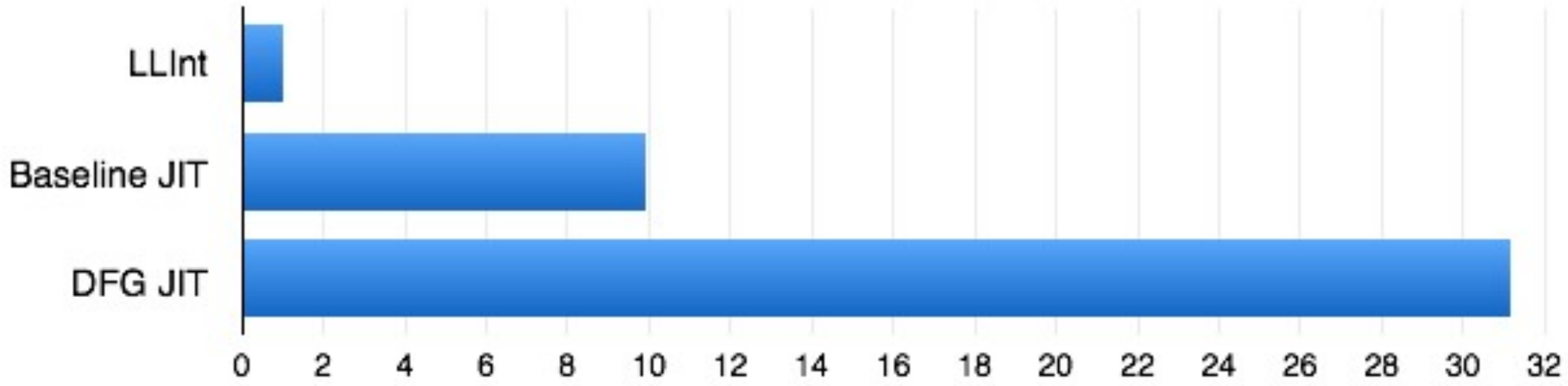


Figure 3. Relative speed-up *(higher is better)* on the Richards benchmark from each of the three tiers.

In Cog

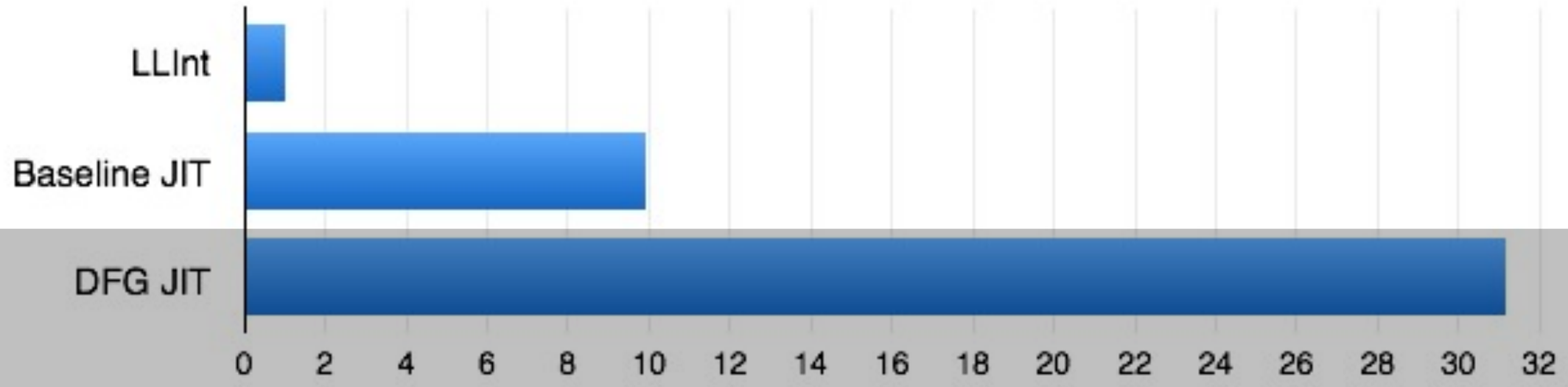


Figure 3. Relative speed-up *(higher is better)* on the Richards benchmark from each of the three tiers.

What is Sista ?

- Implementing the next JIT optimization level

Goals

- Smalltalk performance
- Code readability

Smalltalk performance

The Computer Language Benchmarks Game

Smalltalk performance

The Computer Language Benchmarks Game

Program Source Code	CPU secs	Elapsed secs	Memory KB	Code B	≈ CPU Load
binary-trees					
Smalltalk VisualWorks	65.60	65.67	316,312	722	0% 0% 0% 100%
Java	16.28	16.30	511,448	584	1% 1% 0% 100%
k-nucleotide					
Smalltalk VisualWorks	313.19	313.44	343,320	1153	0% 0% 0% 100%
Java	53.49	53.53	908,608	1630	1% 1% 0% 100%
spectral-norm					
Smalltalk VisualWorks	95.34	95.40	27,236	438	0% 0% 0% 100%
Java	16.26	16.27	16,416	950	1% 1% 0% 100%
pidigits					
Smalltalk VisualWorks	27.28	27.30	177,184	652	0% 0% 0% 100%
Java	4.12	4.13	18,024	938	1% 1% 1% 100%
fannkuch-redux					
Smalltalk VisualWorks	601.47	601.61	21,952	838	0% 0% 0% 100%
Java	67.40	67.43	15,720	1282	1% 1% 0% 100%
fasta					
Smalltalk VisualWorks	44.63	44.66	21,956	1315	0% 1% 1% 100%
Java	4.91	4.91	27,668	2457	0% 1% 1% 100%
n-body					
Smalltalk VisualWorks	263.11	263.24	21,956	1652	0% 0% 0% 100%
Java	24.54	24.55	15,604	1424	0% 0% 1% 100%
reverse-complement					

Smalltalk performance

The Computer Language Benchmarks Game

Smalltalk is 4x~25x slower than Java

Smalltalk performance

The Computer Language Benchmarks Game

Smalltalk is 4x~25x slower than Java

Our goal is 3x faster:
1.6x~8x times slower than Java
+ Smalltalk features

Code Readability

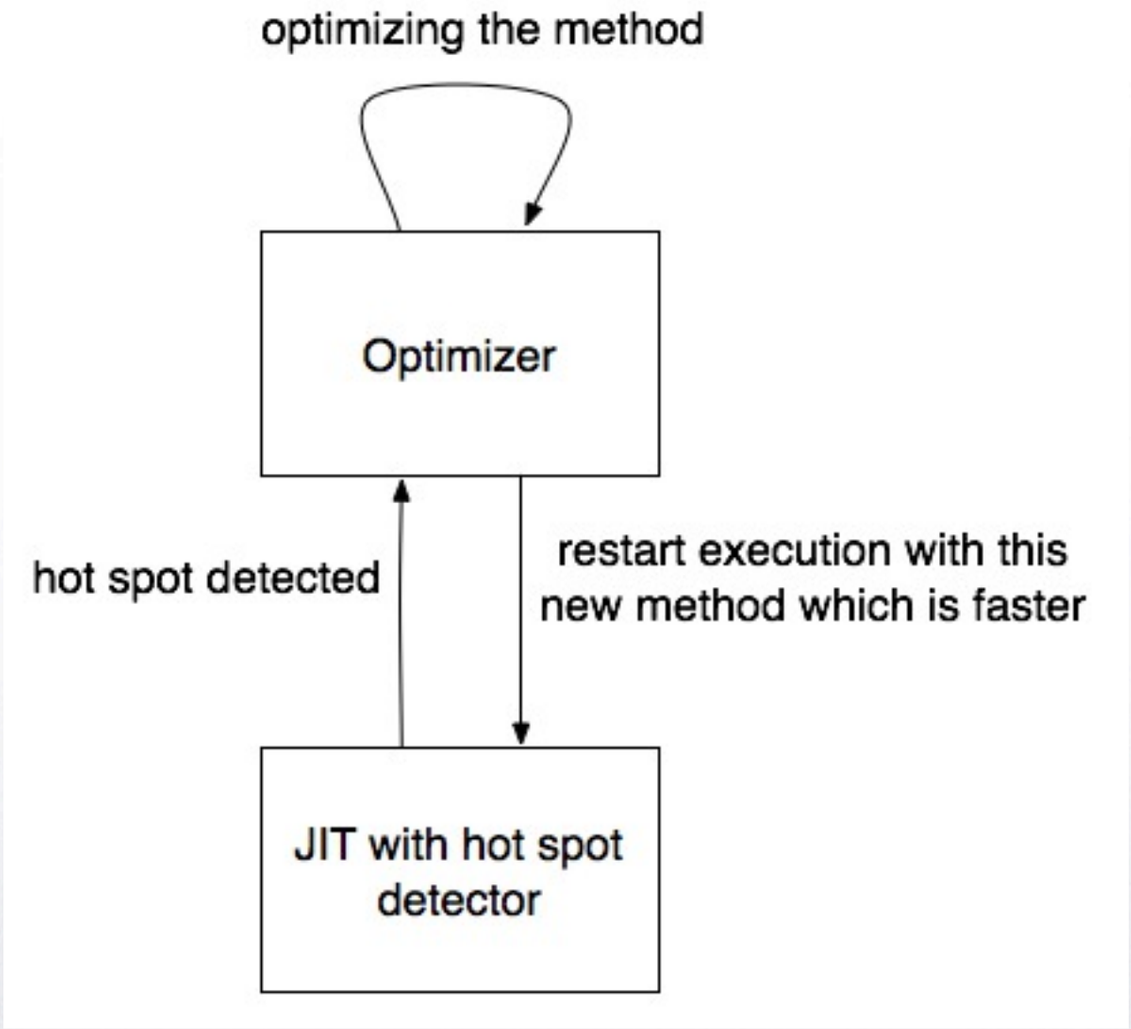
- Messages optimized by the bytecode compiler overused in the kernel

- #do: => #to:do:

Adaptive recompilation

- Recompiles on-the-fly portion of code frequently used based on the current environment and previous executions

Optimizing a method



Example

```
display: listOfDrinks
    listOfDrinks do: [ :drink |
        self displayOnScreen: drink ]
```

```
do: aBlock
    "Refer to the comment in Collection|do:."
    1 to: self size do:
        [:index | aBlock value: (self at: index)]
```

Example

- Executing #display: with over 30 000 different drinks ...

```
display: listOfDrinks
    listOfDrinks do: [ :drink |
        self displayOnScreen: drink ]
```

Hot spot detector

- Detects methods frequently used

Hot spot detector

- JIT adds counters on machine code
- Counters are incremented when code execution reaches it

- When a counter reaches threshold, the optimizer is triggered

Example

```
display: listOfDrinks
    listOfDrinks do: [ :drink |
        self displayOnScreen: drink ]
```

Cannot detect hot spot

```
do: aBlock
    "Refer to the comment in Collection|do:."
    1 to: self size do:
        [:index | aBlock value: (self at: index)]
```

Can detect hot spot
(#to:do: compiled inlined)

Hot spot detected

```
display: listOfDrinks
    listOfDrinks do: [ :drink |
        self displayOnScreen: drink ]
```

```
do: aBlock
    "Refer to the comment in Collection|do:."
    1 to: self size do:
        [:index | aBlock value: (self at: index)]
```

Over 30 000 executions

Optimizer

- What to optimize ?

```
do: aBlock
    "Refer to the comment in Collection|do:."
    1 to: self size do:
        [:index | aBlock value: (self at: index)]
```

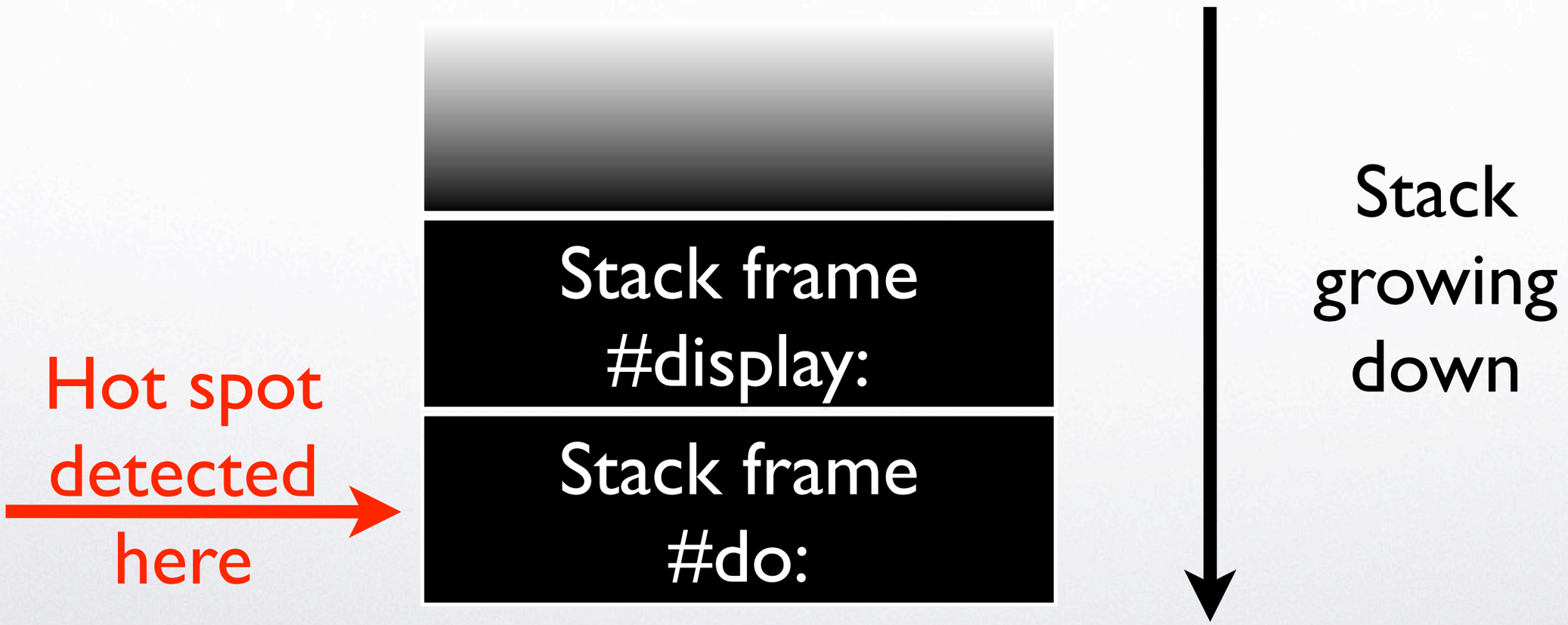
Optimizer

- What to optimize ?
- Block evaluation is costly

```
do: aBlock
    "Refer to the comment in Collection|do:."
    1 to: self size do:
        [:index | aBlock value: (self at: index)]
```

Optimizer

- What to optimize ?



Optimizer

- What to optimize ?

Hot spot detected here

Stack frame #display:

Stack frame #do:

```
display: listOfDrinks
    listOfDrinks do: [ :drink |
        self displayOnScreen: drink ]
```

```
do: aBlock
    "Refer to the comment in Collection|do:."
    1 to: self size do:
        [:index | aBlock value: (self at: index)]
```

Find the best method

```
display: listOfDrinks
    listOfDrinks do: [ :drink |
        self displayOnScreen: drink ]
```

```
do: aBlock
    "Refer to the comment in Collection|do:"
    1 to: self size do:
        [:index | aBlock value: (self at: index)]
```

Find the best method

```
display: listOfDrinks
    listOfDrinks do: [ :drink |
        self displayOnScreen: drink ]
```

```
do: aBlock
    "Refer to the comment in Collection|do:"
    1 to: self size do:
        [:index | aBlock value: (self at: index)]
```

- To be able to optimize the block activation, we need to optimize both #example and #do:

Inlining

- Replaces a function call by its callee

```
display: listOfDrinks
    listOfDrinks do: [ :drink |
        self displayOnScreen: drink ]
```

```
do: aBlock
    "Refer to the comment in Collection|do:"
    1 to: self size do:
        [:index | aBlock value: (self at: index)]
```

Inlining

- Replaces a function call by its callee

```
display: listOfDrinks
    listOfDrinks do: [ :drink |
        self displayOnScreen: drink ]
```

```
do: aBlock
    "Refer to the comment in Collection|do:"
    1 to: self size do:
        [:index | aBlock value: (self at: index)]
```

- Speculative: what if listOfDrinks is not an Array anymore ?

Inlining

- Replaces a function call by its callee

```
display: listOfDrinks
    listOfDrinks do: [ :drink |
        self displayOnScreen: drink ]
```

```
do: aBlock
    "Refer to the comment in Collection|do:"
    1 to: self size do:
        [:index | aBlock value: (self at: index)]
```

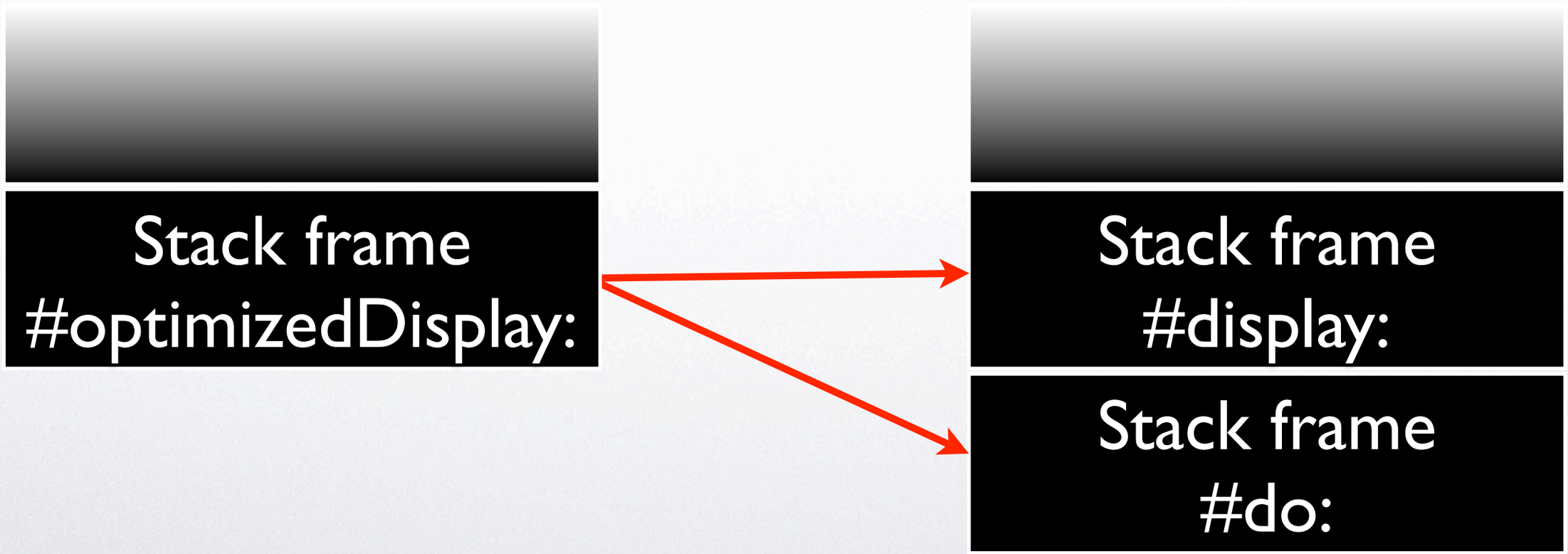
- Speculative: what if listOfDrinks is not an Array anymore ?
- Type-feedback from inline caches


Guard

- Guard checks: listOfDrinks hasClass: Array
- If a guard fails, the execution stack is *dynamically deoptimized*
- If a guard fails more than a threshold, the optimized method is uninstalled

Dynamic deoptimization

- On Stack replacement
- PCs, variables and methods are mapped.



Optimizations

- 1) #do: is inlined into #display:

```
display: listOfDrinks
    listOfDrinks do: [ :drink |
        self displayOnScreen: drink ]
```

```
do: aBlock
    "Refer to the comment in Collection|do:"
    1 to: self size do:
        [:index | aBlock value: (self at: index)]
```

```
display: listOfDrinks
    guard check: listOfDrinks hasClass: Array.
    1 to: listOfDrinks size do:
        [ :index |
            [ :drink | self displayOnScreen: drink ] value: (listOfDrinks at: index) ]
```

Optimizations

- 2) Block evaluation is inlined

```
display: listOfDrinks
    guard check: listOfDrinks hasClass: Array.
    1 to: listOfDrinks size do:
        [ :index |
            [ :drink | self displayOnScreen: drink ] value: (listOfDrinks at: index) ]
```

```
display: listOfDrinks
    guard check: listOfDrinks hasClass: Array.
    1 to: listOfDrinks size do:
        [ :index | self displayOnScreen: (listOfDrinks at: index) ]
```

Optimizations

- 3) at: is optimized
- at: optimized ???

```
display: listOfDrinks
    guard check: listOfDrinks hasClass: Array.
    1 to: listOfDrinks size do:
        [ :index | self displayOnScreen: (listOfDrinks at: index) ]
```

At: implementation

```
object at: index
```

- VM implementation
  - Is index an integer ?
  - Is object a variable-sized object, a byte object, ... ? (Array, ByteArray, ... ?)
  - Is index within the bound of the object ?
  - Answers the value

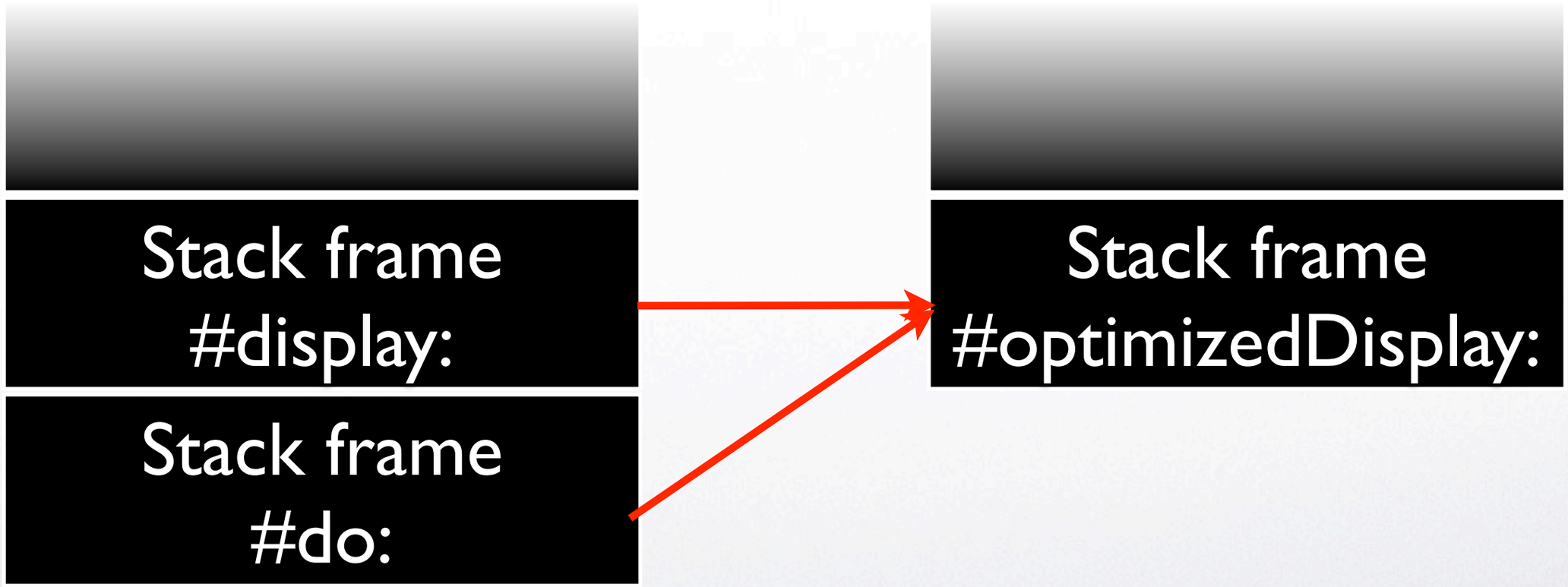
Optimizations

- 3) at: is optimized
  - index isSmallInteger
  - 1 <= index <= listOfDrinks size
  - listOfDrinks class == Array
- uncheckedAt1: (at: for variable size objects with in-bounds integer argument)

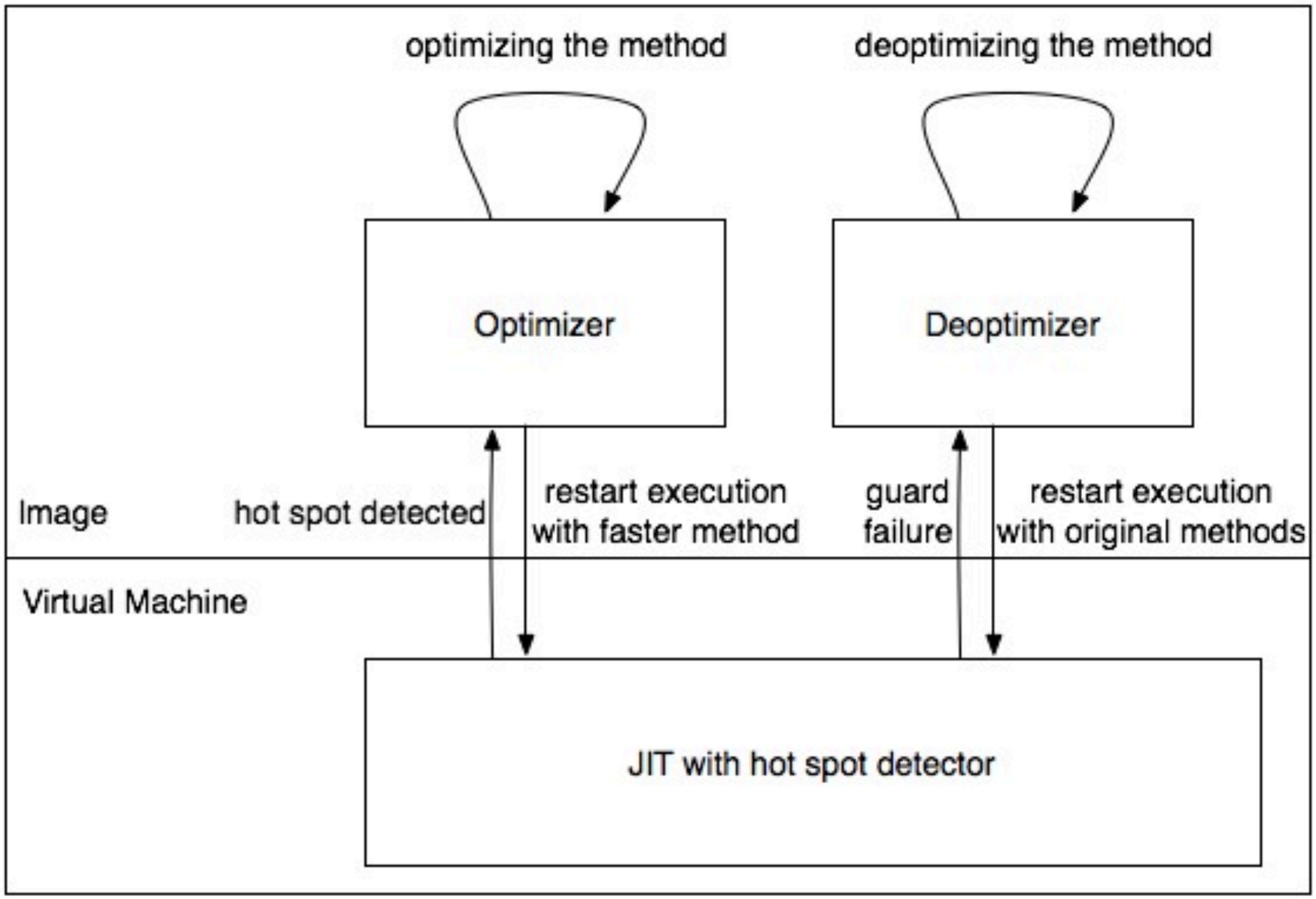
```
display: listOfDrinks
   guard check: listOfDrinks hasClass: Array.
   1 to: listOfDrinks size do:
      [ :index | self displayOnScreen: (listOfDrinks uncheckedAt1: index) ]
```

Restarting

- On Stack replacement



Our approach



In-image Optimizer

- Inputs
  - Stack with hot compiled method
  - Branch and type information
- Actions
  - Generates and installs an optimized compiled method
  - Generates deoptimization metadata
  - Edit the stack to use the optimized compiled method

In-image Deoptimizer

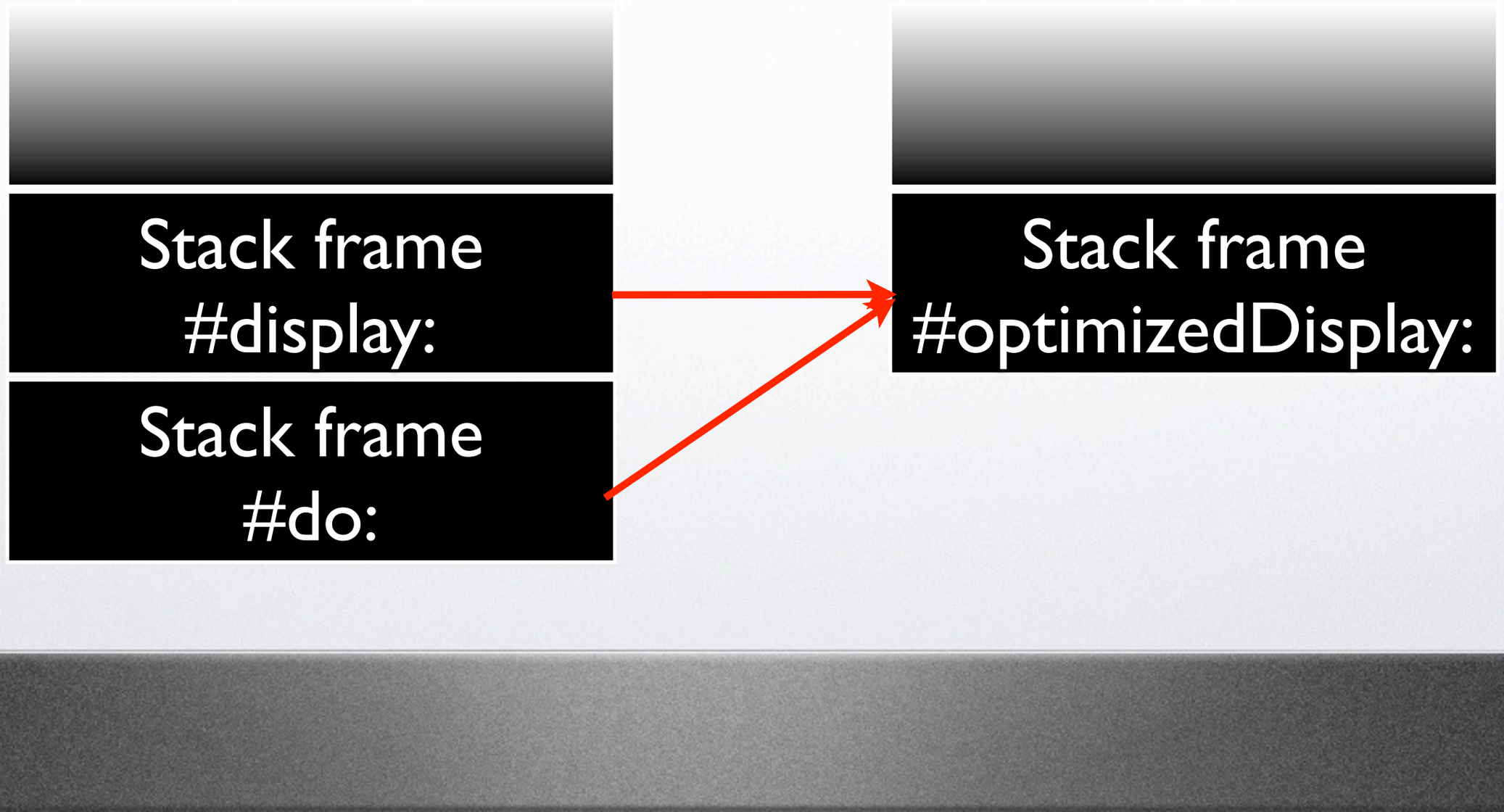
- Inputs
  - Stack to deoptimize (failing guard, debugger, ...)
  - Deoptimization metadata
- Actions
  - Deoptimize the stack
  - May discard the optimized method

Advantages

- Optimizer / Deoptimizer in smalltalk
  - Easier to debug and program in Smalltalk than in Slang / C
- Editable at runtime
- Lower engineering cost

Advantages

On Stack replacement done with Context manipulation



Cons

- What if the optimizer is triggered while optimizing ?

Advantages

- CompiledMethod to optimized CompiledMethod
- Snapshot saves compiled methods


Cons

- Some optimizations are difficult
  - Instruction selection
  - Instruction scheduling
  - Register allocation

Critical optimizations

- Inlining (Methods and Closures)
- Specialized at: and at:put:
- Specialized arithmetic operations

Interface VM - Image

- Extended bytecode set
- CompiledMethod introspection primitive
- VM callback to trigger optimizer /deoptimizer

Extended bytecode set

- Guards
- Specialized inlined primitives
  - at:, at:put:
  - +, - , <=, =, ...

Introspection primitive

- Answers
  - Type info for each message send
  - Branch info for each conditional jump
- Answers nothing if method not in machine code zone

VM callback

- Selector in specialObjectsArray
- Sent to the active context when hot spot / guard failure detected

Stability

- Low engineering resources
- No thousands of human testers

Stability

- Gemstone style ?
- Large test suites

Stability

- RMOD research team

- Inventing new validation techniques
- Implementing state-of-the-art validators

Stability goal

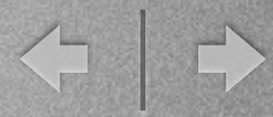
- Both bench and large test suite run on CI
- Anyone could contribute

Anyone can contribute

Anyone can contribute

Stability goal

- Both bench and large test suite run on CI
- Anyone could contribute

Status

- A full round trip works
  - Hot spot detected in the VM
  - In-image optimizer finds a method to optimize, optimizes it an install it
    - method and closure inlining
  - Stack dynamic deoptimization

Status: hot spot detector

- Richards benchmark
  - Cog: 341ms
  - Cog + hot spot detector: 351ms
- 3% overhead on Richards
- Detects hot spots
- Branch counts ⇒ basic block counts


Next steps

- Dynamic deoptimization of closures is not stable enough
- Efficient new bytecode instructions
- Stabilizing bounds check elimination
- lowering memory footprint
- Invalidation of optimized methods
- On stack replacement

Thanks

- Stéphane Ducasse
- Marcus Denker

- Yaron Kashai

Conclusion

- We hope to have a prototype running for Christmas
- We hope to push it to production in around a year

Demo

Questions

RMod

Inria
INVENTEURS DU MONDE NUMÉRIQUE

cadence™