Clean Blocks

Nahuel Palumbo, Inria Evref
Marcus Denker, Inria Evref

It looks so simple

```
[ 1 + 2 ] value
```

- Block of code
- It is an object!
- [] creates the object
- #value evaluates it

But it is not

```
| temp myBlock |

temp := 1.
myBlock := [ 1 + temp].
^  myBlock
```

- The block can access temps from the outer method (or block)
- The block could access self and ivars
- the block can survive the execution of the method where the temp is defined

Creation vs Activation

- Two operations

```
[ … ]
```

Block Creation: VM creates instance of FullBlockClosure

```
aBlock value
```

Block Activation: the VM executes the CompledBlock

Block creation at runtime

- We need access to the temporary variables from outer blocks / the method
  - And: We need access to self (for ivar access, too)
- We have to hand over runtime data to the closure!
- Slow: object allocation

Bytecode View

```
Chart >> verticalTick
^ decorations detect: #isVerticalTick ifNone: [ nil ]
```

```
<07> pushRcvr: 7 "Push ’decorations’ variable”
<20> pushConstant: #isVerticalTick "Push symbol #isVerticalTick"
<F9 01 00> fullClosure: [ nil ] "Create and Push the Closure"
<A2> send: detect:ifNone:
<5C> returnTop
```

Optimizations needed

- Creating blocks is slow

- What can we optimize?

Clean Blocks

- What if a block does not access any variables from an outer method ?
- And no self
  - and no ivars (as we need self to read the ivar)
- And no return (as it needs the home context)

Pre-create Clean Blocks

- We do not need any information from runtime
- We can create the block at compile time
- And store the block in the literal frame, not the CompiledBlock
- A clean block is exactly the “naive” block idea we had at the start: it’s just a wrapper around a method that implements #value methods

Bytecode View

```
Chart >> verticalTick
^ decorations detect: #isVerticalTick ifNone: [ nil ]
```

```
<07> pushRcvr: 7 "Push ’decorations’ variable"
<20> pushConstant: #isVerticalTick”
<21> pushConstant: [ nil ] "Push the Block Closure”
<A2> send: detect:ifNone:
```

Clean Blocks Creation

```
OCCompilationContext optionCleanBlockClosure: true.
cleanEnabled := [ [1+2] ] bench.

OCCompilationContext optionCleanBlockClosure: false.
cleanDisabled := [ [1+2] ] bench.
```

Clean Blocks are created **9x faster** than Full Blocks

What about constants?

- An even more trivial block: a block that just returns a literal
- Happens more often than you think!
- e.g. at:IfAbsent:

```
Morph>>#minHeight
    "answer the receiver's minHeight"
    ^ self
        valueOfProperty: #minHeight
        ifAbsent: [2]
```

Constant Blocks

- Specialized subclass
  - constant is state

```
ConstantBlockClosure>>#value
^literal
```

```
ConstantBlockClosure>>#value: anObject
self numArgsError: 1
```

Constant Blocks Execution

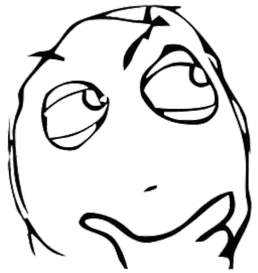
```
OCCompilationContext optionConstantBlockClosure: true.
block := [ nil ].
constEnabled := [ block value ] bench.

OCCompilationContext optionConstantBlockClosure: false.
block := [ nil ].
constDisabled := [ block value ] bench.
```

Constant Blocks are executed **1.4x faster** than Full Blocks

Evaluation

Static Analysis

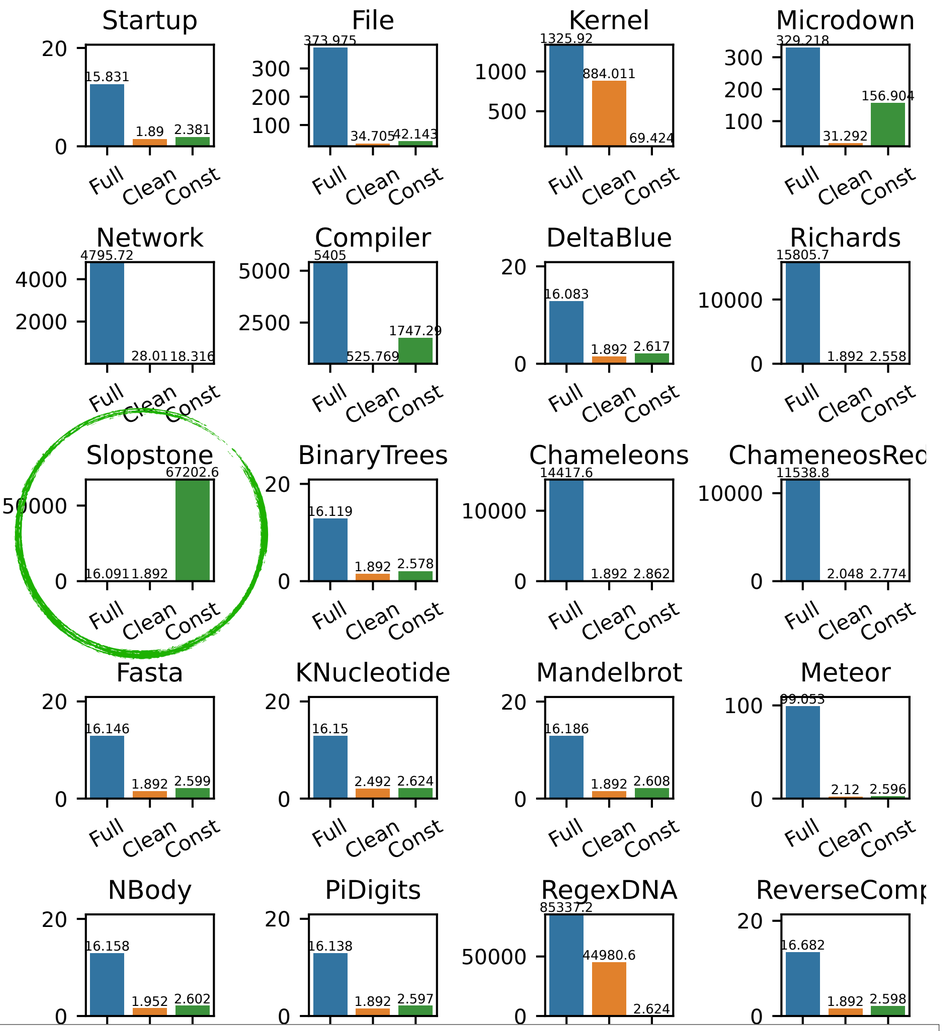
How many Clean Blocks are in a new image?



Blocks	Quantity	Percentage
Full Blocks	26060	71%
Clean Blocks	8219	22%
Constant Blocks	2365	6%
All	**36644**	**100%**

Closure Activations

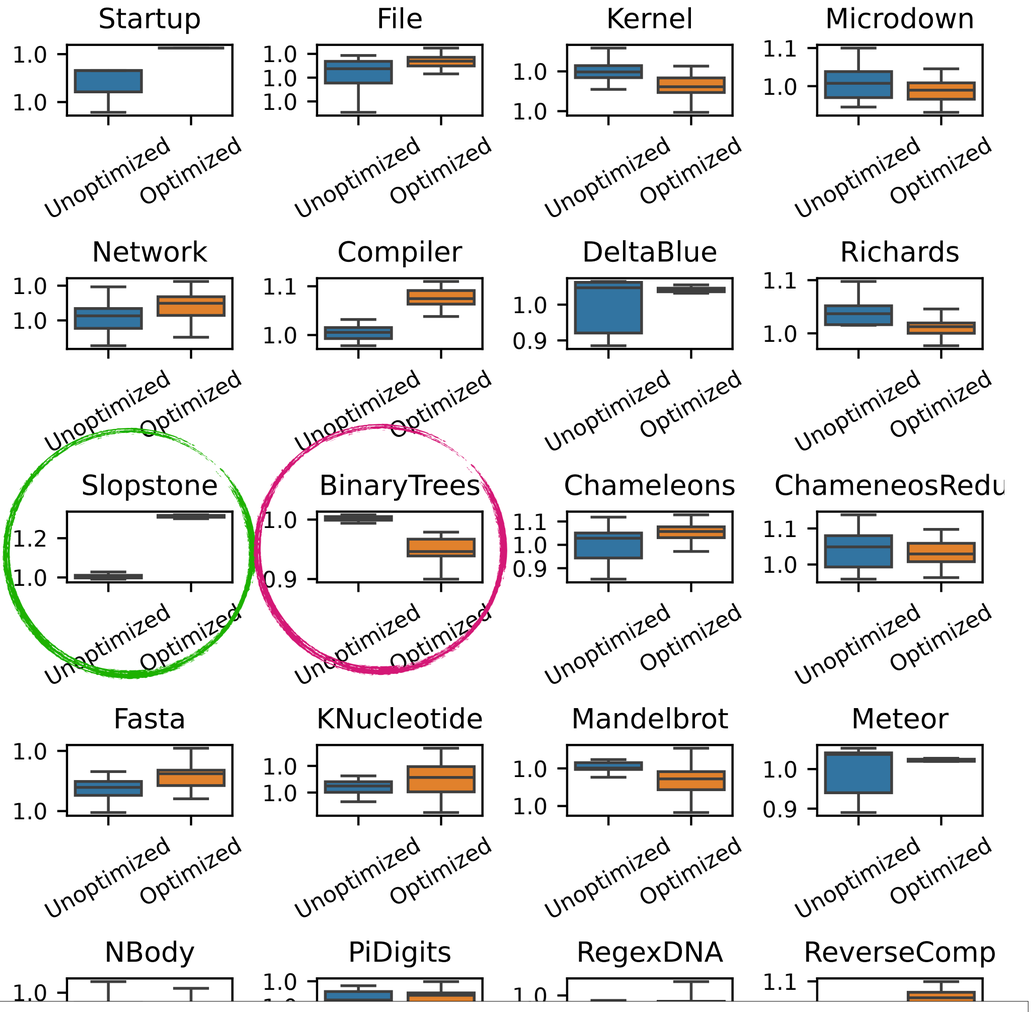
- **76%** Full Closures
- **24%** Clean & Constant Closures
  - **40%** Clean Closures
  - **60%** Constant Closures



Number of activations for Full, Clean, and Constant Blocks per benchmark. Numbers are in K- activations (x1000).

Speed Up

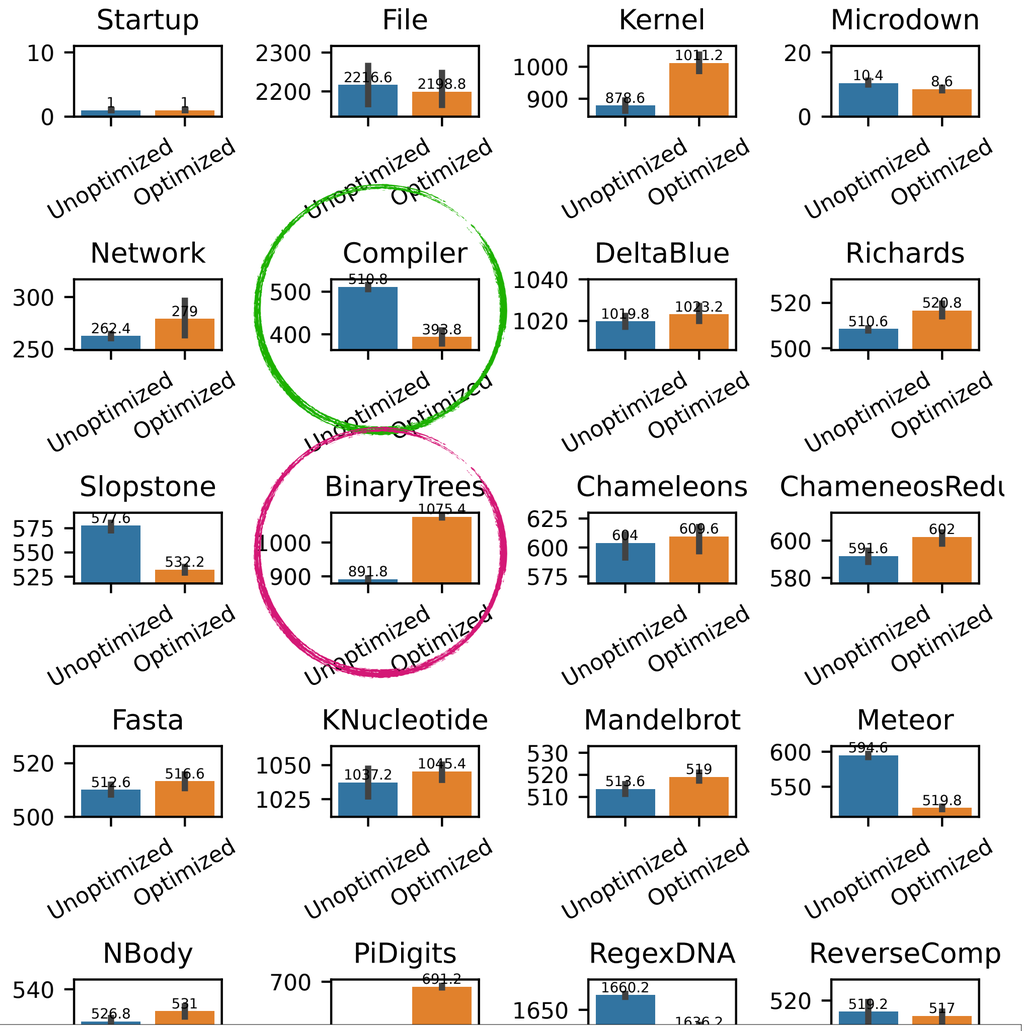
- **1.35x** I  ase (Slopstone)
- **1.02x** On average
- **0.95x** Worst case (BinaryTrees)



Speed up using Clean & Constant Blocks relative to Full Blocks. Higher is better.

GC Overhead

- **0.77x** Best case (Compiler)
- **1.01x** On average
- **1.21x** Worst case (BinaryTrees)



Garbage Collection overhead for each configuration per benchmark. Time is in milliseconds. Lower is better.

Future Work

- Analyse the Garbage Collector in detail
- Measure Full Closure creation time
- Continue optimizing Full Closures
  - Avoid the outer context
  - Force Clean Blocks, accessing through reflection?