A Tour to Spur for Non-VM Experts

Guille Polito, Christophe Demarey
ESUG 2016, 22/08, Praha

From a user point of view

We are working on the new Pharo Kernel

- Bootstrap: create an image from scratch
  - Classes
  - Global objects
  - Processes and contexts

- Image Initialization: What is the correct order?

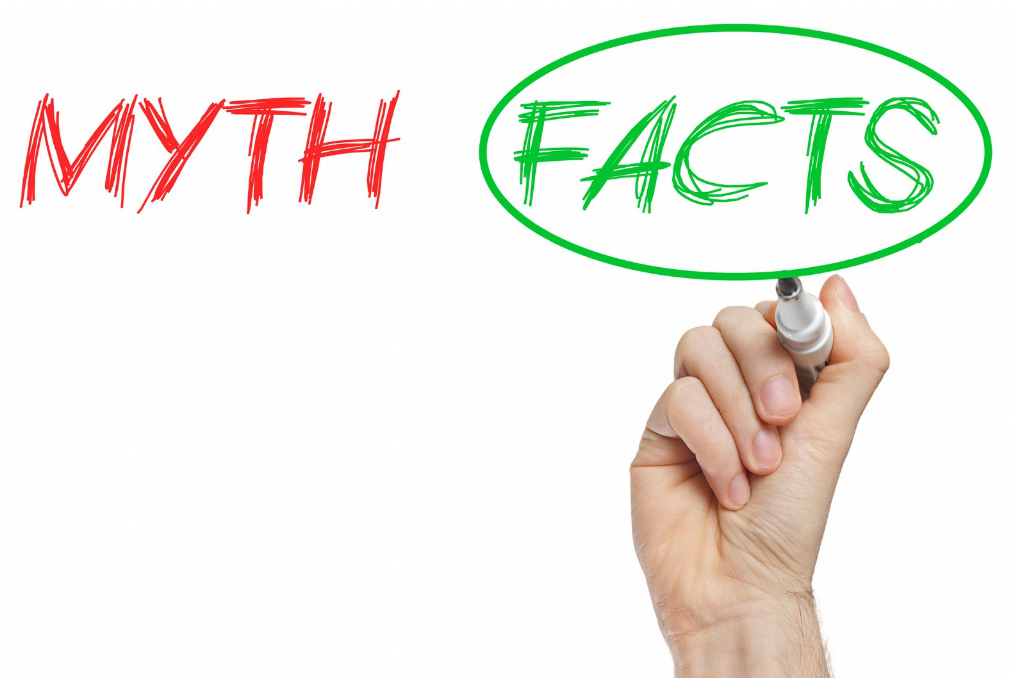
BTW, see our talk on this ;)

Mission Pharo Kernel, Thursday 10 am

What is this talk about?

Dec 14, 2015; 11:08am

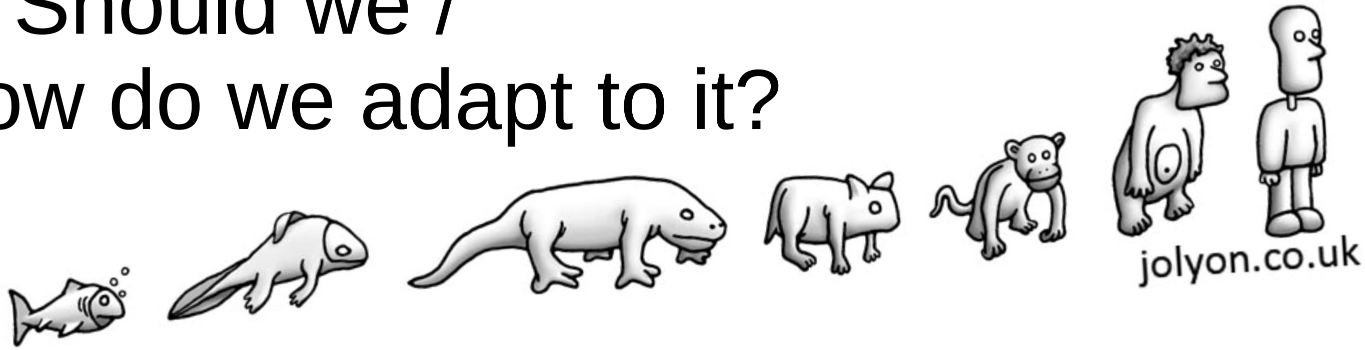
[IMPORTANT] Starting migration to Spur VM



Initial Motivation to look into Spur

1) How do we move to Spur as fast as possible?

2) Should we / How do we adapt to it?



3) What are the risks?

Motivation of this talk #1: Education

Explain what is Spur

Determine if a problem comes from **image side** or **VM side**?

Motivation of this talk #2: Understanding the Impact

Is my application **compatible**?

Will It **break**? Do I have to **port** it?

Part 1:
Demolishing Myths

MYTH

What is **Spur**?

Spur is **not a new Virtual Machine**

Its underlying execution engine is the same as in Cog
(same bytecode, same interpreter, same JIT compiler)

MYTH

Spur is **not** a new Garbage Collector

It **just** implements a new garbage collector
(which, BTW, is not new...)

MYTH

Spur is **not a new Object Format**.

It **just** implements a new object format
(which, BTW, is just the means to an end)

MYTH

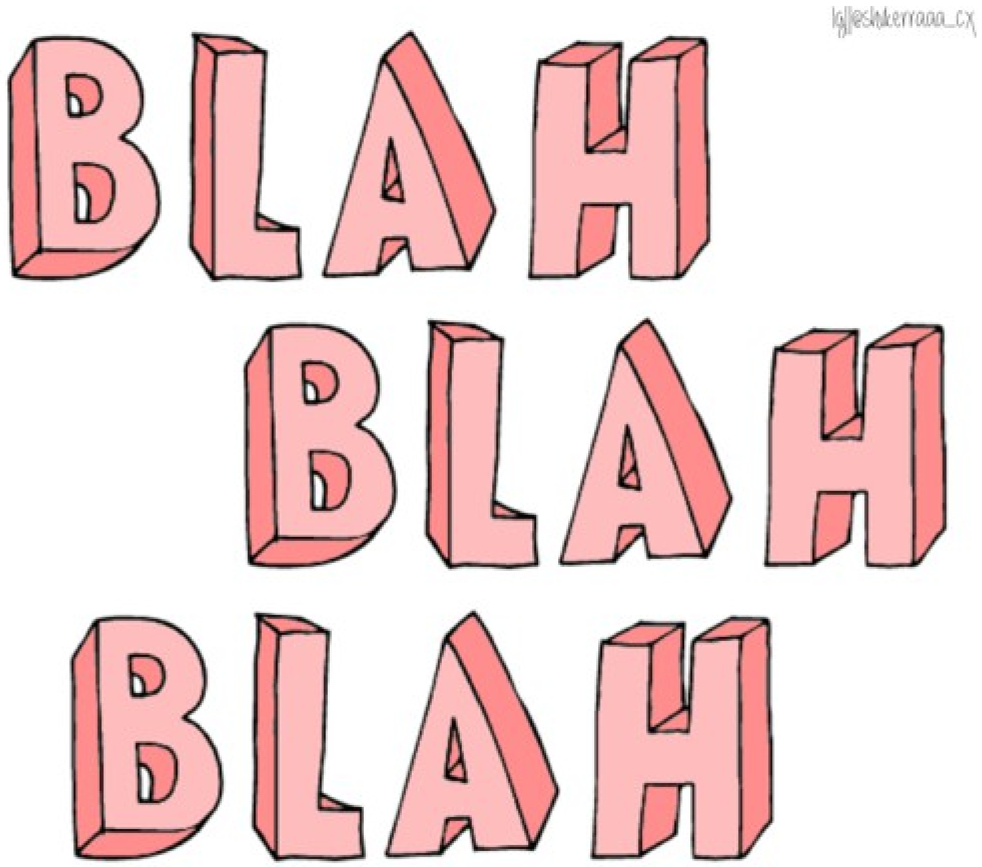
So... what is Spur?

Spur is a **new Memory Manager for Cog VM**.

- New object representation in memory
  (that allows ephemerons, pinned objects,...)

- New memory organization of Pharo images
  (that allows to better manage resources)

FACT


ig||@shakerraaa_cx
BLAH
BLAH
BLAH

Spur in a Nutshell

It's a Cog VM

+ **64 bits** support

+ **faster: x1.8 speedup**

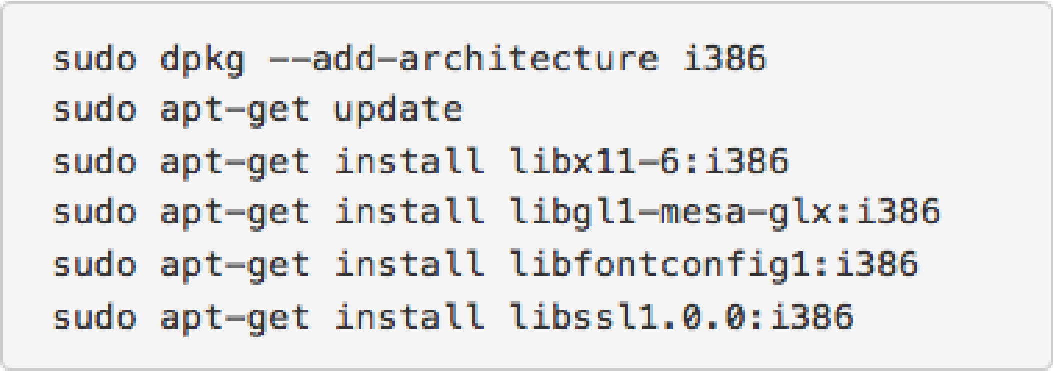
+ **larger images (> 2 Go)**

+ **ephemeron** support

and more ...

Spur > 64-bits support

- No more need to install 32-bits libraries

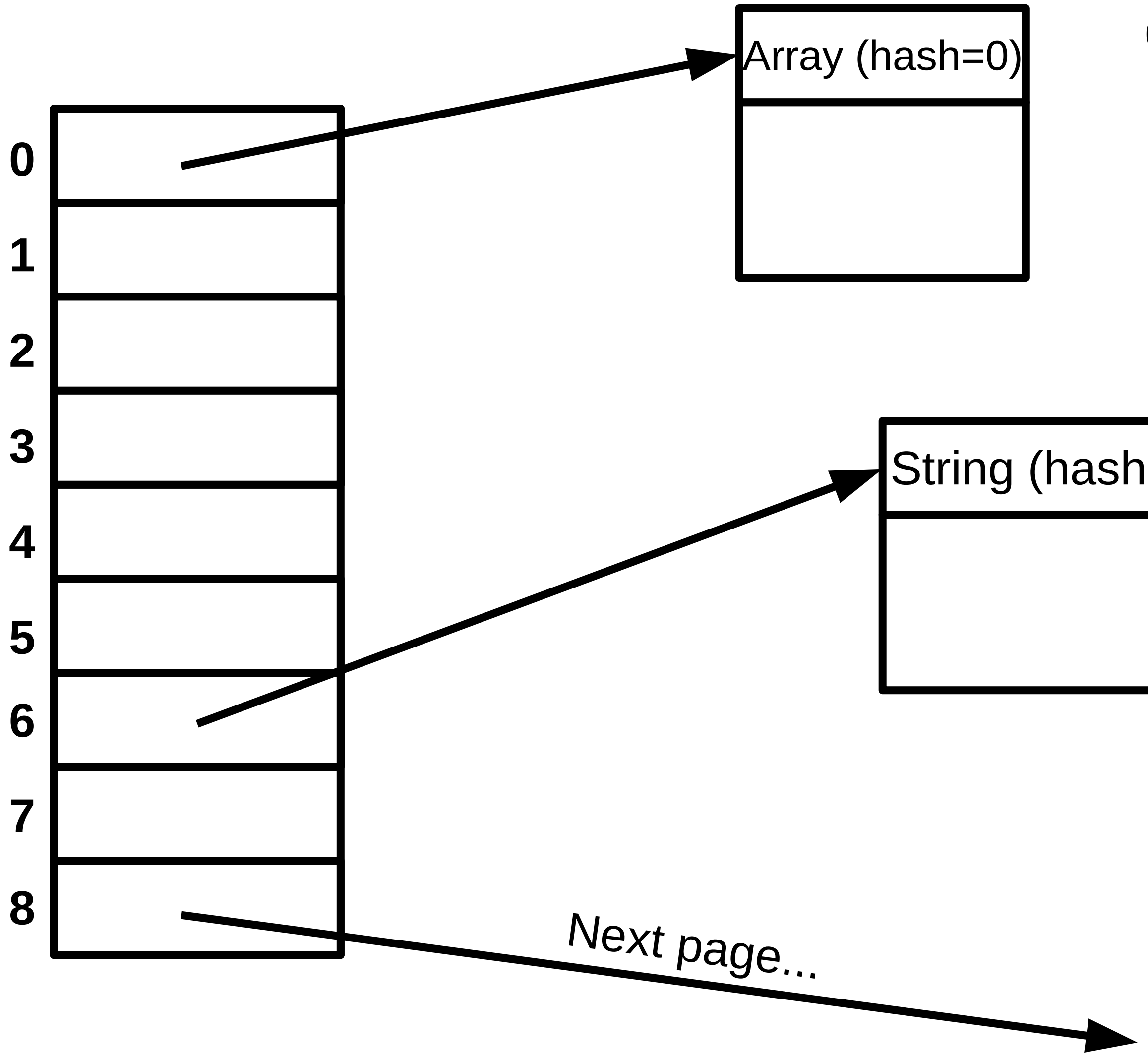


```
sudo dpkg --add-architecture i386
sudo apt-get update
sudo apt-get install libx11-6:i386
sudo apt-get install libgl1-mesa-glx:i386
sudo apt-get install libfontconfig1:i386
sudo apt-get install libssl1.0.0:i386
```

- Images with size > 2 Go

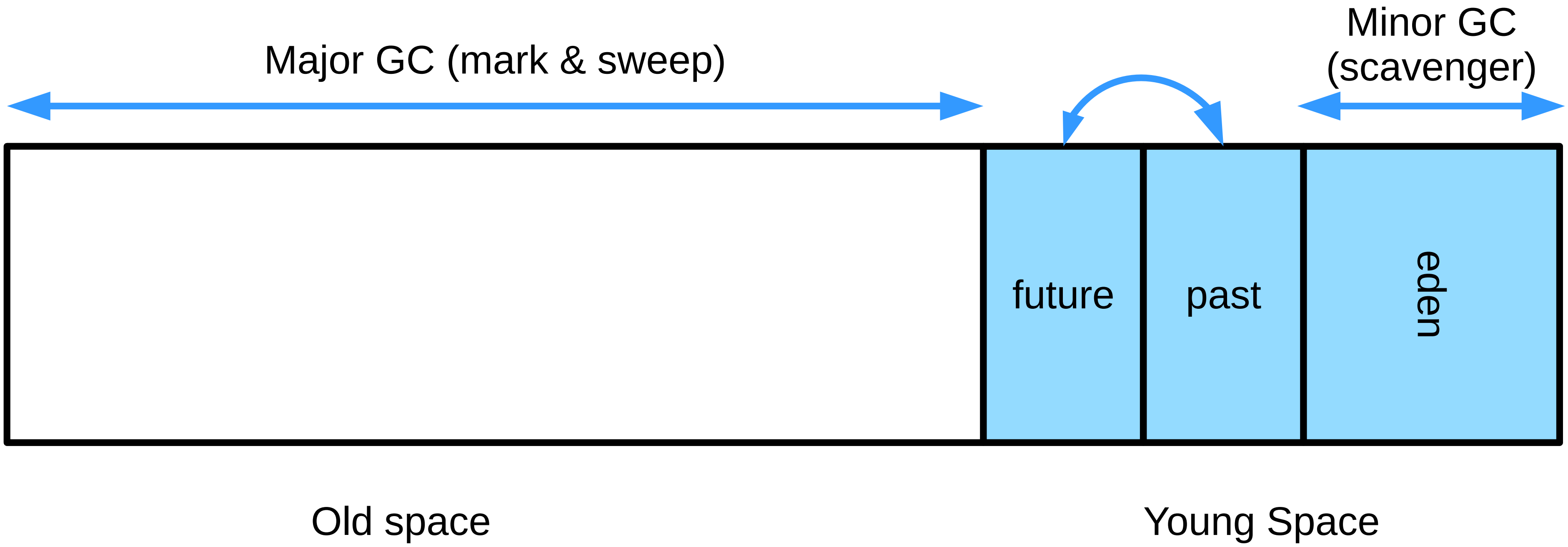
Spur > faster access to classes
Class Table

- Direct access to class objects



Spur > faster Garbage Collector

- “Young objects die young (and fast)”
- Added survivor segments (future and past) to the young space
  => allows more minor GC instead of major GC



Spur > Fast become

No more hangs in large images when using #become:

(e.g. Moose with a big famix model)

Why? Spur introduces forwarders

- prevents to scan the whole memory
- replaces pointers when they are accessed
- implemented by a partial read barrier[1]

Cheap in most cases (just one indirection)

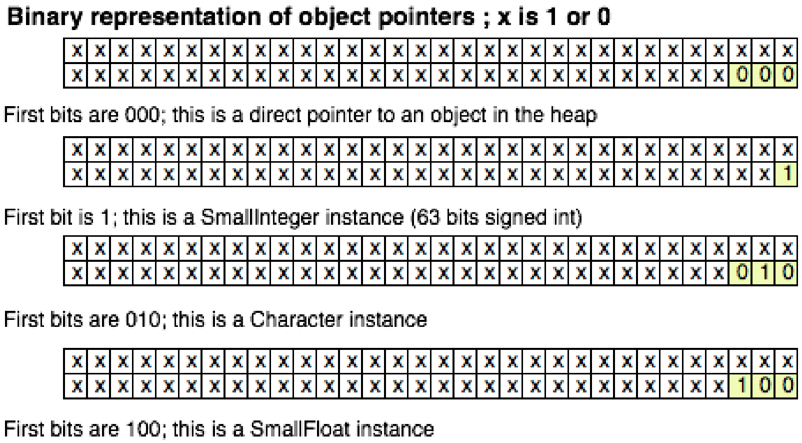
Costly if you rely a lot on primitives fallback

[1] Eliot Miranda, Clément Bera. A Partial Read Barrier for Efficient Support of Live Object-Oriented Programming. ISMM'15

Spur > faster
Immediate objects

New immediate objects

- Character
- Float (only 64-bits)



Binary representation of object pointers ; x is 1 or 0

```
x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x
x x x x x x x x x x x x x x x x x x x x x x x x x x x x x 0 0 0
```

First bits are 000; this is a direct pointer to an object in the heap

```
x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x
x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x 1
```

First bit is 1; this is a SmallInteger instance (63 bits signed int)

```
x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x
x x x x x x x x x x x x x x x x x x x x x x x x x x x x x 0 1 0
```

First bits are 010; this is a Character instance

```
x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x
x x x x x x x x x x x x x x x x x x x x x x x x x x x x x 1 0 0
```

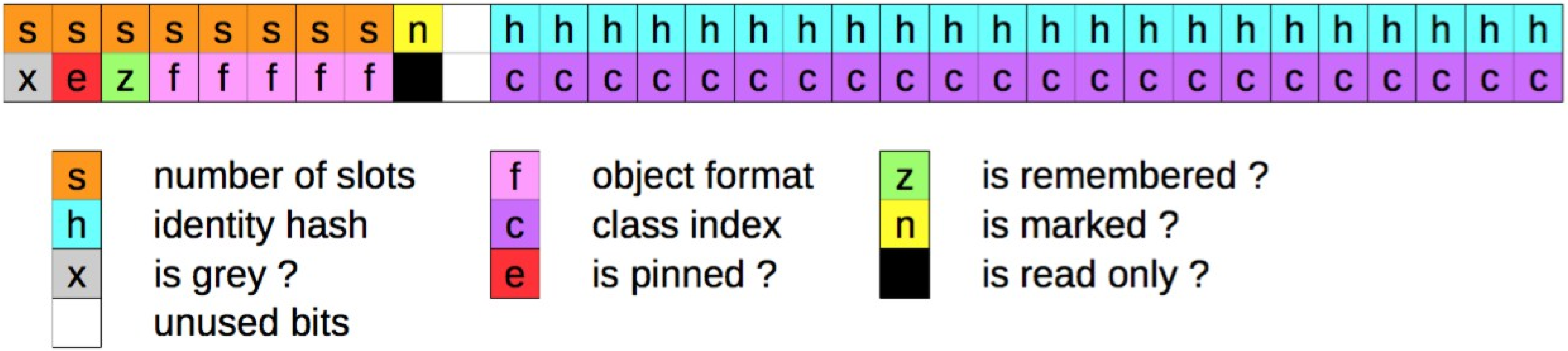
First bits are 100; this is a SmallFloat instance

Speed-up in wide strings

Speed-up in float arithmetic and memory saving

Spur > other features

Spur object format:



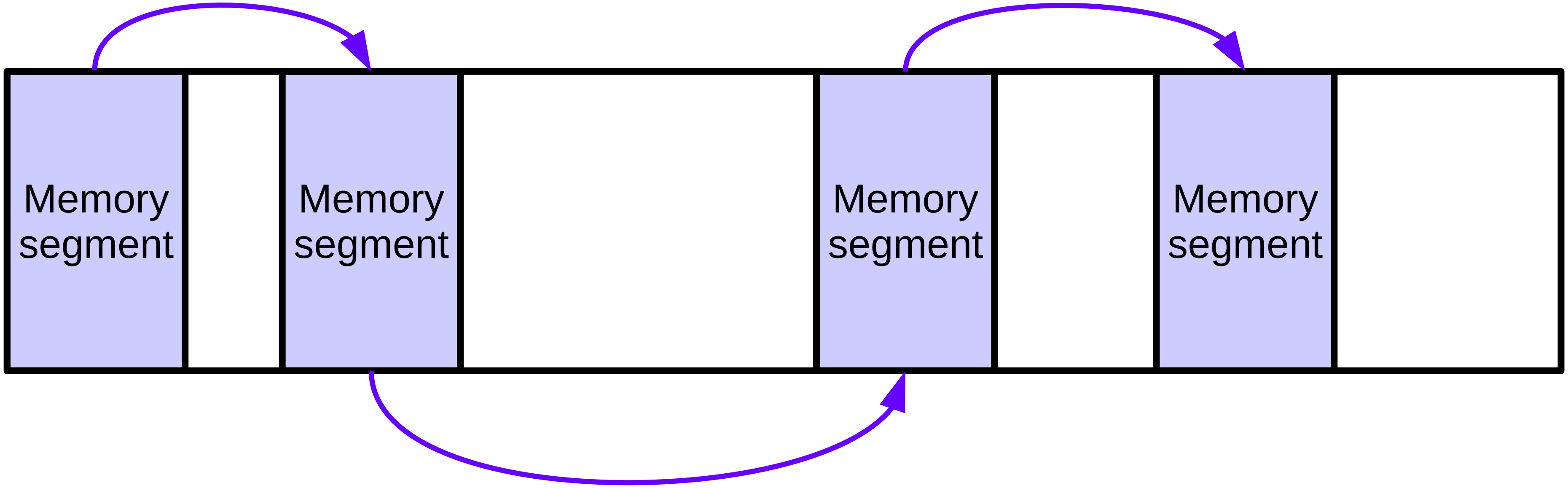
s	s	s	s	s	s	s	s	n		h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h
x	e	z	f	f	f	f	f	■		c	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c

s	number of slots	f	object format	z	is remembered ?
h	identity hash	c	class index	n	is marked ?
x	is grey ?	e	is pinned ?	■	is read only ?
	unused bits				

- All classes are compact
  => only two kind of headers (3 before Spur)
- Support for pinned-objects (**see UFFI talk on Friday**)
- Ongoing support of read-only objects
- Still 2 unused bits

Spur > scalability

- Memory is now divided in several segments
- No more need to have a contiguous chunk of memory



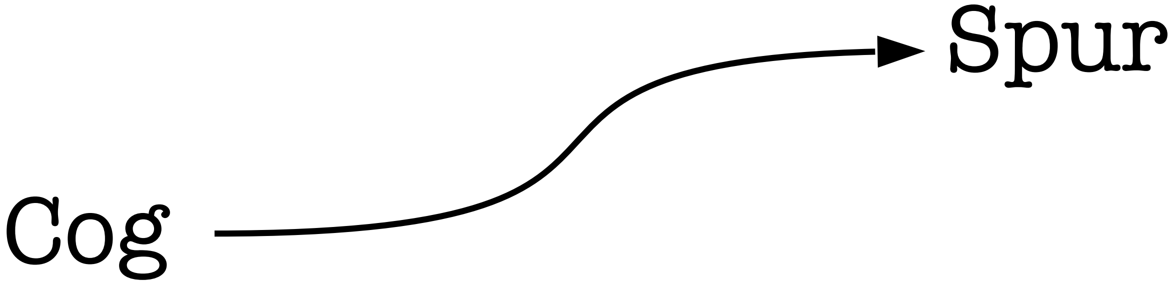
Spur > reliability

- Ephemeron finalization support
- Avoid memory leaks

BTW, see our OTHER talk on this ;)

A Weak Pharo story, Thursday, 3 pm

Part 2:
Porting applications and frameworks to Spur



Cog

Spur

How do I port my *application* to **Spur**?

Porting Applications

TO-DO LIST
~~NOTHING~~

Porting Applications

**JUST
DO
NOTHING.**

Porting Applications

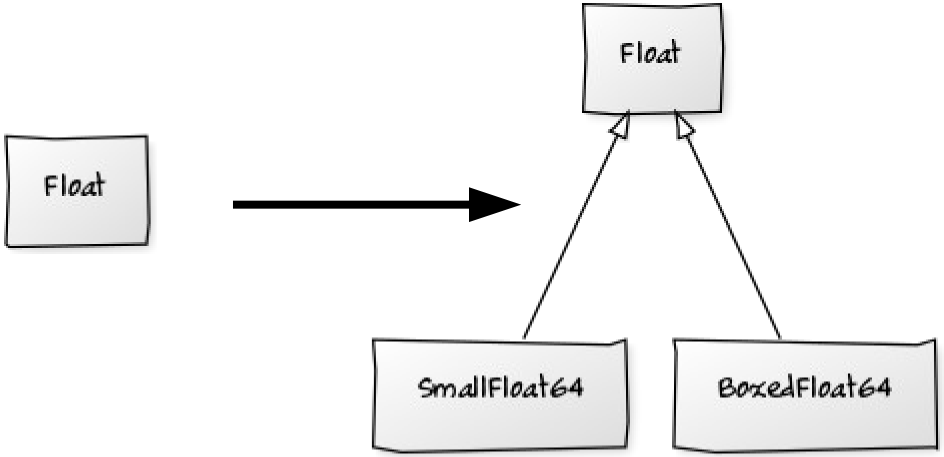
Okay, maybe just wait that your developer friends port your favorite frameworks.

Porting Frameworks/Libraries

I have work to do I have work to do
I have work to do I have work to do
I have work to do I have work to do
I have work to do I have

Porting Basics #1

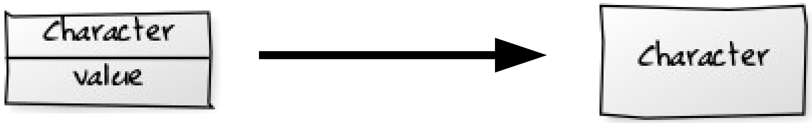
The number hierarchy changed



- Beware if you have visitors
- Beware if you have overrides

Porting Basics #2

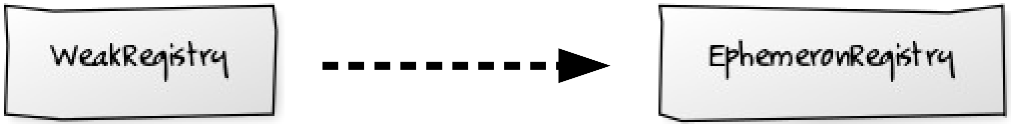
Character is now immediate



- Beware if you have overrides that use the internal state

Porting Basics #3

New (enhanced) ephemeron finalization



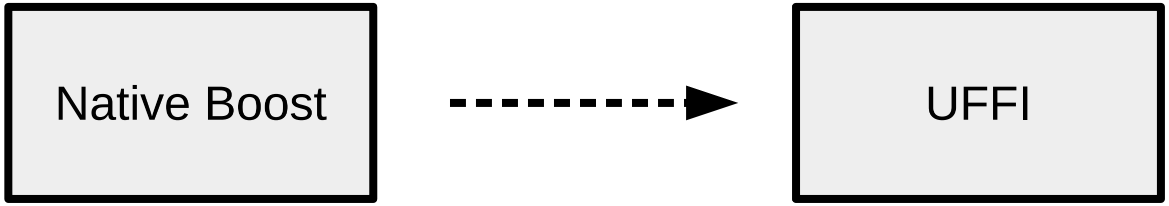
- If you need finalization you'll probably want to use the new one

BTW, see our OTHER talk on this ;)

A Weak Pharo story, Thursday, 3 pm

Porting Basics #4

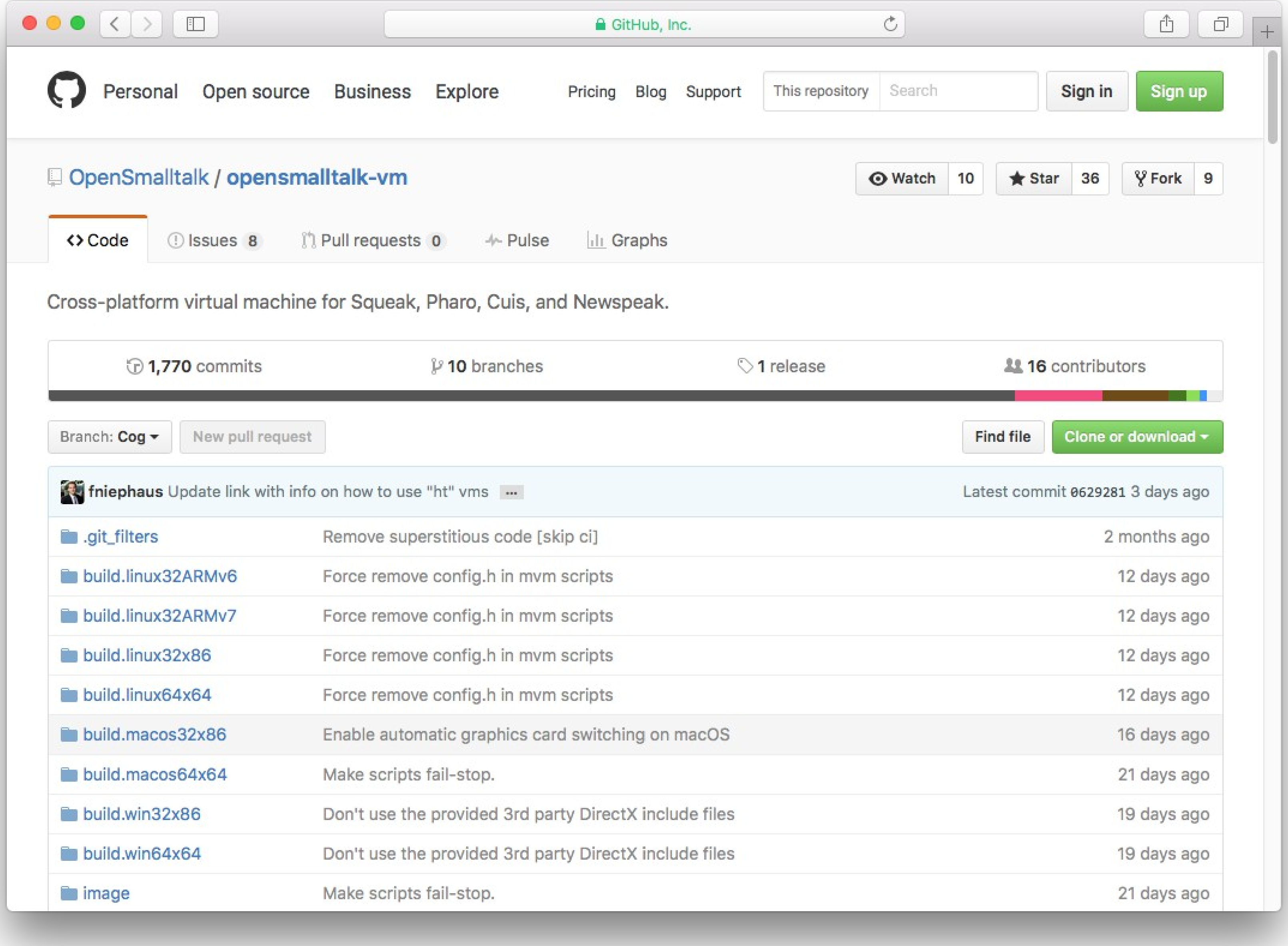
Native Boost is being deprecated



- If you are using FFI, you will need to review your bindings

Spur Behind the Scenes

VM development hosted on GitHub: OpenSmalltalk / opensmalltalk-vm



OpenSmalltalk / opensmalltalk-vm

Cross-platform virtual machine for Squeak, Pharo, Cuis, and Newspeak.

1,770 commits · 10 branches · 1 release · 16 contributors

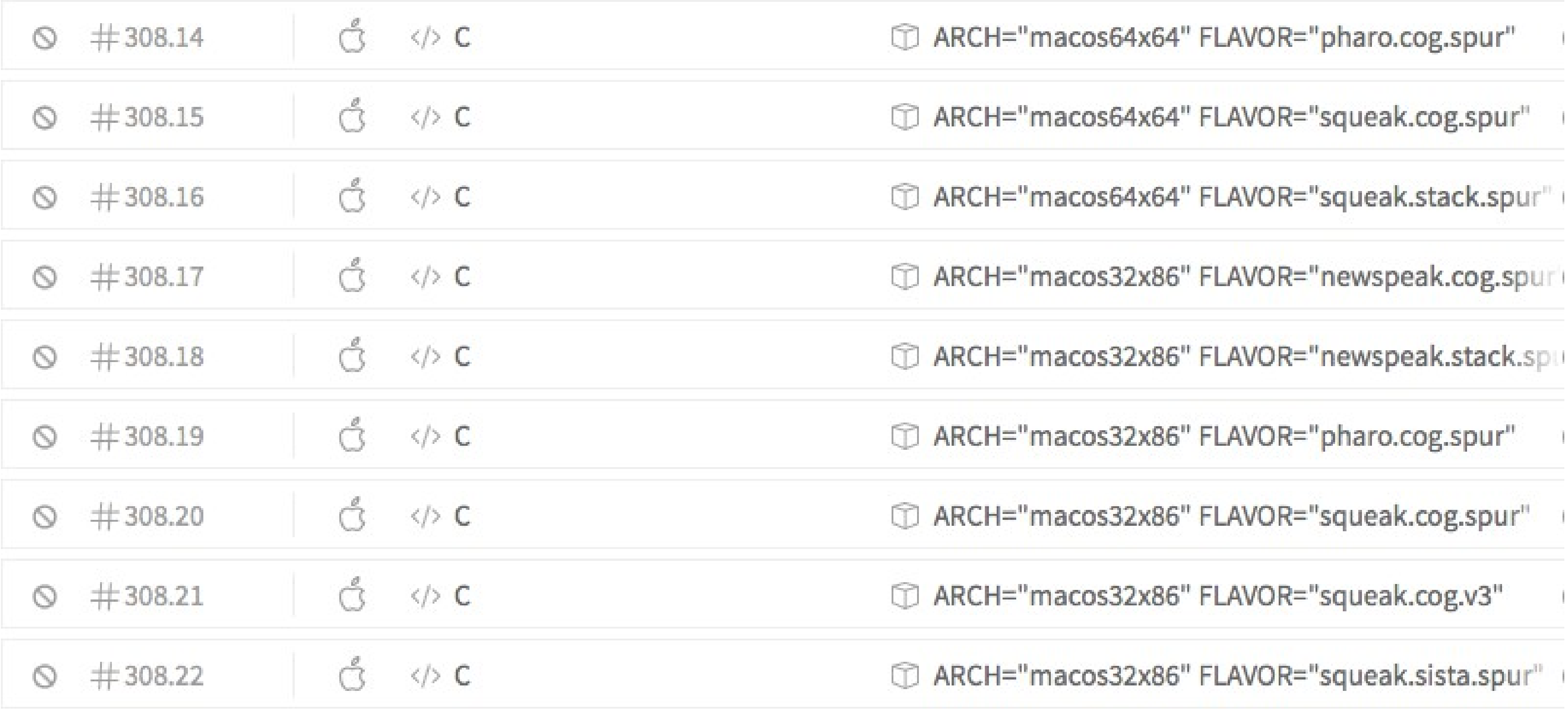
fniephaus Update link with info on how to use "ht" vms … Latest commit 0629281 3 days ago

.git_filters	Remove superstitious code [skip ci]	2 months ago
build.linux32ARMv6	Force remove config.h in mvm scripts	12 days ago
build.linux32ARMv7	Force remove config.h in mvm scripts	12 days ago
build.linux32x86	Force remove config.h in mvm scripts	12 days ago
build.linux64x64	Force remove config.h in mvm scripts	12 days ago
build.macos32x86	Enable automatic graphics card switching on macOS	16 days ago
build.macos64x64	Make scripts fail-stop.	21 days ago
build.win32x86	Don't use the provided 3rd party DirectX include files	19 days ago
build.win64x64	Don't use the provided 3rd party DirectX include files	19 days ago
image	Make scripts fail-stop.	21 days ago

Why is it a good news?

- Brings together the VM community
- Easier to contribute
  - Pull requests
  - Issue tracker
  - Documentation: https://github.com/OpenSmalltalk/opensmalltalk-vm/blob/Cog/CONTRIBUTING.md

VM build all flavors through Travis CI



⊘	#308.14	</> C	ARCH="macos64x64" FLAVOR="pharo.cog.spur"
⊘	#308.15	</> C	ARCH="macos64x64" FLAVOR="squeak.cog.spur"
⊘	#308.16	</> C	ARCH="macos64x64" FLAVOR="squeak.stack.spur"
⊘	#308.17	</> C	ARCH="macos32x86" FLAVOR="newspeak.cog.spur
⊘	#308.18	</> C	ARCH="macos32x86" FLAVOR="newspeak.stack.sp
⊘	#308.19	</> C	ARCH="macos32x86" FLAVOR="pharo.cog.spur"
⊘	#308.20	</> C	ARCH="macos32x86" FLAVOR="squeak.cog.spur"
⊘	#308.21	</> C	ARCH="macos32x86" FLAVOR="squeak.cog.v3"
⊘	#308.22	</> C	ARCH="macos32x86" FLAVOR="squeak.sista.spur"

Still missing VM tests. Upcoming?

Where to find VM binaries?

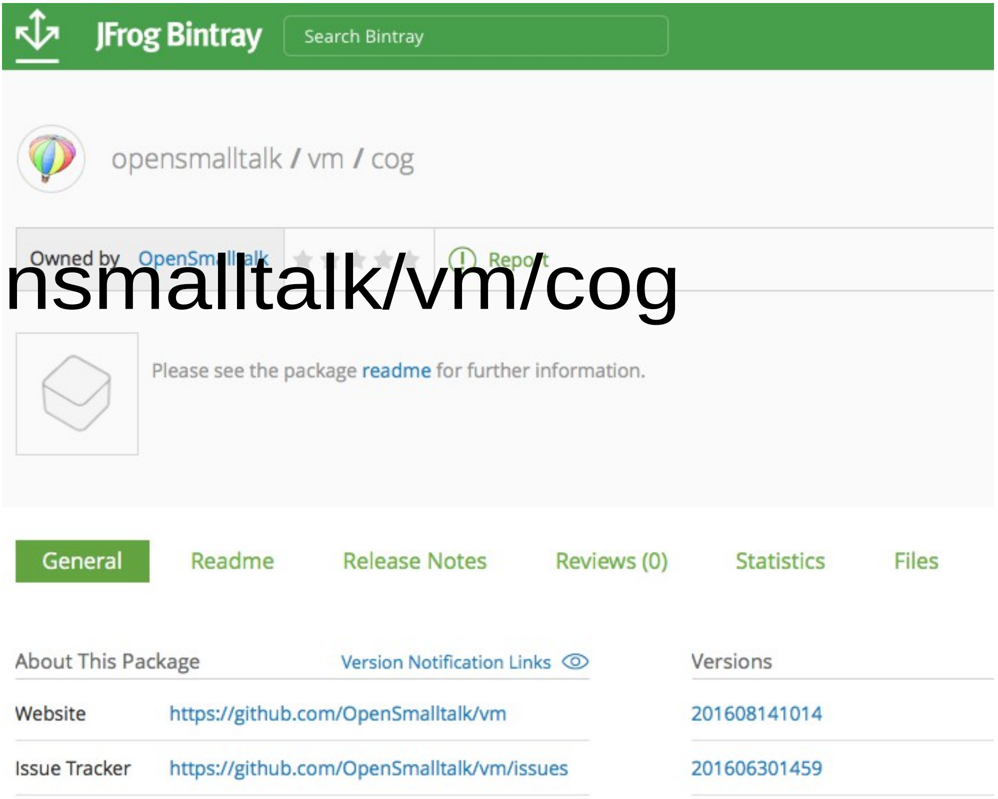
- Pharo

Virtual Machines

http://files.pharo.org/vm/

Squeak, NewSpeak

https://bintray.com/opensmalltalk/vm/cog



opensmalltalk / vm / cog

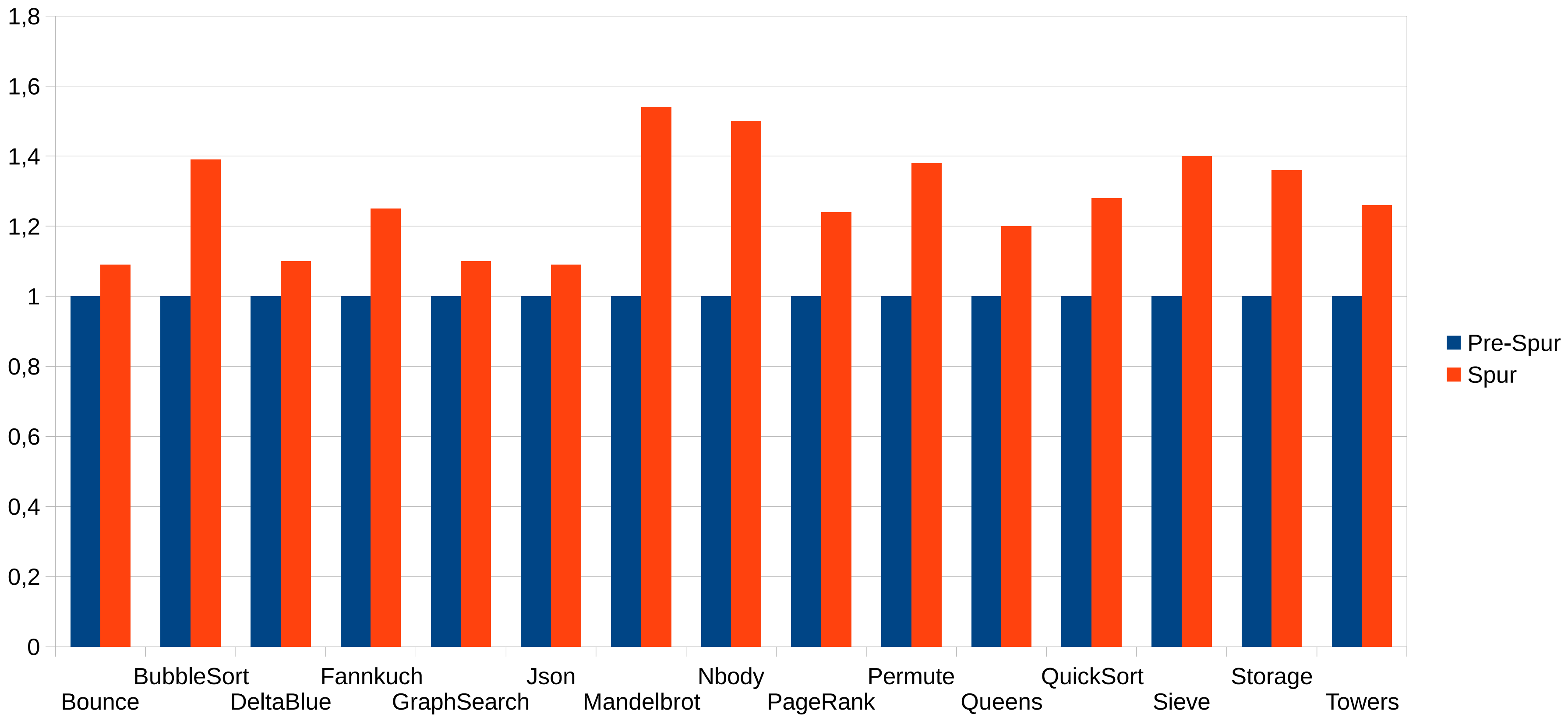
Owned by OpenSmalltalk

Please see the package readme for further information.

About This Package	Version Notification Links	Versions
Website	https://github.com/OpenSmalltalk/vm	201608141014
Issue Tracker	https://github.com/OpenSmalltalk/vm/issues	201606301459

Conclusion

Should I move to Spur?



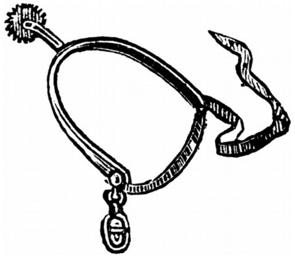
by Stefan Marr, Apr 06, 2015

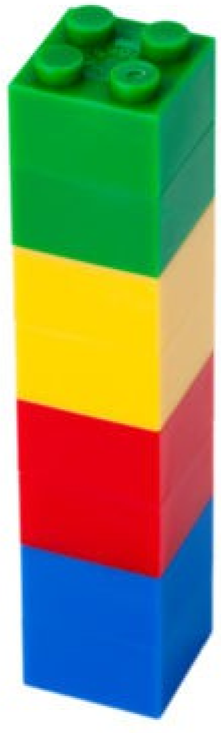
Should I move to Spur?

- ✓ 64-bits support
- ✓ Increased performances: x1.8 speedup
- ✓ Scalability, Reliability and open to new features
- – image not compacting anymore (will be fixed soon)

EXTRA SLIDES!

Dissecting Spur...



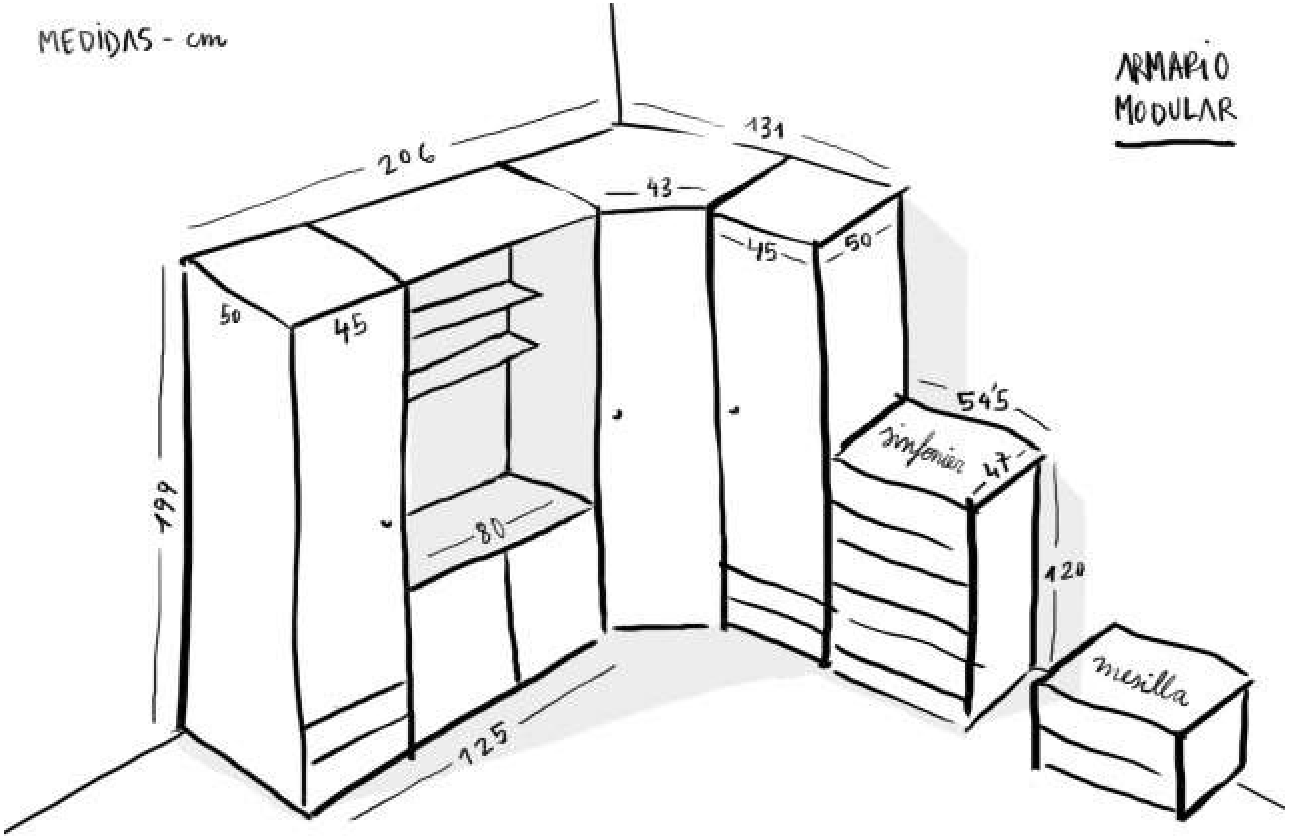


1) Class tables

2) Forwarders

3) Ephemeron Finalization

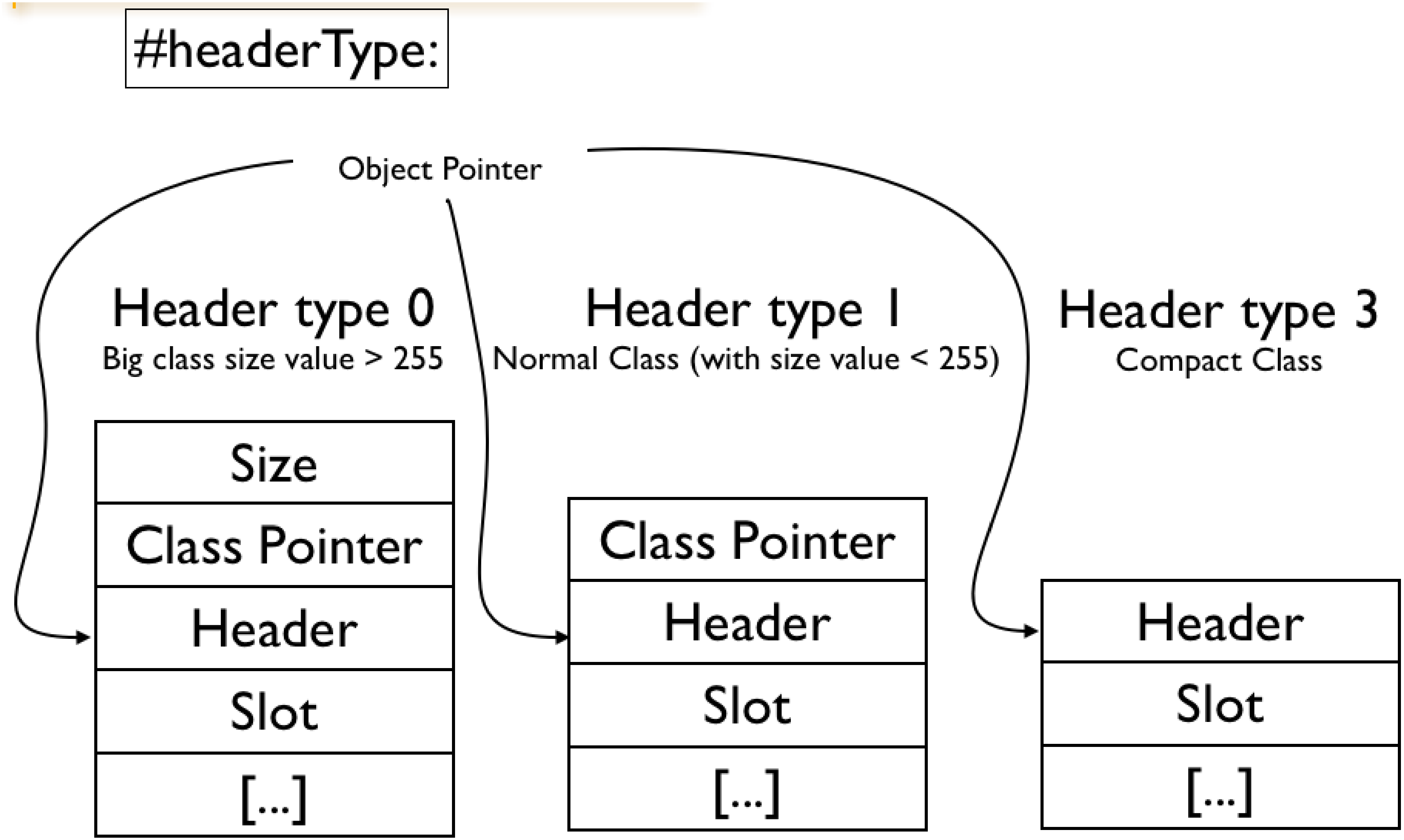
4) The Scavenger GC



Chapter 1

Classes are in Tables
(and they hide in tables)

1.1 The old object header...



#headerType:

Object Pointer

Header type 0
Big class size value > 255

Size
Class Pointer
Header
Slot
[...]

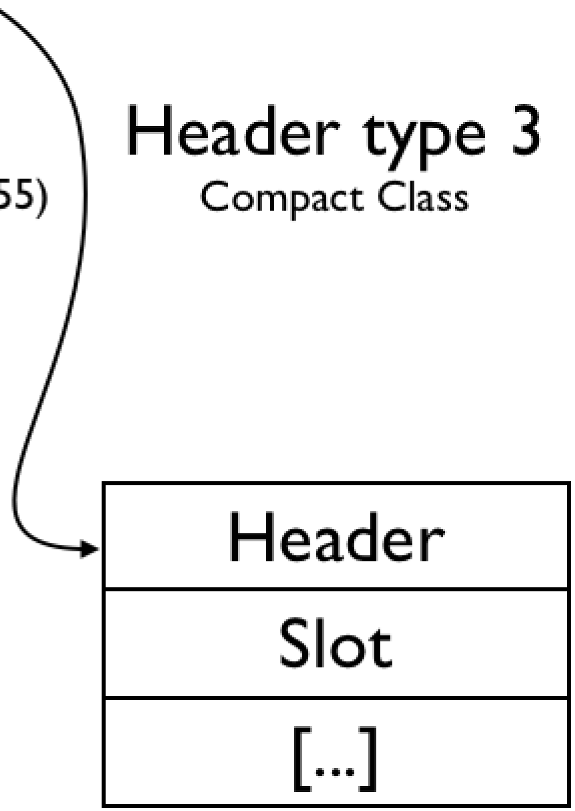
Header type 1
Normal Class (with size value < 255)

Class Pointer
Header
Slot
[...]

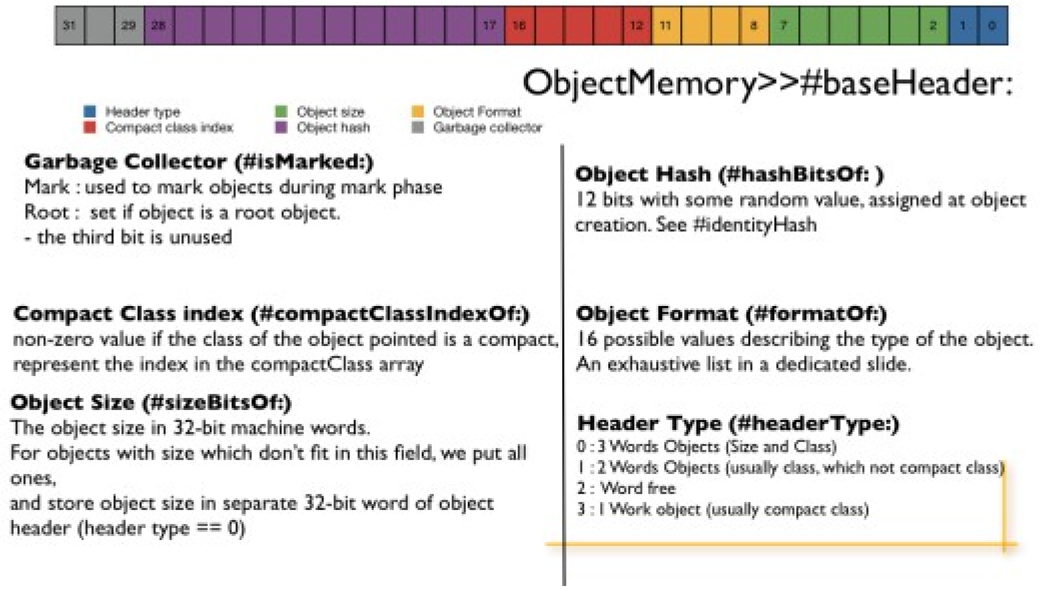
Header type 3
Compact Class

Header
Slot
[...]

1.1 Compact classes



Smalltalk compactClassesArray



ObjectMemory>>#baseHeader:

🟦 Header type
🟥 Compact class index
🟩 Object size
🟪 Object hash
🟨 Object Format
⬜ Garbage collector

Garbage Collector (#isMarked:)
Mark : used to mark objects during mark phase
Root : set if object is a root object.
- the third bit is unused

Compact Class index (#compactClassIndexOf:)
non-zero value if the class of the object pointed is a compact, represent the index in the compactClass array

Object Size (#sizeBitsOf:)
The object size in 32-bit machine words.
For objects with size which don't fit in this field, we put all ones,
and store object size in separate 32-bit word of object header (header type == 0)

Object Hash (#hashBitsOf:)
12 bits with some random value, assigned at object creation. See #identityHash

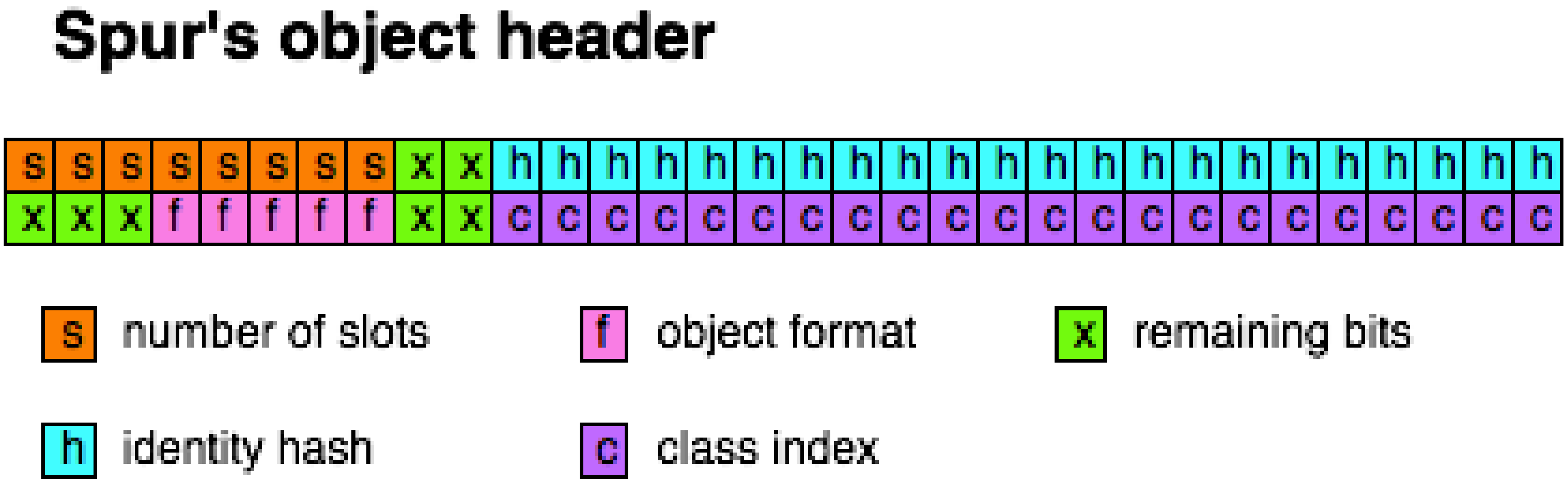
Object Format (#formatOf:)
16 possible values describing the type of the object.
An exhaustive list in a dedicated slide.

Header Type (#headerType:)
0 : 3 Words Objects (Size and Class)
1 : 2 Words Objects (usually class, which not compact class)
2 : Word free
3 : 1 Work object (usually compact class)

1.1 Cons of the old object header

- A full word is used to indicate an object's class
  - 4G classes in 32 bits
  - 16E (2^60) classes in 64 bits (!!)
- Three different headers
  => checks for the header type are common

1.2 New class header



Spur's object header

s	s	s	s	s	s	s	s	x	x	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h
x	x	x	f	f	f	f	f	x	x	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c	c

s number of slots

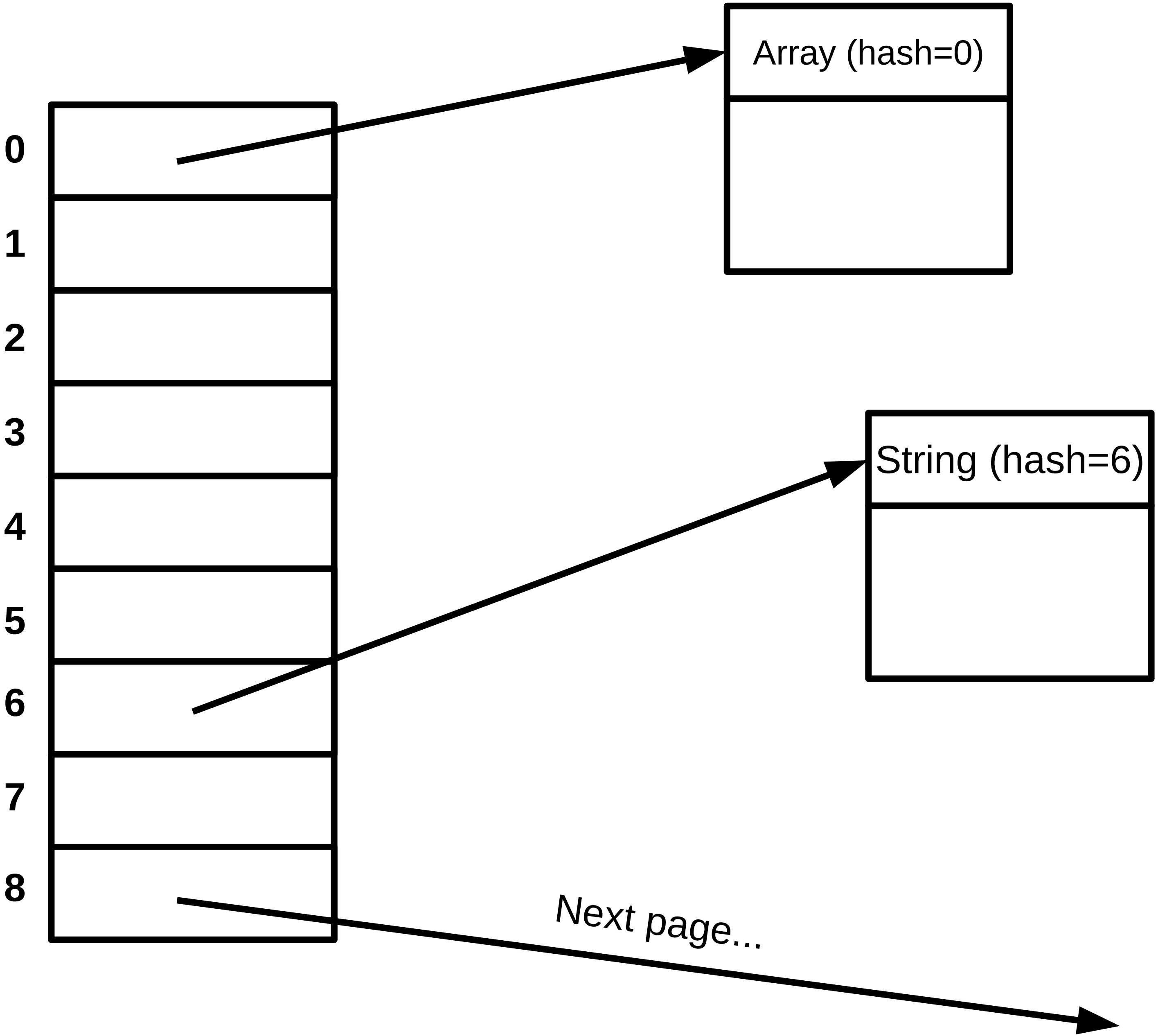
f object format

x remaining bits

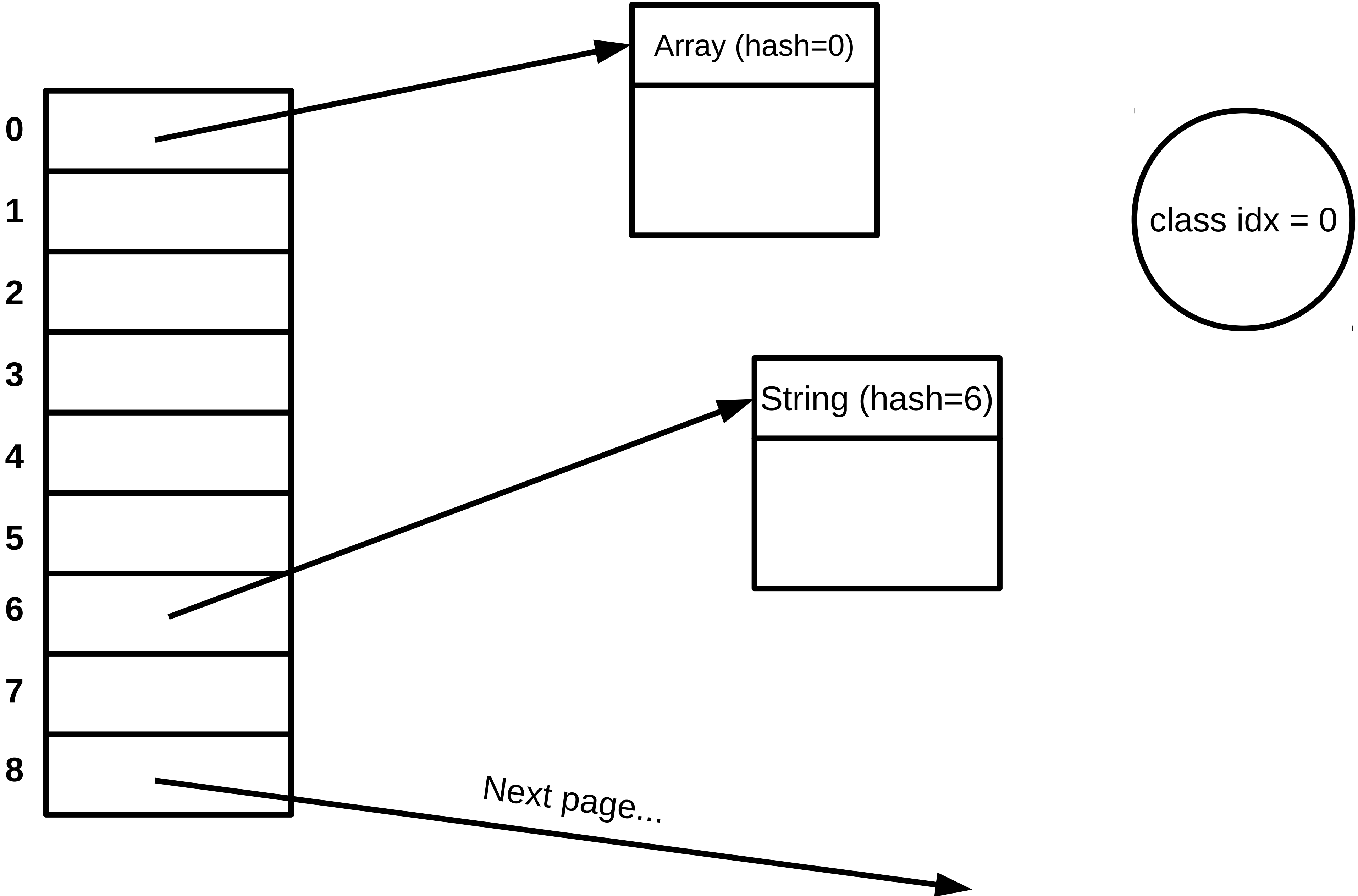
h identity hash

c class index

1.2 Class table



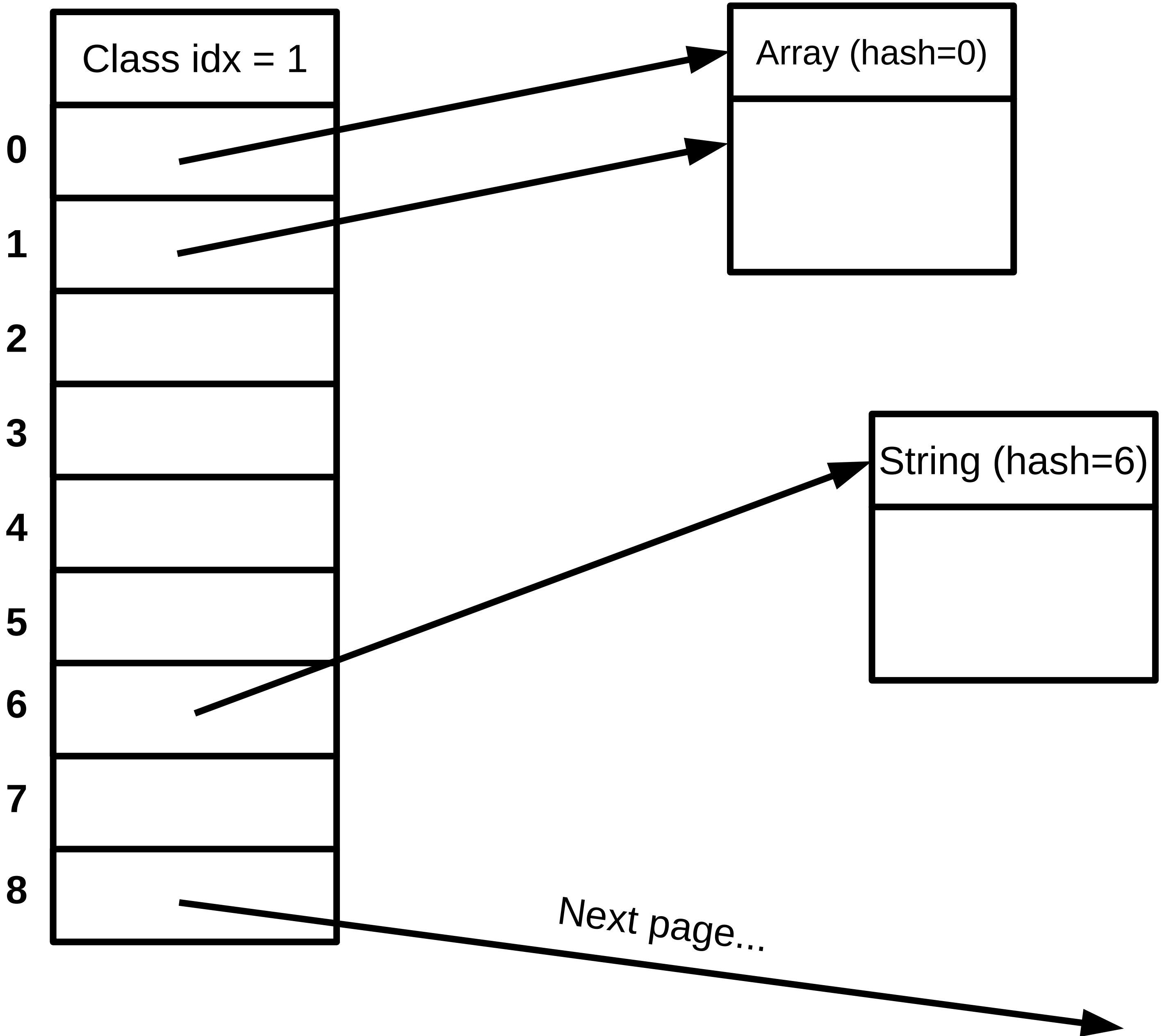
1.2 Class table



1.2 Pros of the new object header

- 2^22 classes (4M). Still enough and efficient.
- Compatible with 64bits
- All classes are compact => only two kind of headers

1.2 Hidden objects



1.2 Hidden objects?

- The class table is an object (and lives in the heap)

- Its class index is “hidden”:
  - **Array allInstances**

    will iterate objects by class index

- In the class table:
  - Indexes 0-15 are reserved for tagged objects
  - Indexes 16-32 are reserved for hidden classes

1.3 Maintaining the class table

Classes are normal objects...

They are created with no special primitives...

But...

How does the VM know an object is a class to put it into the class table?

1.3 Identifying classes by definition

A class is an object that is instantiated:

A class enters the class table upon instantiation

1.3 But the index is the hash!

But... hashes are assigned lazily for all objects:

Classes, on instance-side,
define a special hash method

```
Behavior >> basicIdentityHash

    <primitive: 175>
    self primitiveFailed
```

```
Object >> basicIdentityHash

    <primitive: 75>
    self primitiveFailed
```

Chapter 1 - Conclusions

- Classes are organized in tables
- All classes are compact

- Simpler object header
- Still place for 4M classes

- On the image side, is almost transparent

Chapter 2

The forwarder plague

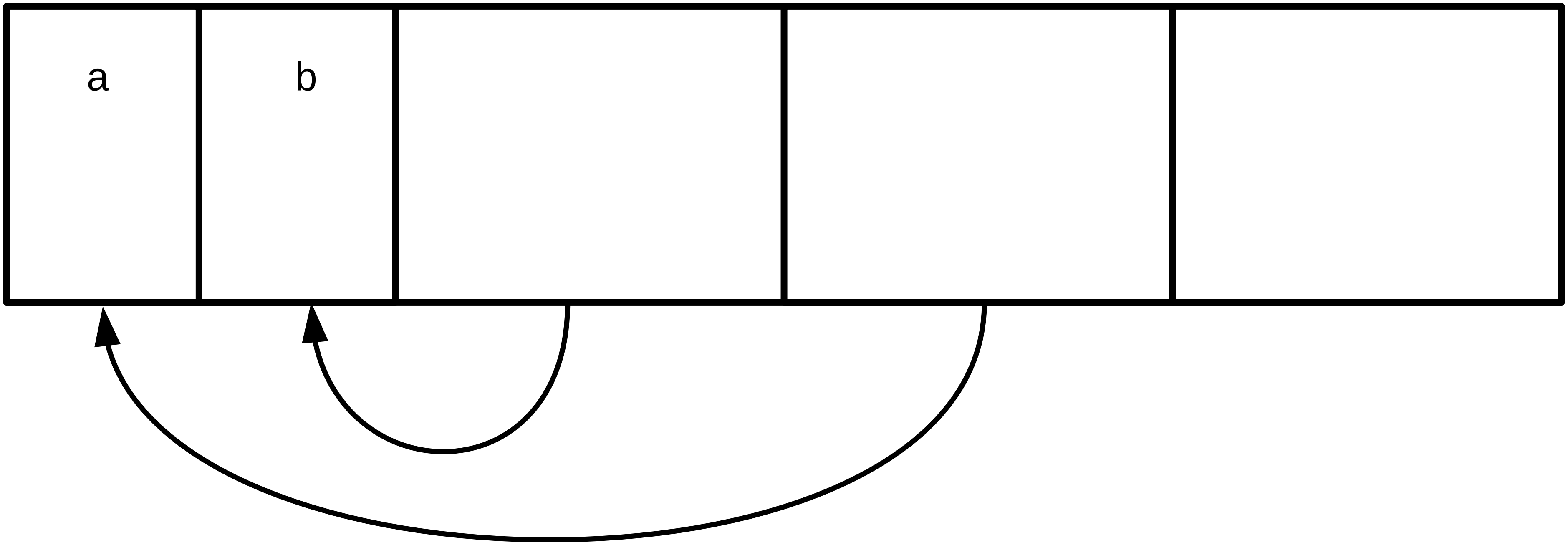
2.1 Become

- Swaps two objects
  - (actually, swaps two object's identity)
- Useful for:
  - Updating and migrating objects
  - Install proxies
  - Replace an object's behavior

2.1 The old become

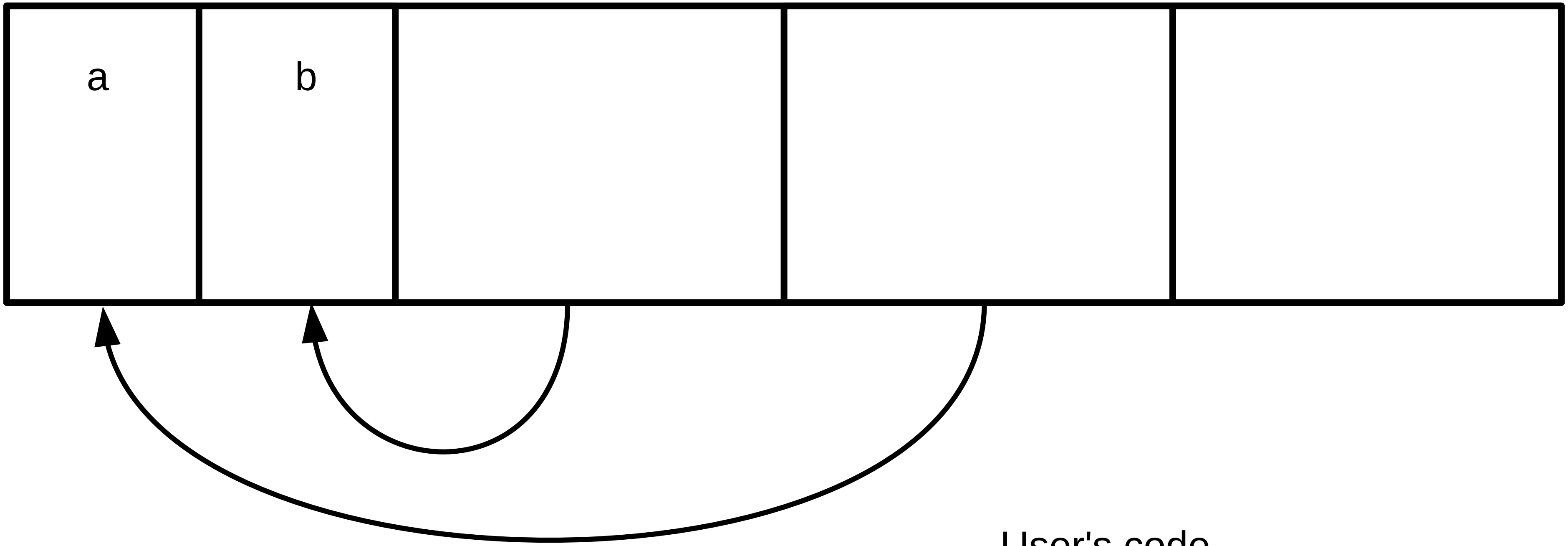
- Full scanned all memory
- And was SLOOOOOW

2.1 Lazy become



Eliot Miranda, Clément Bera. A Partial Read Barrier for Efficient Support of Live Object-Oriented Programming. ISMM'15

2.1 Lazy become

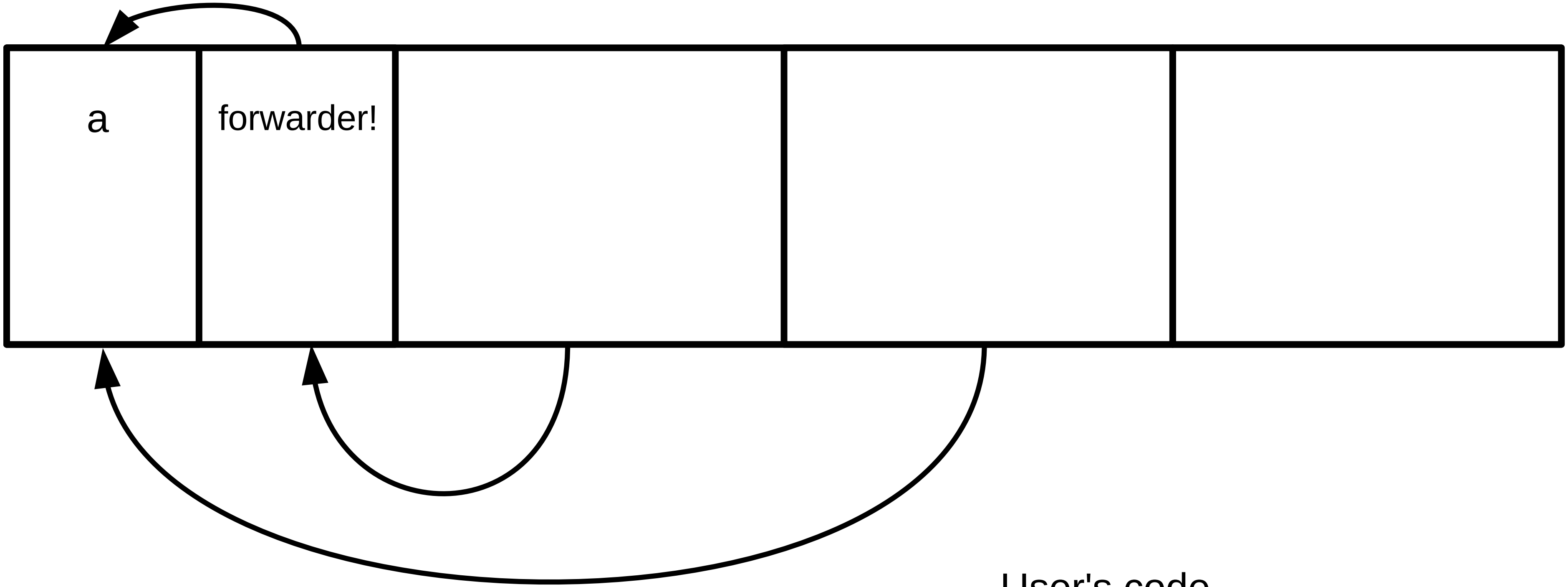


User's code

b becomeForward: a.

Eliot Miranda, Clément Bera. A Partial Read Barrier for Efficient Support of Live Object-Oriented Programming. ISMM'15

2.1 Lazy become

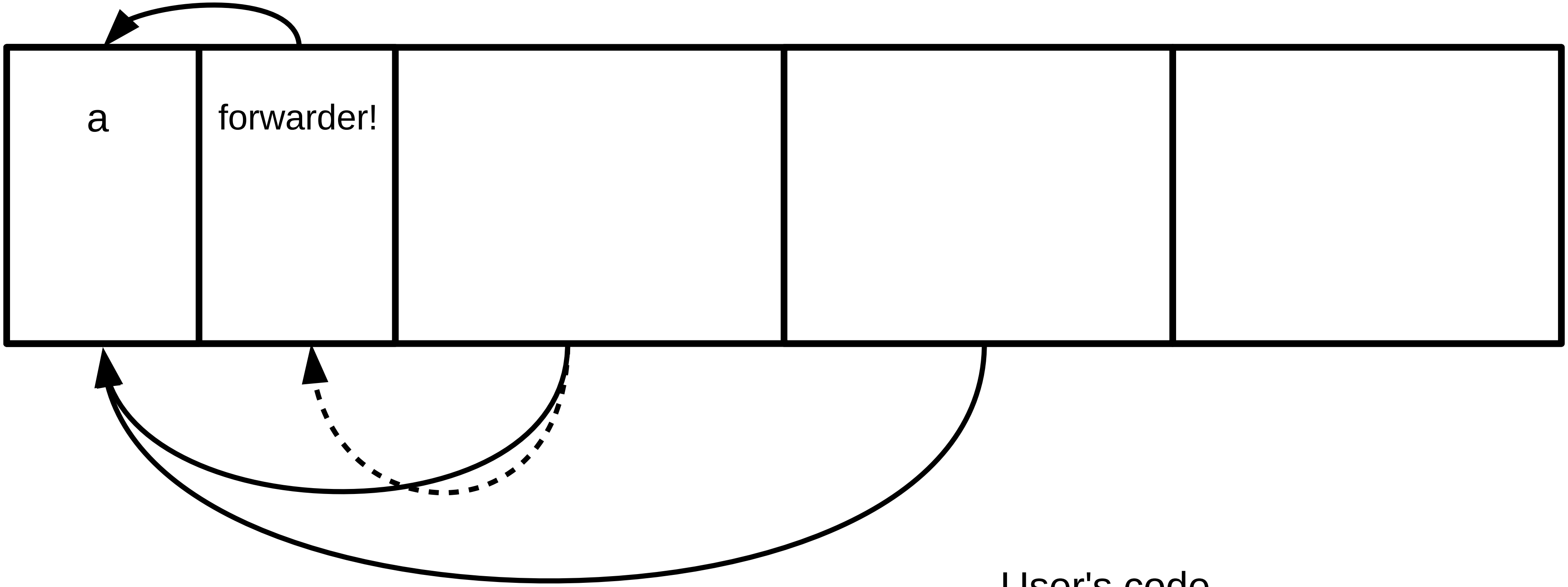


User's code

b becomeForward: a.

Eliot Miranda, Clément Bera. A Partial Read Barrier for Efficient Support of Live Object-Oriented Programming. ISMM'15

2.1 Lazy Become



User's code

```
b becomeForward: a.
b doSomething
```

Eliot Miranda, Clément Bera. A Partial Read Barrier for Efficient Support of Live Object-Oriented Programming. ISMM'15

2.2 The read barrier

- A full read barrier would be too expensive
  - (on every read, on every primitive, on every message send...)

- The read barrier is implemented in two places:
  - Message send lookup failure
  - Primitive failure

Eliot Miranda, Clément Bera. A Partial Read Barrier for Efficient Support of Live Object-Oriented Programming. ISMM'15

2.2 Message send lookup failure

```
method := (self lookupSelector: selector inClass: class).
method ifNil: [
    (receiver isForwarder) ifTrue: [
            receiver := receiver forward.
            “scan also the objects in the stack” ].
    method := (self lookupSelector: selector inClass: class).
].
```

Eliot Miranda, Clément Bera. A Partial Read Barrier for Efficient Support of Live Object-Oriented Programming. ISMM'15

2.2 Primitive failure

```
self performPrimitive: primitiveNumber.
self primitiveFailed ifTrue: [
    “scan the stack looking for forwarders and retry”
    self performPrimitive: primitiveNumber.
].
```

Eliot Miranda, Clément Bera. A Partial Read Barrier for Efficient Support of Live Object-Oriented Programming. ISMM'15

2 Conclusions

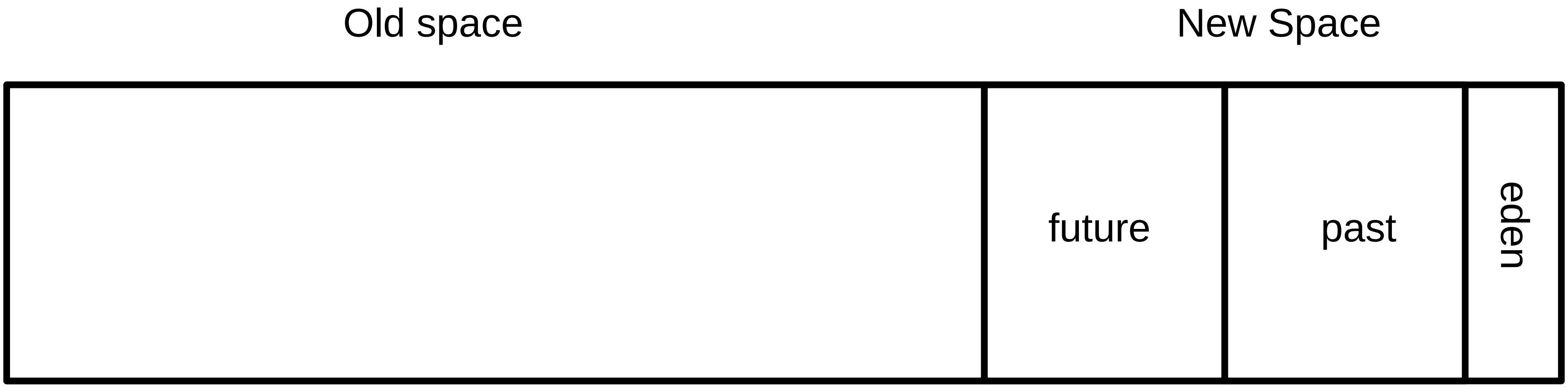
- Become does not need full scan anymore
- A forwarder replaces the object *in place*
- Two-way become copies object at the end

- Forwarders are bypassed using a partial read barrier:
  - Message lookup failure
  - Primitive failure
- No noticeable overhead

3.5 Scavenger GC

- “Young Objects Die Young (and quick)”
- Young objects are created in **eden**
- Objects are “tenured” after surviving several generations
- Tenured objects go to **old space**

3.5 Scavenger GC



Old space

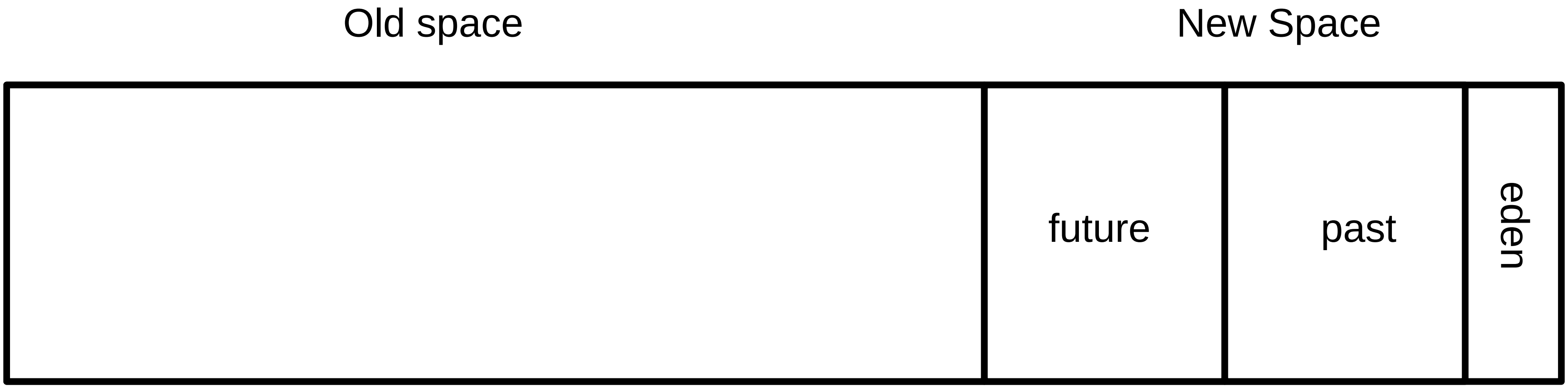
New Space

future

past

eden

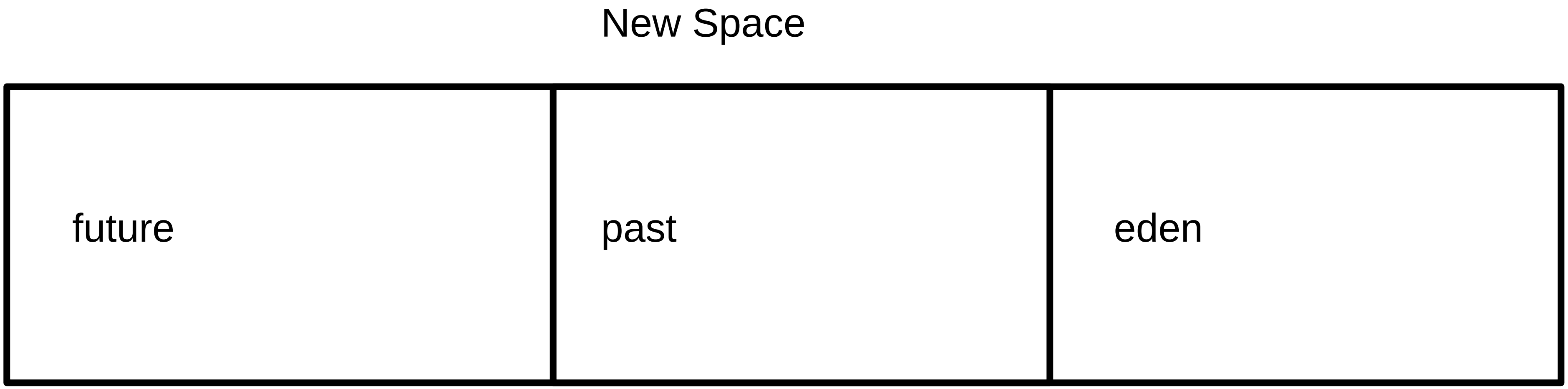
3.5 Scavenger GC



- Mark and Sweep (marking collector)
- Runs “every blue moon” on the entire memory
- Slow

- Scavenger (copying collector)
- Runs often, only in new space
- Object tenure (to old space) depends on the ratio of allocation

3.5 Scavenger GC



New Space

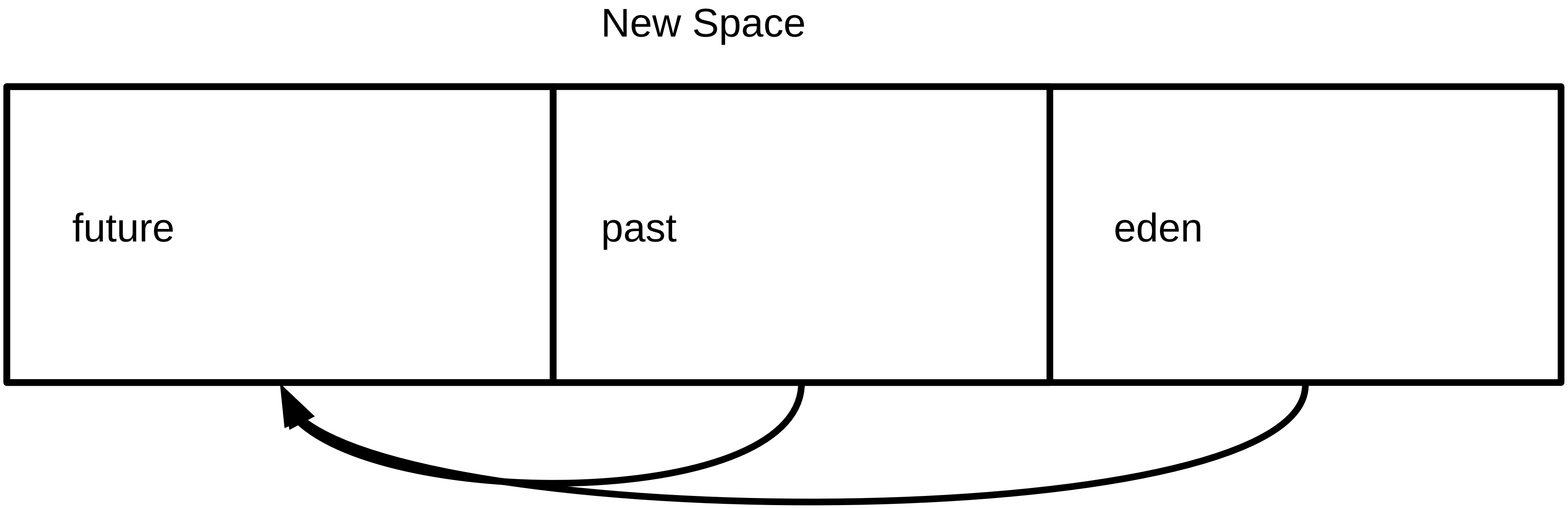
future

past

eden

1) Future is always empty during execution

3.5 Scavenger GC



New Space

future

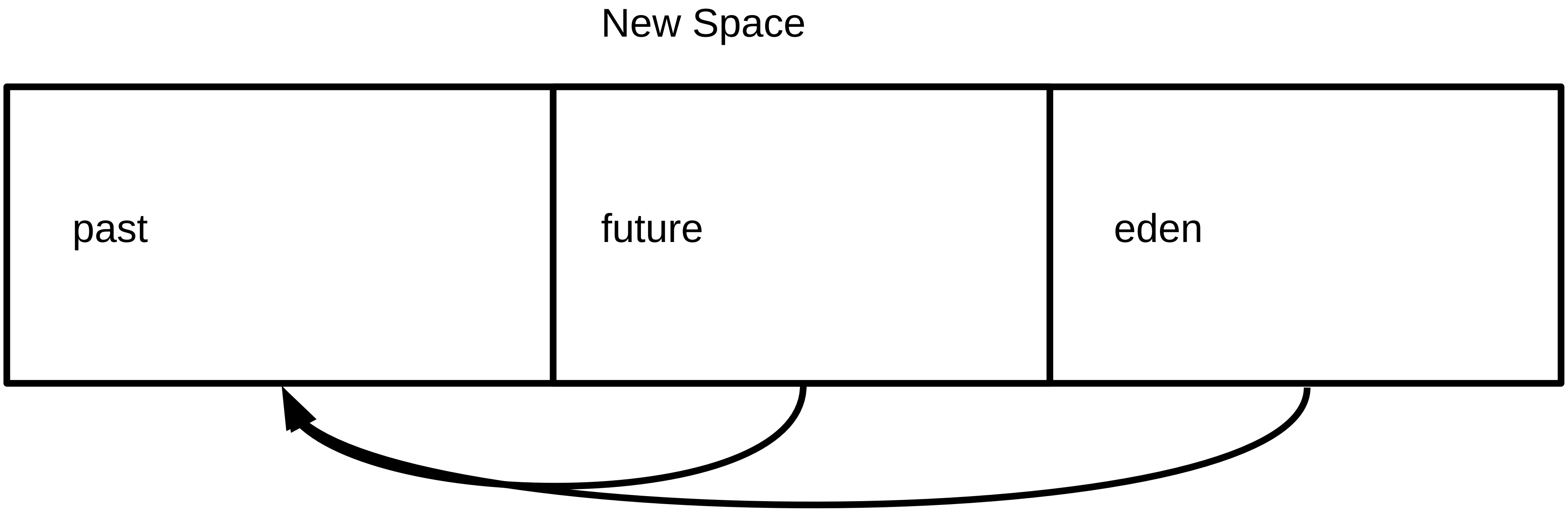
past

eden

1) Future is always empty during execution

2) On a GC, **past** and **eden** objects *that are referenced* are copied to **future**

3.5 Scavenger GC



1) Future is always empty during execution

2) On a GC, **past** and **eden** objects *that are referenced* are copied to **future**

3) Then, future and past spaces are **swapped**

3.5 Scavenger GC

Two questions remain:

- How does the scavenger do a GC without iterating the entire heap?
- How does he know object ages?

3.5 Scavenger GC

Two questions remain:

- How does the scavenger do a GC without iterating the entire heap?

It maintains a set of “objects in new space referenced from old space”

- How does he know object ages?

By their addresses! Lower addresses are younger....

Is that all?

- Pinned objects?
- The finalization queue?
- Memory segments, bridges, …?
- (The not working) Memory compaction?
- New immediate objects?
- ...