RSqueak/VM
Building a fast, malleable VM with students

Hasso-Plattner-Institut Potsdam

Software Architecture Group

Tim Felgentreff, Tobias Pape, Patrick Rein, Robert Hirschfeld

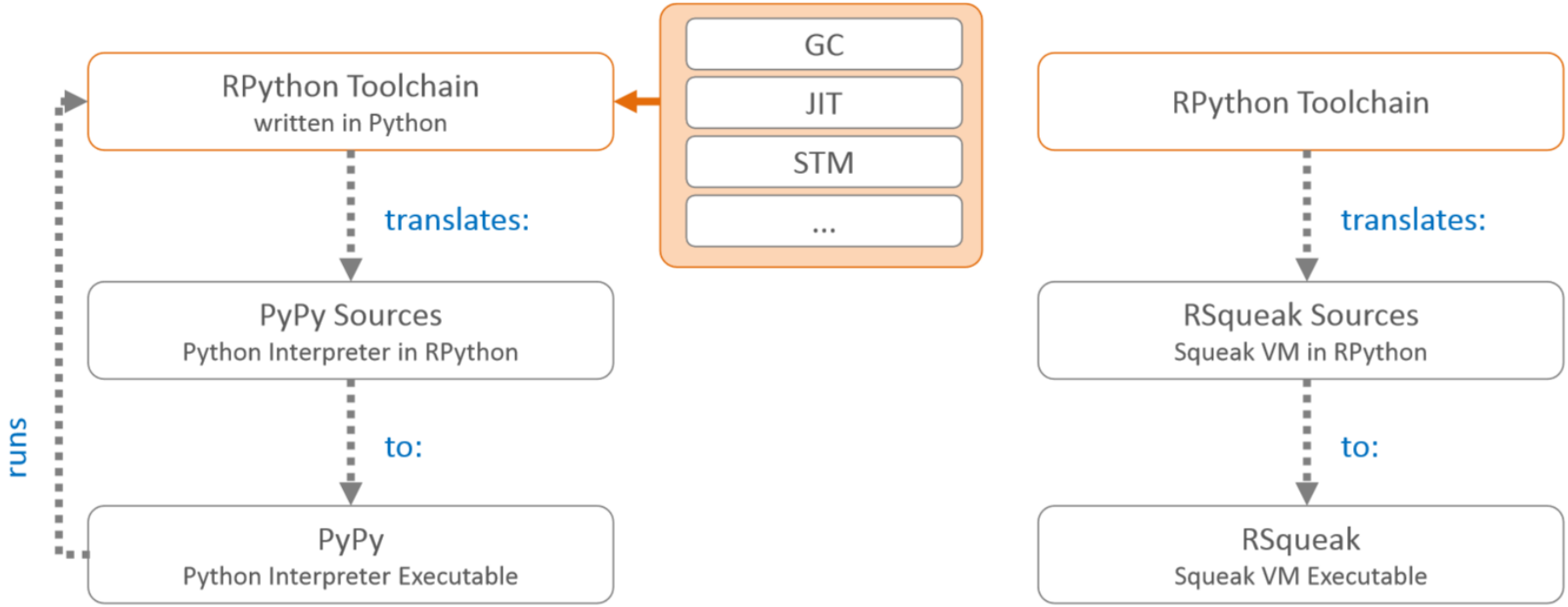
http://www.hpi.uni-potsdam.de/swa/

Goals

- No C or assembler
  - Useful for teaching
- Good performance
  - Think about abstractions and how to lower them
- Small codebase
  - Easy to introduce new students
- Lots of tests
  - Experiments can rely on tests to catch errors

A VM WITHOUT LL CODE

Background: the RPython Toolchain

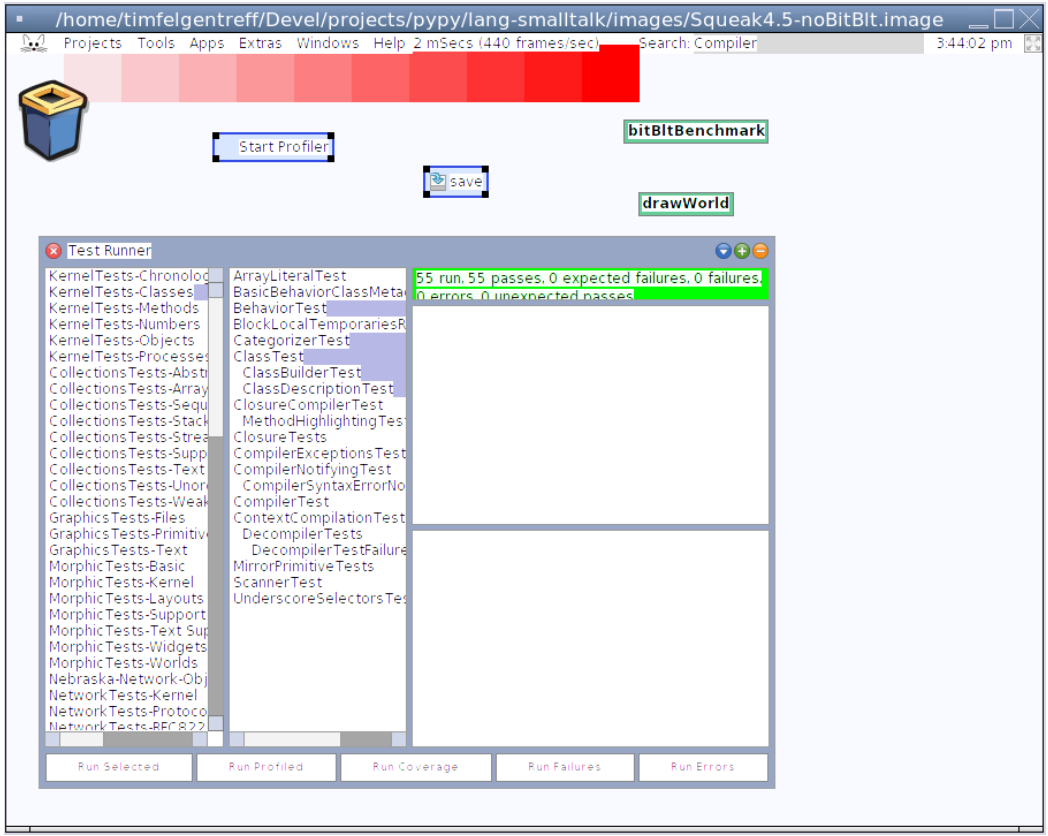


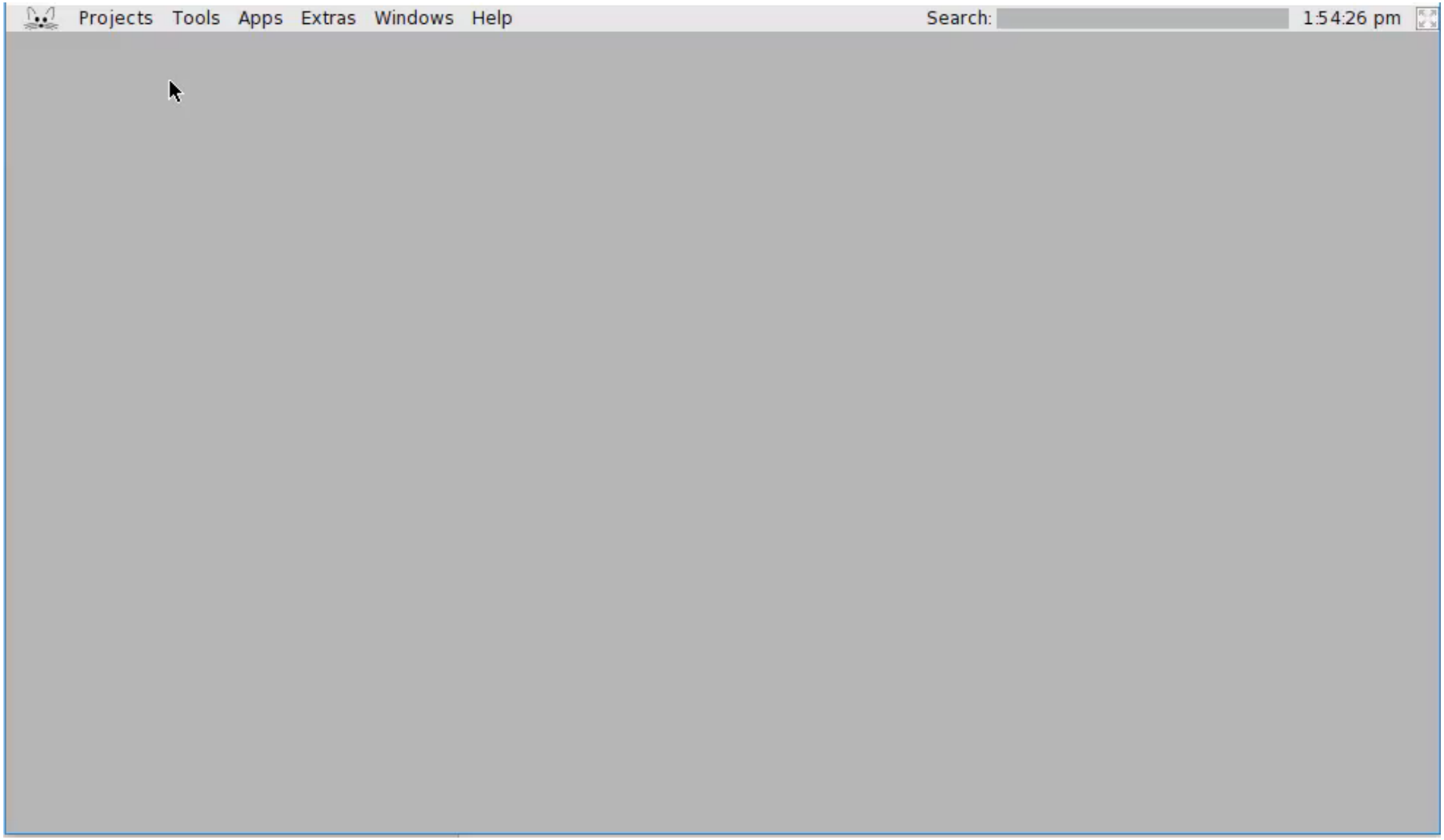
2008: Back to the Future in 1 Week

Slow. Incomplete. Interesting

2013: Tracing Algorithmic Primitives

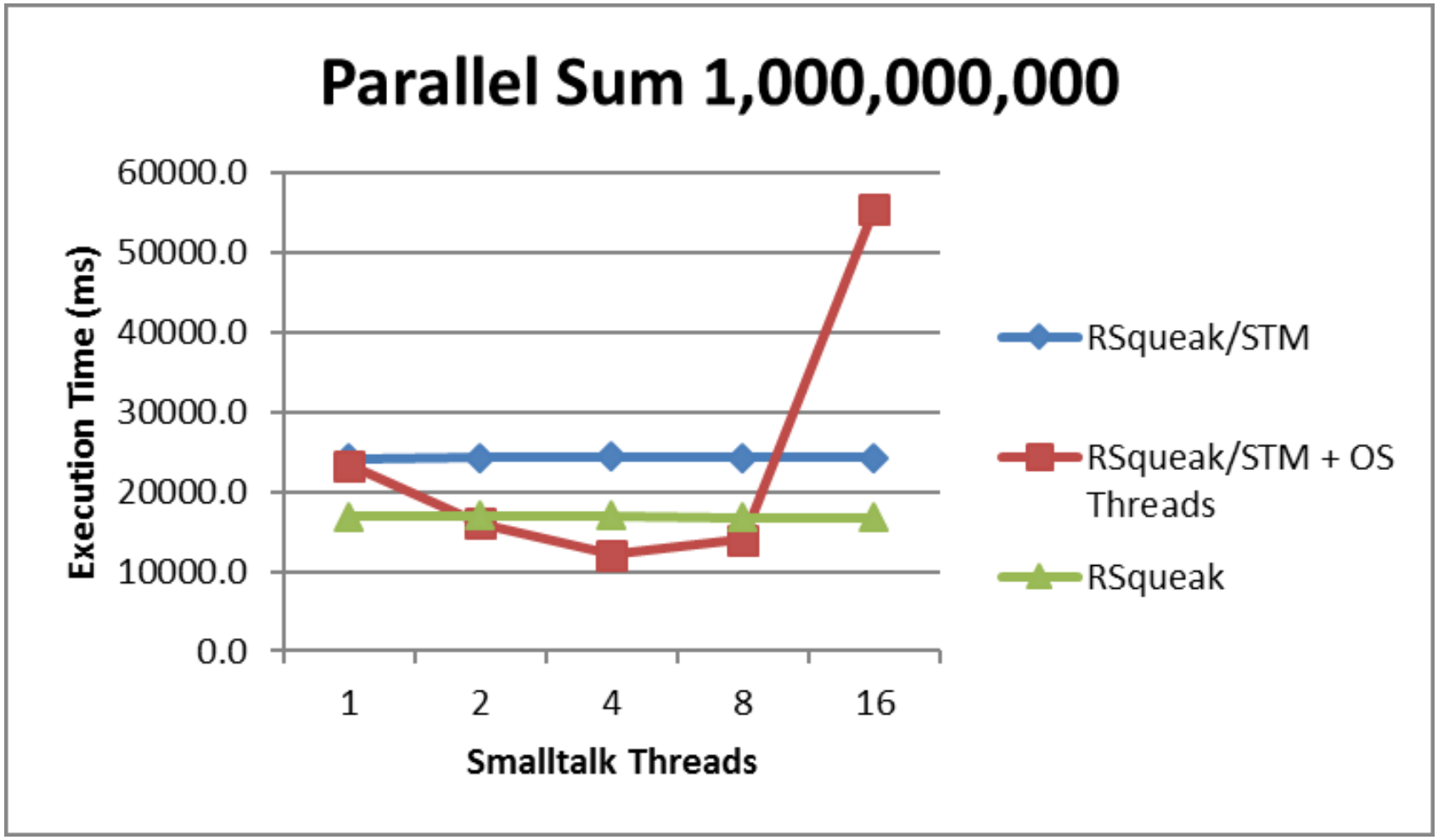
- Lars Wassermann for his Master's thesis transforms Spy VM into RSqueak/VM
- Adds an FFI interface to native Squeak plugins
- Adds JIT annotations
- Supports Closures
- Fast enough to run many primitives from Slang




Projects
Tools
Apps
Extras
Windows
Help
Search:
1:54:26 pm

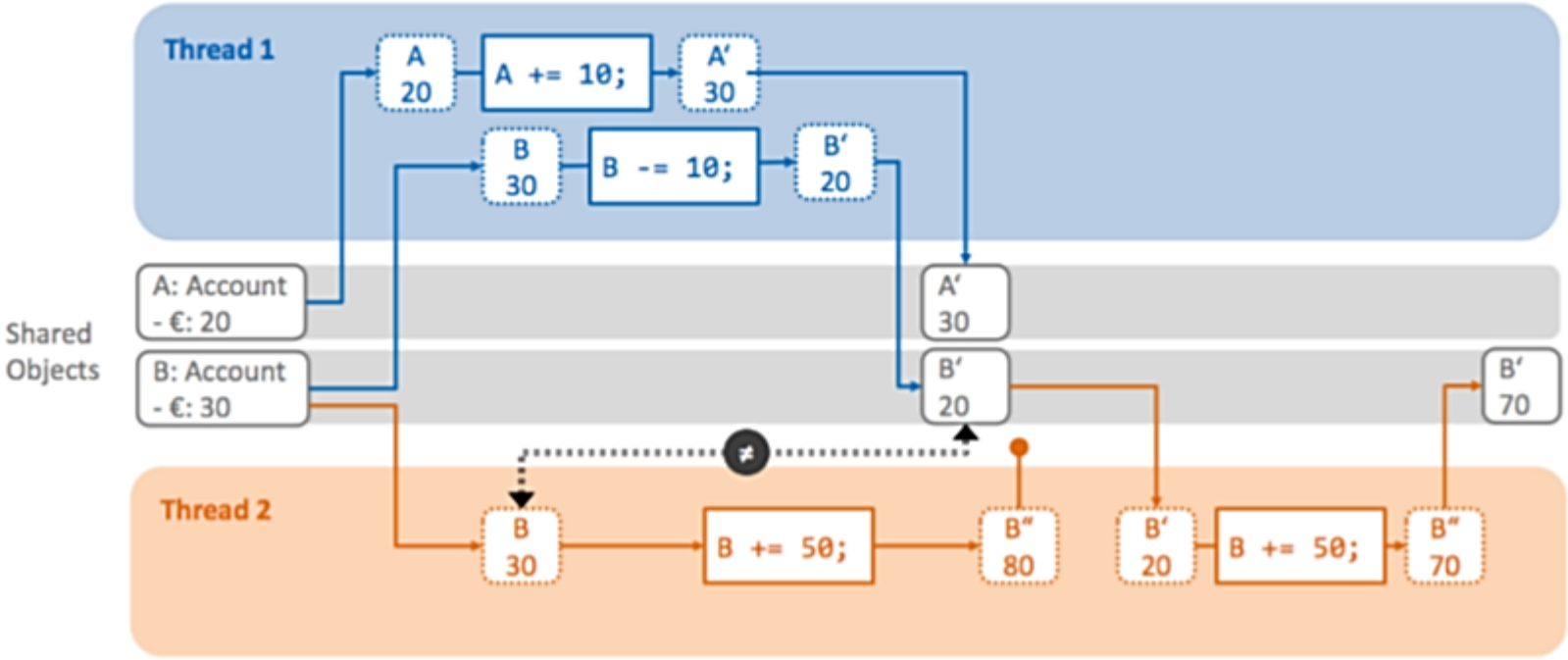
2014: Software Transactional Memory

- 5 students, 3 months



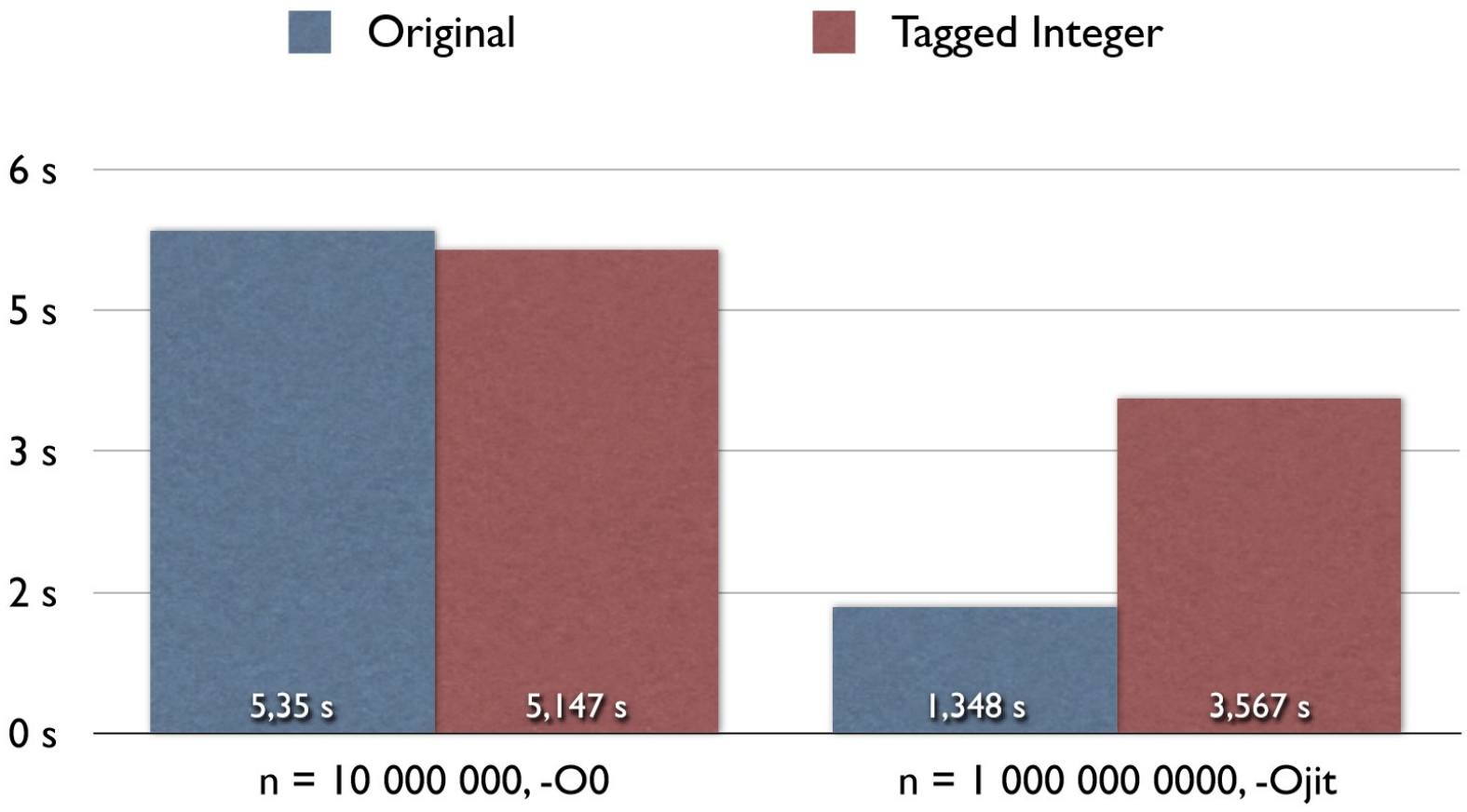
2014: Software Transactional Memory

- Threads see different memory until they commit
- Automatically re-execute conflicts



2014: Tagged vs Boxed Integers

- 3 students, 3 months



2015: Objects as Methods

- 3 students, 3 months

```
+        if not isinstance(w_method, model.W_CompiledMethod):
+            if w_arguments:
+                self.push_all(w_arguments)
+            #fixme: this should be passed to the primitive in another way
+            self.push(w_selector)
+            try:
+                return self._call_primitive(248, interp, argcount, w_method, w_selector)
+            except error.PrimitiveFailedError:
+                #err well, this does not happen
+                as
+                re
```

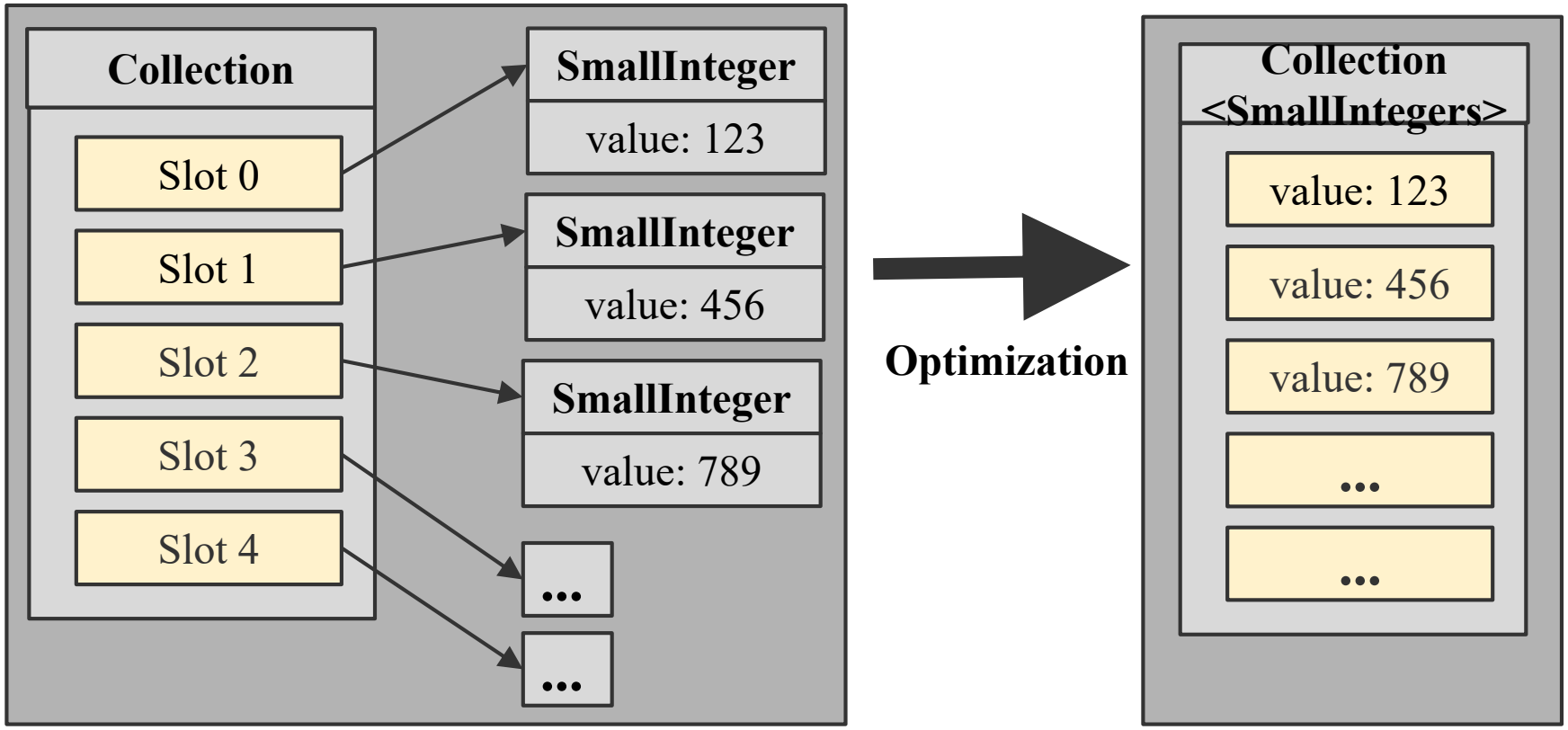
```
+@expose_primitive(VM_INVOKE_OBJECT_AS_METHOD, method_object=True, result_is_new_frame=True
+def func(interp, s_frame, argcount, w_objectAsMethod):
+    w_selector = s_frame.pop()
+    args = s_frame.pop_and_return_n(argcount)
+    arguments_w = interp.space.wrap_list(args)
+    w_rcvr = s_frame.pop()
+    w_newrcvr = w_objectAsMethod
+    s_frame.push(w_newrcvr)
+
+    w_newarguments = [w_selector, arguments_w, w_rcvr]
+
+    return s_frame._sendSpecialSelector(interp, w_newrcvr, "runWithIn", w_newarguments);
+
```

2015: Allocation Removal Strategies

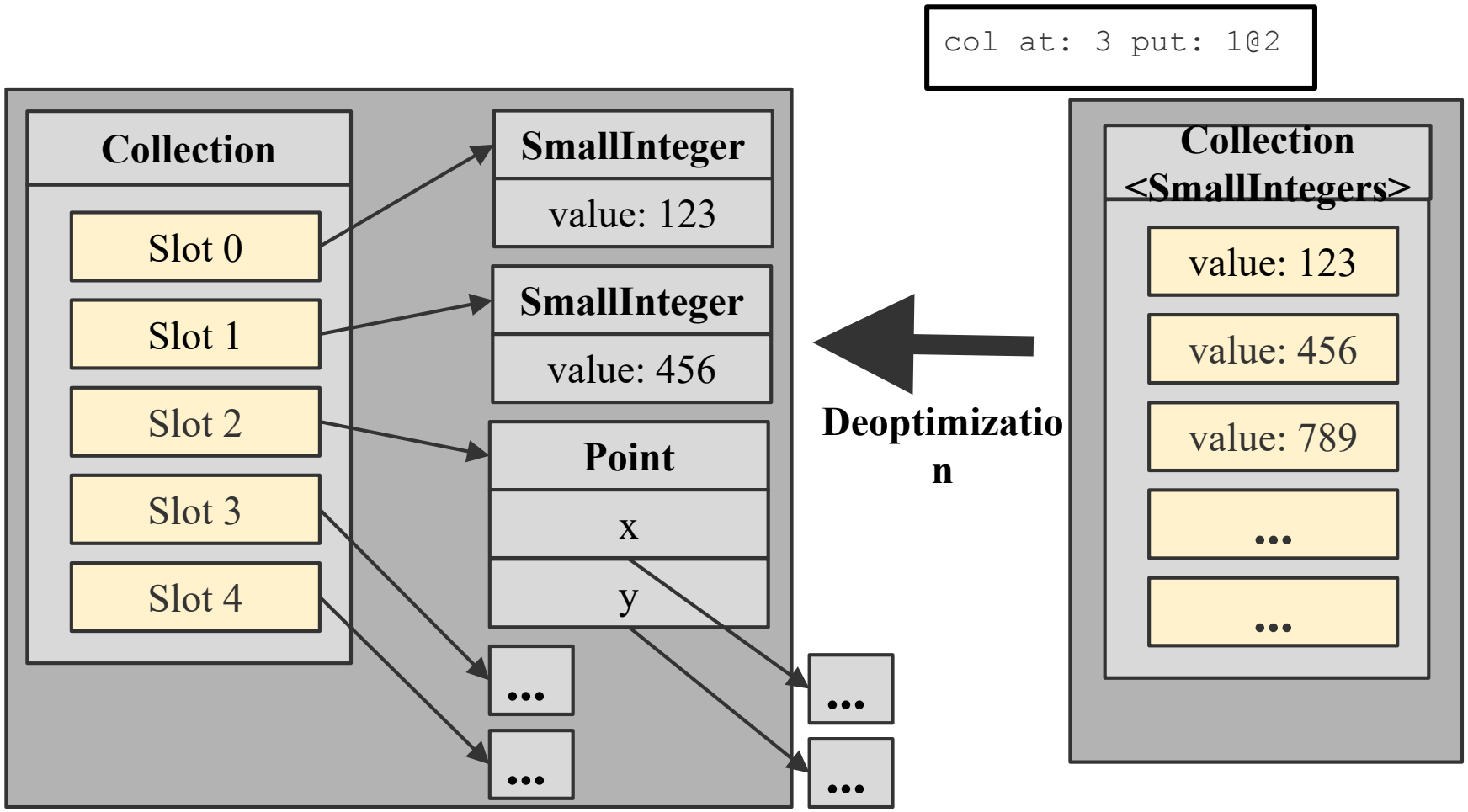
- 1 student, 6 months
- adds a generic interface in RPython (and uses it in RSqueak) to avoid allocations of special objects in homogeneous objects

BitBlt benchmark	in C	in Smalltalk
Interpreter VM	650ms	389,660ms
	1 x C	599 x C
Cog JIT VM	790ms	336,490ms
	1 x C	423 x C
R/SqueakVM	880ms	20,310ms
	1 x C	23 x C

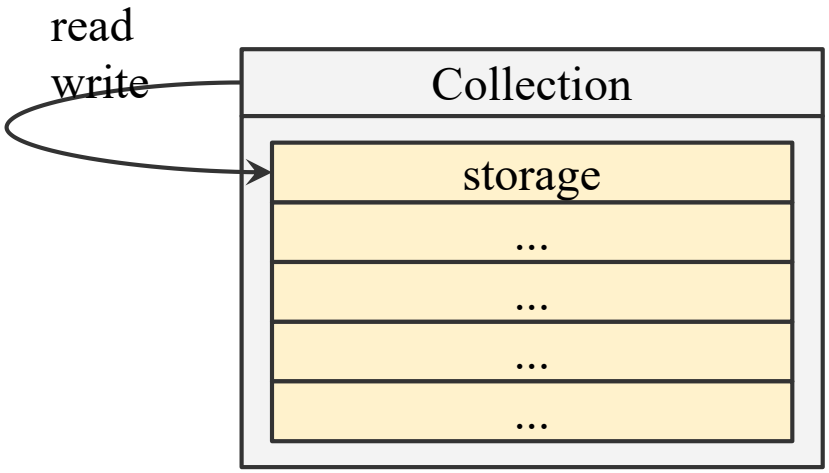
Storage Strategies



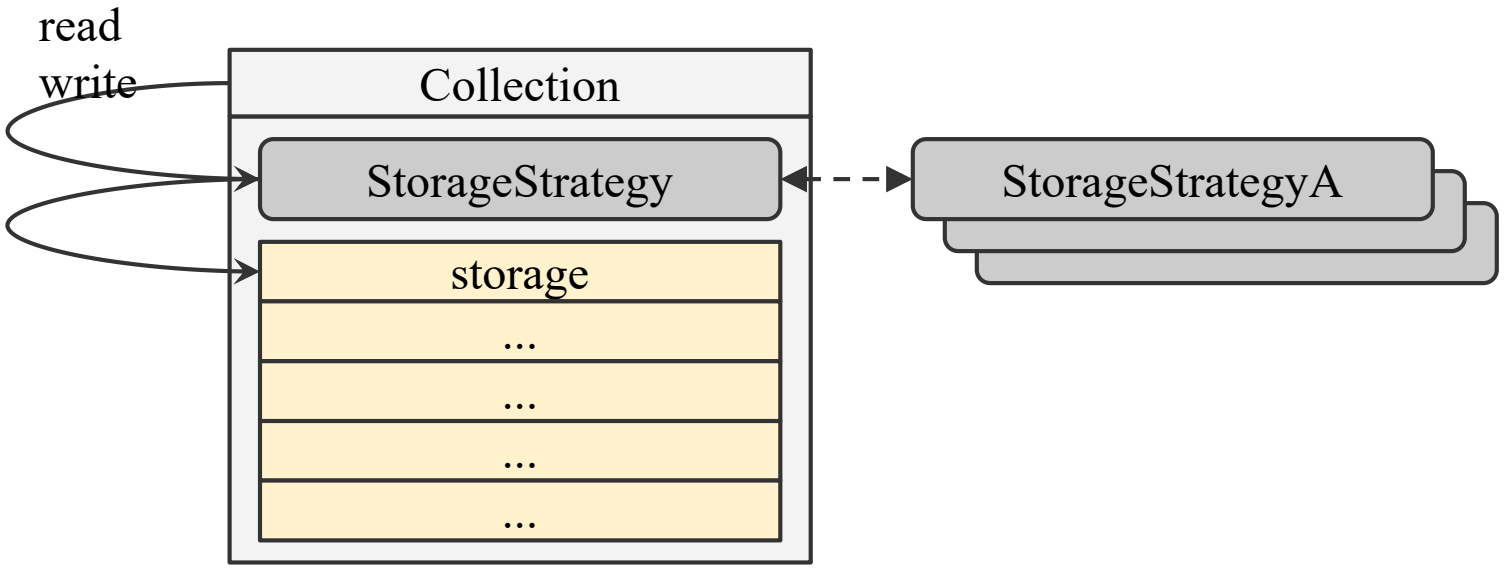
Storage Strategies



Storage Strategies



Storage Strategies



Increment size of OrderedCollection

```
i215 = int_add_ovf(i213, 1)
p227 = new_with_vtable(ConstClass(W_SmallInteger))
setfield_gc(p227, i215, W_SmallInteger.inst_value)
setarrayitem_gc(p147, 2, p227)
```

Without Strategies

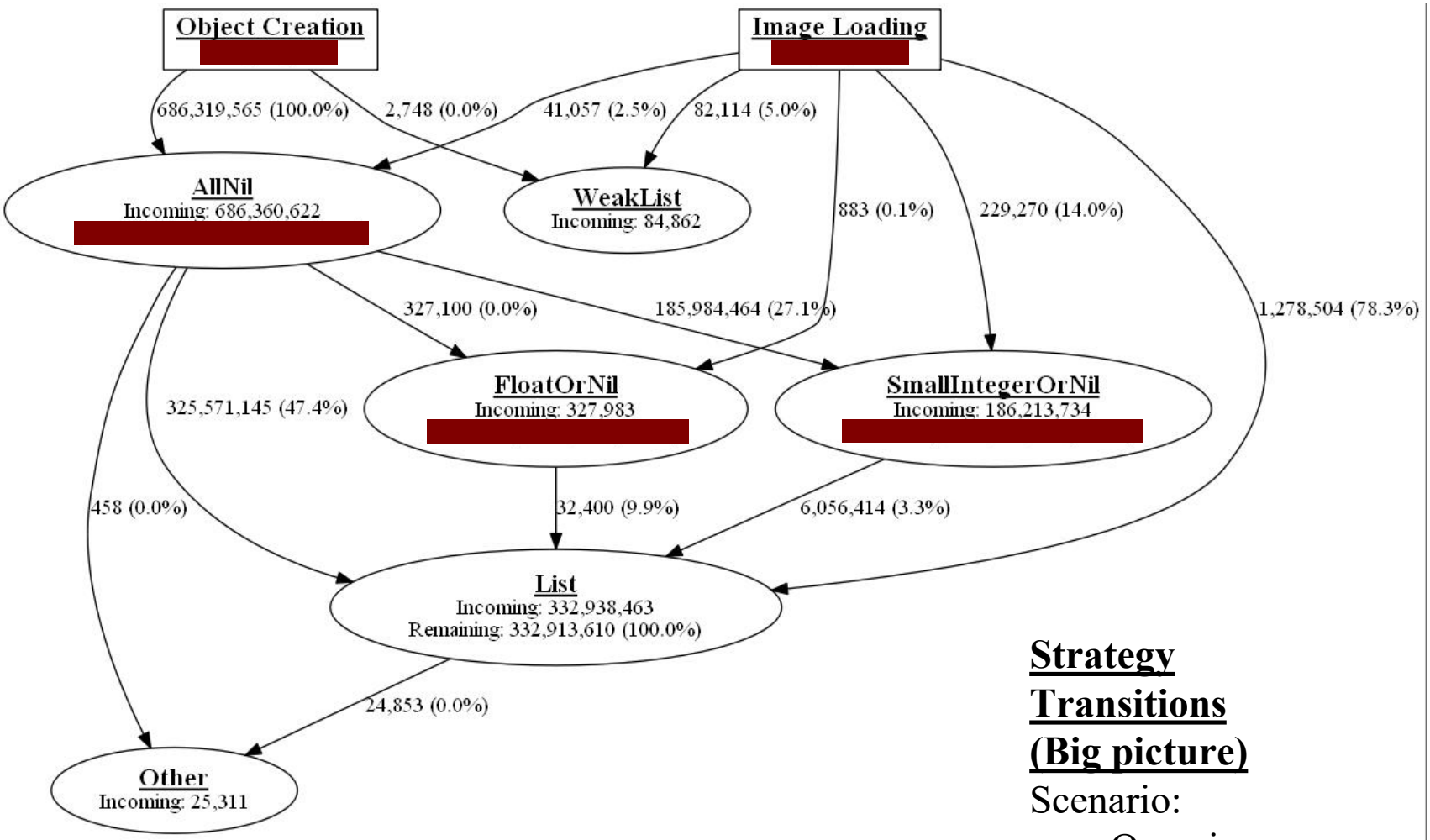
Store new value in Array

```
i207 = int_add_ovf(i194, 2)
p231 =
new_with_vtable(ConstClass(W_SmallIntege
r))
setfield_gc(p231, i207,
W_SmallInteger.inst_value)
setarrayitem_gc(p220, i222, p231)
```

With Strategies

Store new value in Array

```
i204 = int_add_ovf(i189,
2)
i219 = int_ne(i204,
2147483647)
guard_true(i219)
setarrayitem_gc(p211,
i213, i204)
```



Strategy Transitions (Big picture)

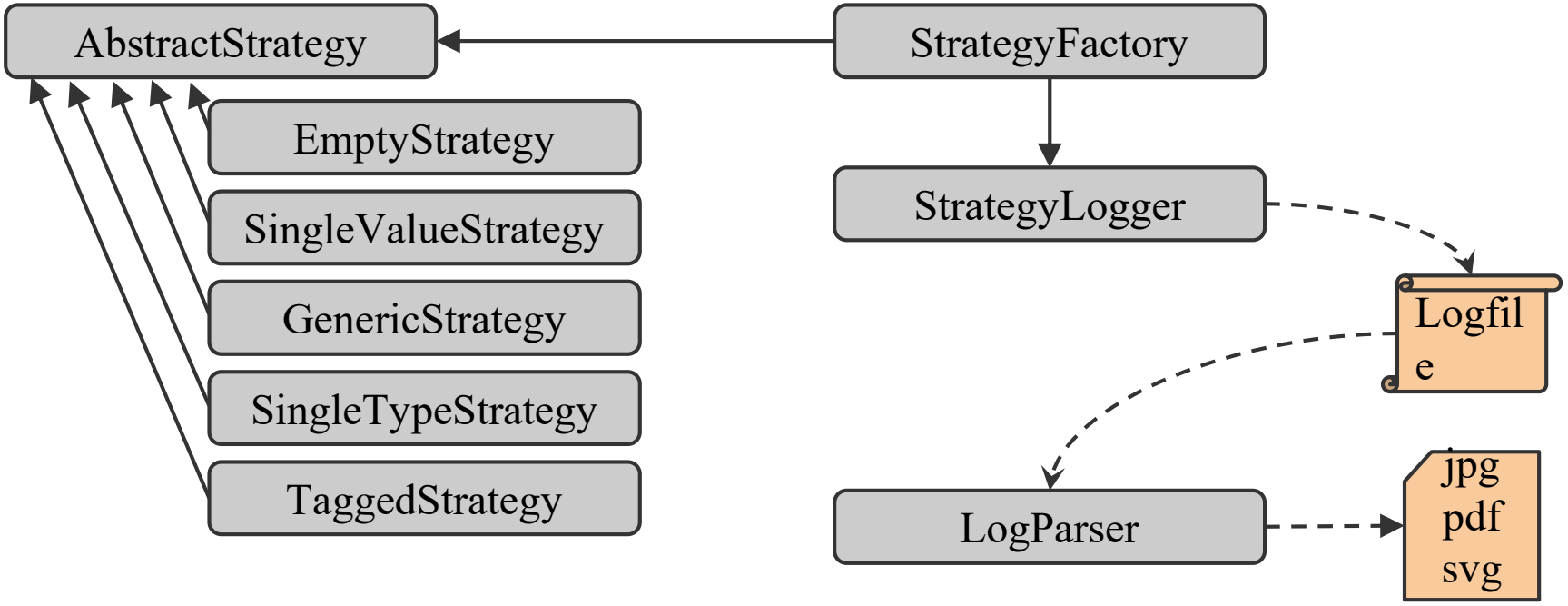
Scenario:

- Open image
- Use browser
- Close image

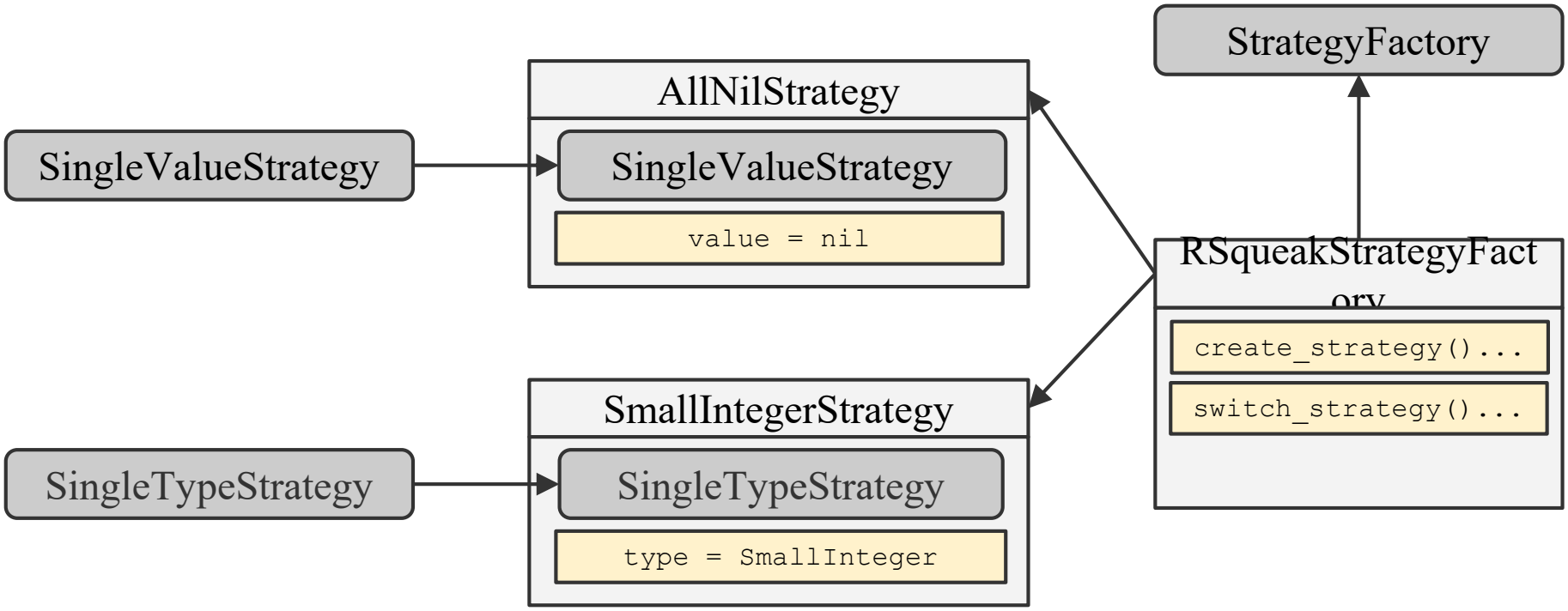
Evaluation: Performance

Benchmark	Without Strategies	With Strategies	Performance Change
AStar1	80.64 ms ±1.92	55.81 ms ±2.61	±6.58
AStar2	351.17 ms ±8.15	288.81 ms ±2.68	±2.25
BinaryTree	187.47 ms ±1.00	174.94 ms ±1.77	+7.26 % ±1.01
BlowfishDecryption	422.16 ms ±2.05	429.16 ms ±2.49	±0.64
BlowfishEncryption	423.85 ms ±2.00	427.15 ms ±2.47	±0.63
DeltaBlue	99.86 ms ±2.24	98.31 ms ±2.16	+1.57 % ±2.62
NBody	271.38 ms ±2.15	274.09 ms ±2.16	±0.94
Richards	165.97 ms ±2.80	162.95 ms ±4.14	+2.11 % ±2.29
SplayTree	781.16 ms ±2.25	469.92 ms ±2.89	±0.97

rstrategies: Architecture



rstrategies: Usage



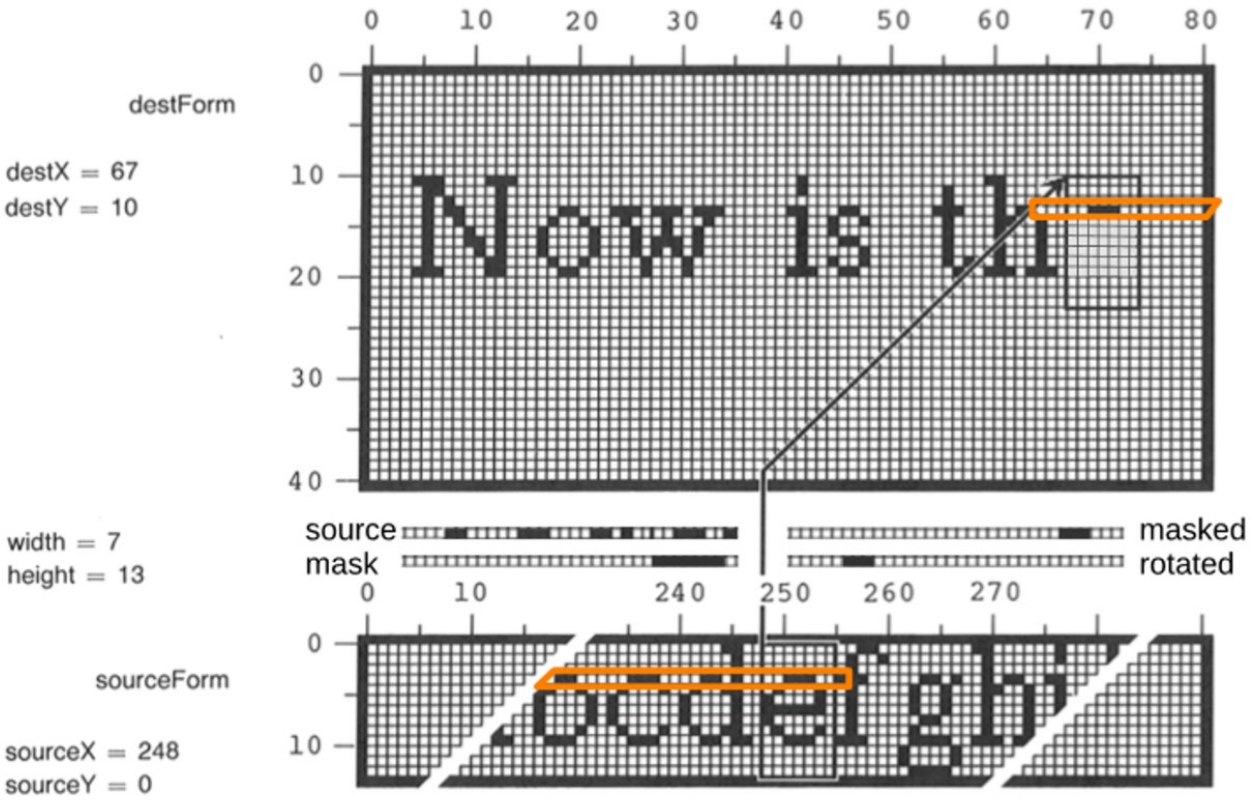
rstrategies: Usage

```
@rstrat.strategy(generalize=[
    SmallIntegerOrNilStrategy,
    FloatOrNilStrategy,
    ListStrategy])
class AllNilStrategy(AbstractStrategy):
    repr_classname = "AllNilStrategy"
    import_from_mixin(rstrat.SingleValueStrategy)
    def value(self): return self.space.w_nil
```

The tracing JIT optimizations from up high

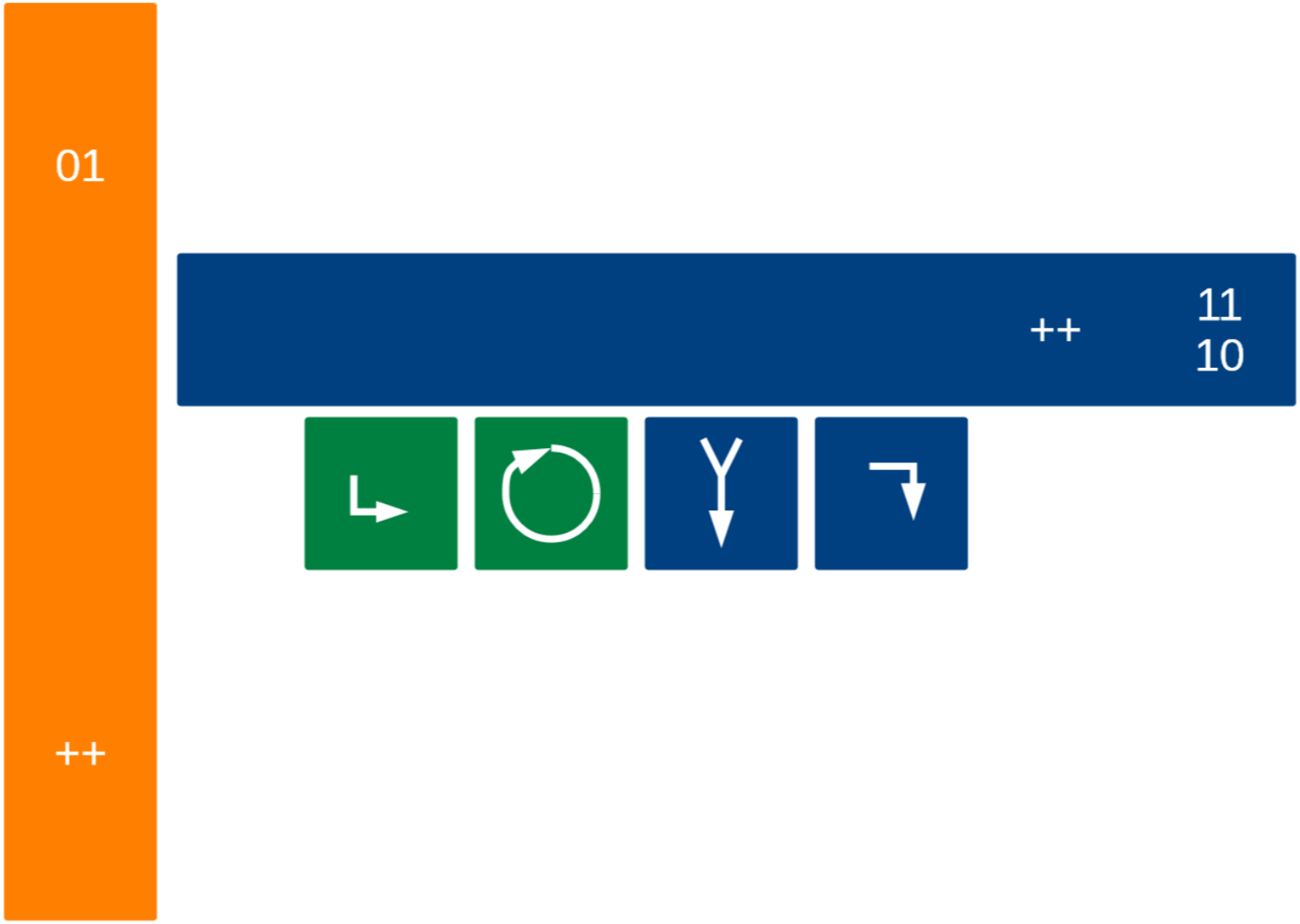
SIDETRACK

BitBlt

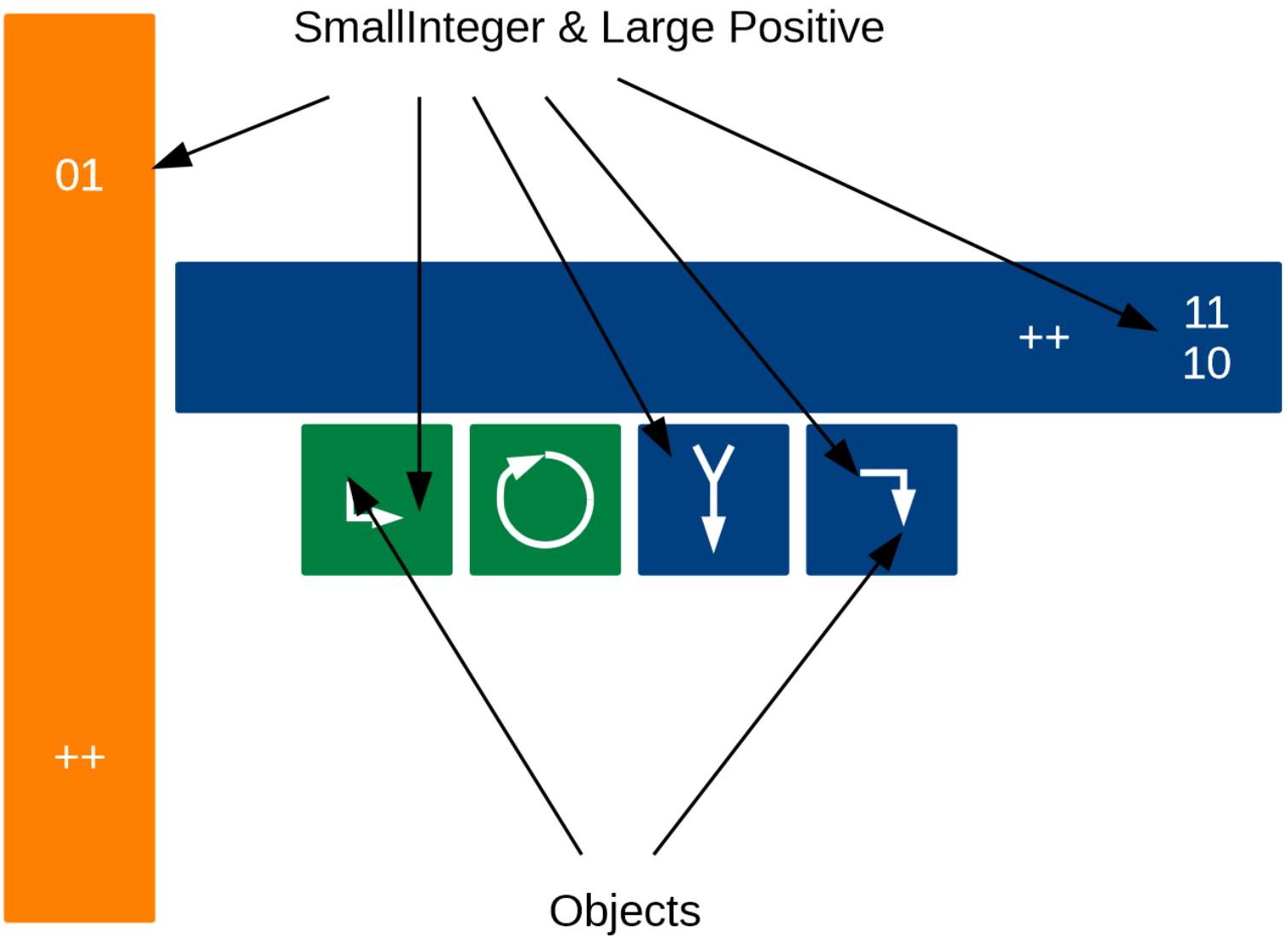


[Smalltalk-80 Bluebook]

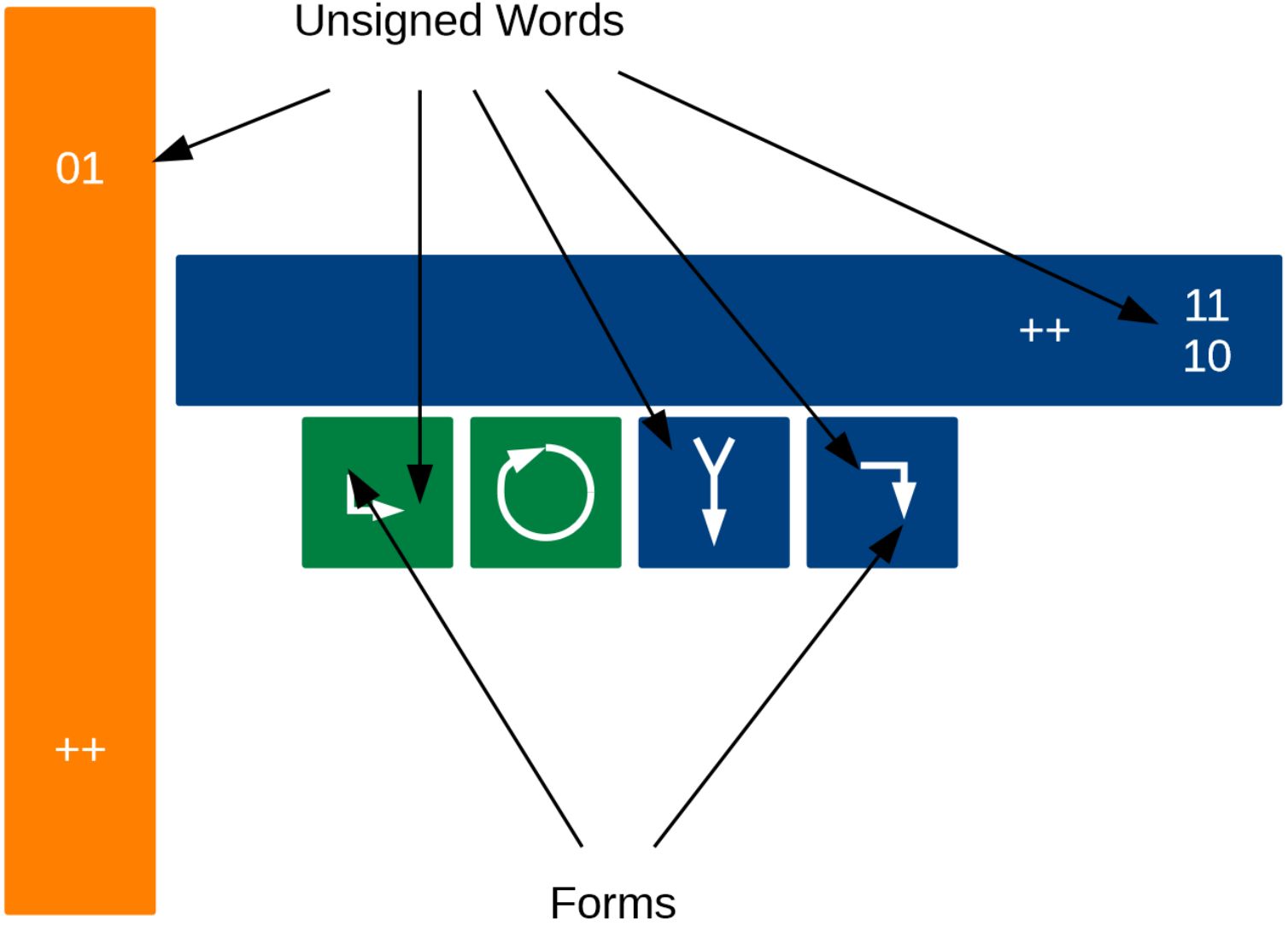
Algorithm Structure



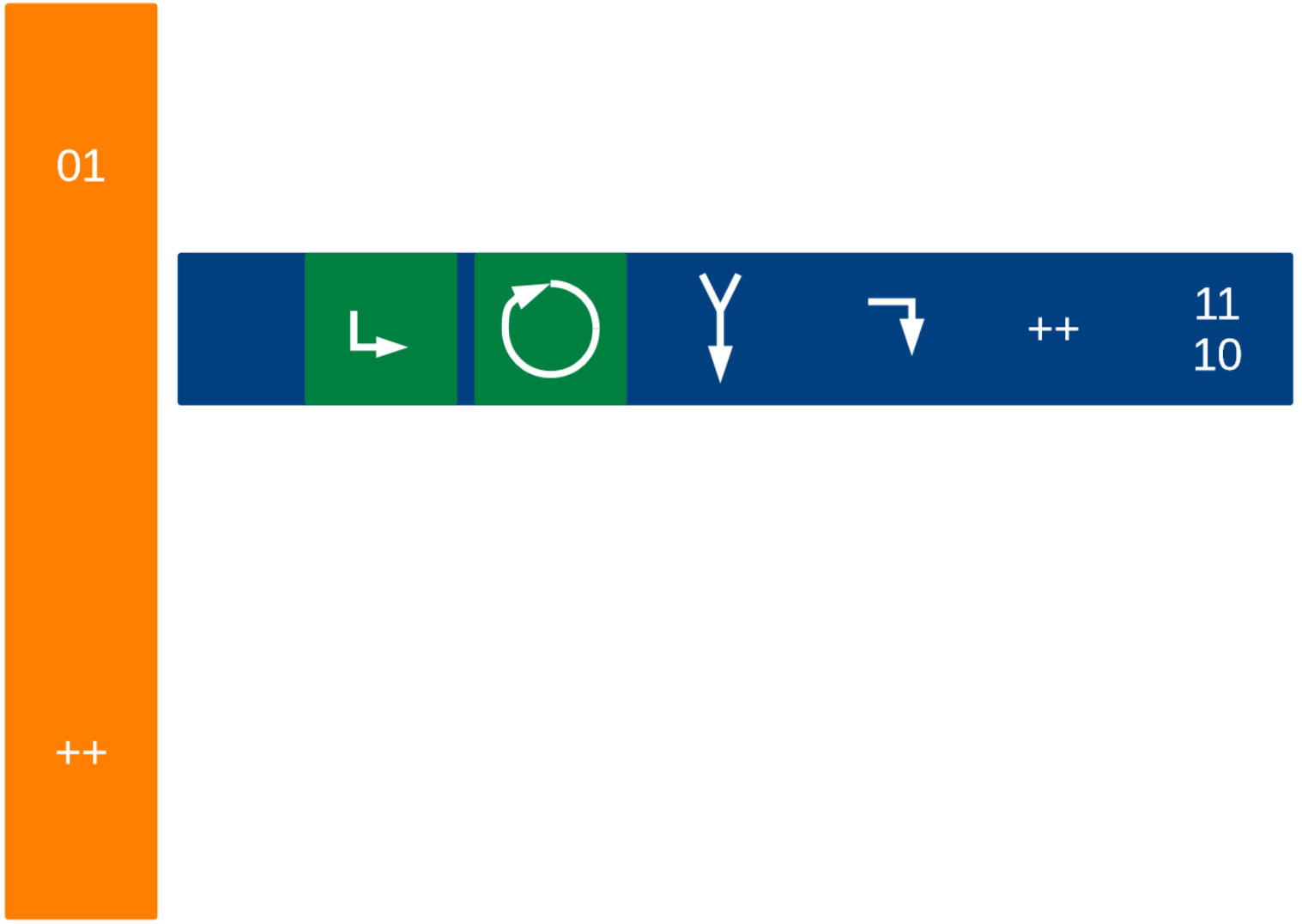
Type Specialization



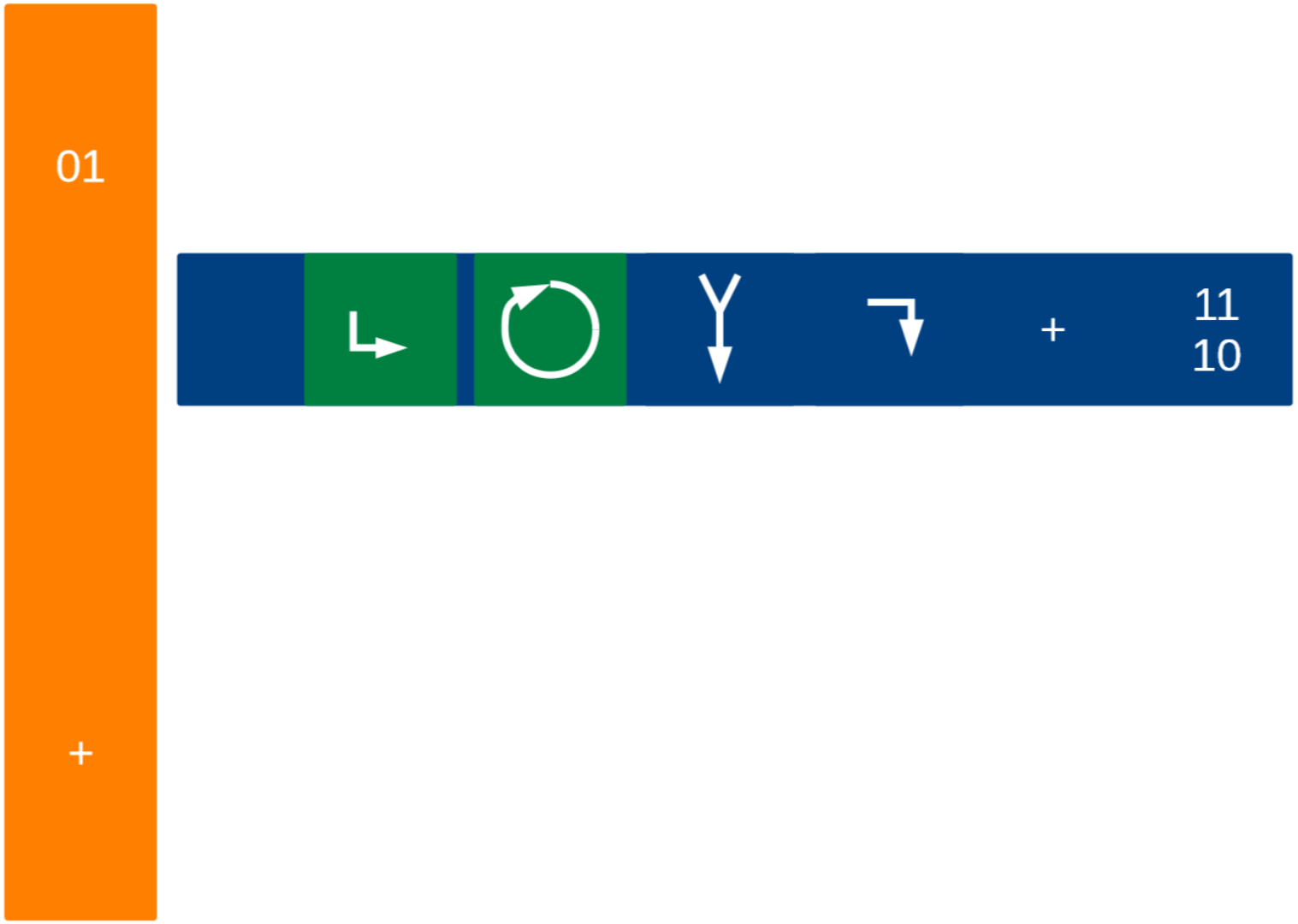
Type Specialization



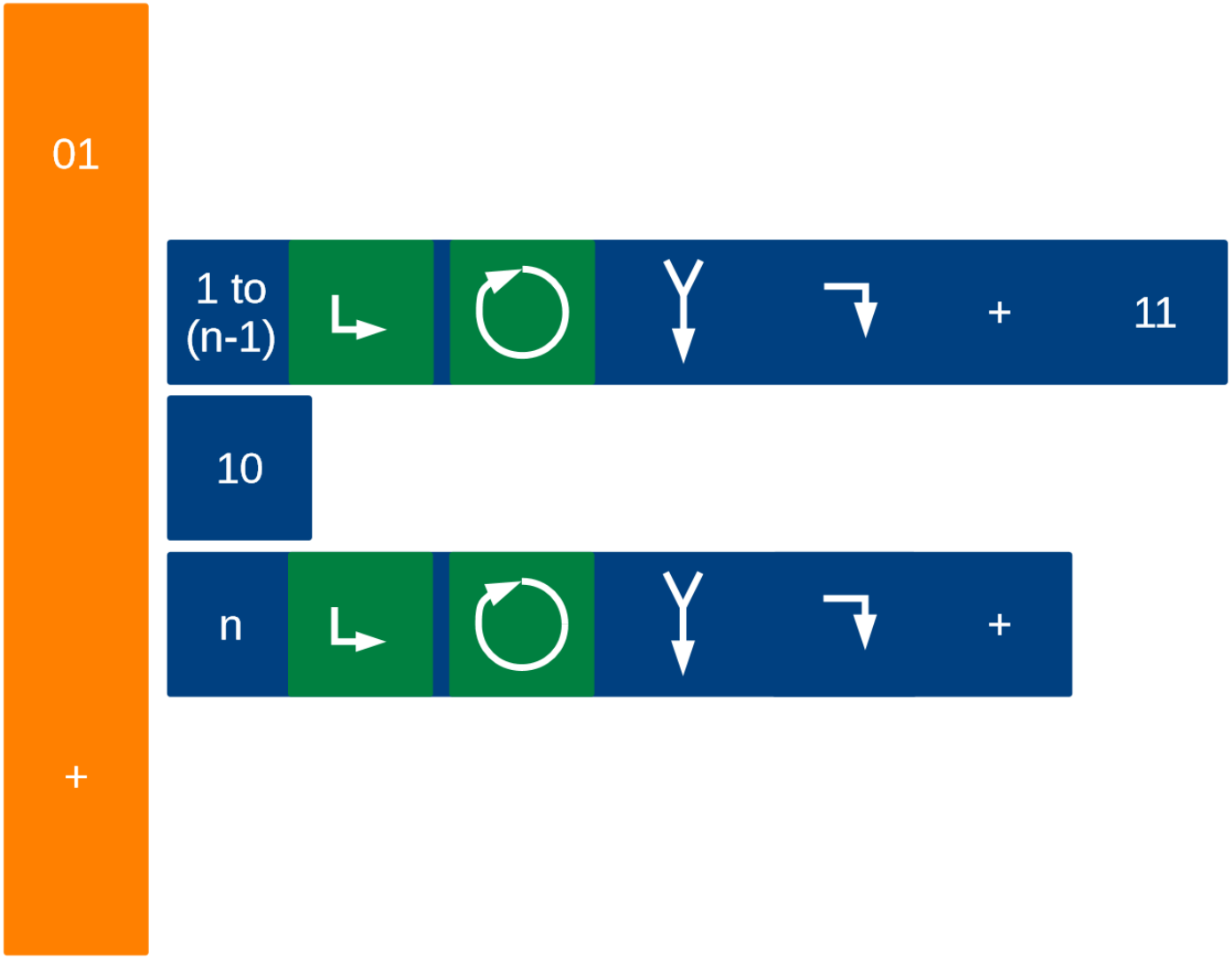
Inlining



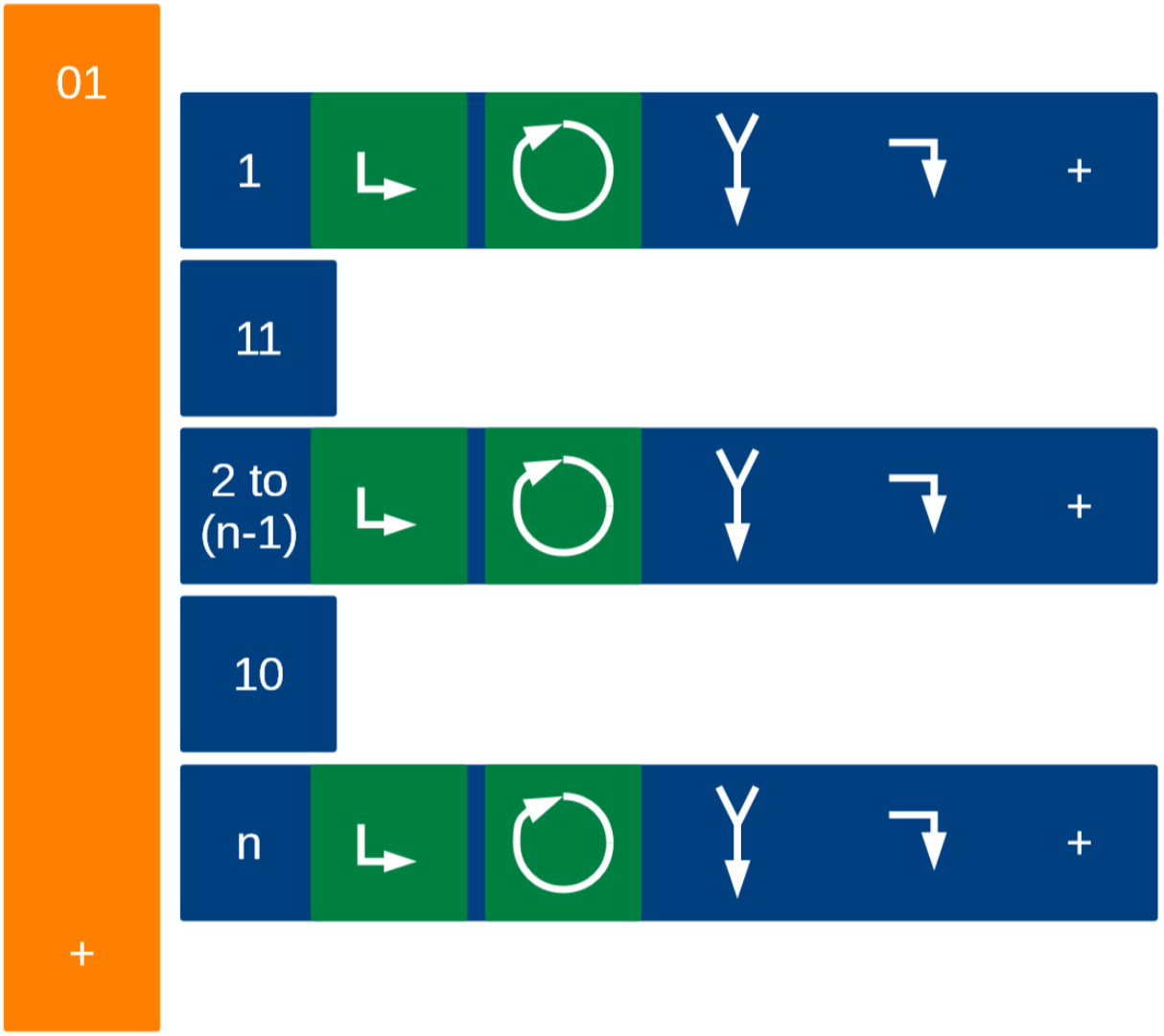
Folding



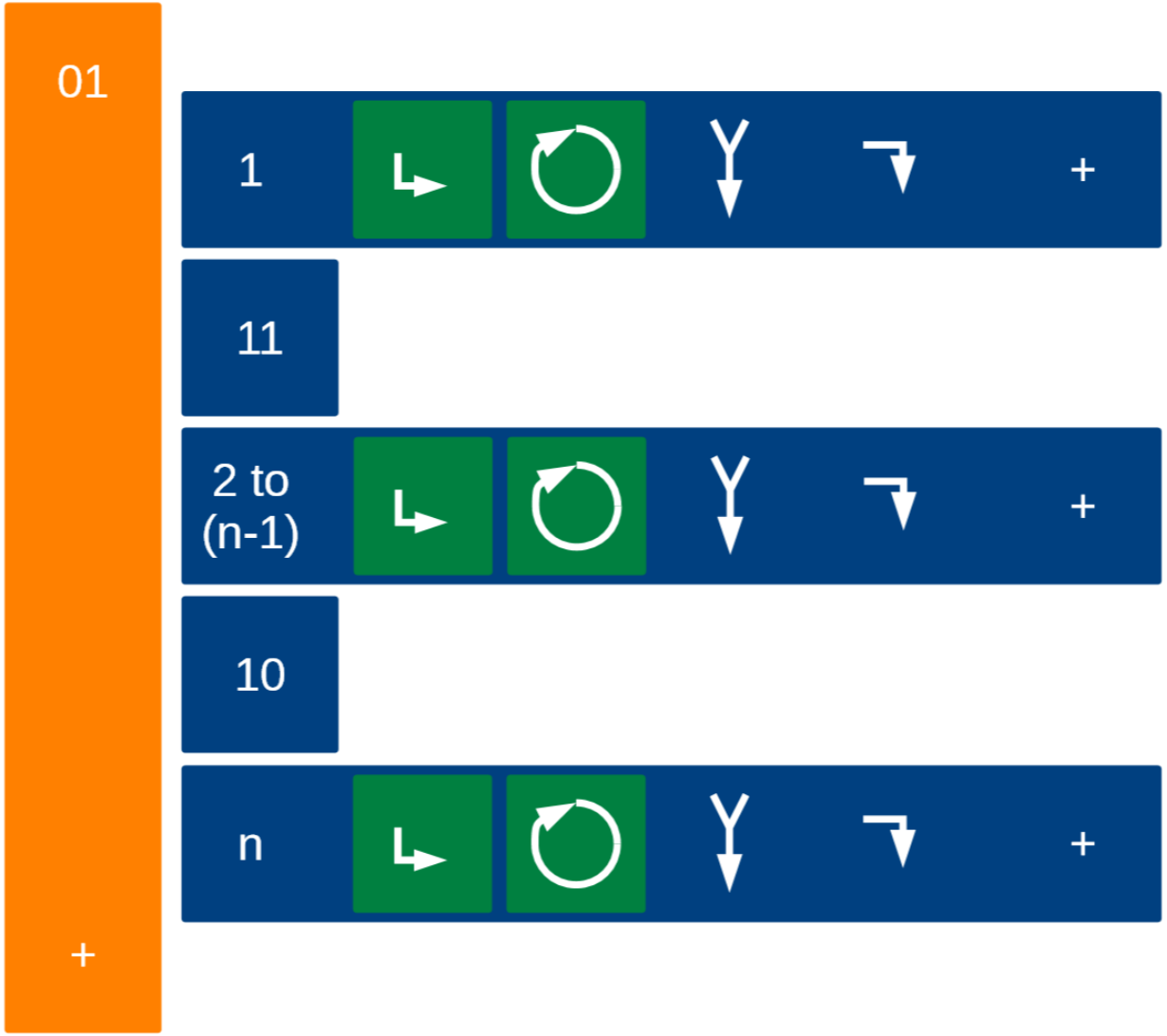
Loops



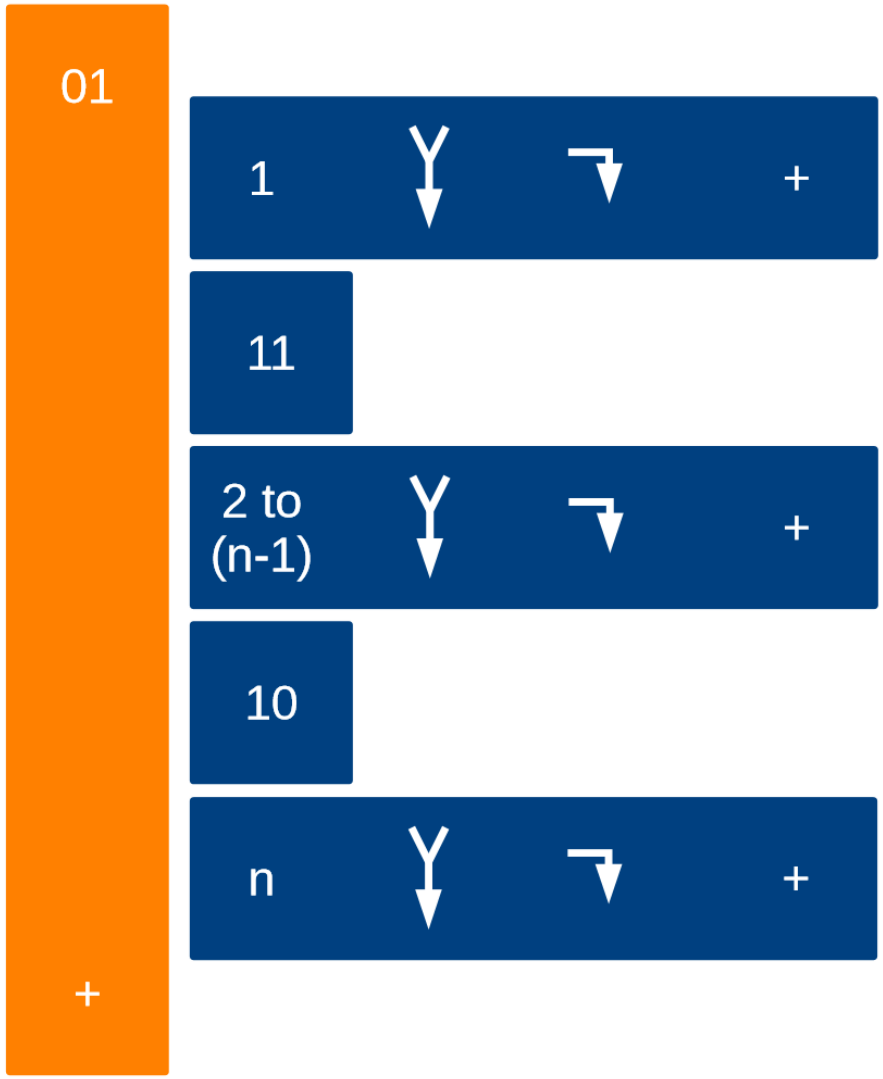
Loops



Branch Pruning

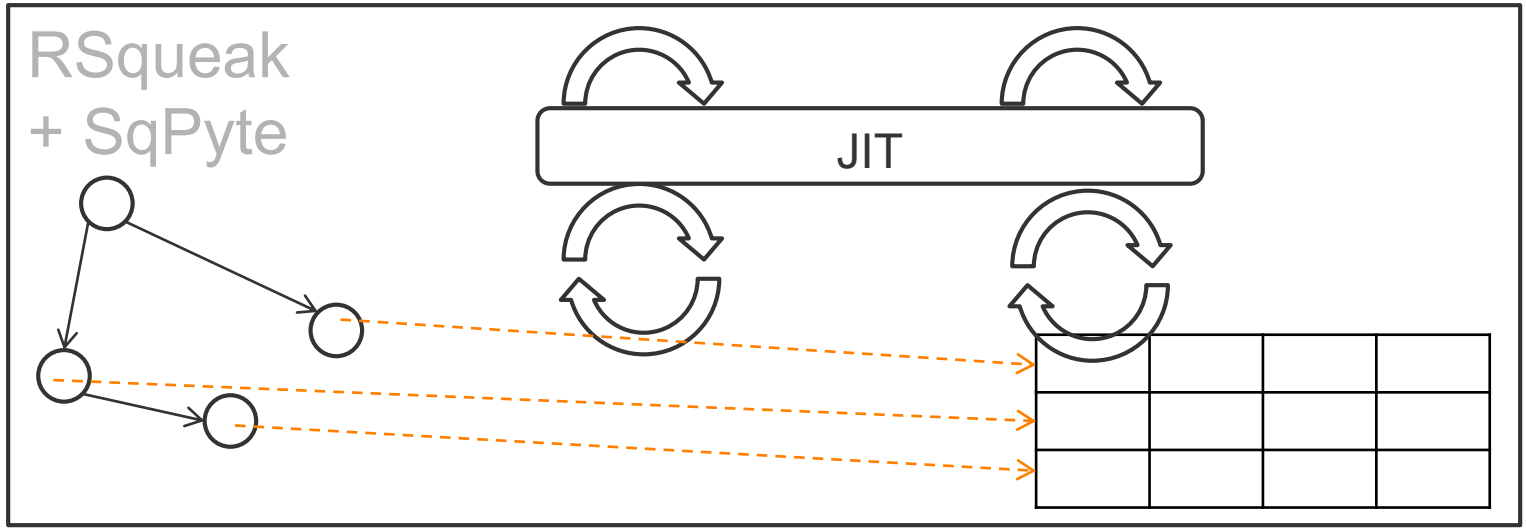


Branch Pruning



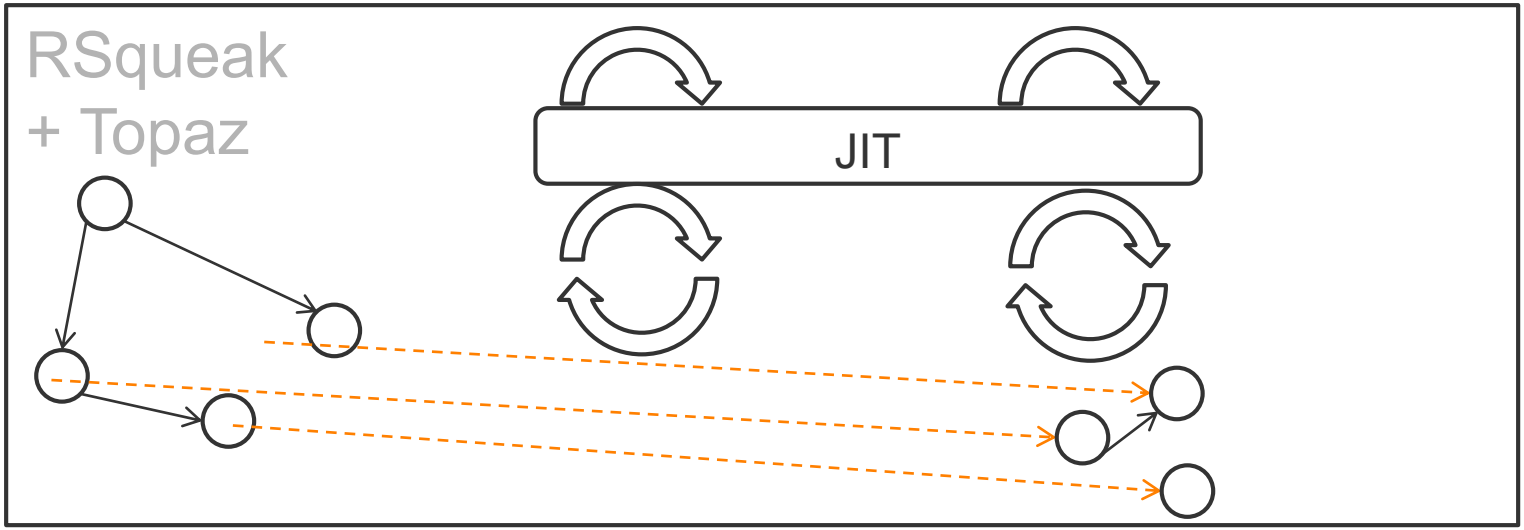
2016: RSqueak + SQLite

- Joint-execution JIT and shared object space for SQLite and RSqueak
- 5 students, 3 months
- ~25% speed-up



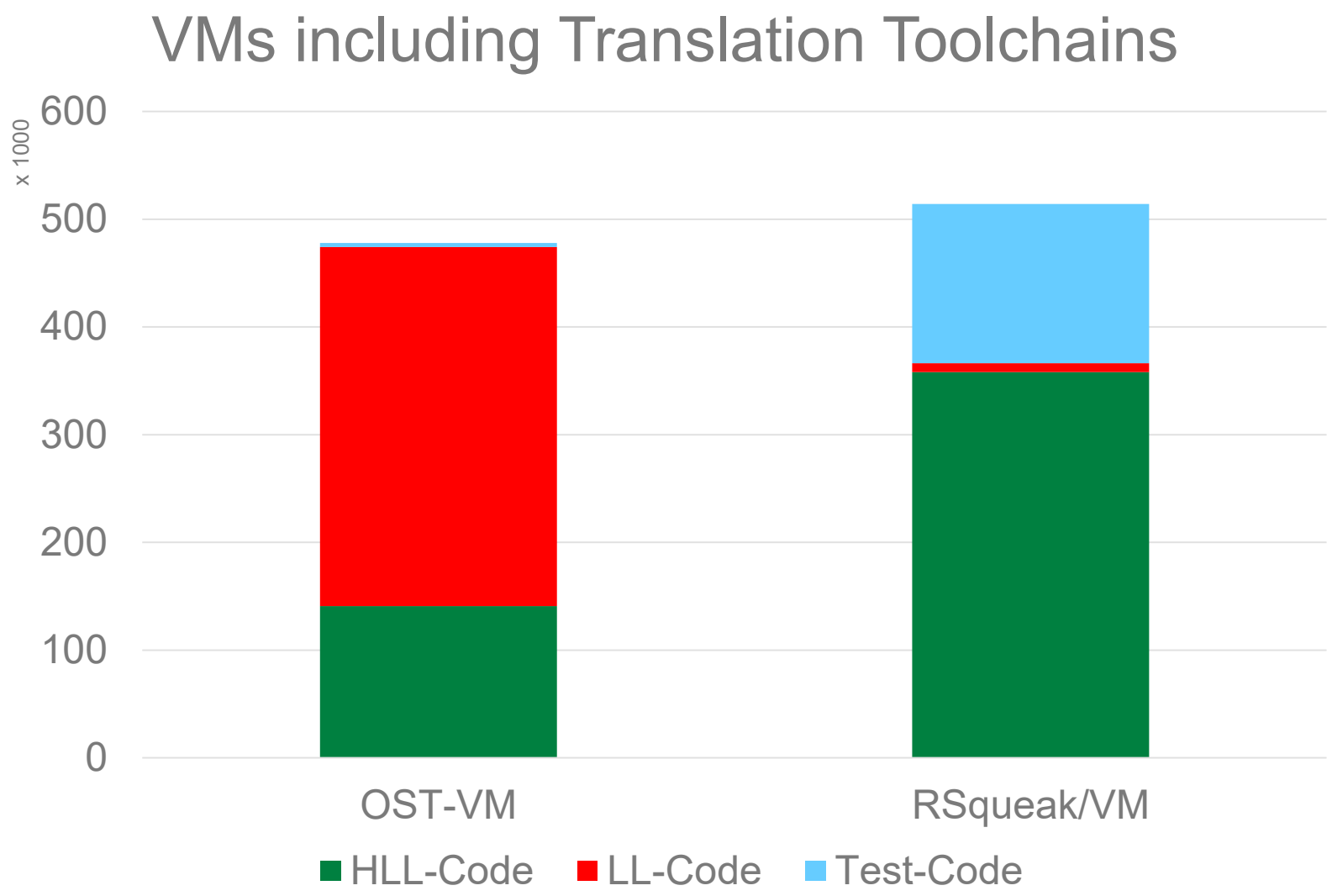
2016: RSqueak + Topaz Ruby

- Joint-execution JIT and shared object space for Topaz and RSqueak
- me, 2 days

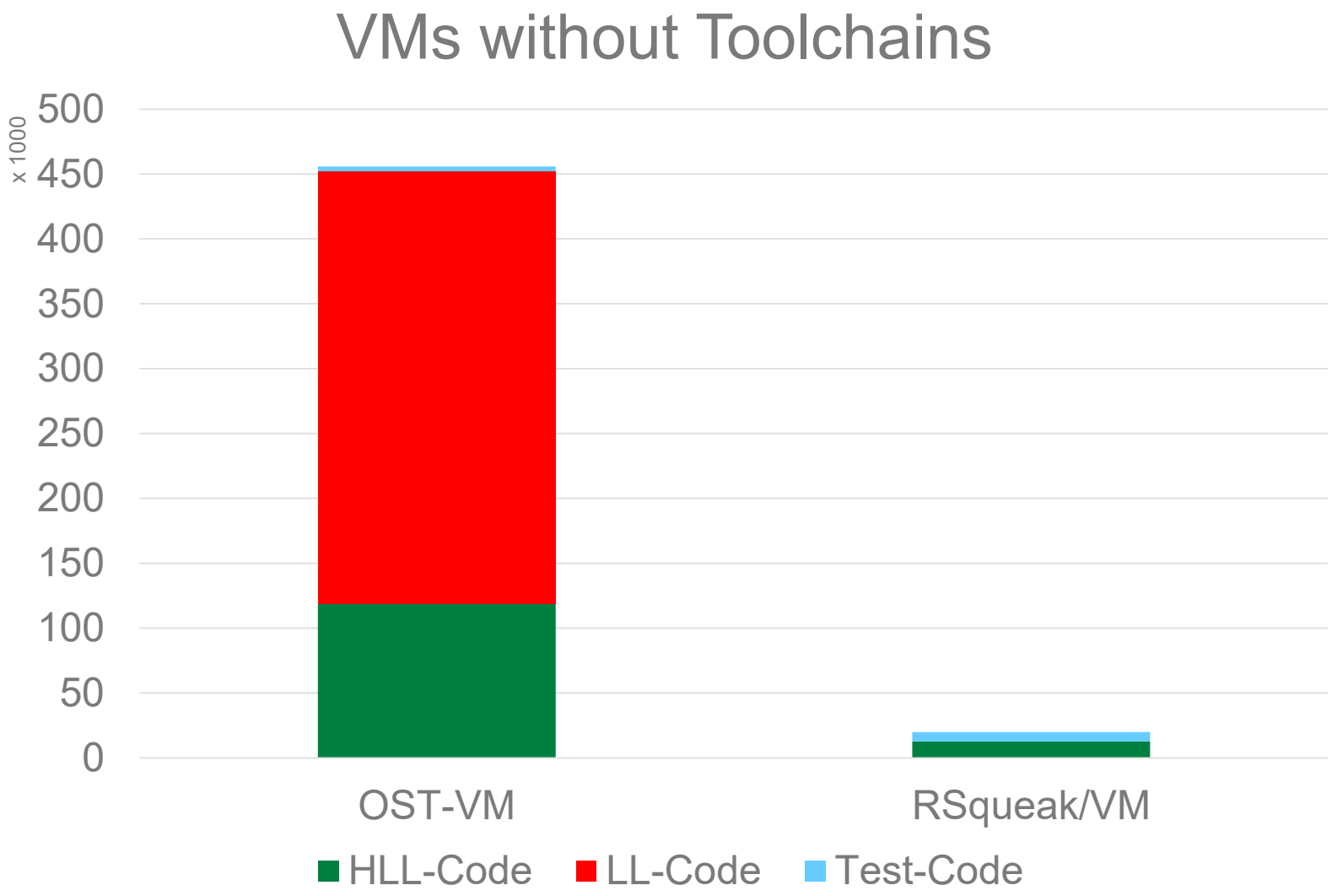


THE (SMALL-ISH) CODEBASE

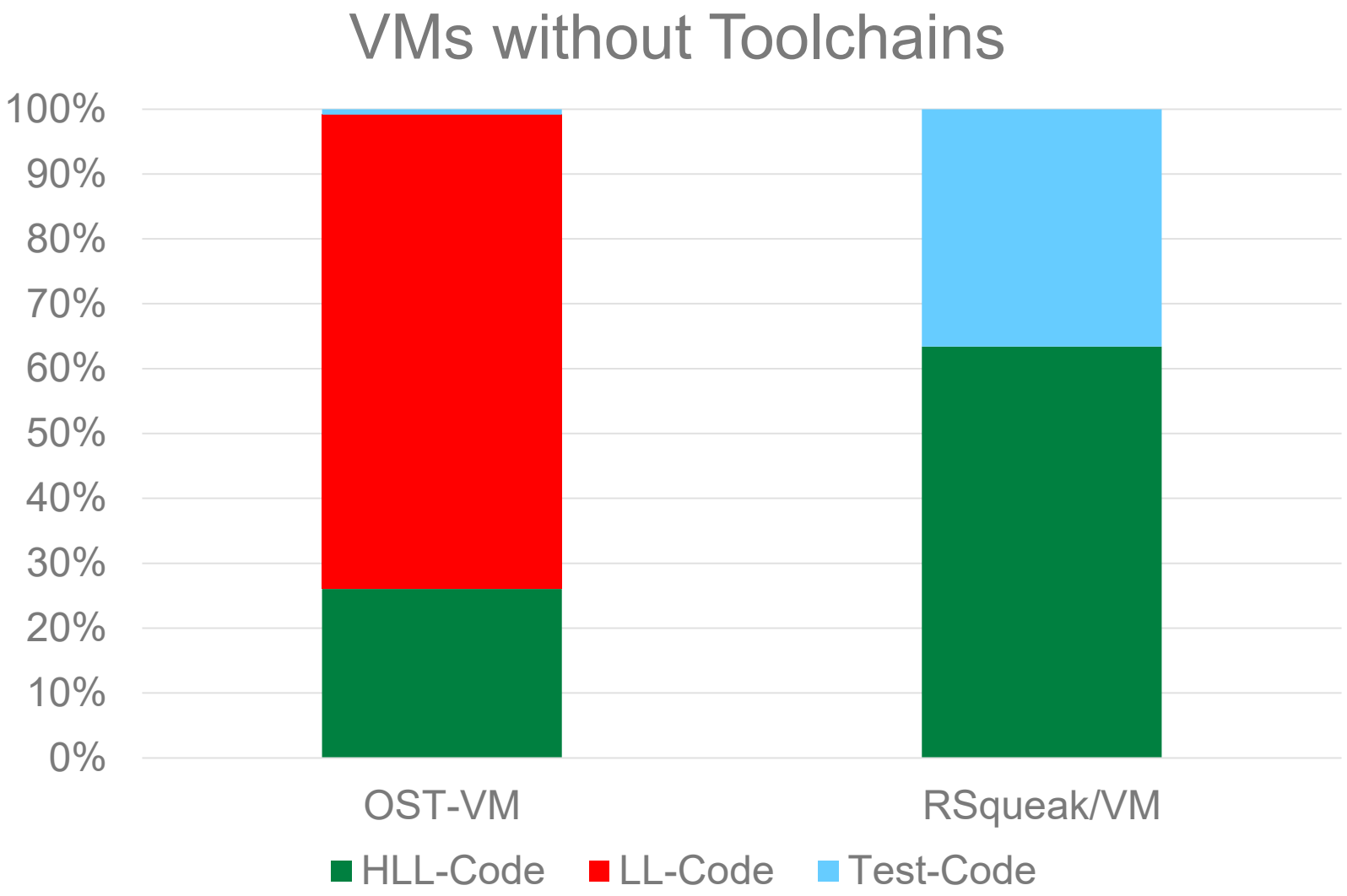
Absolute Size of Codebases



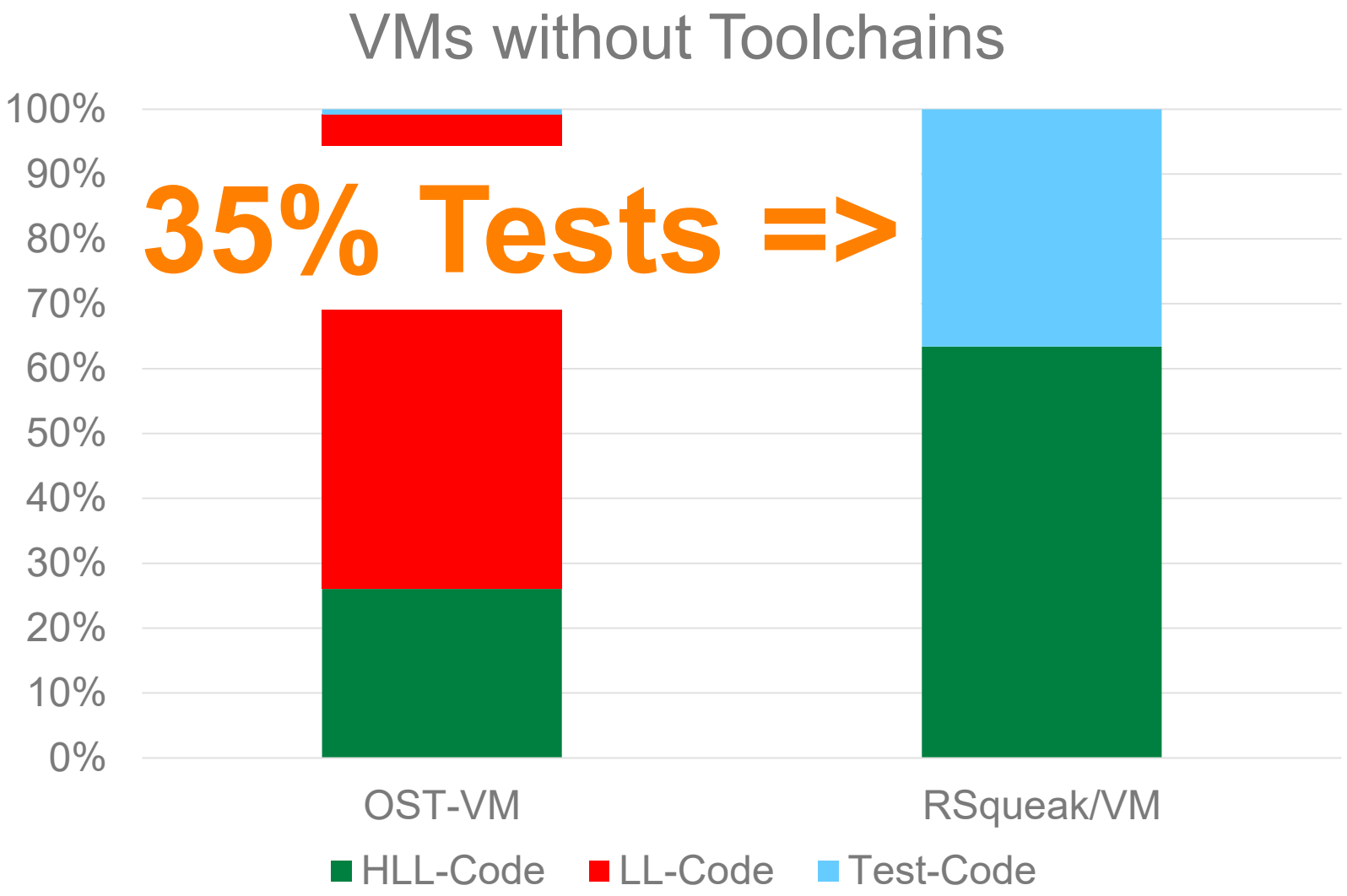
Code of VMs without Translation bits



Code of VMs without Translation bits



Code of VMs without Translation bits



Performance Tests

speed.squeak.org/changes/

SQUEAK VM SPEED CENTER

Home About

Changes | Timeline | Comparison

Environment

fb12ce8ws06

Executable

interpreter

rsqueakvm
- rsqueakvm-linux2
- rsqueakvm64-linux2

cog
- cog-linux2
- cog64-linux2

Options

Trend for last 10 revisions

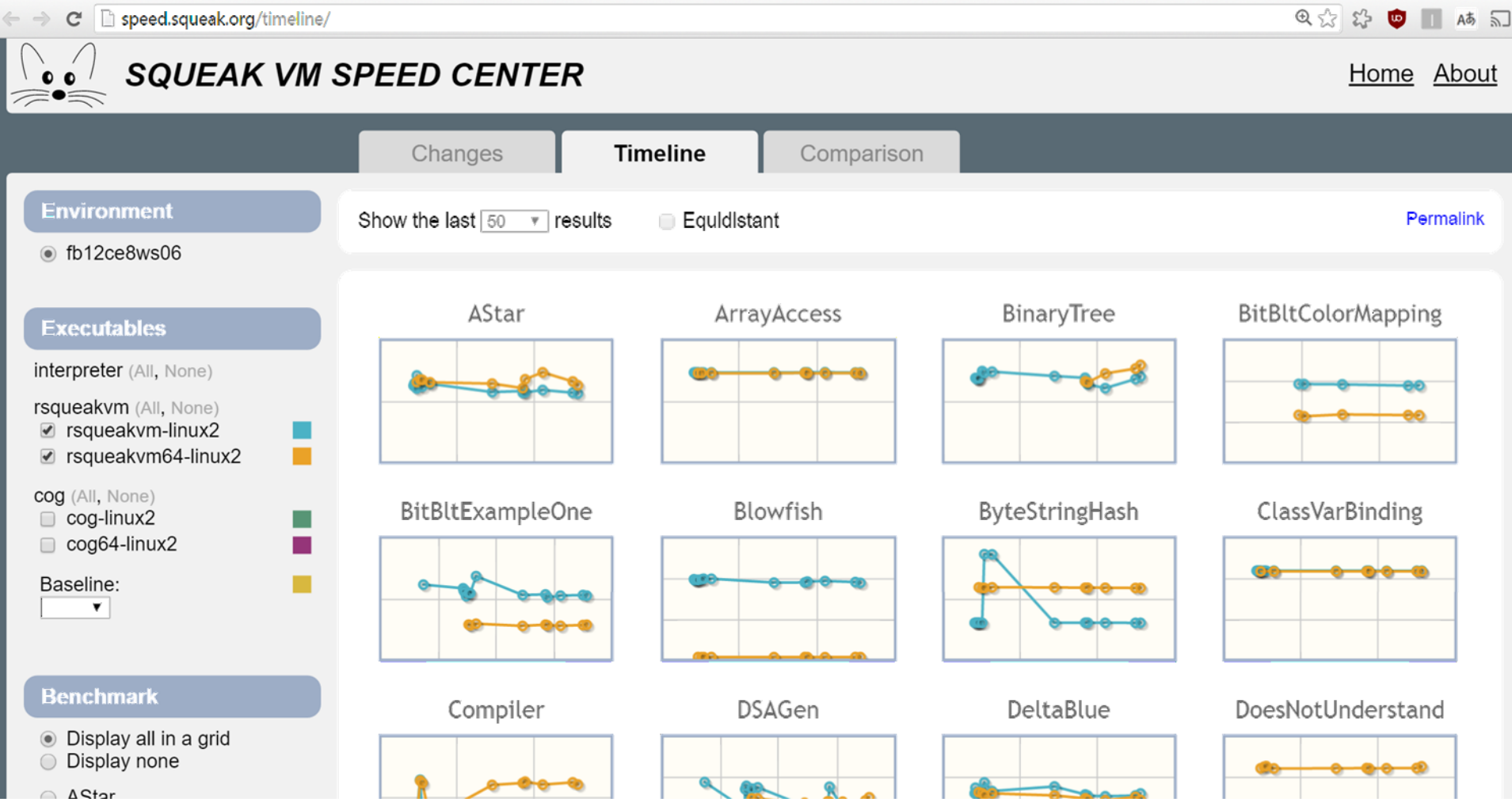
Results for revision Aug 18, 08:04 - 6810238b55 ▾ Permalink

Benchmark	Time in seconds	Std dev	Change	Trend
AStar	111.50	0.42	-1.24	-14.43%
ArrayAccess	727.43	0.72	0.10	-0.02%
BinaryTree	13.90	1.40	2.96	-2.96%
BitBltColorMapping	93552.00	108.00	0.32	-
BitBltExampleOne	105.00	11.00	-0.94	-2.17%
Blowfish	3775.00	189.00	-1.10	-4.46%
ByteStringHash	604.73	0.78	0.07	-38.31%
ClassVarBinding	1093.70	1.10	0.05	0.04%
Compiler	649.10	5.60	-0.41	-24.75%
DSAGen	60898.00	1100.00	-4.04	-12.57%
DeltaBlue	158.10	1.10	3.35	-5.86%
DoesNotUnderstand	620.90	2.00	0.16	-2.02%
Fannkuch	1086.17	0.40	-6.13	-0.59%

Revision

Commit	6810238b55
Date	Aug. 18, 2016, 8:04 a.m.

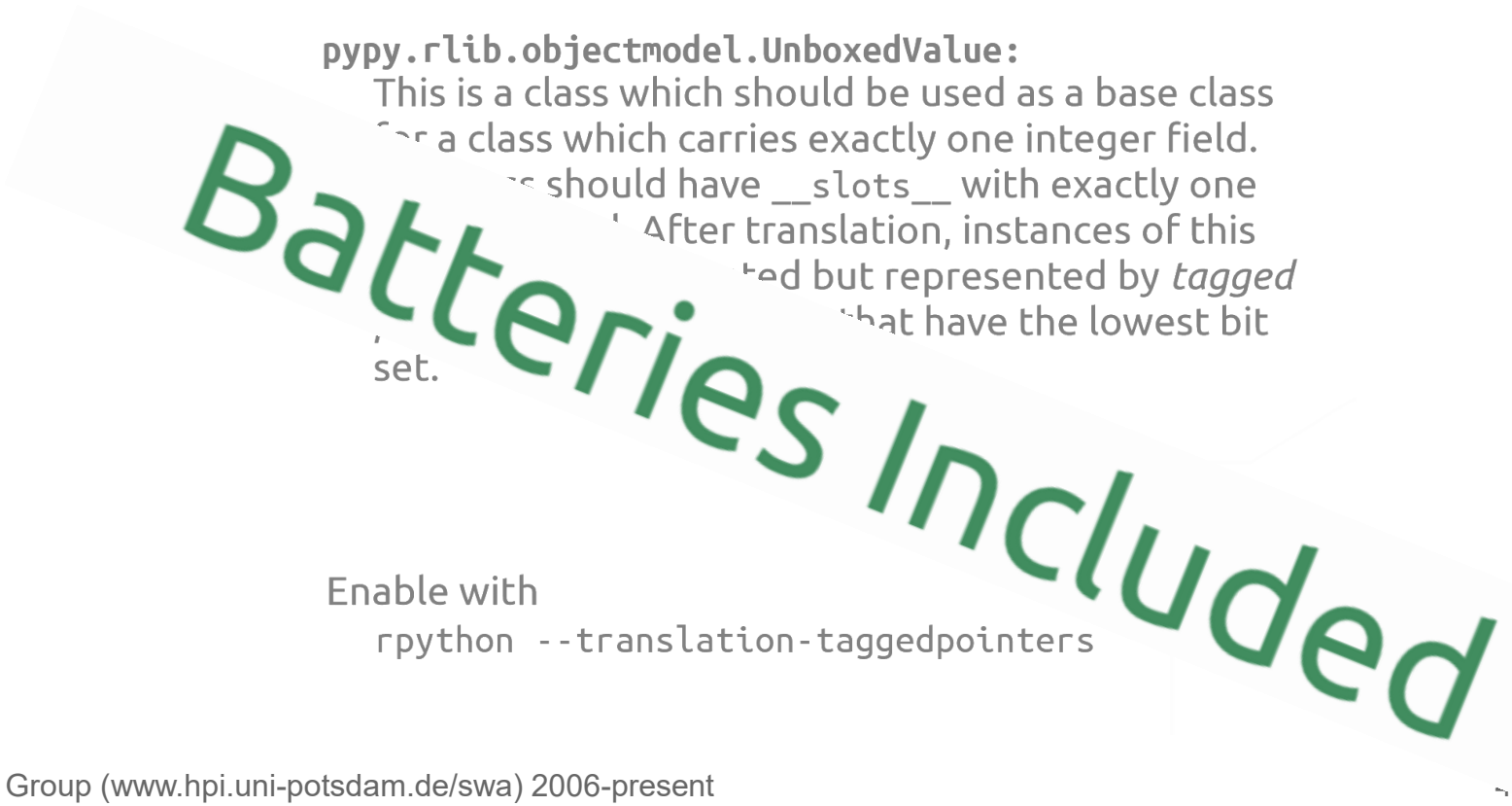
Performance Tests



IMPRESSIONS FROM STUDENTS

Feedback

- Taking advantage of the (R)Python standard library is enjoyable (compared to C-based projects)
- PyPy source documentation very helpful



Feedback

- Writing (R)Python is nice, but...
- RPython toolchain slow (4 min.) and highly complex
- Complexity of RPython toolchain and STM
  - Documentation: PyPy source code
  - Debug segmentation faults in translated code
  - Read and instrument toolchain code (transformations, etc.)

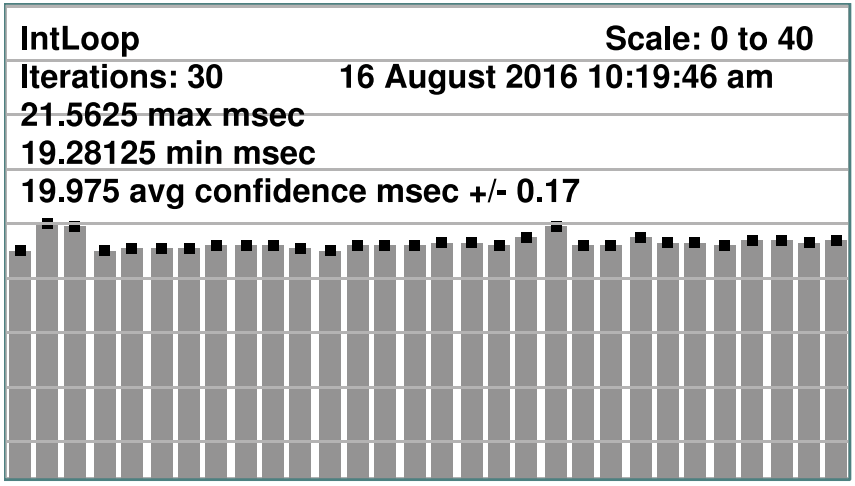
Feedback

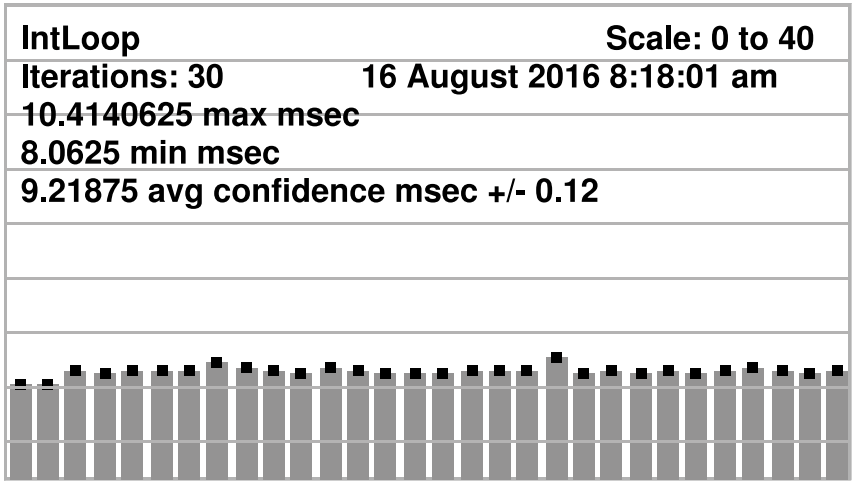
DETAILS ON THE PROJECT SETUP

From a non-technical perspective, a problem we encountered was the **huge roundtrip times** (on our machines up to 600s, 900s with JIT enabled). This led to a **tendency of bigger code changes** ("Before we compile, let's also add this"), **lost flow** ("What where we doing before?") and different compiled interpreters in parallel testing ("How is this version different from the others?") As a consequence it was harder to test and correct errors. While this is not as much of a problem for other RPython VMs, RSqueakVM needs to execute the entire image, which makes **running it untranslated even slower**.

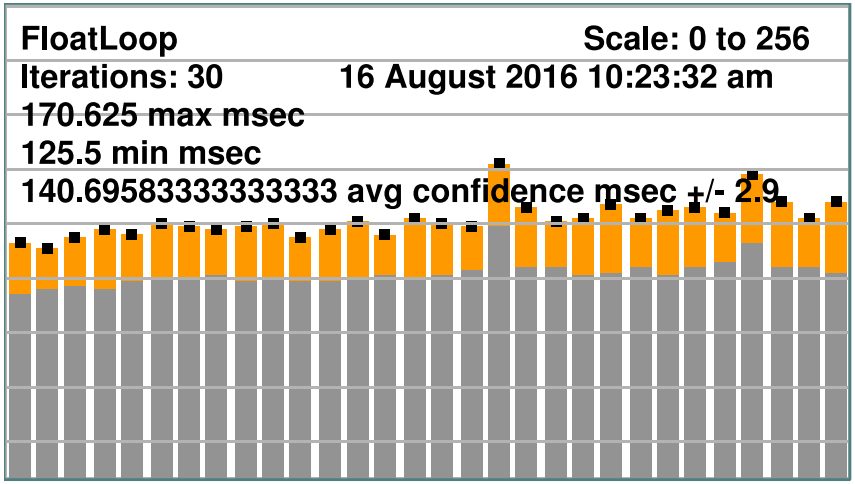
INT TAGGING

ALLOCATION REMOVAL

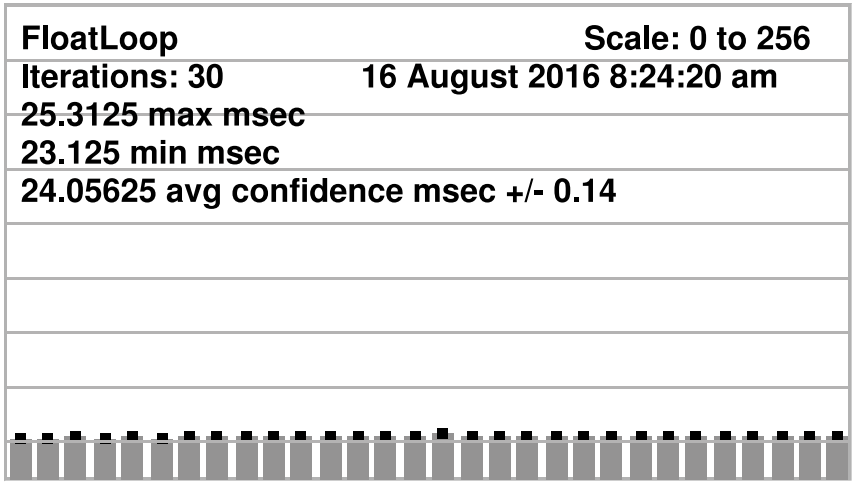




FLOAT WORDS



ALLOCATION REMOVAL

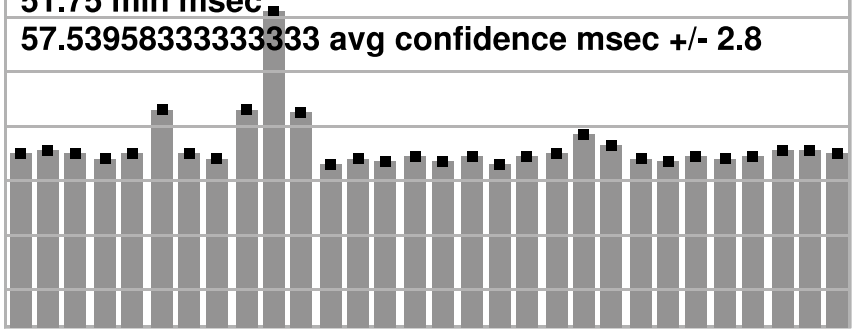


BITBLT PLUGIN

BitBltExampleOne Scale: 0 to 150
Iterations: 30 16 August 2016 10:33:14 am
1.447265625 max msec
1.150390625 min msec
1.2194986979166667 avg confidence msec +/- 0.017

AGGRESSIVE INLINING

BitBltExampleOne Scale: 0 to 150
Iterations: 30 16 August 2016 8:35:43 am
99.9375 max msec
51.75 min msec
57.53958333333333 avg confidence msec +/- 2.8



BitBlt>>benchmark

Rule	Depth	VM	copy	1x1warp	2x2warp	3x3warp
		R	455	763	2703	2831
		R	736	719	2223	2525
		R	624	640	2191	2669
		R	70	95	1650	2518
		R	115	108	1497	1799
		R	18	92	1586	2029

~ Factor <50

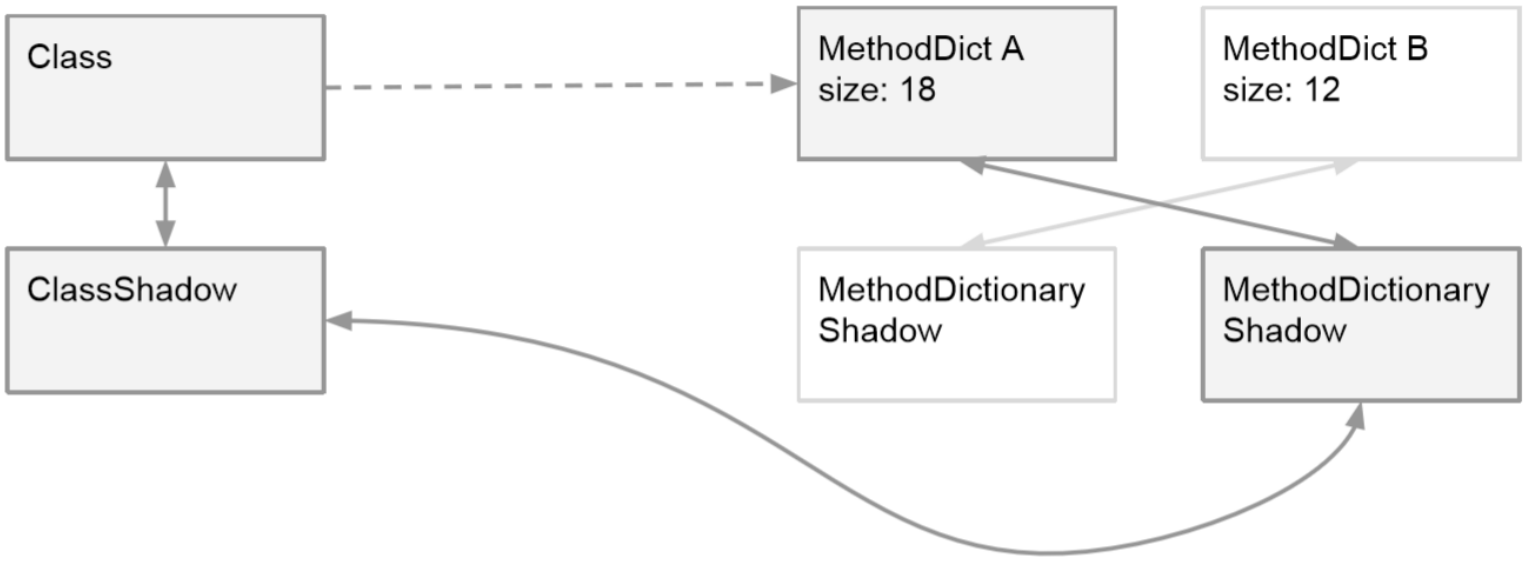
Rule	Depth	VM	copy	1x1warp	2x2warp	3x3warp
			10			59
		R	455	763	2703	2831
			15	19	42	56
		R	736	719	2223	2525
						44
		R	624	640	2191	2669
			4	5		40
		R	70	95	1650	2518
			6	6	27	49
		R	115	108	1497	1799
			1	6		48
		R	18	92	1586	2029

~ Factor <50 Factor 100+

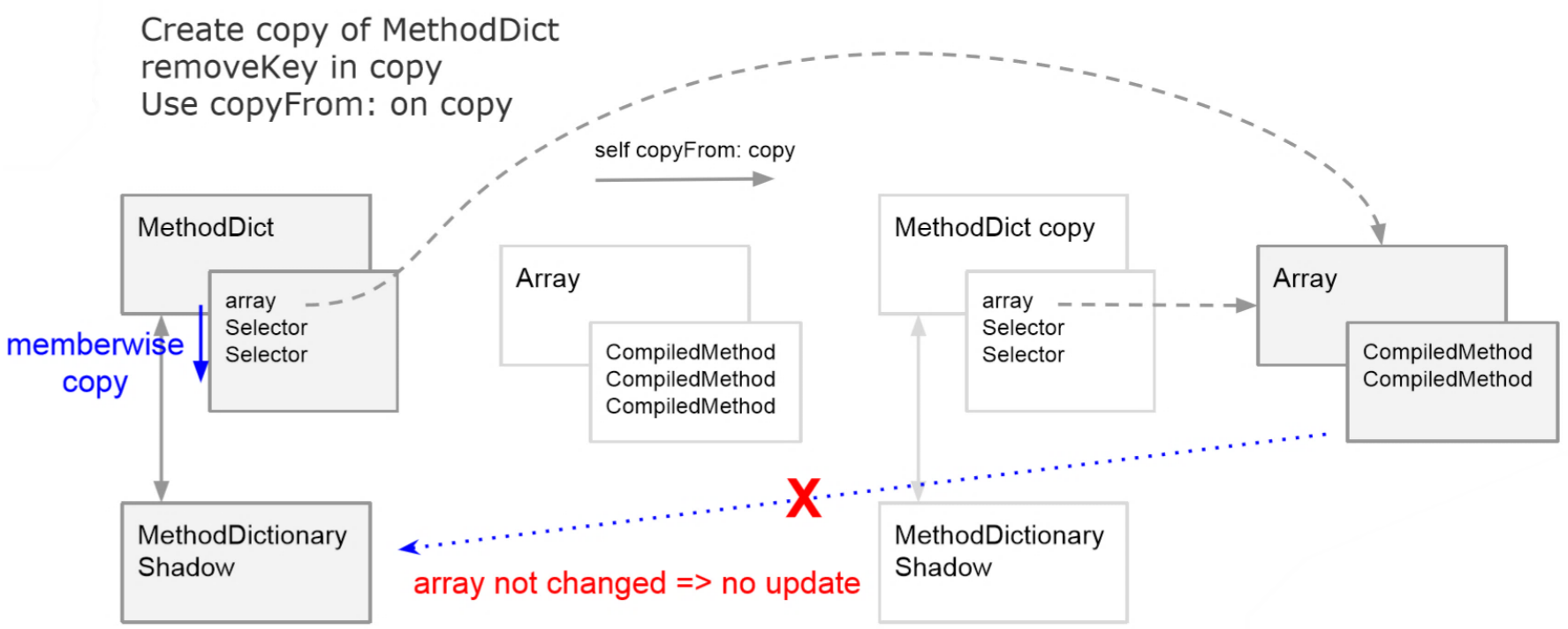
Rule	Depth	VM	copy	1x1warp	2x2warp	3x3warp
			10	**11**	**37**	59
		R	455	**763**	**2703**	2831
			15	19	42	56
		R	736	719	2223	2525
			4	**7**	**26**	44
		R	**624**	**640**	**2191**	2669
			4	5	**23**	40
		R	70	95	**1650**	2518
			6	6	27	49
		R	115	108	1497	1799
			1	6	**22**	48
		R	18	92	**1586**	2029

Growing the method dictionary copies

MethodDictionaries have an interpreter-level Shadow object
In RSqueakVM lookup is done on Shadow
Adding Methods can require MethodDict to grow
Growing of MethodDicts didn't update the Shadow



Removing methods creates copies



Points to Consider

- No interrupts in VM-level primitives
- Performance critical code may still have to be optimized
- GC interaction with user code, but not primitives
- Simulating C semantics impacts results

Array filling

```
| index repOff |
repOff := repStart - start.
index := start - 1.
[(index := index + 1) <= stop] whileTrue: [
    self at: index
         put: (replacement at: repOff + index)]
```

Array filling

```
| index repOff |
repOff := repStart - start.
index := start - 1.
[(index := index + 1) <= stop] whileTrue: [
    self at: index
         put: (replacement at: repOff + index)]
```

Needs bounds checks

– interrupting threads may modify the replacement array

Mandala

```
self isDefined: 'ENABLE_FAST_BLT'
      inSmalltalk: [false
          "there is no current fast path
           specialisation code in-image"]
      ifTrue:[self copyBitsFastPathSpecialised]
      ifFalse: [self copyBitsLockedAndClipped].
```

No Slang code for this – just plain C. Optimization on Smalltalk level required.

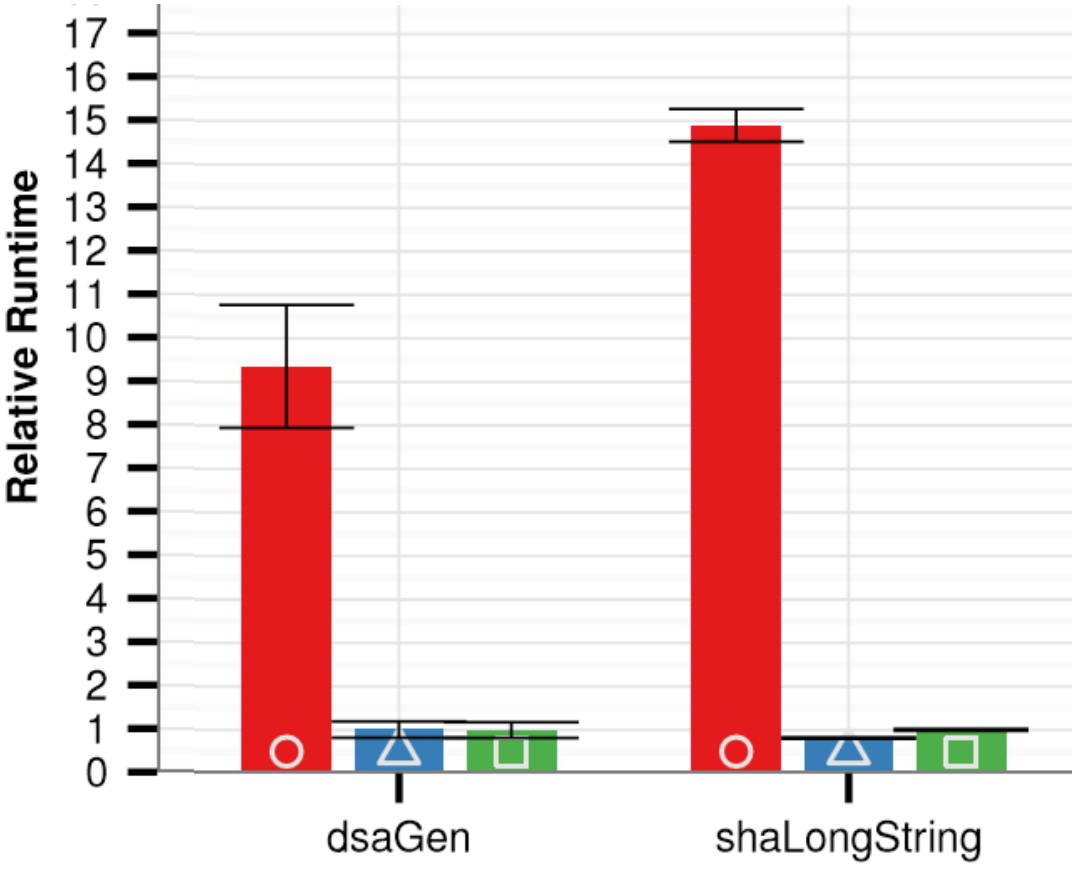
Secure Hash Algorithm

```
self primHasSecureHashPrimitive
      ifTrue: [
        ^ self
          processBufferUsingPrimitives:
             aByteArray]
      ifFalse: [totals := nil].
      "... 26 lines of code using instances of
       ThirtyTwoBitRegister"
```

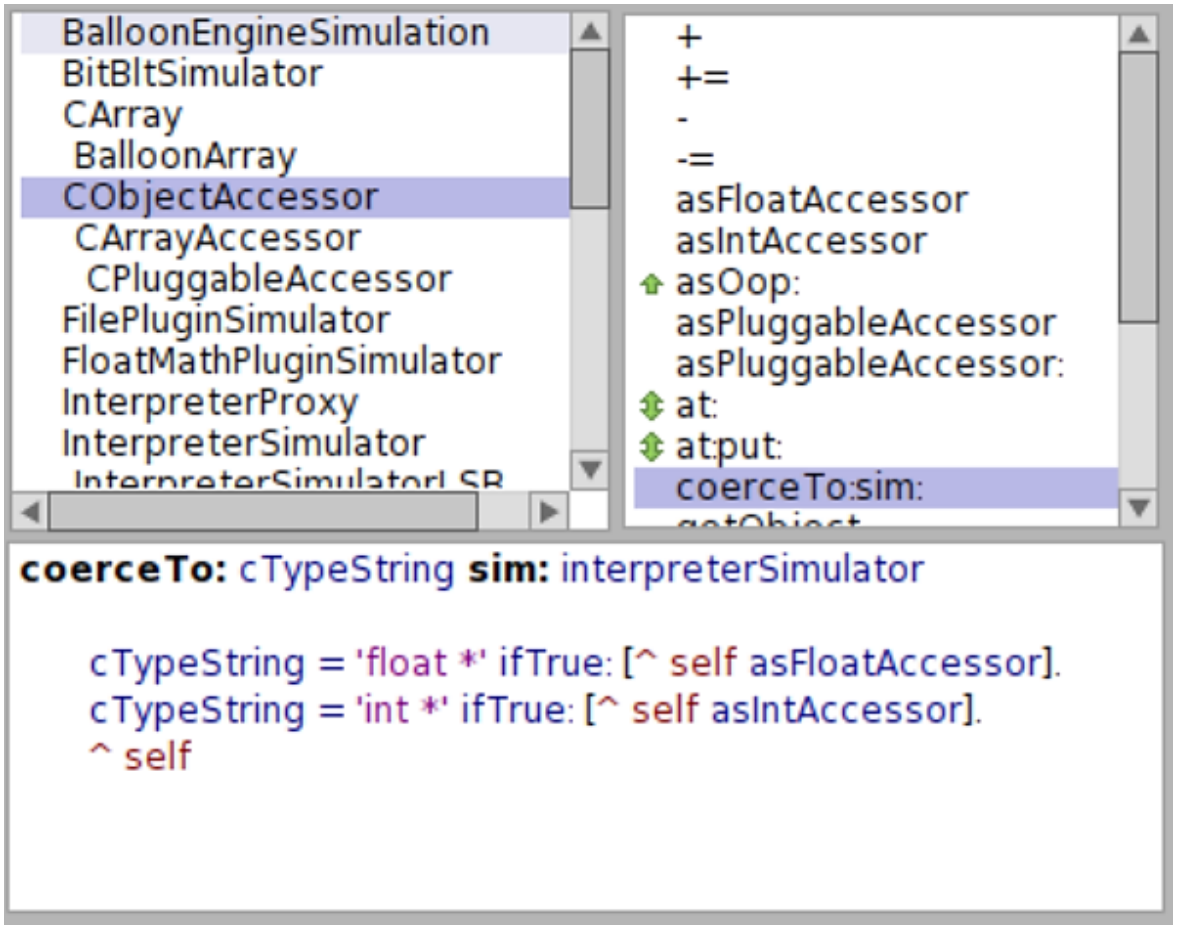
Secure Hash Algorithm

```
self primHasSecureHashPrimitive
      ifTrue: [
        ^ self
          processBufferUsingPrimitives:
             aByteArray]
      ifFalse: [totals := nil].
      "... 26 lines of code using instances of
      ThirtyTwoBitRegister"
```

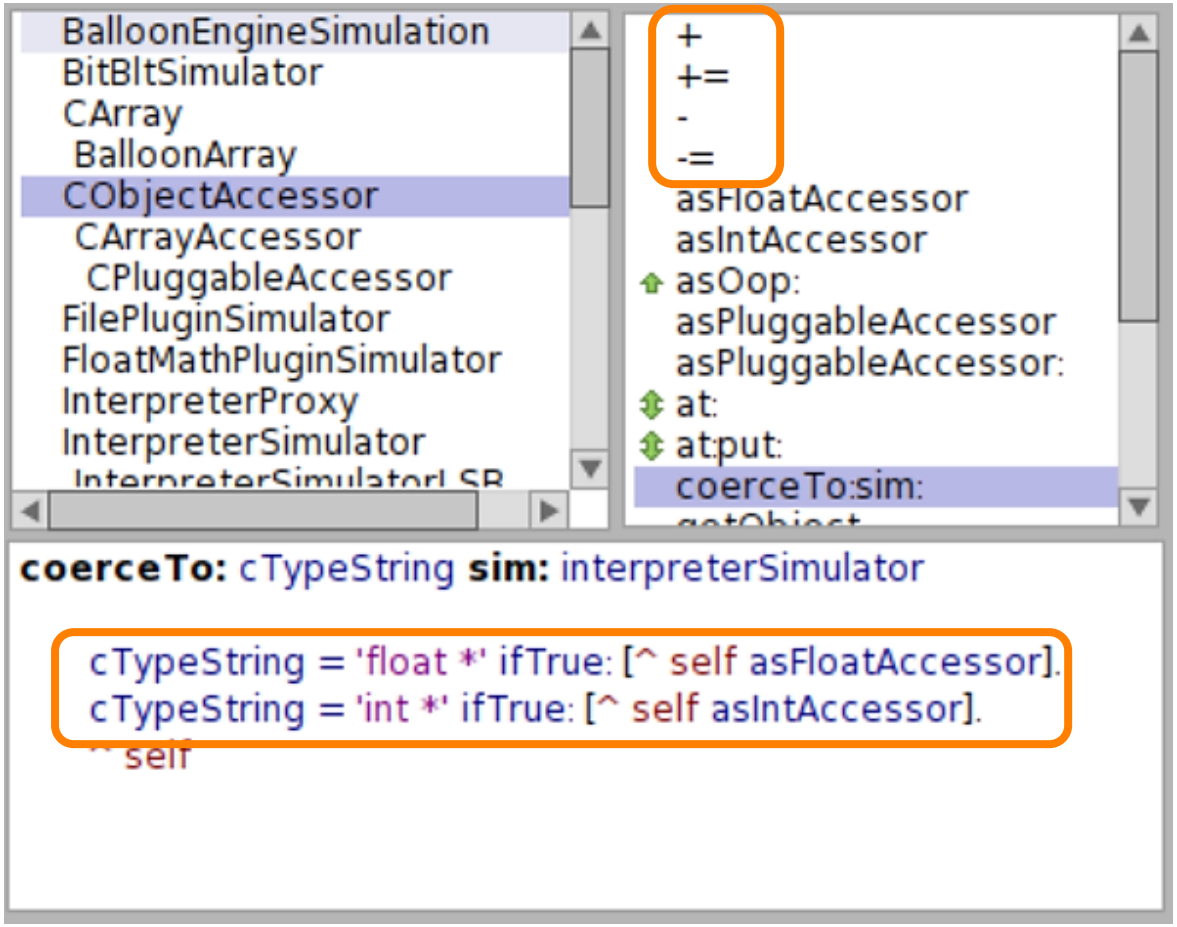
OO-Abstraction of words (with high and low parts stored separately). Instances escape loops and cause GCs


Relative Runtime
17
16
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
0
dsaGen
shaLongString

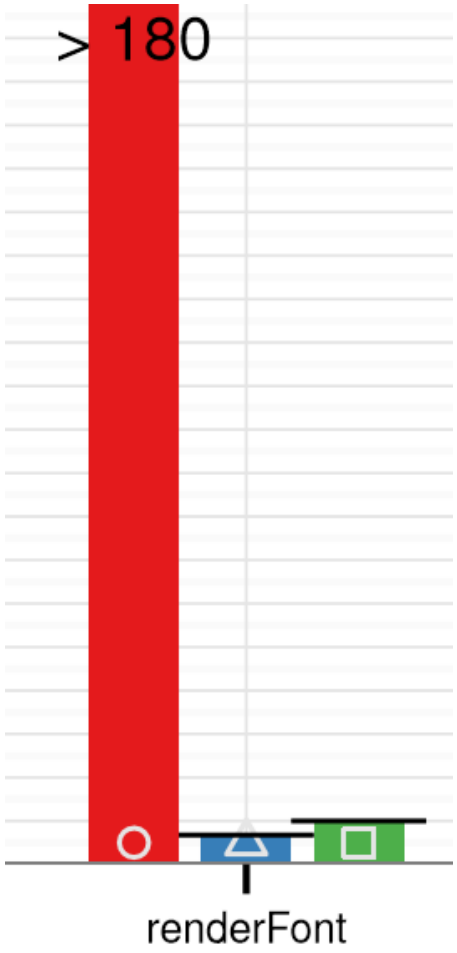
Rendering Fonts



Rendering Fonts

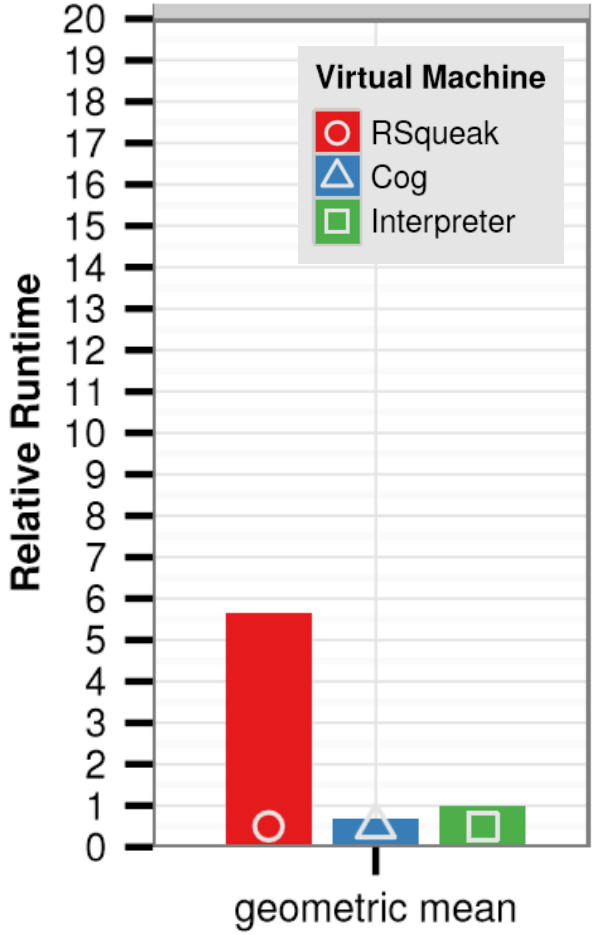


Simulating C pointer arithmetic, casts, …


> 180
renderFont

Our Takeaway

- Useable Performance reduces the need for primitives
- Debugging in practical applications feasible
- Writing new primitives in Smalltalk from the start is an option



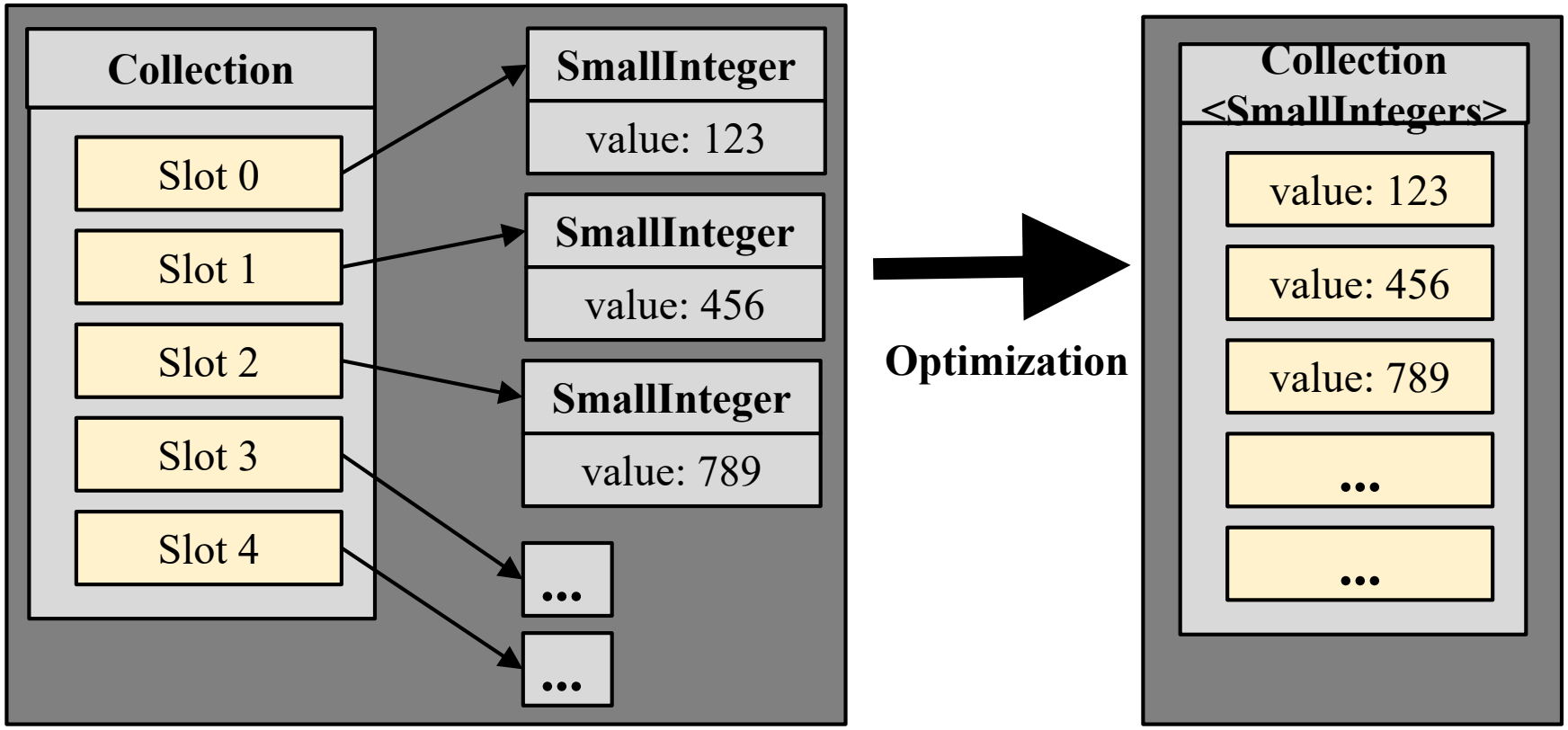
```
(#('VMMaker-Translation to C' 'VMMaker-Building' 'VMMaker-
InterpreterSimulation'
        'VMMaker-JITSimulation' 'VMMaker-SpurMemoryManagerSimulation' 'VMMaker-
PostProcessing')
                gather: [:cat | (Smalltalk organization listAtCategoryNamed:
cat) collect: [:s | (Smalltalk at: s) linesOfCode]]) sum
22171

(#('VMMaker-Interpreter' 'VMMaker-JIT' 'VMMaker-Multithreading' 'VMMaker-
Plugins' 'VMMaker-Plugins-FFI' 'VMMaker-SmartSyntaxPlugins'
     'VMMaker-SpurMemoryManager' 'VMMaker-Support')
                gather: [:cat | (Smalltalk organization listAtCategoryNamed:
cat) collect: [:s | (Smalltalk at: s) linesOfCode]]) sum
118738

sloccount platforms/
ansic:          263736
cpp:            46791
objc:           13036
asm:            9753

------------------------

(#('VMMaker-Tests') gather: [:cat | (Smalltalk organization
listAtCategoryNamed: cat) collect: [:s | (Smalltalk at: s) linesOfCode]]) sum
3837
```

```
find rpython/ -not -path "*test*" -not -path "*_cache*" -not -name "*.pyc" | tr '\n' ' ' |
xargs sloccount
python:            345405
ansic:             8090
asm:               213

sloccount rsdl/
python:            822

find rsqueakvm/ -type f -not -path "*test*" -not -path "*_cache*" -not -name "*.pyc" | tr
'\n' ' ' | xargs sloccount
python:            11823

-------------------------

find rpython -type f -path "*test*" -not -path "*_cache*" -not -name "*.pyc" | tr '\n' ' '
| xargs sloccount
python:            134942
asm:               4956
ansic:             574

sloccount rsdl/test
python:            513

find rsqueakvm/ -type f -path "*test*" -not -path "*_cache*" -not -name "*.pyc" | tr '\n' '
' | xargs sloccount
python:            6786
```

Storage Strategies

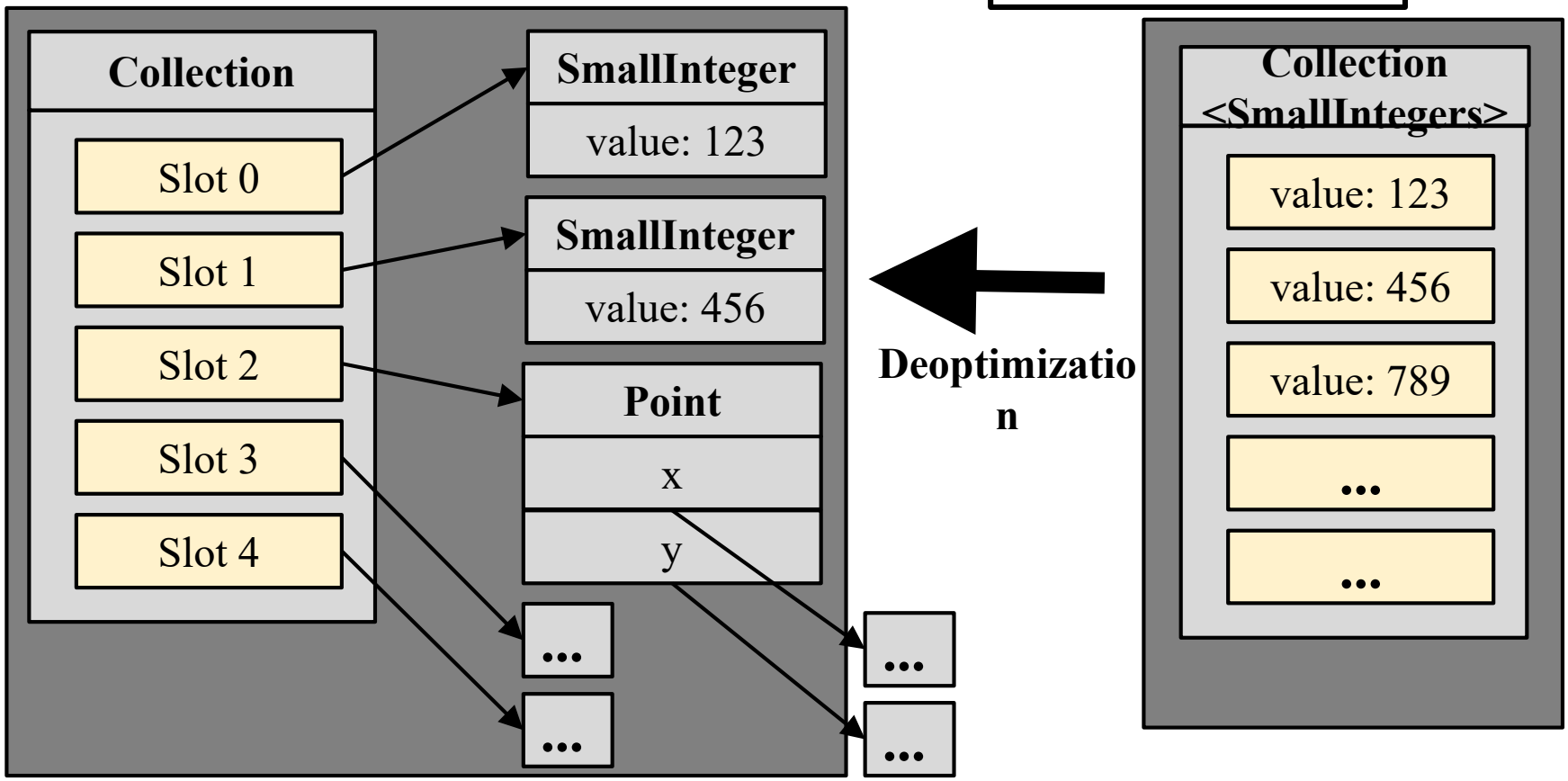
- A kind of Allocation Removal Optimization
  - Less overhead due to memory management
  - Less pressure on garbage collector (less GCs, shorter GCs)
  - Smaller memory footprint of application
- Based on heuristics/speculations
  - Slow deoptimizations possible

Storage Strategies

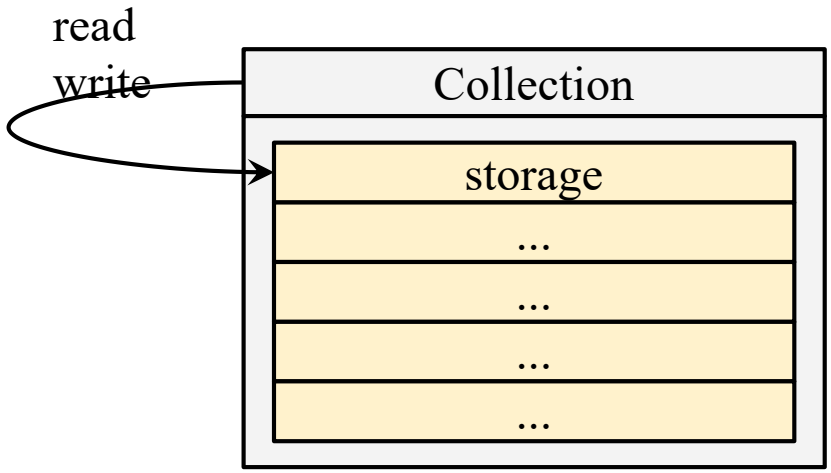


Storage Strategies

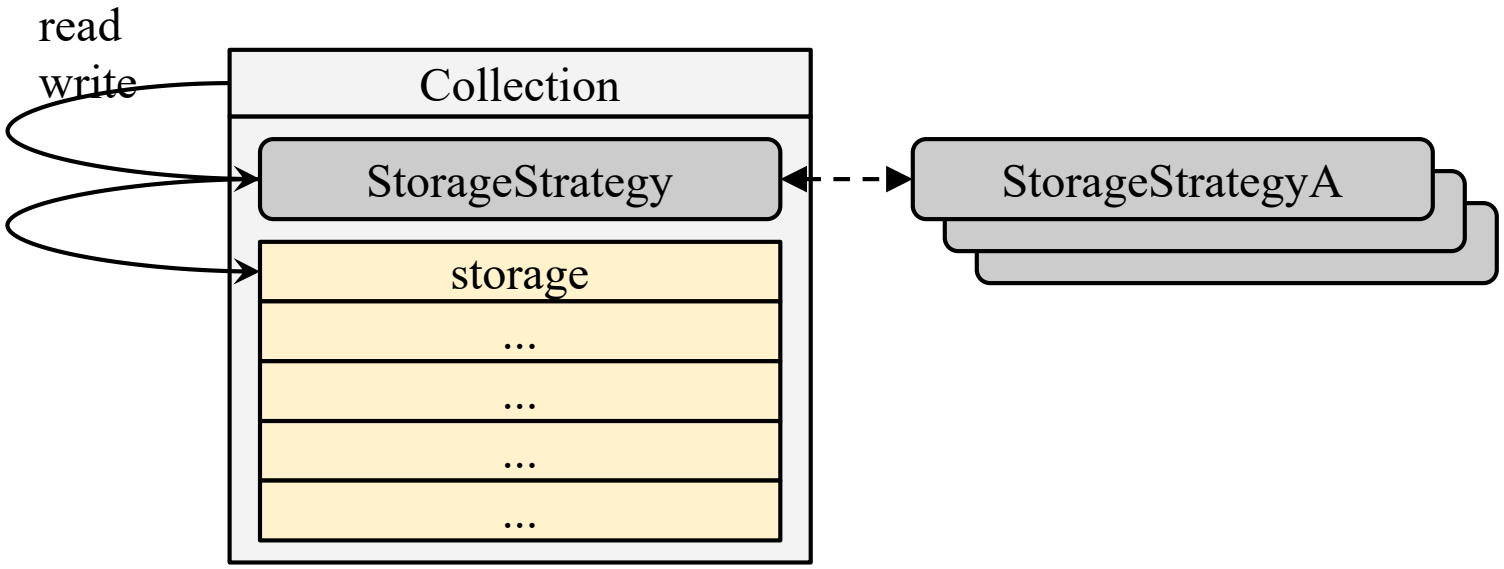
```
col at: 3 put: 1@2
```



Storage Strategies



Storage Strategies



Increment size of OrderedCollection

```
i215 = int_add_ovf(i213, 1)
p227 = new_with_vtable(ConstClass(W_SmallInteger))
setfield_gc(p227, i215, W_SmallInteger.inst_value)
setarrayitem_gc(p147, 2, p227)
```

Without Strategies

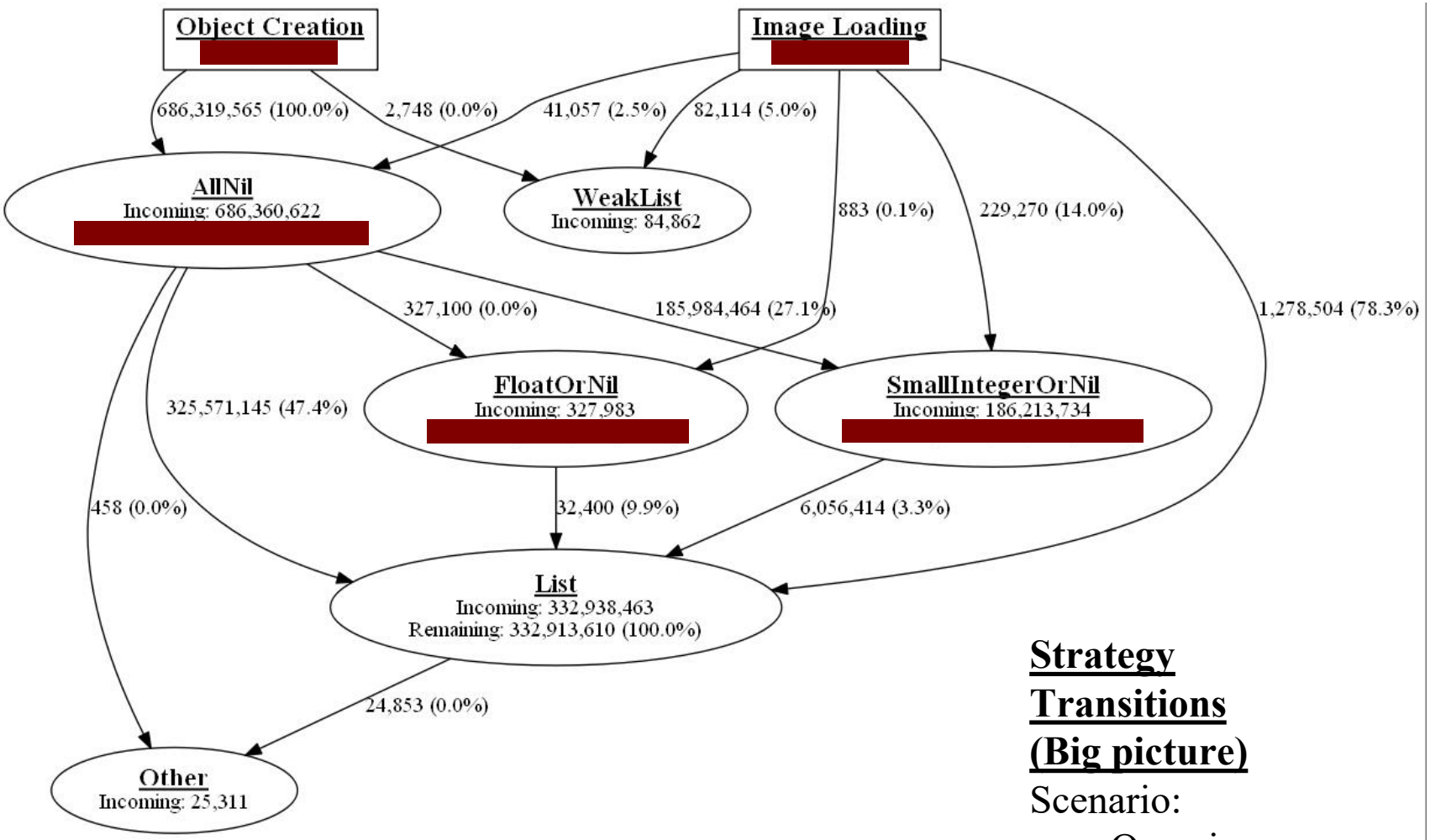
Store new value in Array

```
i207 = int_add_ovf(i194, 2)
p231 =
new_with_vtable(ConstClass(W_SmallIntege
r))
setfield_gc(p231, i207,
W_SmallInteger.inst_value)
setarrayitem_gc(p220, i222, p231)
```

With Strategies

Store new value in Array

```
i204 = int_add_ovf(i189,
2)
i219 = int_ne(i204,
2147483647)
guard_true(i219)
setarrayitem_gc(p211,
i213, i204)
```



Strategy Transitions (Big picture)

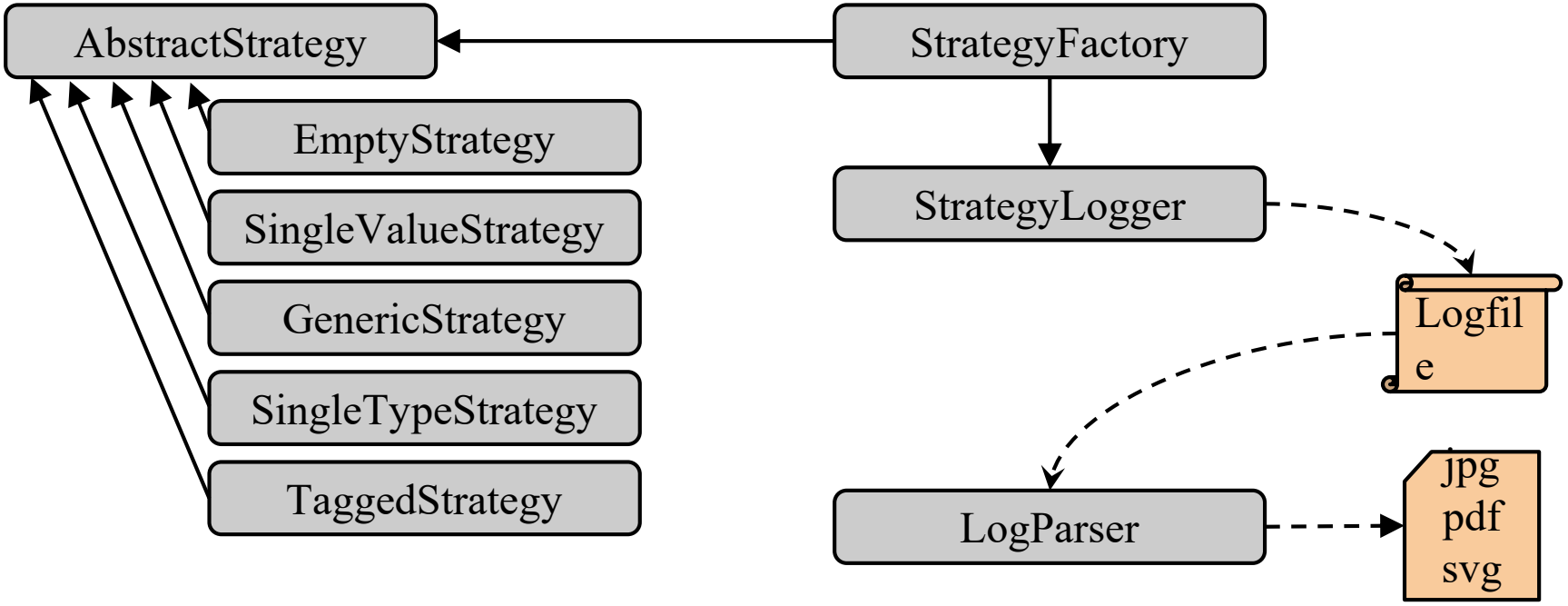
Scenario:

- Open image
- Use browser
- Close image

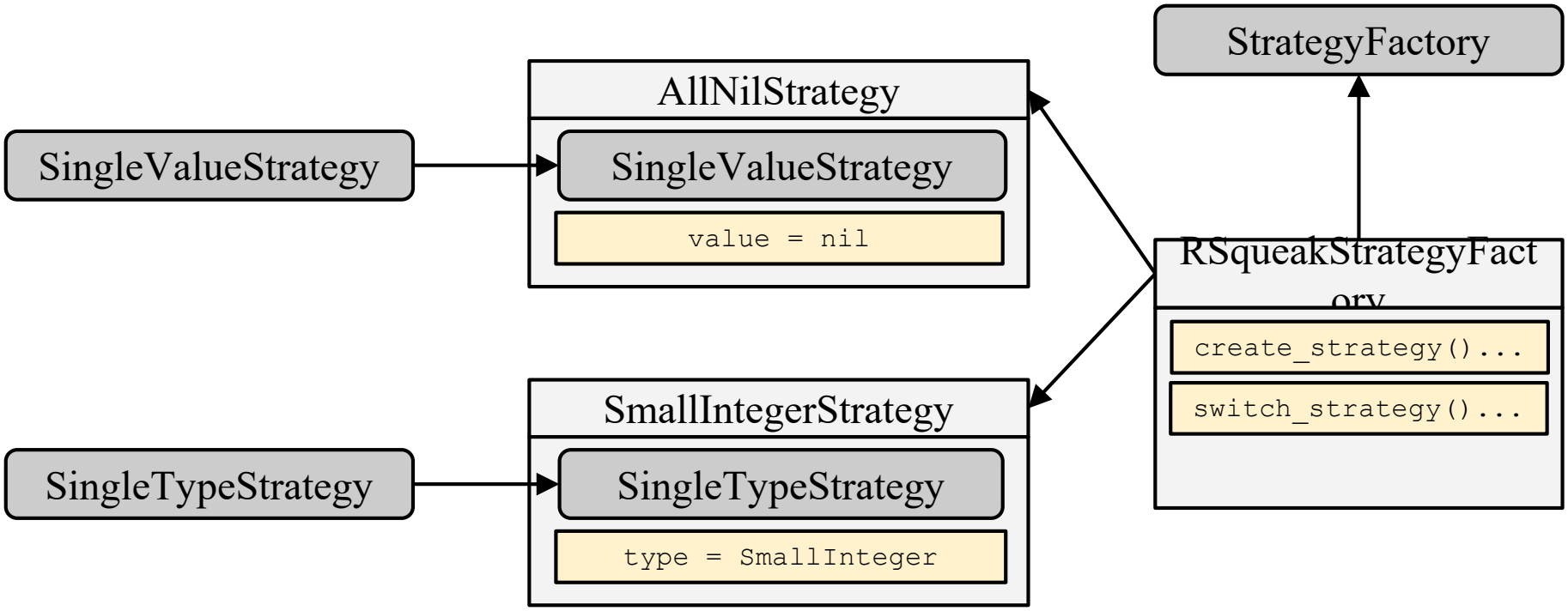
Evaluation: Performance

Benchmark	Without Strategies	With Strategies	Performance Change
AStar1	80.64 ms ±1.92	55.81 ms ±2.61	±6.58
AStar2	351.17 ms ±8.15	288.81 ms ±2.68	±2.25
BinaryTree	187.47 ms ±1.00	174.94 ms ±1.77	+7.26 % ±1.01
BlowfishDecryption	422.16 ms ±2.05	429.16 ms ±2.49	±0.64
BlowfishEncryption	423.85 ms ±2.00	427.15 ms ±2.47	±0.63
DeltaBlue	99.86 ms ±2.24	98.31 ms ±2.16	+1.57 % ±2.62
NBody	271.38 ms ±2.15	274.09 ms ±2.16	±0.94
Richards	165.97 ms ±2.80	162.95 ms ±4.14	+2.11 % ±2.29
SplayTree	781.16 ms ±2.25	469.92 ms ±2.89	±0.97

rstrategies: Architecture



rstrategies: Usage



Example: RSqueak VM

```
@rstrat.strategy(generalize=[
    SmallIntegerOrNilStrategy,
    FloatOrNilStrategy,
    ListStrategy])
class AllNilStrategy(AbstractStrategy):
    repr_classname = "AllNilStrategy"
    import_from_mixin(rstrat.SingleValueStrategy)
    def value(self): return self.space.w_nil
```