



Objects Layout in Memory

Virtual Machine presentations cycle

Carolina Hernández Phillips
PhD student in RMod Team

- Objects Layouts
 - Fixed, Indexable Pointer, Indexable Byte
- Object's Header
 - Format, number of slots, identity hash...



This presentation is about how objects are stored in memory

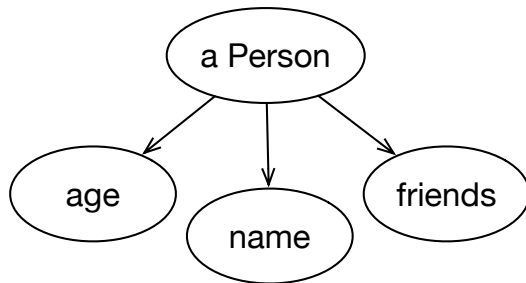
Motivations?

- Understand implementation design decisions
 - Modify the implementation
 - **Garbage Collection** algorithms
 - Custom images **bootstrapping**
 - Memory **optimizations**
- Develop custom **debugging** tools
- Implement ad-hoc **inspectors**



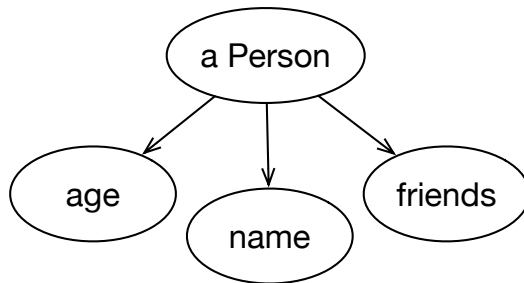
Example: Let's create a Person object

```
Object subclass: #Person  
  instanceVariableNames: 'age name friends'  
  classVariableNames: ''  
  package: '_UnpackagedPackage'|
```



Example: Let's create a Person object

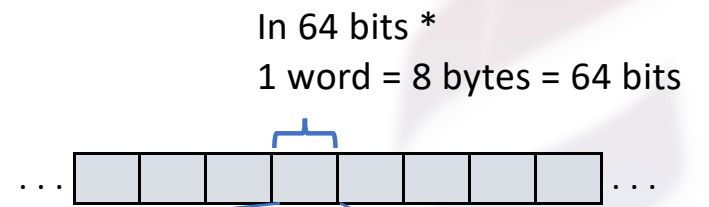
```
Object subclass: #Person
  instanceVariableNames: 'age name friends'
  classVariableNames: ''
  package: '_UnpackagedPackage'
```



```
Playground
Page
person := Person new
  age: 33;
  friends: {Person new. Person new};
  name: 'Caro';
  yourself.
```

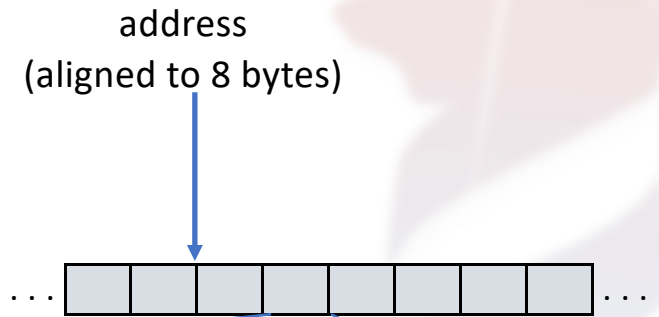
The memory:

1 0 0 1 1 0 1 0 0 1 1 1 0 1 0 0 1 1 1 0 0 1 1 0 1 0 0 1 1 1 0 1 0 0 1 1 1 0 0 1 1 0 1 0 0 1 1 1 0 0 1 1 0 1 0 0 1 1 1 0 1 0 0 1 1 1 0 0 1 1 0 1 1 0 1 1 0 1



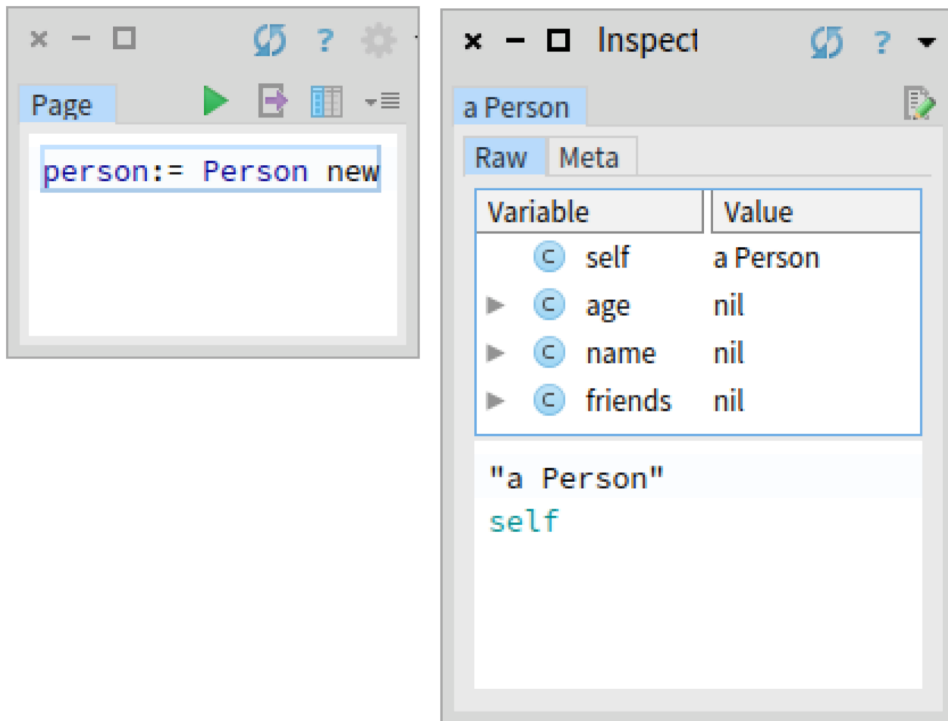
* In 32 bits
1 word = 4 bytes = 32 bits

The memory:



1 0 0 1 1 0 1 0 0 1 1 1 0 1 0 0 1 1 1 0 0 1 1 0 1 0 0 1 1 1 0 1 0 0 1 1 1 0 0 1 1 0 1 0 0 1 1 1 0 0 1 1 0 1 0 0 1 1 1 0 1 0 0 1 1 1 0 0 1 1 0 1 1 0 1

a Person object in memory



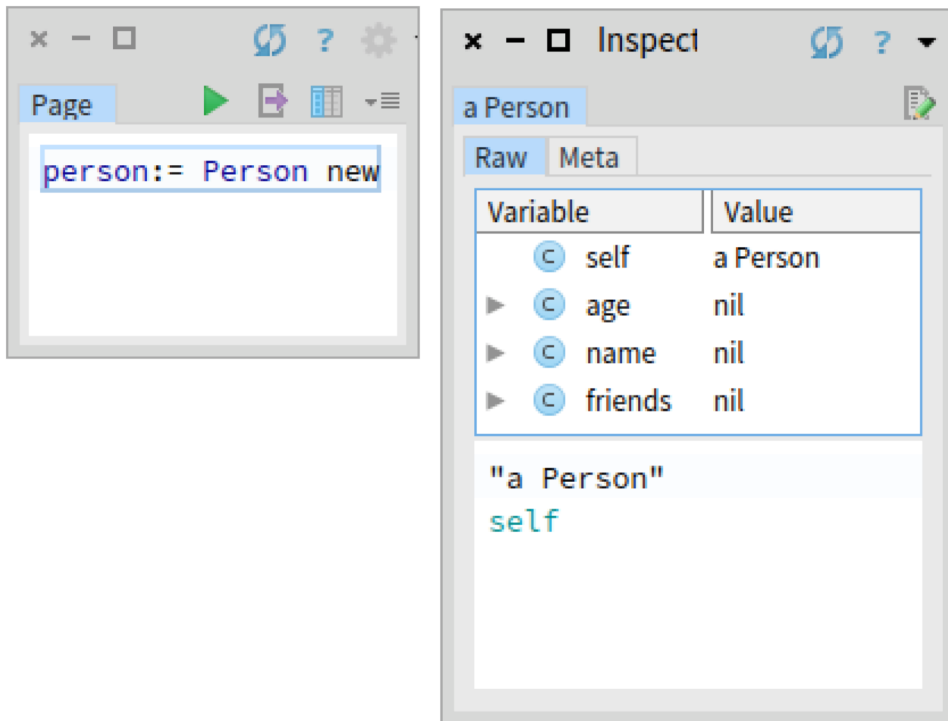
The image shows two windows from a development environment. The left window, titled "Page", contains the code `person := Person new`. The right window, titled "Inspect", shows the details of the object `a Person`. It has two tabs: "Raw" and "Meta". The "Raw" tab displays a table of variables and their values:

Variable	Value
self	a Person
age	nil
name	nil
friends	nil

Below the table, the "Meta" tab shows the object's class name, `"a Person"`, and the `self` variable.



a Person object in memory

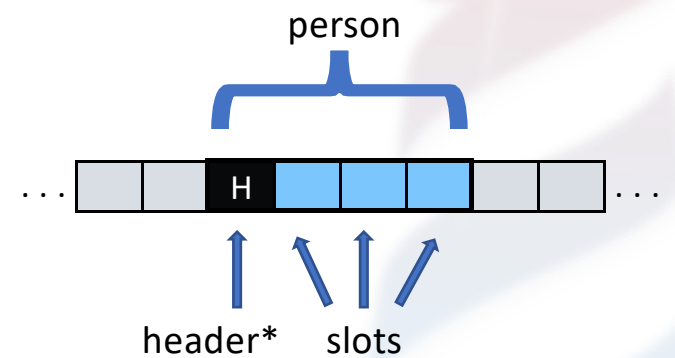


The image shows a REPL window on the left with the code `person := Person new` and an inspect window on the right. The inspect window shows the following table:

Variable	Value
self	a Person
age	nil
name	nil
friends	nil

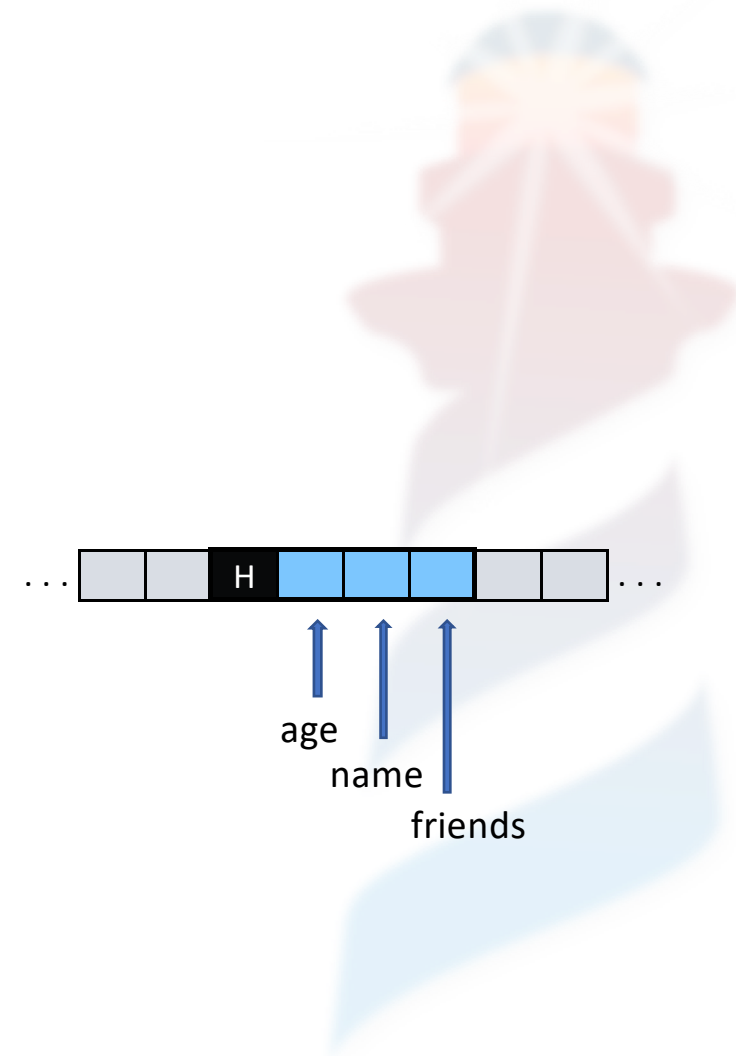
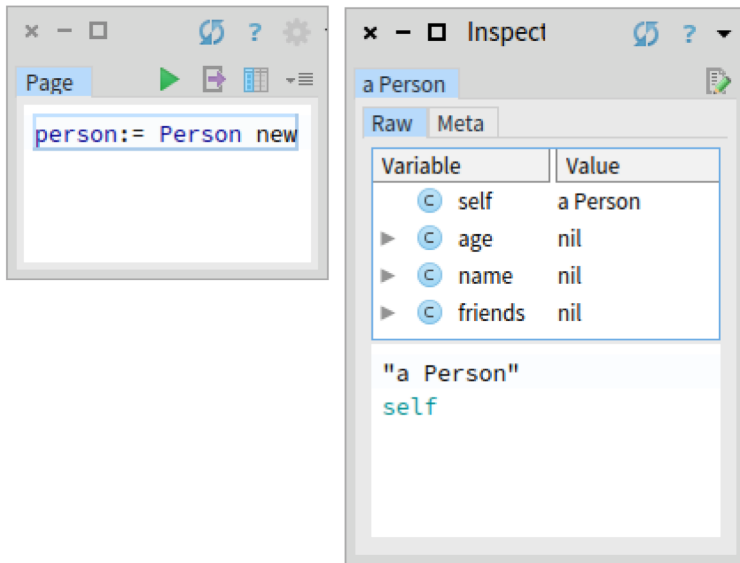
Below the table, the inspect window shows the object's class and its superclass:

```
"a Person"  
self
```

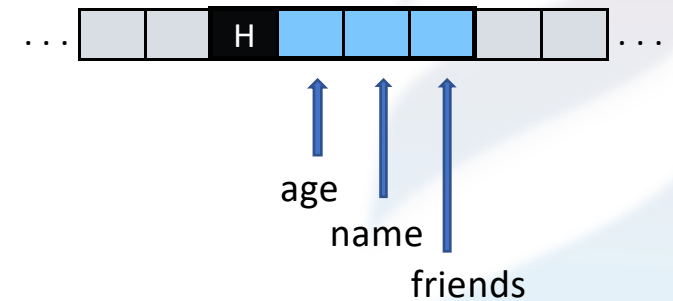
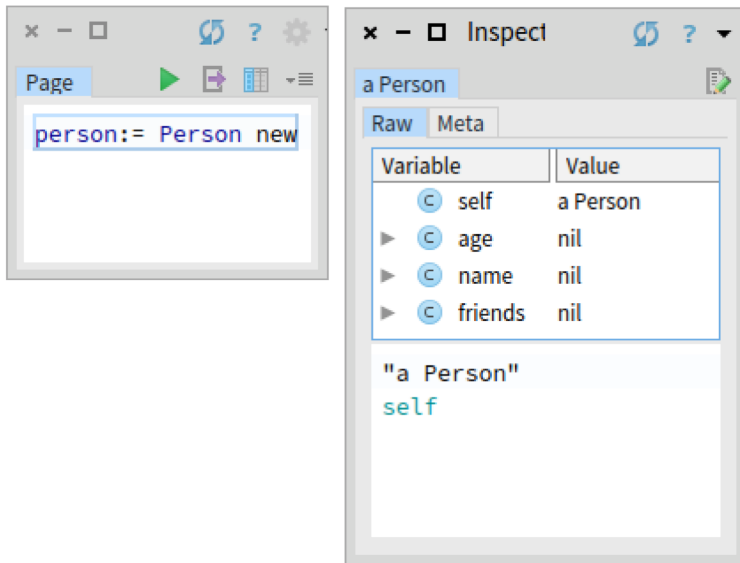


* In 32 bits
header size is 64 bits = 2 words

a Person object in memory



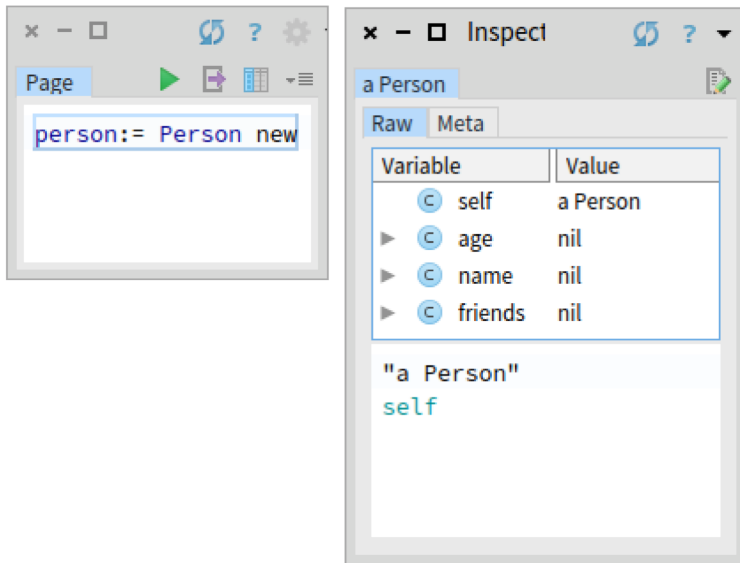
a Person object in memory: Fixed Layout



What is a Layout?

It is the structure of an object in memory

a Person object in memory: Fixed Layout

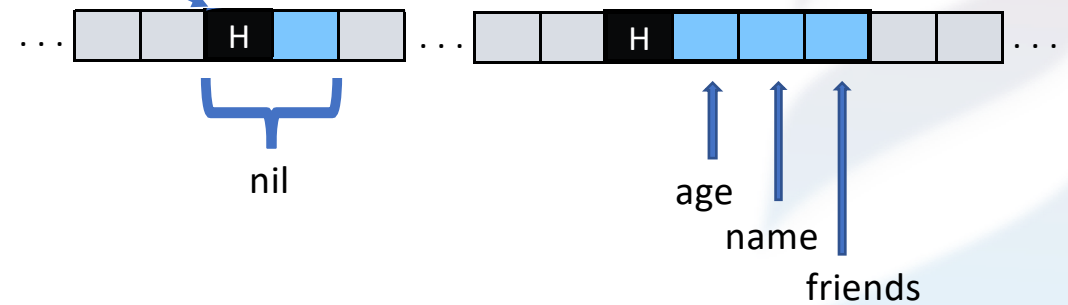


The image shows two windows from a development environment. The left window, titled 'Page', contains the code `person := Person new`. The right window, titled 'Inspect', shows the state of the object 'a Person'. It has a 'Raw' tab selected, displaying a table of variables and their values:

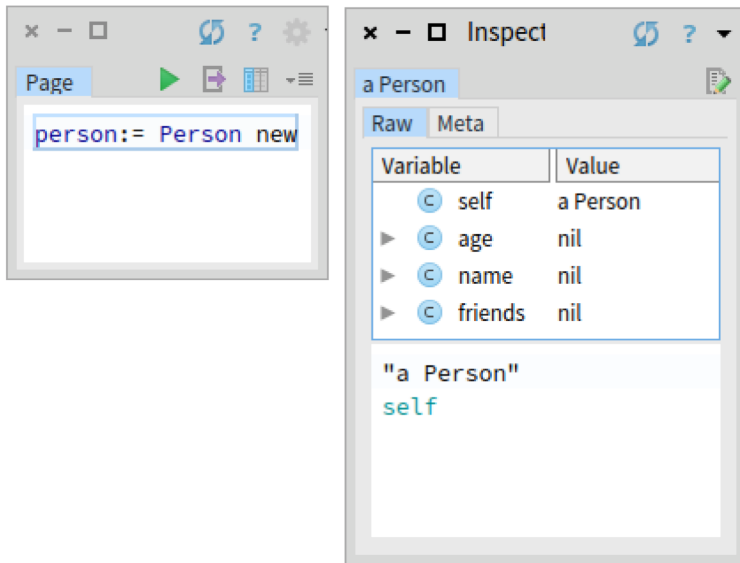
Variable	Value
self	a Person
age	nil
name	nil
friends	nil

Below the table, the object's class is shown as '"a Person"' and its superclass as 'self'.

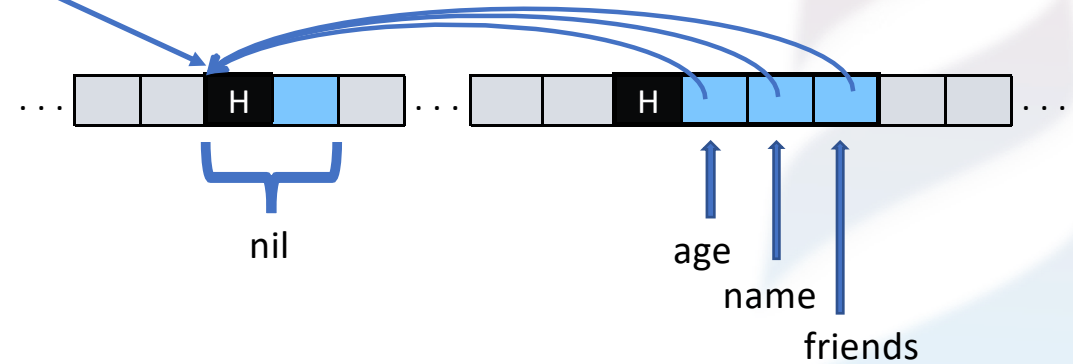
address = 24 = 2r11000



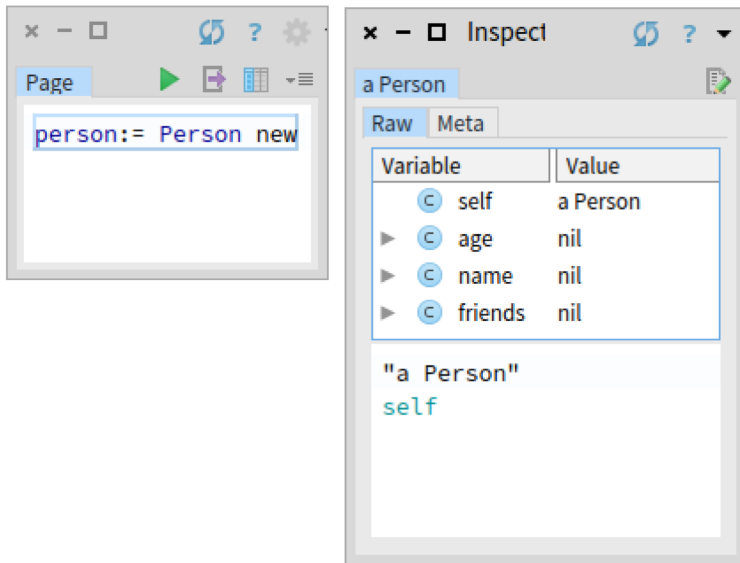
a Person object in memory: Fixed Layout



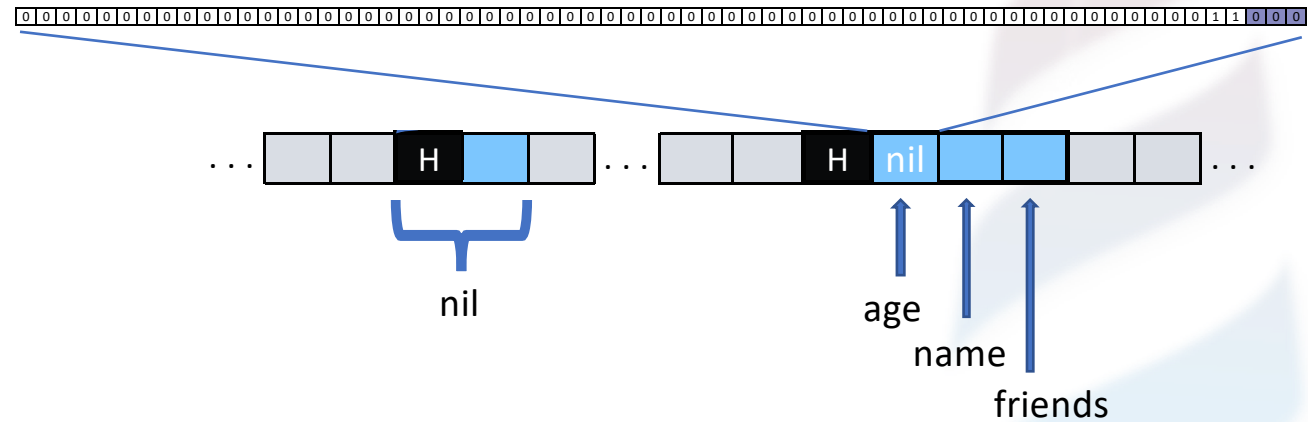
address = 24 = 2r11000



a Person object in memory: Fixed Layout

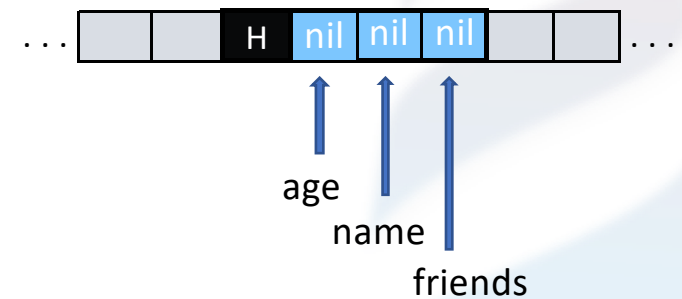


address = 24 = 2r11000

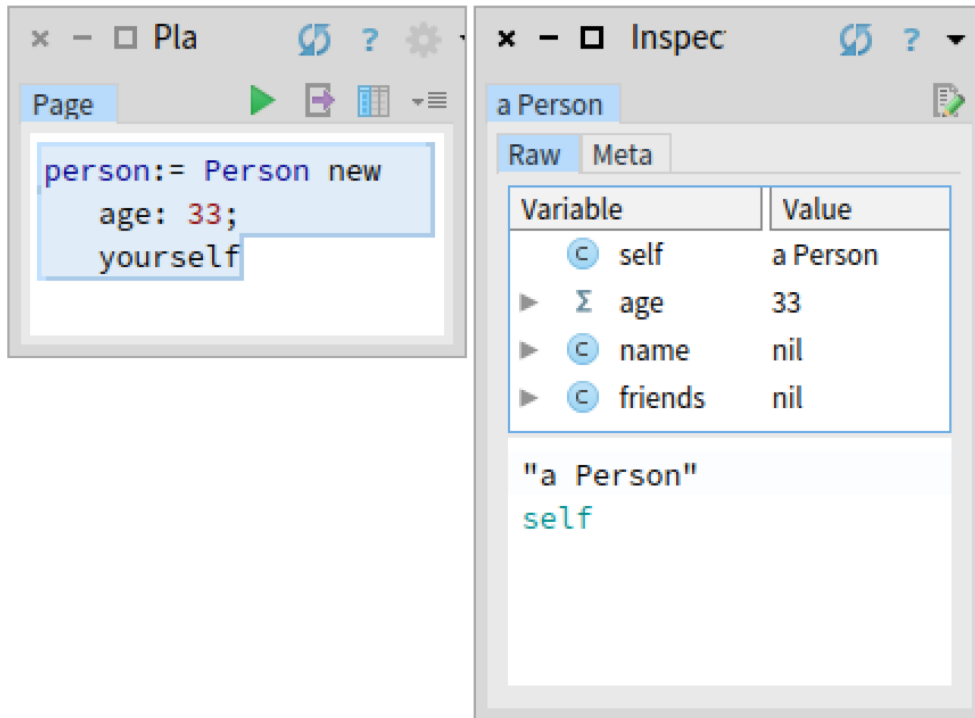


a Person object in memory: Fixed Layout

- Each slot is an **instance variable**
- Objects created sending the message **new**
- Number of slots is **constant** and defined by the class
- Each slot is initialized with a valid value: in this case with 'nil'
- Slots contain **Oop** (ordinary object pointer)
 - **references** (address of another object)
 - **values** (immediates)



Setting the age



The image shows two windows from a REPL environment. The left window, titled 'Pla', contains the following code:

```
person := Person new  
  age: 33;  
  yourself
```

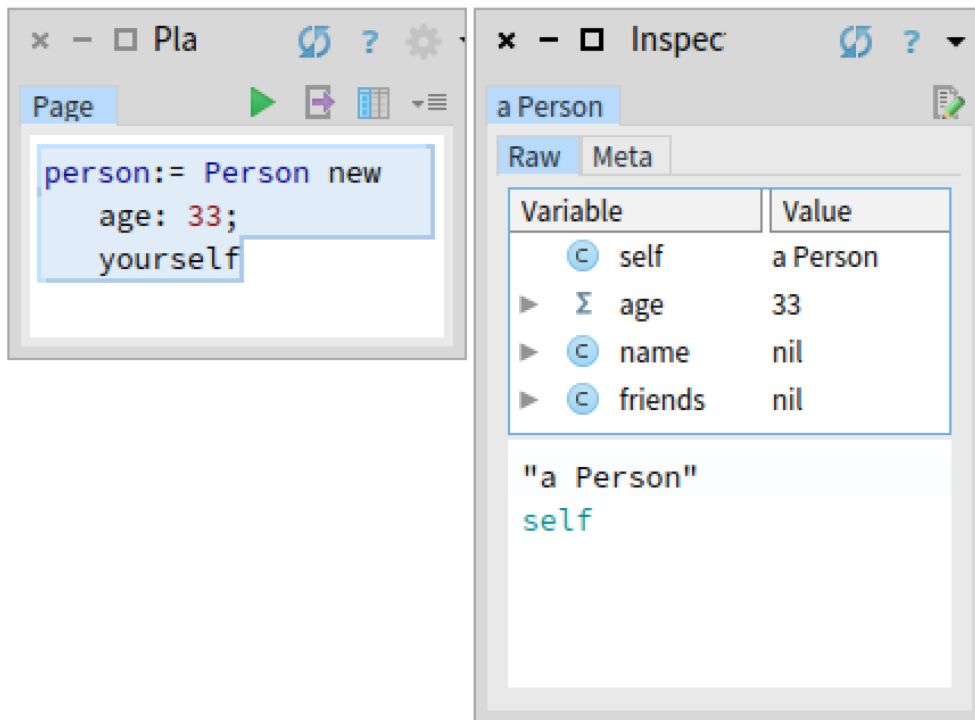
The right window, titled 'Inspect', shows the state of the object 'a Person'. It has a table with the following data:

Variable	Value
self	a Person
age	33
name	nil
friends	nil

Below the table, the text '"a Person"' and 'self' are displayed.



Setting the age



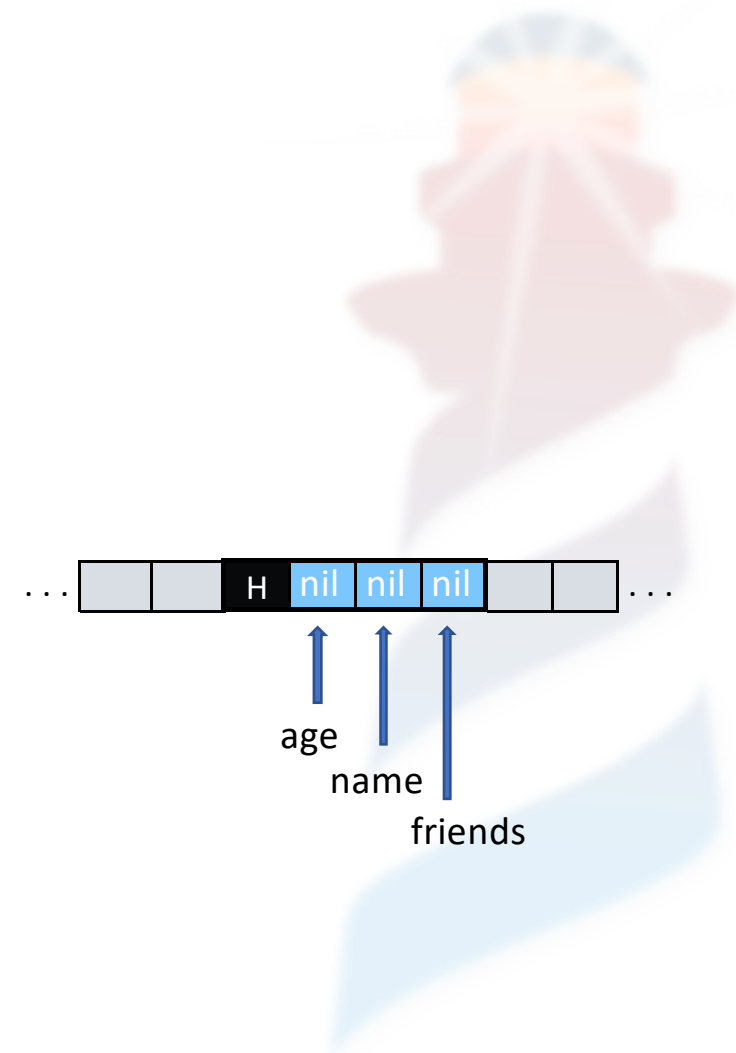
The image shows two windows from a REPL environment. The left window, titled 'Pla', contains the following code:

```
person := Person new
  age: 33;
  yourself
```

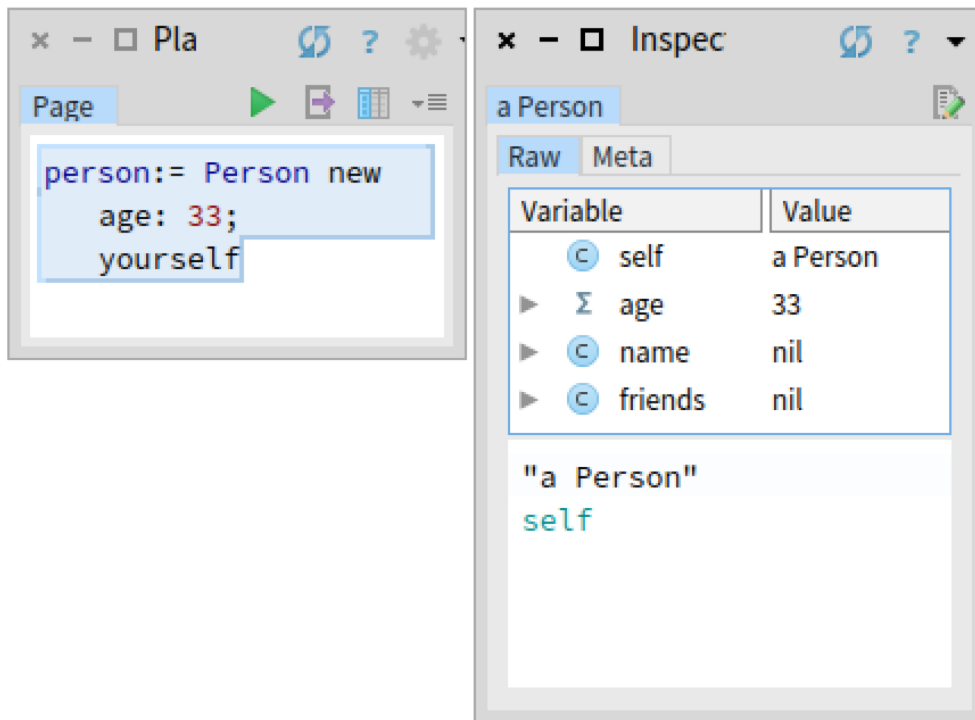
The right window, titled 'Inspect', shows the state of the object 'a Person'. It has two tabs: 'Raw' and 'Meta'. The 'Raw' tab is active and displays a table of variables and their values:

Variable	Value
self	a Person
age	33
name	nil
friends	nil

Below the table, the 'Meta' tab is visible, showing the string '"a Person"' and the variable 'self'.



Setting the age



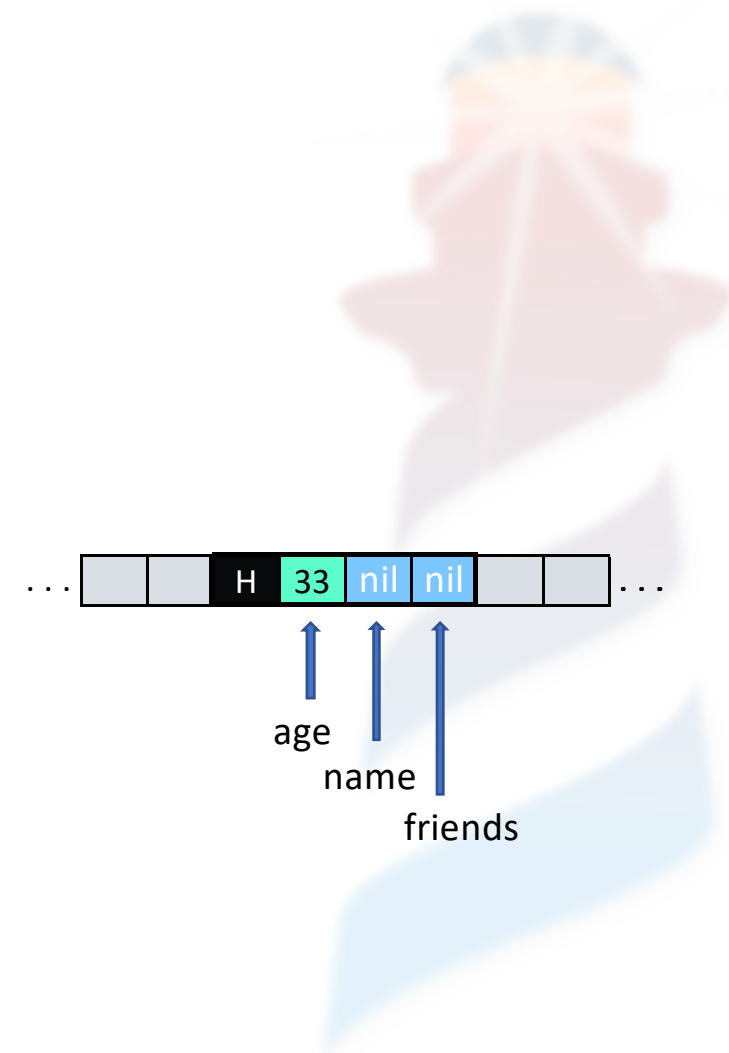
The image shows two windows from a REPL environment. The left window, titled 'Pla', contains the following code:

```
person := Person new  
  age: 33;  
  yourself
```

The right window, titled 'Inspect', shows the state of the object 'a Person' with the following table:

Variable	Value
self	a Person
age	33
name	nil
friends	nil

Below the table, the object's class is shown as 'a Person' and the object itself as 'self'.



Setting the friends

```
Playground
```

```
Page
```

```
ana := Person new.  
bob := Person new.  
friends := Array new:2
```

```
an Array [2 items] ...
```

Index	Item
1	nil
2	nil



Setting the friends

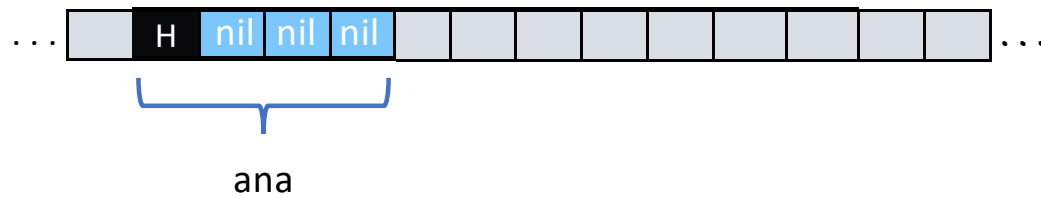
```
Playground
```

```
Page
```

```
ana := Person new.  
bob := Person new.  
friends := Array new:2
```

```
an Array [2 items] ...
```

Index	Item
1	nil
2	nil



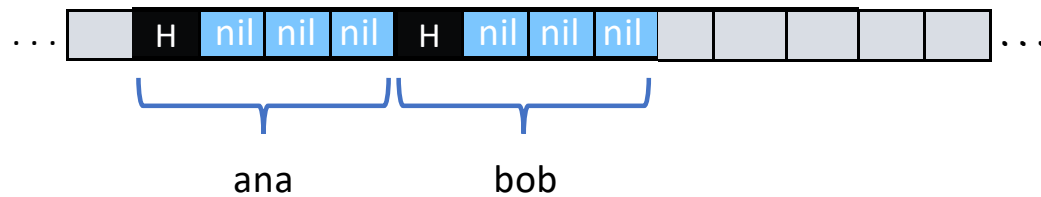
Setting the friends

```
Playground
```

```
Page
```

```
ana := Person new.  
bob := Person new.  
friends := Array new:2
```

Index	Item
1	nil
2	nil

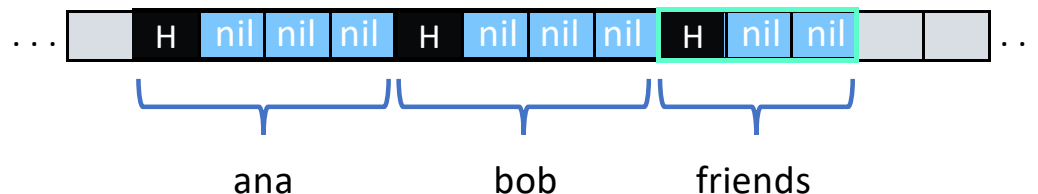


Setting the friends: Indexable Pointer layout

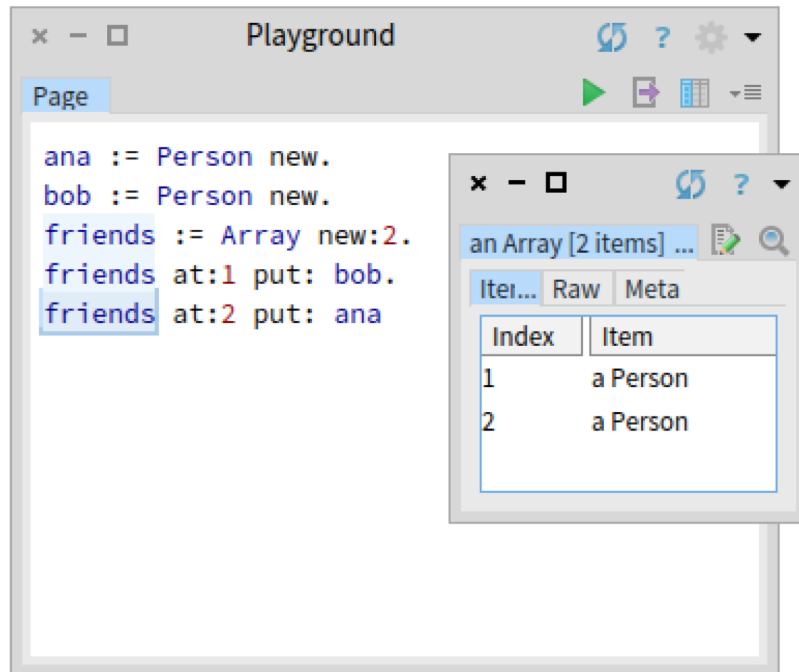
```
Playground  
Page  
ana := Person new.  
bob := Person new.  
friends := Array new:2
```

Index	Item
1	nil
2	nil

- **Variable number of slots**
 - Defined dynamically at the moment of **allocation**
- Object created by sending the message **new:**
- Objects **can't grow or shrink!**
- Size of slots is 1 word
- Slots contain **Oop**



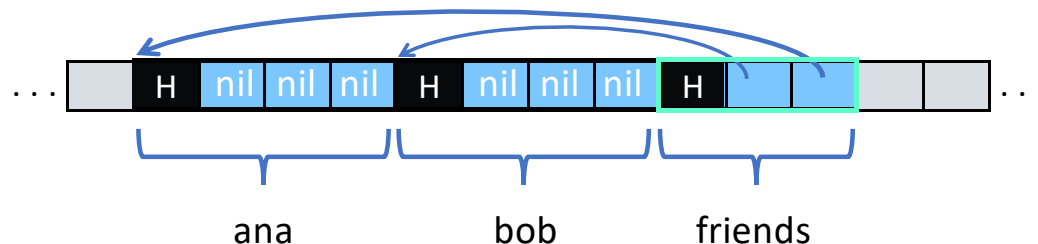
Setting the friends: Indexable Pointer layout



```
ana := Person new.  
bob := Person new.  
friends := Array new:2.  
friends at:1 put: bob.  
friends at:2 put: ana
```

Index	Item
1	a Person
2	a Person

- **Variable number of slots**
 - Defined dynamically at the moment of **allocation**
- Object created by sending the message **new:**
- Objects **can't grow or shrink!**
- Size of slots is 1 word
- Slots contain **Oop**



Setting the friends: Indexable Pointer layout

The Playground window contains the following code:

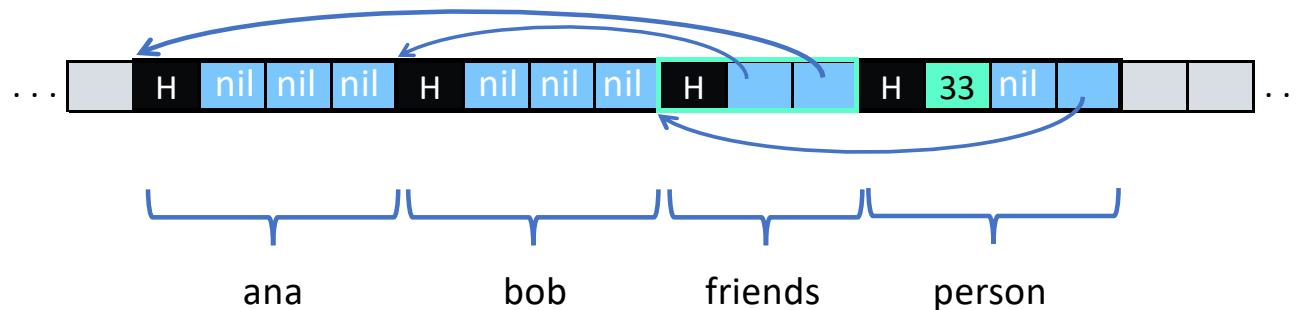
```
ana := Person new.  
bob := Person new.  
friends := Array new:2.  
friends at:1 put: bob.  
friends at:2 put: ana.  
  
person := Person new  
age: 33;  
friends: friends;  
yourself
```

The Inspector window shows the following details for 'a Person':

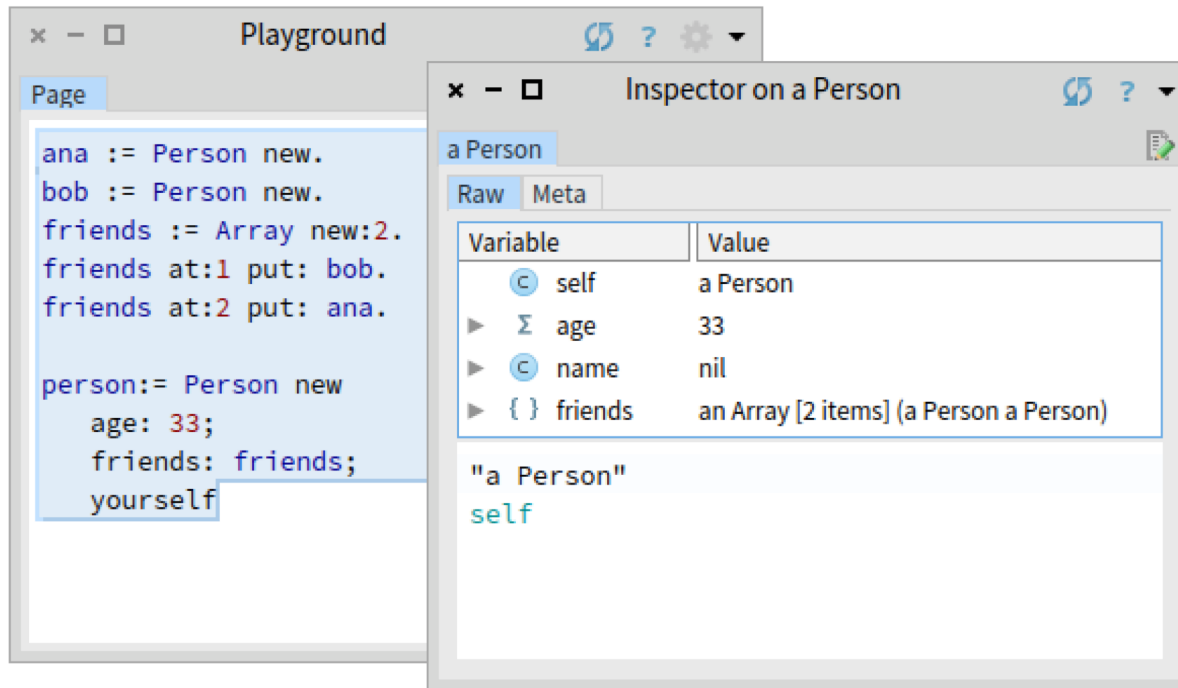
Variable	Value
self	a Person
age	33
name	nil
friends	an Array [2 items]

Below the table, the string representation is shown as: "a Person" self

- **Variable number of slots**
 - Defined dynamically at the moment of **allocation**
- Object created by sending the message **new:**
- Objects **can't grow or shrink!**
- Size of slots is 1 word
- Slots contain **Oop**



Setting the name



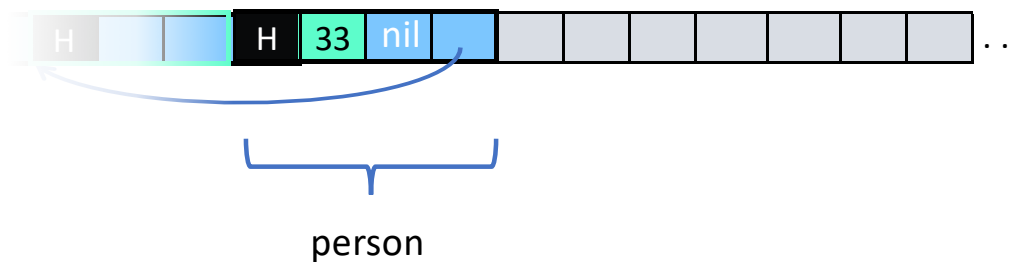
The screenshot shows a REPL window titled "Playground" with the following code:

```
ana := Person new.  
bob := Person new.  
friends := Array new:2.  
friends at:1 put: bob.  
friends at:2 put: ana.  
  
person:= Person new  
  age: 33;  
  friends: friends;  
  yourself
```

Next to it is an "Inspector on a Person" window showing the state of the object "a Person":

Variable	Value
self	a Person
age	33
name	nil
friends	an Array [2 items] (a Person a Person)

Below the inspector, the string "a Person" is shown, followed by the word "self".



Setting the name

Playground

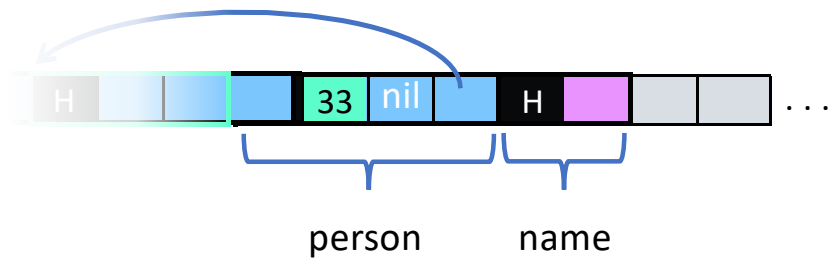
```
Page  
friends at:2 put: ana.  
  
person:= Person new  
  age: 33;  
  friends: friends;  
  yourself.  
  
name := 'Caro'
```

Inspector

a ByteString ('Caro')

Variable	Value
self	'Caro'
1	67
2	97
3	114
4	111

'''Caro''
self



Setting the name: Indexable Byte layout

Playground

```

Page
friends at: put: ana.

person:= Person new
  age: 33;
  friends: friends;
  yourself.

name := 'Caro'
        
```

Inspector

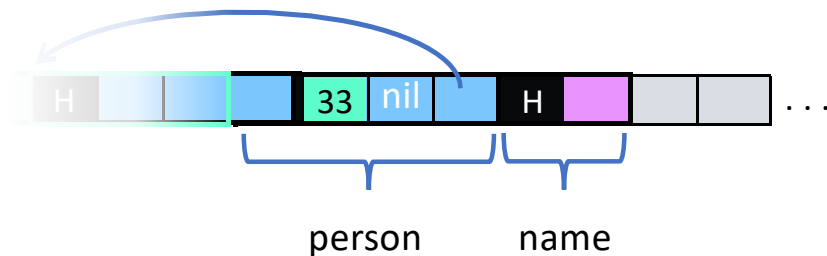
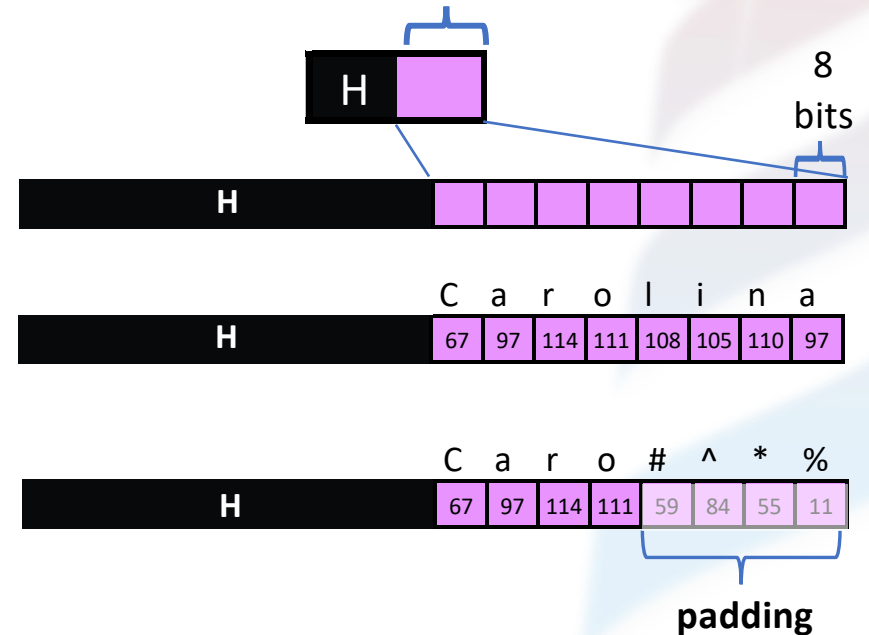
a ByteString ('Caro')

Variable	Value
self	'Caro'
Σ 1	67
Σ 2	97
Σ 3	114
Σ 4	111

""Caro""
self

ByteString (8-bit)
1 slot = 8-bits

1 word = 8 bytes = 64 bits



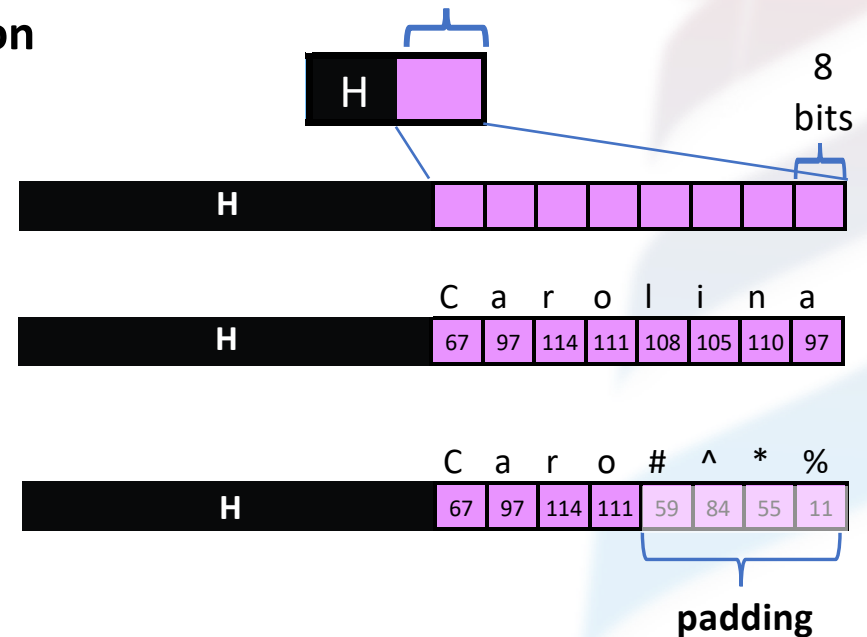
Setting the name: Indexable Byte layout

- **Variable number of slots**
 - Slots **do not contain Oop**, they contain **raw bytes**
 - Defined dynamically at the moment of **allocation**
 - Created by sending the message **new**:
 - Objects **can't grow or shrink!**
- **Size of slots depends on the object's layout**
 - 8-bit (eg: ByteArray)
 - 16 bit
 - 32-bit (eg: FloatArray, IntegerArray)
 - 64-bit

ByteString (8-bit)

1 slot = 8-bits

1 word = 8 bytes = 64 bits



Finally! A Person in memory

```
Object subclass: #Person  
  instanceVariableNames: 'age name friends'  
  classVariableNames: ''
```

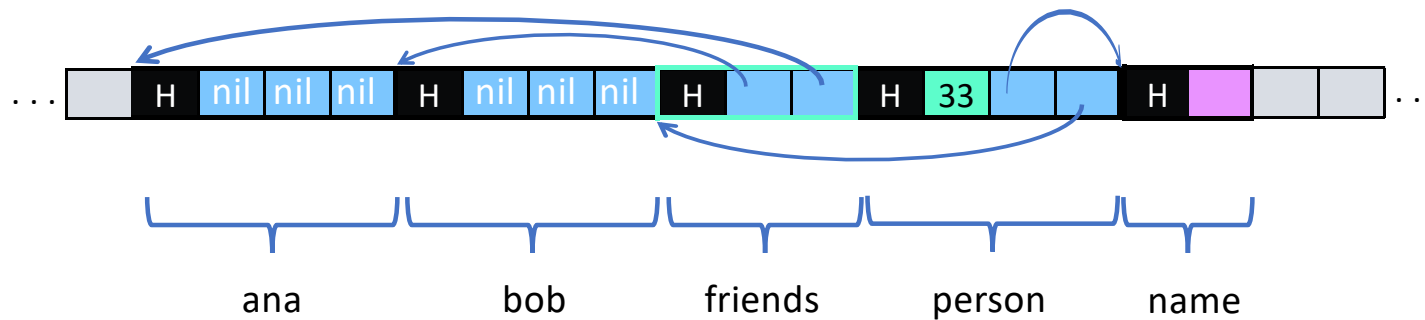
The screenshot shows a REPL window titled "Playground" with the following code:

```
ana := Person new.  
bob := Person new.  
friends := Array new: 2.  
friends at:1 put: bob.  
friends at:2 put: ana.  
  
person := Person new  
  age: 33;  
  friends: friends;  
  yourself.  
  
name := 'Caro'.  
person name: name.  
person
```

Below the code is an "Inspector on a Person" window showing the state of the object:

Variable	Value
self	a Person
name	'Caro'
age	33
friends	an Array [2 items] (a Person a Person)
self	an Array [2 items] (a Person a Person)
1	a Person
2	a Person

Below the inspector is a text area containing the string "a Person" and the word "self" with a cursor.



Special layout: Indexable with Instance Variables layout

Examples:

- Context
- CompiledMethod
- **MethodDictionary**

The screenshot shows the Ruby Inspector interface with three panes:

- Left Pane:** Shows the instance variables of a `MethodDictionary` object.

Variable	Value
{ } self	a MethodDictionary [3 items]
#friends:	
1	nil
2	nil
3	nil
4	nil
5	nil
6	nil
7	nil
8	nil
9	nil
#age:	
11	nil
12	nil
13	nil
14	nil
15	nil
16	nil
17	nil
18	nil
19	nil
20	nil
21	nil
22	nil
23	nil
24	nil
25	nil
26	nil
#name:	
28	nil
29	nil
30	nil
31	nil
32	nil
tally	3
array	an Array [32 items] (Person>>#friends:)
- Middle Pane:** Shows the instance variables of an `Array` object.

Variable	Value
{ } self	an Array [32 items] (Person>>#friends:)
1	Person>>#friends:
2	nil
3	nil
4	nil
5	nil
6	nil
7	nil
8	nil
9	nil
{ } 10	Person>>#age:
11	nil
12	nil
13	nil
14	nil
15	nil
16	nil
17	nil
18	nil
19	nil
20	nil
21	nil
22	nil
23	nil
24	nil
25	nil
26	nil
{ } 27	Person>>#name:
28	nil
29	nil
30	nil
31	nil
32	nil
- Right Pane:** Shows the compiled method `Person>>#friends:`.

Variable	Value
{ } self	Person>>#friends:
bc 25	64
bc 26	202
bc 27	88
bc 28	29
bc 29	7
bc 30	165
bc 31	253

The bottom of the inspector shows the source code for the `MethodDictionary` object:

```
"a
MethodDictionary(#age:->Person>>#age:
#friends:->Person>>#friends:
#name:->Person>>#name:)"
```

Indexable with Instance Variables layout

Example: MethodDictionary



Inspector on a MethodDictionary [3 items] (#age:->Person>>#age: #friends:->Person>>#friends: #name:)

Variable	Value
{ } self	a MethodDictionary [3 items]
#friends:	
1	nil
2	nil
3	nil
4	nil
5	nil
6	nil
7	nil
8	nil
9	nil
#age:	
10	nil
11	nil
12	nil
13	nil
14	nil
15	nil
16	nil
17	nil
18	nil
19	nil
20	nil
21	nil
22	nil
23	nil
24	nil
25	nil
26	nil
#name:	
27	nil
28	nil
29	nil
30	nil
31	nil
32	nil
tally	3
array	an Array [32 items] (Person>>#friends:)

```

a
MethodDictionary(#age:->Person>>#age:
#friends:->Person>>#friends:
#name:->Person>>#name: )
...
  
```

an Array [32 items] (Person>>#friends:)

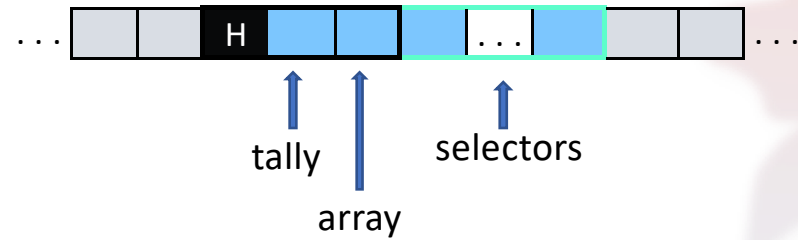
Variable	Value
{ } self	Person>>#friends:
1	Person>>#friends:
2	nil
3	nil
4	nil
5	nil
6	nil
7	nil
8	nil
9	nil
{ } 10	Person>>#age:
11	nil
12	nil
13	nil
14	nil
15	nil
16	nil
17	nil
18	nil
19	nil
20	nil
21	nil
22	nil
23	nil
24	nil
25	nil
26	nil
{ } 27	Person>>#name:
28	nil
29	nil
30	nil
31	nil
32	nil

```

"Person>>#friends:"
self
  
```

Indexable with Instance Variables layout

Example: MethodDictionary



Inspector on a MethodDictionary [3 items] (#age:->Person>>#age: #friends:->Person>>#friends: #name:)

a MethodDictionary [3 items] (#age:->Pers...)

Variable	Value
{ } self	a MethodDictionary [3 iter...
#friends:	
1	nil
2	nil
3	nil
4	nil
5	nil
6	nil
7	nil
8	nil
9	nil
#age:	
10	nil
11	nil
12	nil
13	nil
14	nil
15	nil
16	nil
17	nil
18	nil
19	nil
20	nil
21	nil
22	nil
23	nil
24	nil
25	nil
26	nil
#name:	
27	nil
28	nil
29	nil
30	nil
31	nil
32	nil
tally	3
array	an Array [32 items] (Perso...

an Array [32 items] (Person>>#friends:...

Variable	Value
{ } self	an Array [32 items] (Perso...
1	Person>>#friends:
2	nil
3	nil
4	nil
5	nil
6	nil
7	nil
8	nil
9	nil
{ } 10	Person>>#age:
11	nil
12	nil
13	nil
14	nil
15	nil
16	nil
17	nil
18	nil
19	nil
20	nil
21	nil
22	nil
23	nil
24	nil
25	nil
26	nil
{ } 27	Person>>#name:
28	nil
29	nil
30	nil
31	nil
32	nil

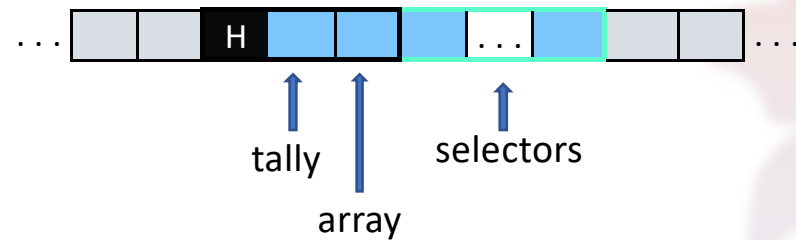
a CompiledMethod (Person>>#frie...)

Variable	Value
{ } self	Person>>#friends:
bc 25	64
bc 26	202
bc 27	88
bc 28	29
bc 29	7
bc 30	165
bc 31	253

```
"Person>>#friends:"
self
```

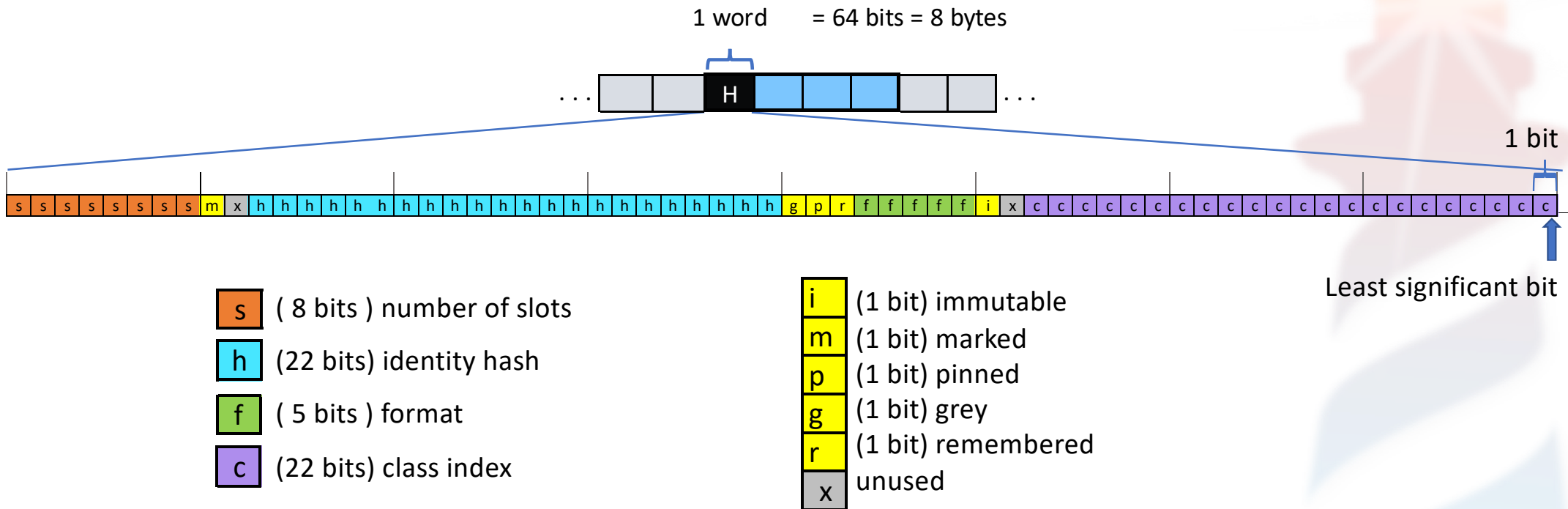
Indexable with Instance Variables layout

Example: MethodDictionary



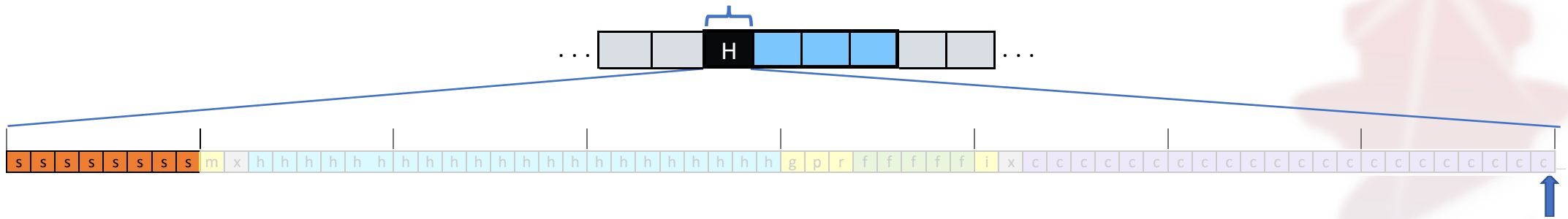
- **Hybrid** between Fixed and Indexable layout
 - A **fixed number of slots** containing instance variables
 - In the example: tally and array
 - Plus a **variable number of slots**
 - In the example: selectors

The Header

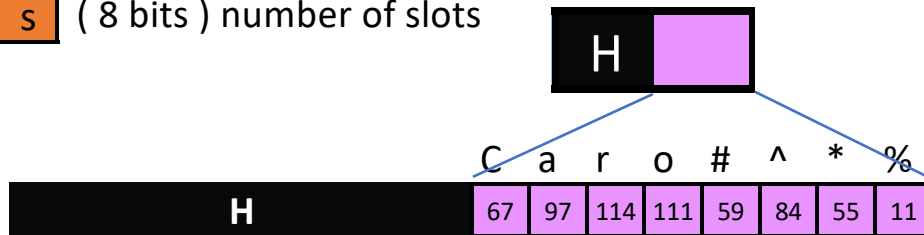


The Header: number of Slots (number of words)

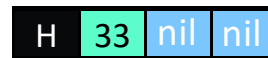
1 word = 64 bits = 8 bytes



s (8 bits) number of slots



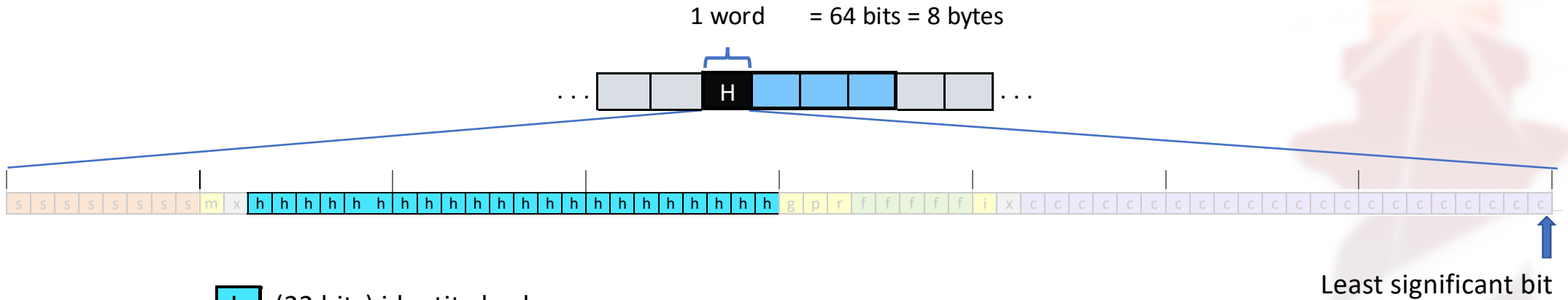
-> 1 word



-> 3 words

- $0 \leq \text{number of slots} < 255$
- When number of slots is 255, the object is a Big Object

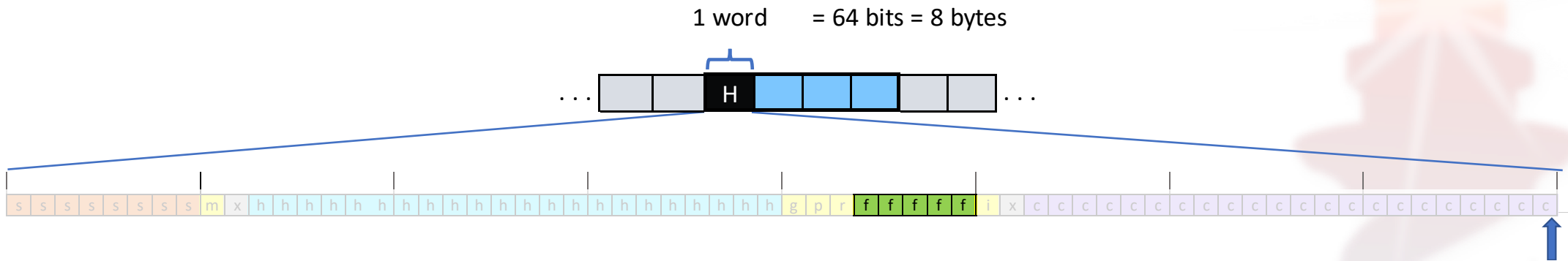
The Header: identity hash



h (22 bits) identity hash

- **Unique identifier** of the object
- Check ProtoObject >> basicIdentityHash

The Header: format



f (5 bits) format

- Stores the layout of the object
 - 0 = 0 sized objects (UndefinedObject True False et al)
 - 1 = non-indexable objects with inst vars (Point et al)
 - 2 = indexable objects with no inst vars (Array et al)
 - 3 = indexable objects with inst vars (MethodContext AdditionalMethodState et al)
 - 4 = weak indexable objects with inst vars (WeakArray et al)
 - 5 = weak non-indexable objects with inst vars (ephemerons) (Ephemeron)
 - 6 = unused
 - 7 = immediates (SmallInteger, Character)
 - 8 = unused
 - 9 = 64-bit indexable
 - 10-11 = 32-bit indexable (Bitmap)
 - 12-15 = 16-bit indexable
 - 16-23 = 8-bit indexable
 - 24-31 = compiled methods (CompiledMethod)

Class table

Organized in pages

tag	class
1048	*SmalltalkImage*
1049	*Magnitude*
1052	*Association*
1053	*MethodDictionary*
1056	*ProcessorScheduler*
1057	*System*

tag	class
1026	*CompiledMethod*
1027	*String*
1028	*Process*
1029	*Integer*
1031	*Number*
1032	
1033	
1035	
1036	
1041	
1042	
1043	
1044	
1046	

tag	class
1	*SmallInteger*
1	*SmallInteger*
2	*Character*
32	*LargeNegativeInteger*
33	*LargePositiveInteger*
34	*Float*
35	*Message*
36	*Context*
37	*BlockClosure*
50	*ByteArray*
51	*Array*
51	*Array*
52	*ByteString*
54	*Point*



Big objects

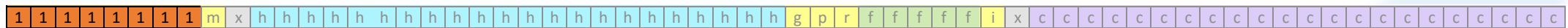
- Objects with more than 254 slots

Example: Object with 260 instance variables

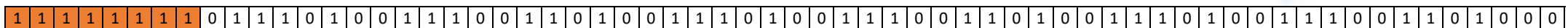
- 2 Headers



- **H**: Normal header's number of slots set as 255 (8 bits set as 1)



- **260**: Extra header starting with 8 bits as 1, and saving the Object's number of slots



Objects size



- Every object must have at least one slot
 - Minimum size is 16 bytes
 - It is hidden from the image side (smalltalk code)
 - In case it becomes into a Forwarder
- Even nil, true and false have one slot!



Objects Layout in Memory

VM presentations

- Layouts:
 - Fixed
 - Immediate
 - Indexable Pointer
 - Indexable Byte
 - Indexable with Instance Variables
- Object
 - Header
 - Class Table
 - Big Objects

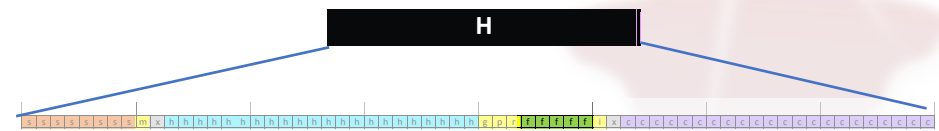
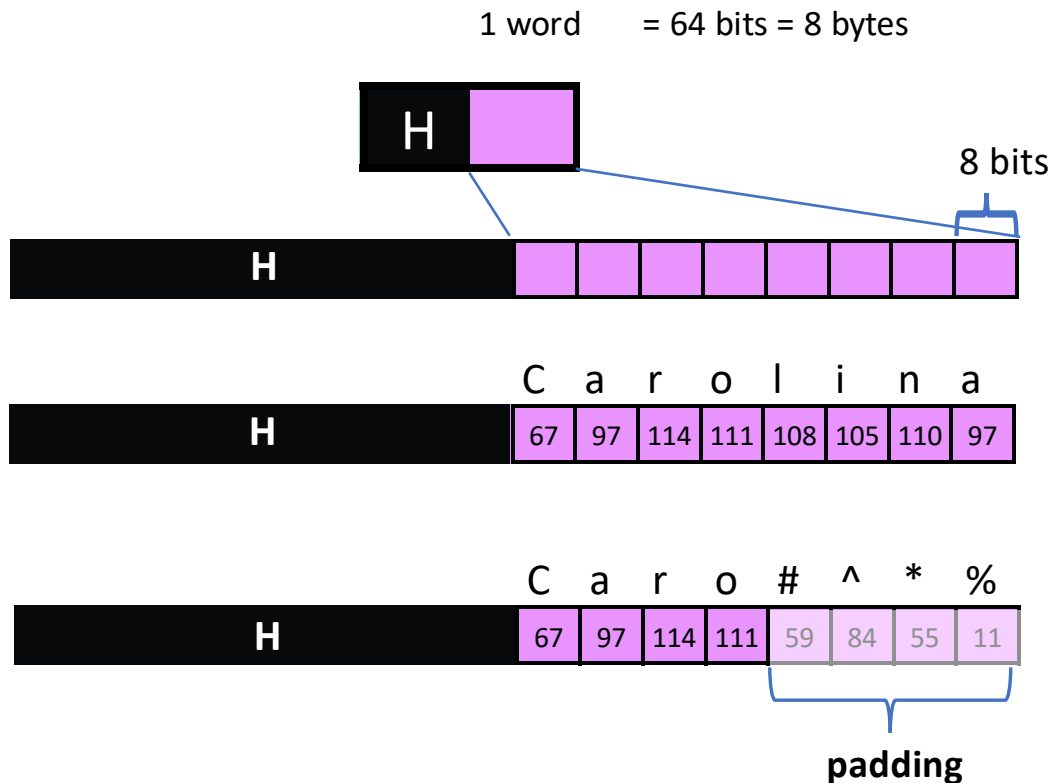


Carolina Hernández Phillips

RMod Team

More about the padding in: Indexable Byte layout

Example: ByteString => 8 bit indexable opaque layout



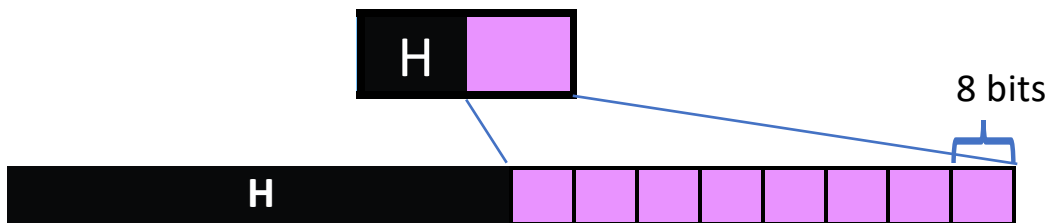
16-23 Object **format** 8 bits indexable opaque layout

- 16 -> padding **0-bits**
- 17 -> padding **8-bits**
- 18 -> padding **16-bits**
- 19 -> padding **24-bits**
- 20 -> padding **32-bits**
- 21 -> padding **40-bits**
- 22 -> padding **48-bits**
- 23 -> padding **56-bits**

More about the padding in: Indexable Byte layout

Example: ByteString => 8 bit indexable opaque layout

1 word = 64 bits = 8 bytes



C a r o l i n a

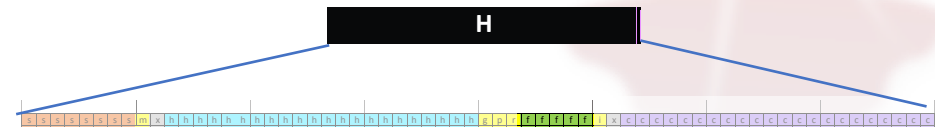


Format = 16
(padding = 0 bits)

C a r o # ^ * %



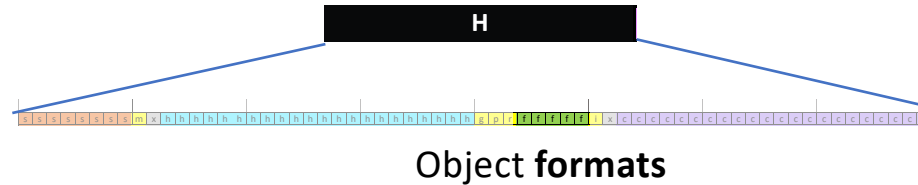
Format = 20
(padding = 32 bits)



16-23 Object format 8 bits indexable opaque layout

- 16 -> padding **0-bits**
- 17 -> padding **8-bits**
- 18 -> padding **16-bits**
- 19 -> padding **24-bits**
- 20 -> padding **32-bits**
- 21 -> padding **40-bits**
- 22 -> padding **48-bits**
- 23 -> padding **56-bits**

More about the padding in: Indexable Byte layout



9 => 64 bits layout

- 9 -> padding **0-bits**

10-11 => 32 bits layout

- 10 -> padding **0-bits**
- 11 -> padding **32-bits**

12-15 => 16 bits layout

- 12 -> padding **0-bits**
- 13 -> padding **16-bits**
- 14 -> padding **32-bits**
- 15 -> padding **48-bits**

16-23 => 8 bits layout

- 16 -> padding **0-bits**
- 17 -> padding **8-bits**
- 18 -> padding **16-bits**
- 19 -> padding **24-bits**
- 20 -> padding **32-bits**
- 21 -> padding **40-bits**
- 22 -> padding **48-bits**
- 23 -> padding **56-bits**