Dynamic Initialization of Collections

Jean Privat - for RMod - 2022-09-22

Jean Privat

- Professor at Université du Québec à Montréal (UQAM) since 2007
  Ph.D. at LIRMM (Montpellier, France) 2006
- Work on OO languages and compilers
  Like other things: VM, OS, free software, cybersecurity…
- Little practical knowledge about Pharo or Smalltalk.
  ➔ Here to learn
  ➔ And observe what you are doing and how you are doing it

And, possibly, try to make myself useful while having fun.

Exprim Instances of Collections with Items Inside

Literal Collections (AST level)

```
#(1 2 3)  #[1 2 3]  ‘abc’ #abc
```

- Fast 👍
- Immutable (read-only) 👍 or 👎 (it depends)
- Literal elements only 👎

  `#(1/2)` or `#(1@2)` can be misunderstood
- Few selected classes only 👎

  Array ByteArray String Symbol

Do It Yourself (basic programmative level)

```
(Array new: 3) at:1 put:10; at: 2 put: 20; at: 3 put:30; yourself

(OrderedCollection new: 3) add: 10; add: 20; add: 30; yourself

(Set new: 3) add: 10; add: 20; add: 30; yourself

(Dictionary new: 3) at: 1 put: 10; at:2 put: 20; at: 3 put: 30; yourself
```

- 2 basic schemes

  `add:`

  `at:put:`
- Use *`yourself`* (not beginner-friendly) 👎
- Very verbose. 👎

  Painful to read and to write

Dynamic Array to the Rescue

```
{1. 2. 3}
```

- Accept any sequences of expressions 👍

```
{1@2. 1/2. Random new. self doSomething. thisContext}
```

- Not in Smalltalk80 (who proposed it first?)
- Only for Array 👎
  - Not other collections
  - This is unfair

DIY With the Help of Dynamic Arrays

```
(Array new: 3) at:1 put:10; at: 2 put: 20; at: 3 put:30; yourself
(OrderedCollection new: 3) add: 10; add: 20; add: 30; yourself
(Set new: 3) add: 10; add: 20; add: 30; yourself
(Dictionary new: 3) at: 1 put: 10; at:2 put: 20; at: 3 put: 30; yourself
```

Can **equally** become

```
{10. 20. 30}
{10. 20. 30} asOrderedCollection
{10. 20. 30} asSet
{1->10. 2->20. 3->30} asDictionary
```

But one seems **more equal** than the others (hint, it is Array)

This is Unfair (and Outrageous)

Can we extend the dynamic {} syntax to **other** collections

Important: this it **not** a proposal about **performance**

We are discussing **language specification**

A modest proposal...

Syntax?

Prefix (or suffix) the syntactic construction with the name of the class?

- {:Set 1. 2. 3}
- {Set: 1. 2. 3}
- {Set| 1. 2. 3}
- {1. 2. 3}:Set
- Other ideas?

Follow-up questions: accept user-defined classes? Expressions?

- {:ColorArray Color blue. Color white. Color red}
- {:(self species) 1. 2. 3}

Semantic?

The following constructions should be equivalent

```
(Set new: 3) add: 10; add: 20; add: 30; yourself
Set withAll: {10. 20. 30}
{10. 20. 30} asSet
{:Set 10. 20. 30}
```

Could the proposal (last one) just be some syntactic sugar of the first form?

Could the bytecode compiler (Opal) do it transparently?

1st issue how to distinguish `add:` vs `at:put:` ?

Explicit list of known classes? (bad)
Ask the class at compile time? (the class should be known at compile time).
Something else?

```
{:Set 1. 2. 3}
```

```
{1. 2. 3} asSet
```

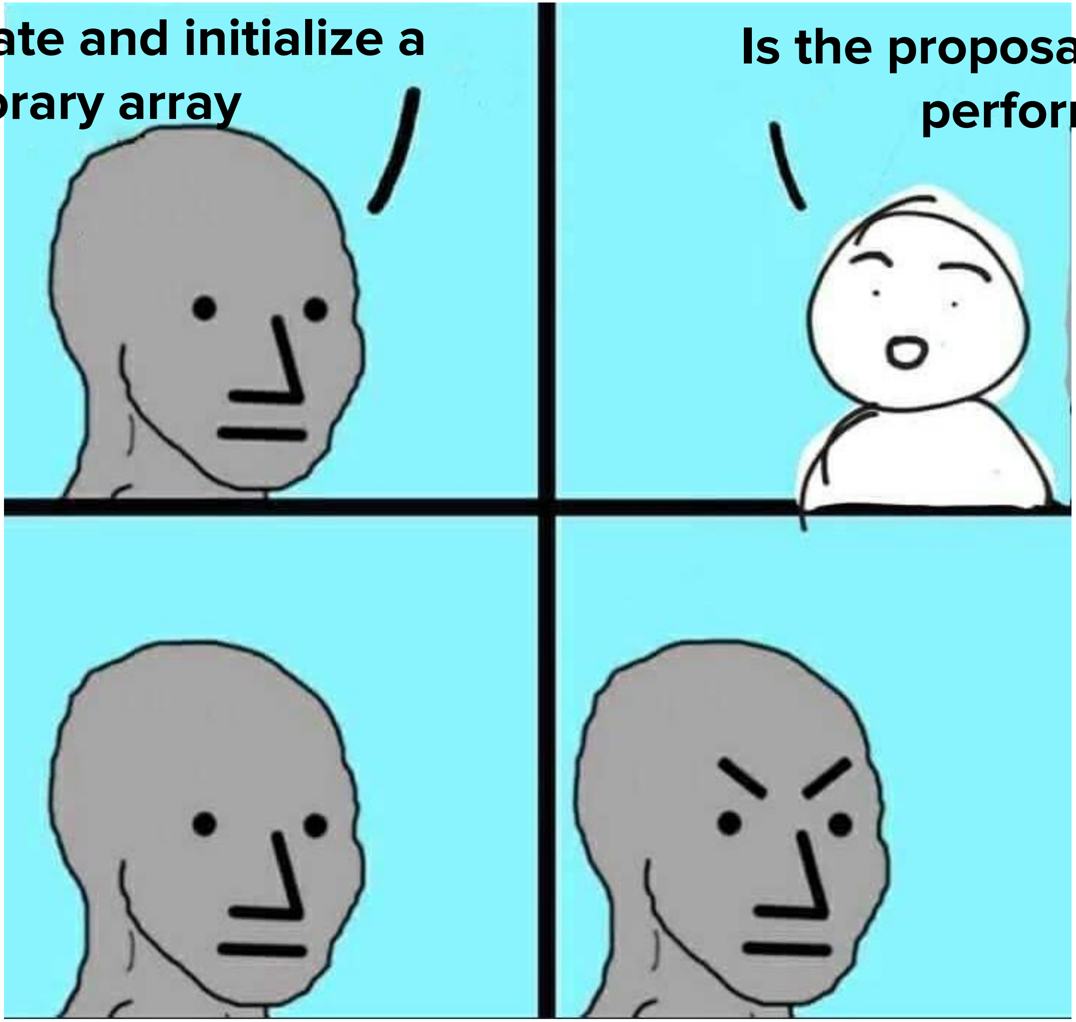
Pros and Cons of asSet

Pros 👍

- Short. Basically only the items and a class information
- A non-magic message send
- I can debug it
- Redefine it
- Inspect senders
- Etc.

Cons 👎

- ?


But, you allocate and initialize a useless temporary array
Is the proposal about performance?

Let’s talk about performance.

What is the speed of current code?

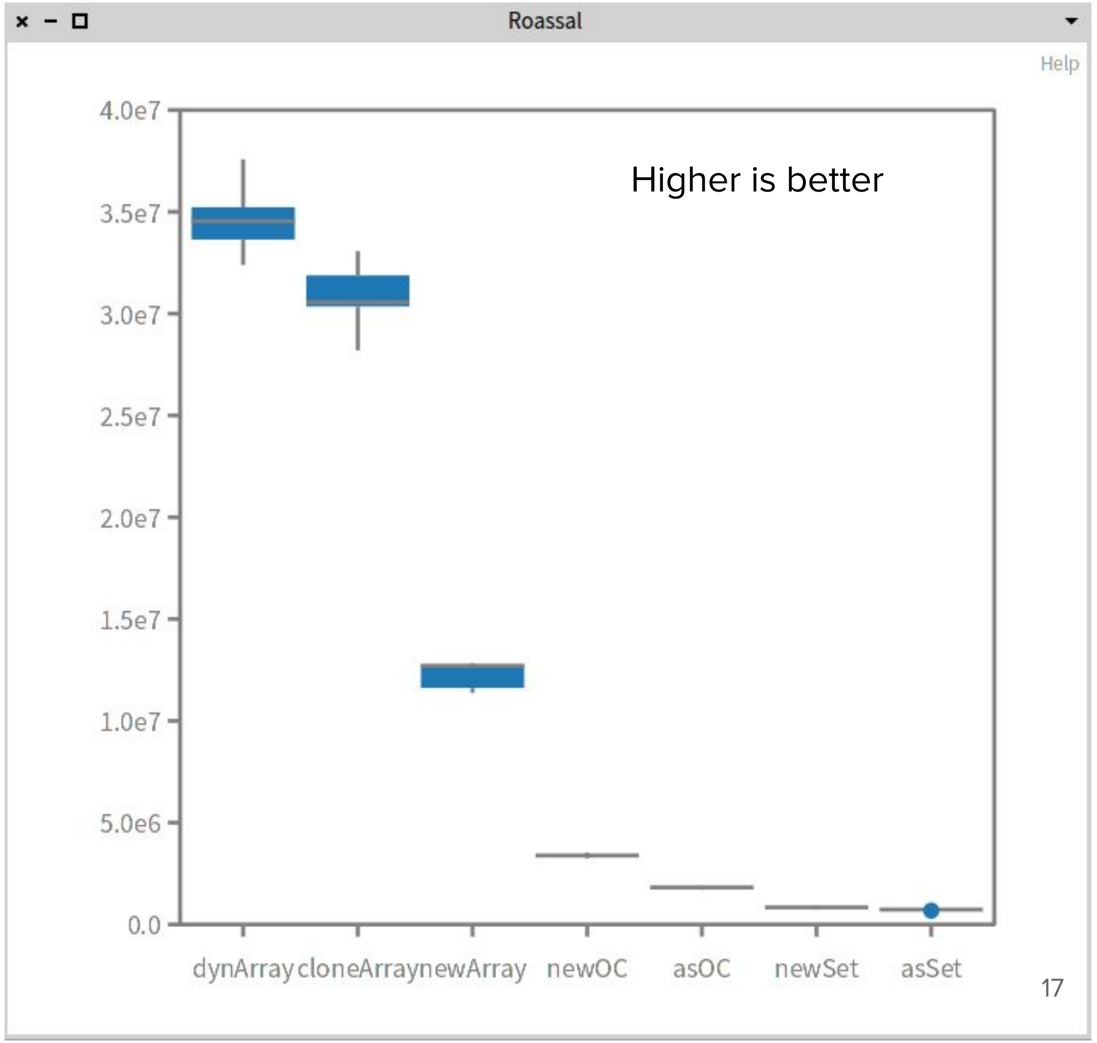
- dynArray: `{1. 2. 3. 4. 5. 6. 7. 8. 9. 10}.`
- cloneArray: `#(1 2 3 4 5 6 7 8 9 10) clone.`
- newArray: `(Array new: 10) at:1put:1; at:2put:2; at:3put:3; at:4put:4; at:5put:5; at:6put:6; at:7put:7; at:8put:8; at:9put:9; at:10put:10; yourself.`
- newOC: `(OrderedCollection new: 10) add:1; add:2; add:3; add:4; add:5; add:6; add:7; add:8; add:9; add:10; yourself.`
- asOC: `{1. 2. 3. 4. 5. 6. 7. 8. 9. 10} asOrderedCollection.`
- newSet: `(Set new: 10) add:1; add:2; add:3; add:4; add:5; add:6; add:7; add:8; add:9; add:10; yourself.`
- asSet: `{1. 2. 3. 4. 5. 6. 7. 8. 9. 10} asSet.`

Old noisy laptop. Debian testing. x86_64. Pharo11. PharoVM9.
5 executions of 5 seconds each, using `BlockClosure>>benchFor:`

Numbers!

Dynamic arrays are insanely **fast**!
➜ x3 faster than manual at:put:
➜ Even faster than clone!
How is that possible?
➜ Special byte code instruction to pop all elements and push an allocated and filled array

asX cause an overhead
➜ 50% overhead for OC
➜ 20% overhead of Set
Can we improve?



Optimize all the things!!!

Current Code for asSet

```
Collection>>asSet
    ^Set withAll: self

Set>>asSet
    ^self

Collection class>>withAll: aCollection
    ^(self new: aCollection size) addAll: aCollection; yourself
```

This Is Very Elegant!

Improving asSet with double dispatch

```
Array>>asSet
    ^Set newFromArray: self

Collection class>>newFromArray: anArray
    | newCollection size |
    size := anArray size.
    newCollection := self new: size.
    1 to: size do: [:i| newCollection add: (anArray at: i)].
    ^newCollection
```

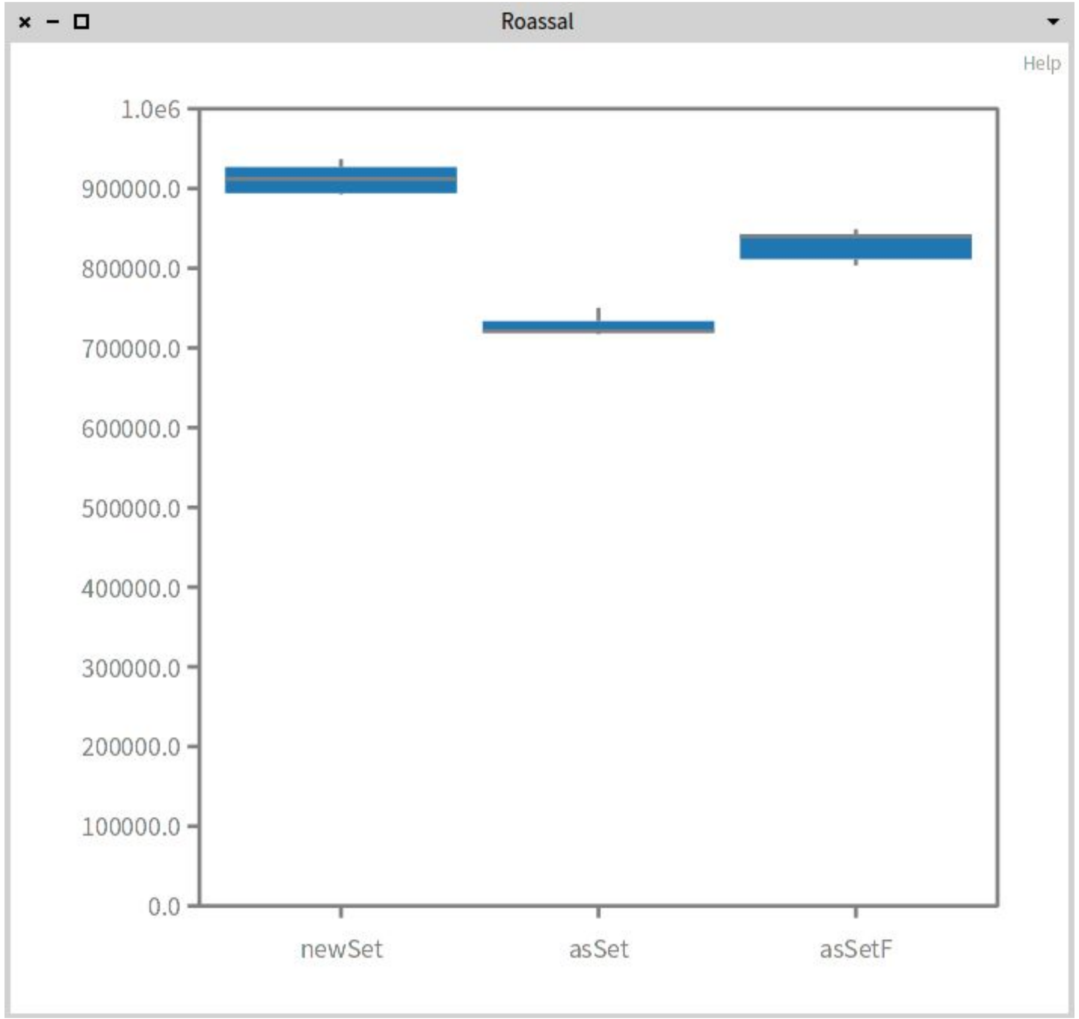
Numbers

newSet: 912k/s (base)

asSet (old): 728k/s (-21%)

asSet (fast): 829k/s (-10%)

Not that bad!



Current code for asOrderedColletion

```
Collection>>asOrderedCollection
    ^ self as: OrderedCollection

OrderedCollection>>asOrderedCollection
    self species == OrderedCollection ifTrue: [ ^self ].
    ^super asOrderedCollection

Object>>as: aSimilarClass
    aSimilarClass == self class ifTrue: [ ^self ].
    ^aSimilarClass newFrom: self

OrderedCollection class>>newFrom: aCollection
    | newCollection |
    newCollection := self new: aCollection size.
    newCollection addAll: aCollection.
    ^newCollection
```

Improving asOrderedCollection by hijacking

```
Array>>asOrderedCollection
    ^ OrderedCollection newFromArray: self

OrderedCollection class>>newFromArray: anArray
    ^ self basicNew setContents: anArray clone
```

setContents: (private) already exists.

It uses the given array as internal storage.

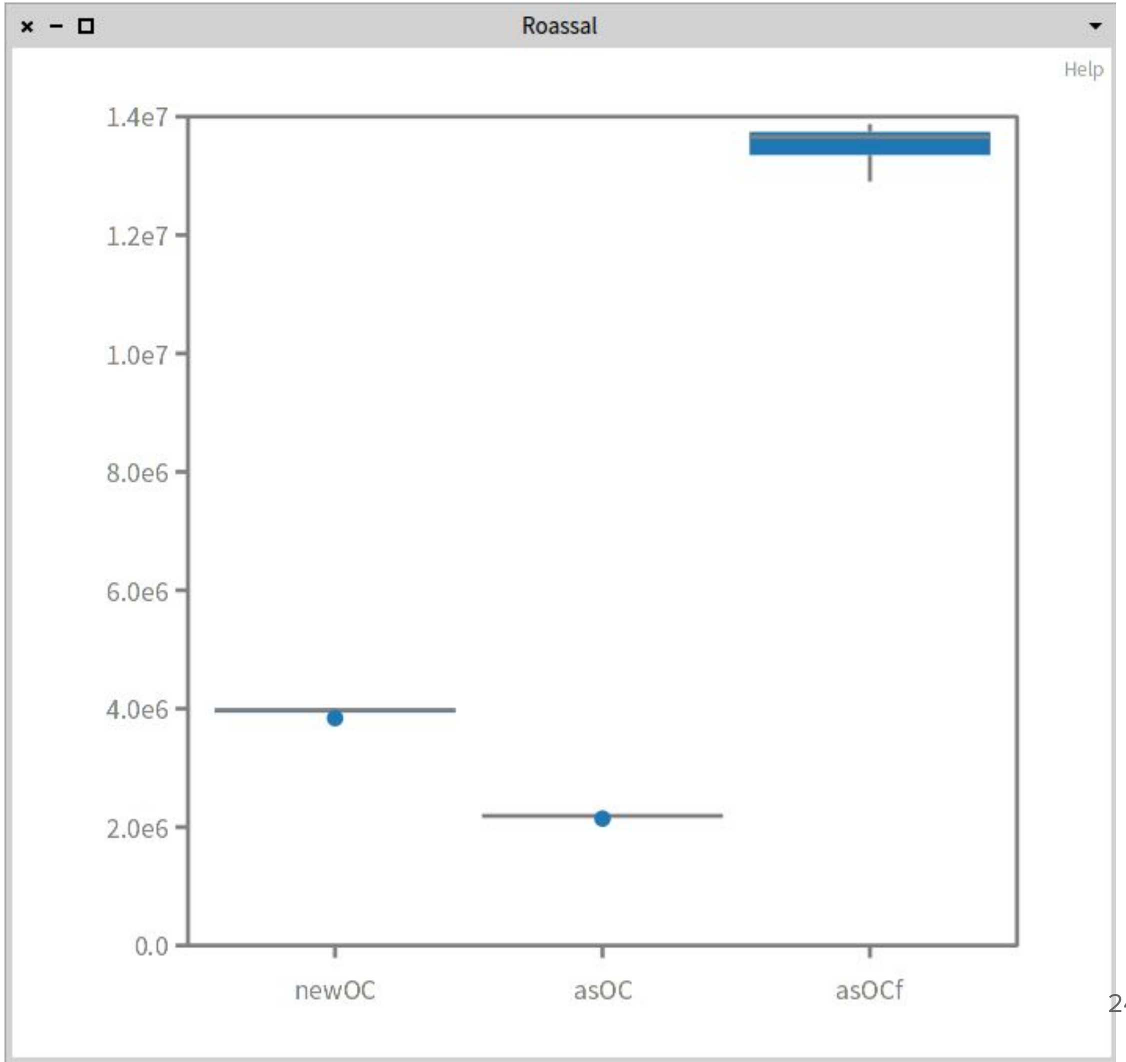
Numbers!

newOC 3.8M/s (base)

asOC (old) 1.9M/s (x0.5)

asOC (fast): 12.3M/s (x3.25)

Nice!



Questions?