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?

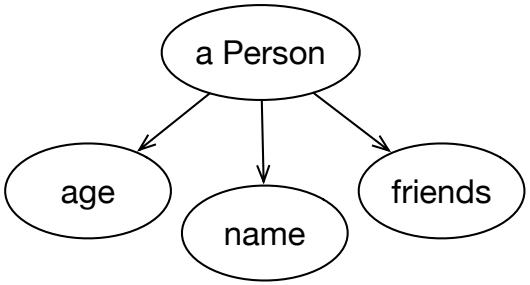
- Understand implementation design decisions
  - Modify the implementation
    - **Garbage Collection** algorithms
    - Custom images **bootstrapping**
    - Memory **optimizations**

- Develop custom **debugging** tools

- Implement ad-hoc **inspectors**

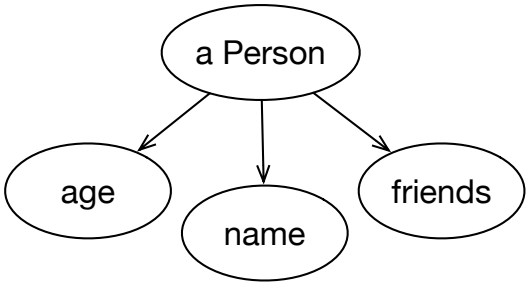
Example: Let's create a **Person object**

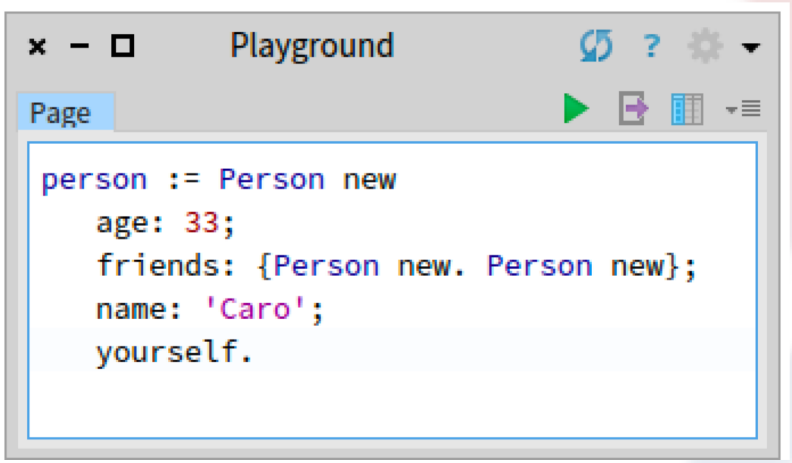
```
Object subclass: #Person
   instanceVariableNames: 'age name friends'
   classVariableNames: ''
   package: '_UnpackagedPackage'
```



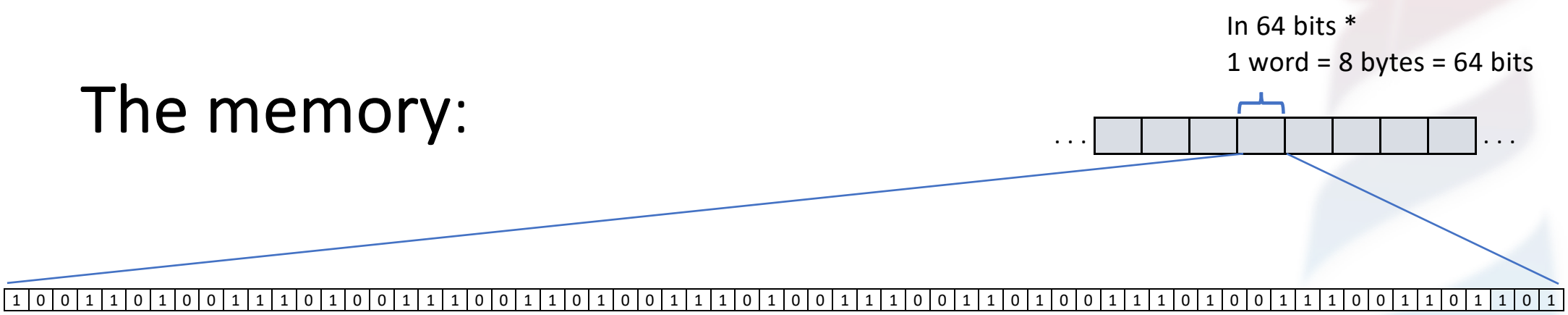
Example: Let's create a **Person object**

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



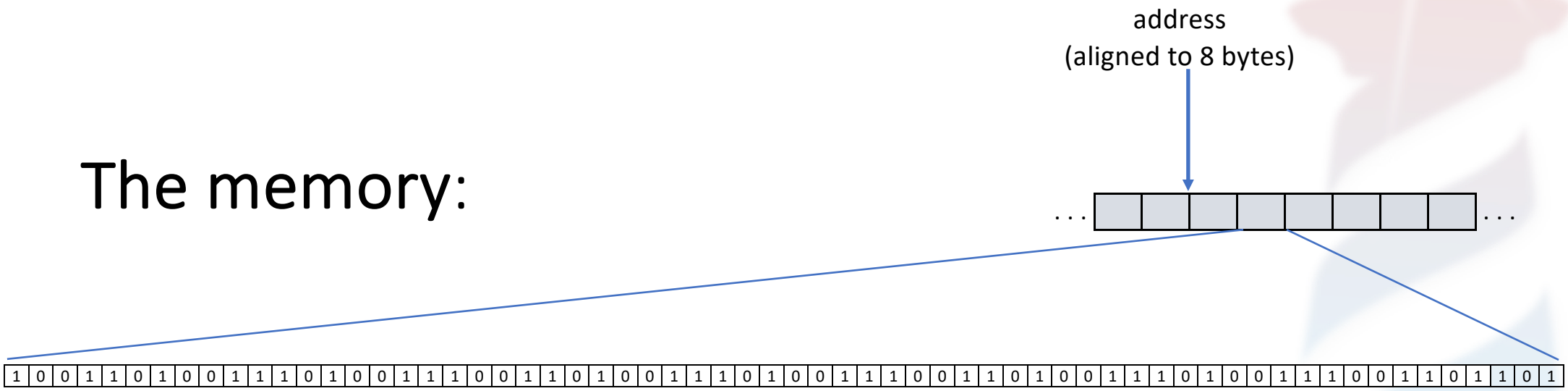


The memory:

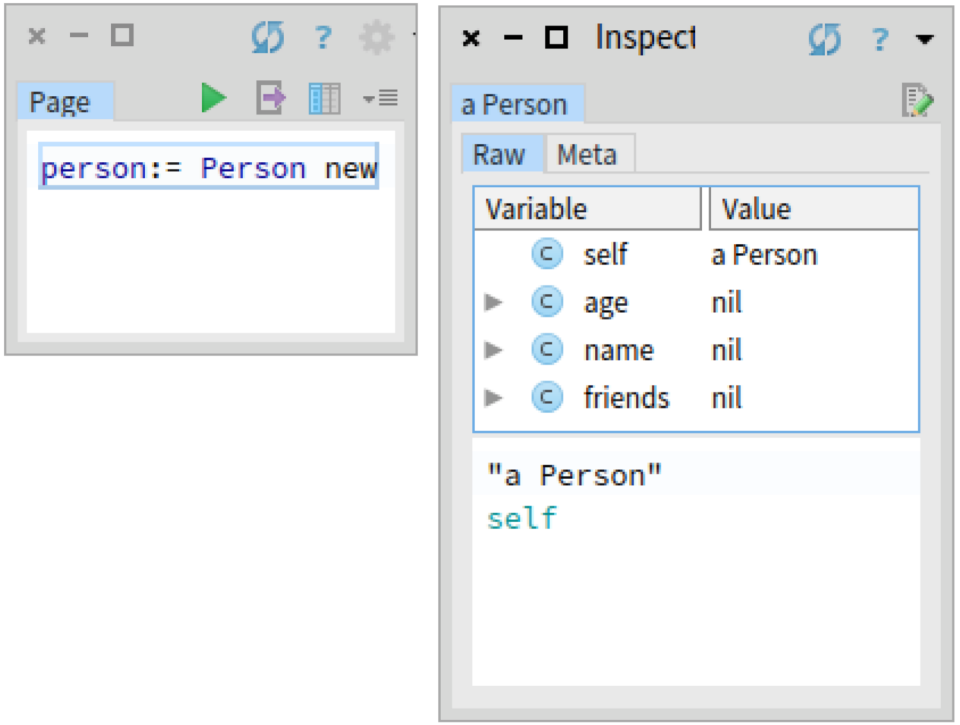


* In 32 bits
1 word = 4 bytes = 32 bits

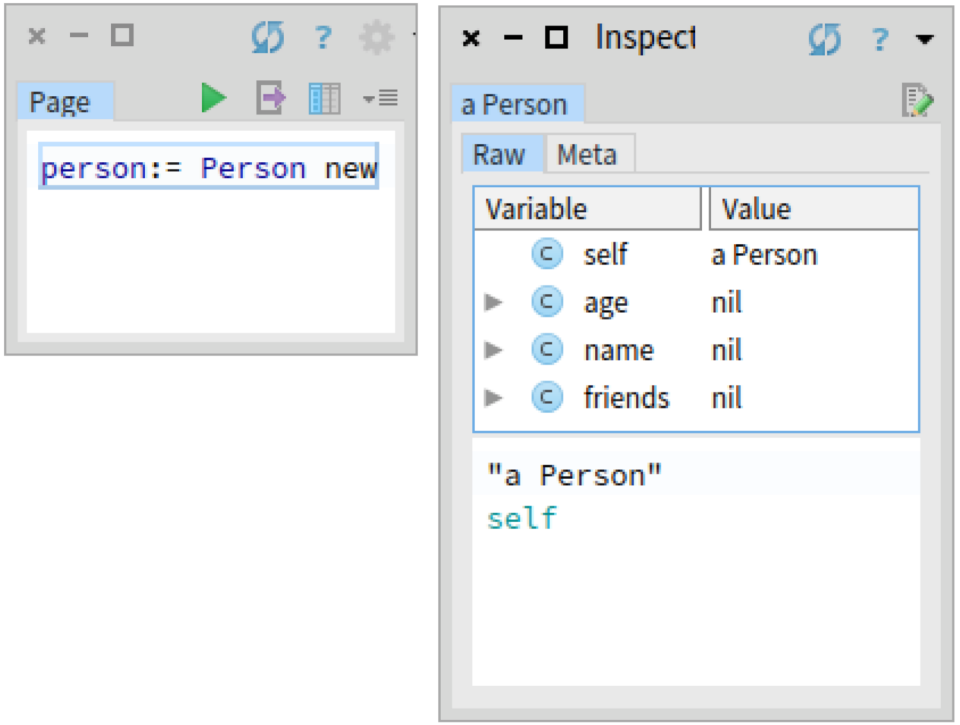
The memory:

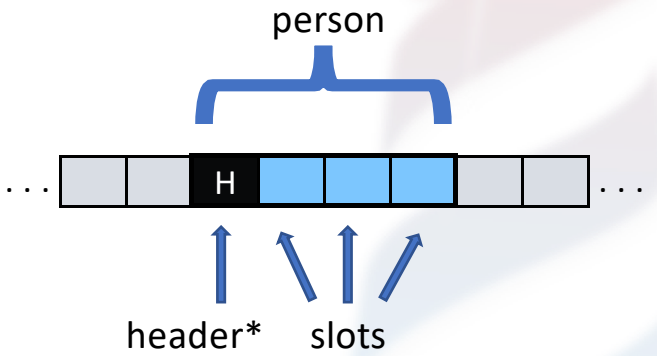

address
(aligned to 8 bytes)
1 0 0 1 1 0 1 0 0 1 1 1 0 1 0 0 1 1 1 0 0 1 1 0 1 0 0 1 1 1 0 1 0 0 1 1 1 0 0 1 1 0 1 0 0 1 1 1 0 1 0 0 1 1 1 0 0 1 1 0 1 1 0 1

a Person object in memory



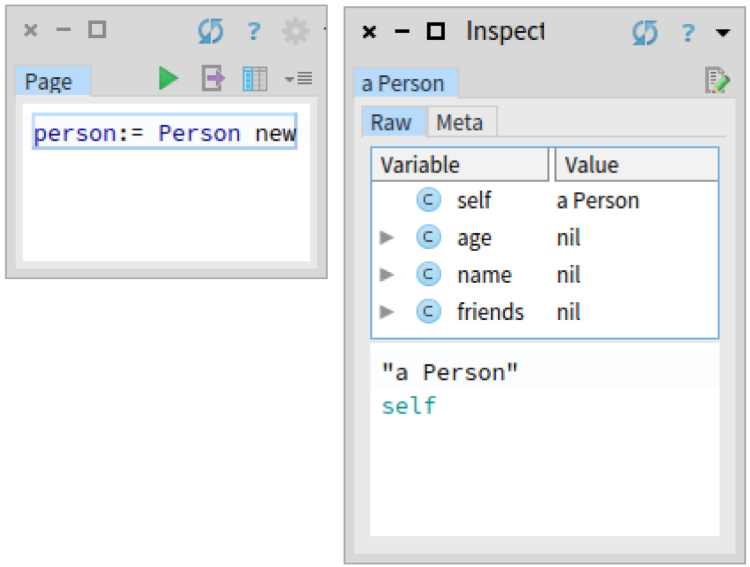
a Person object in memory

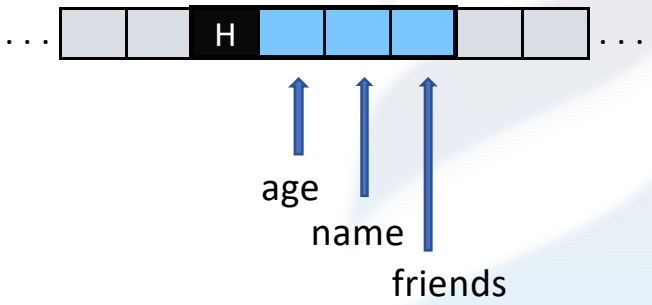




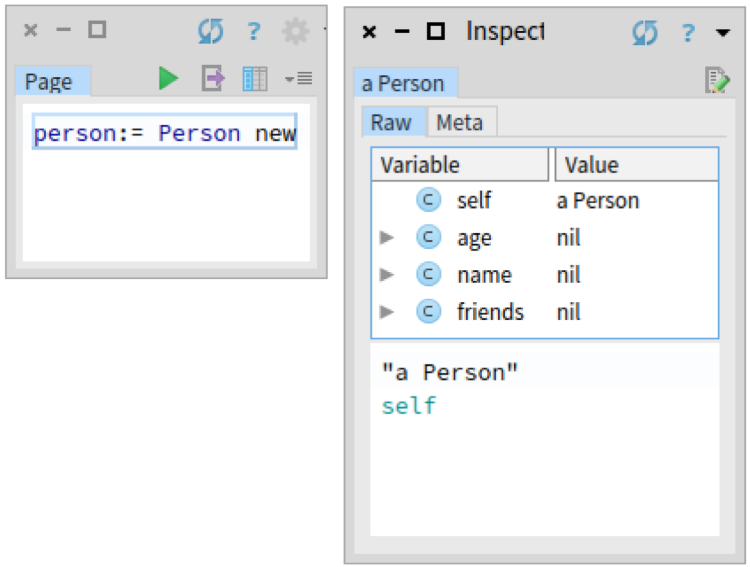
* In 32 bits
header size is 64 bits = 2 words

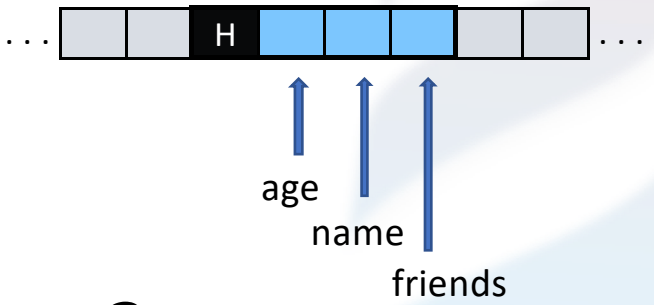
a Person object in memory





a Person object in memory: **Fixed Layout**

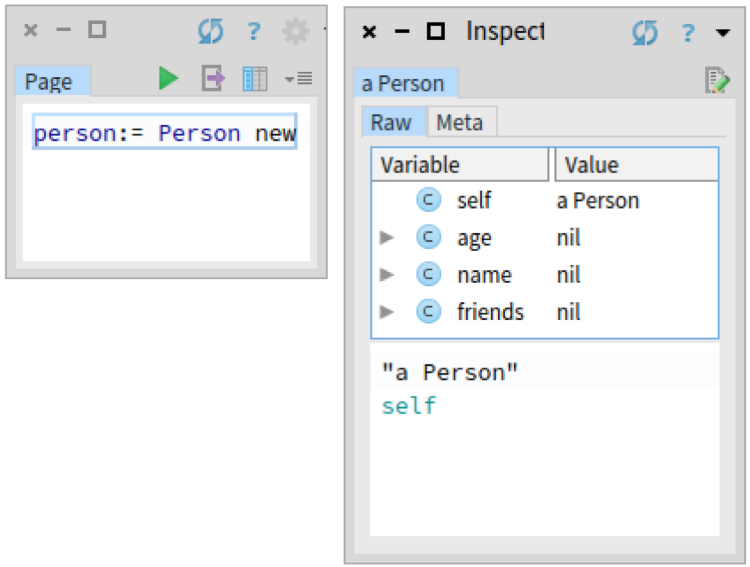


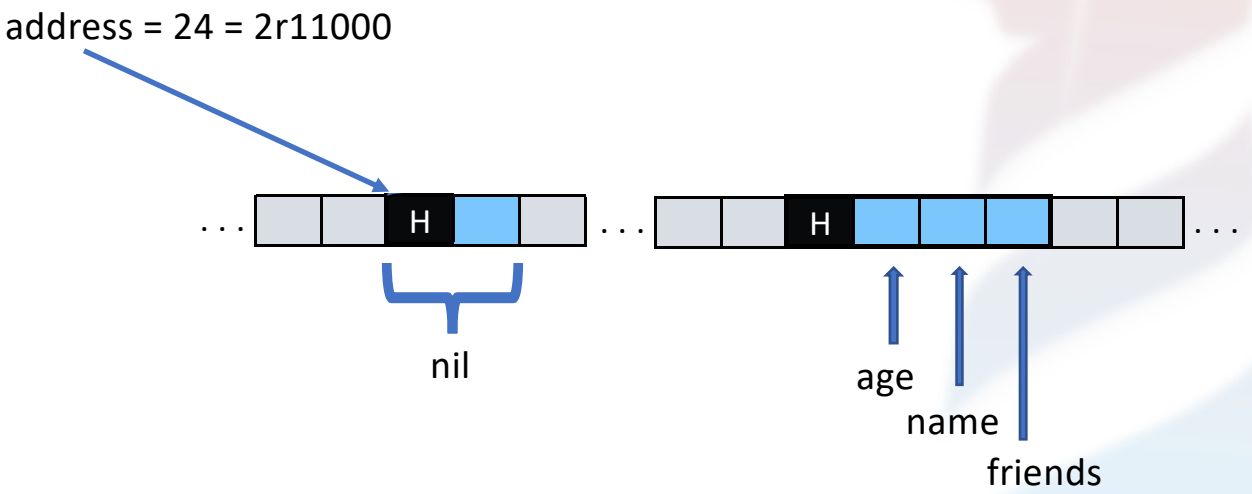


What is a Layout?

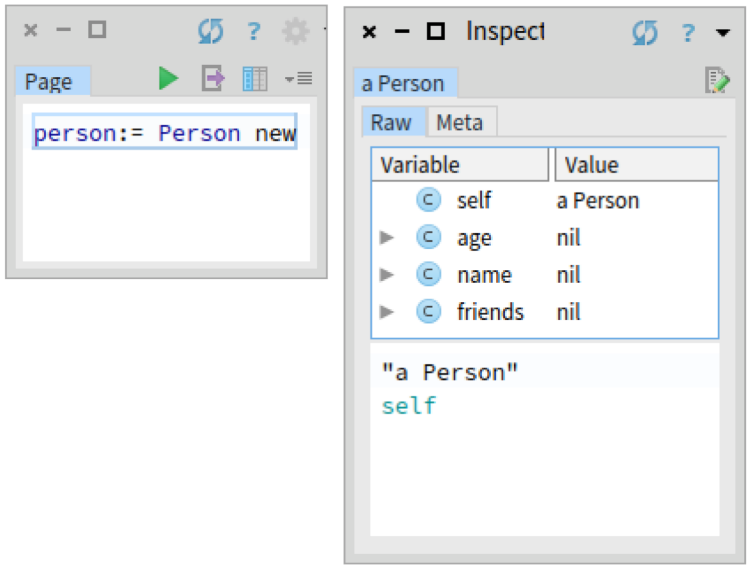
It is the structure of an object in memory

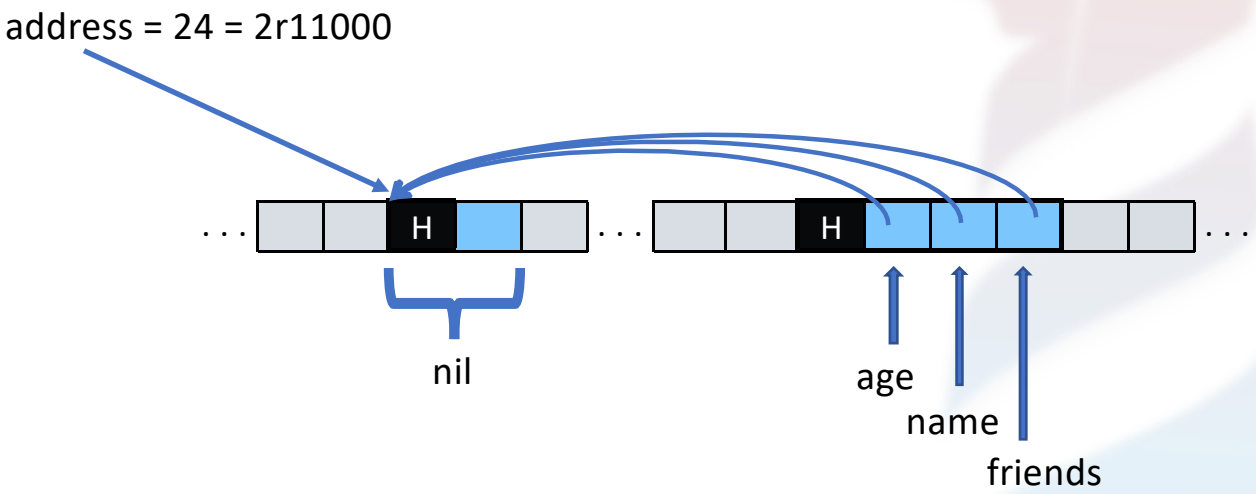
a Person object in memory: **Fixed Layout**



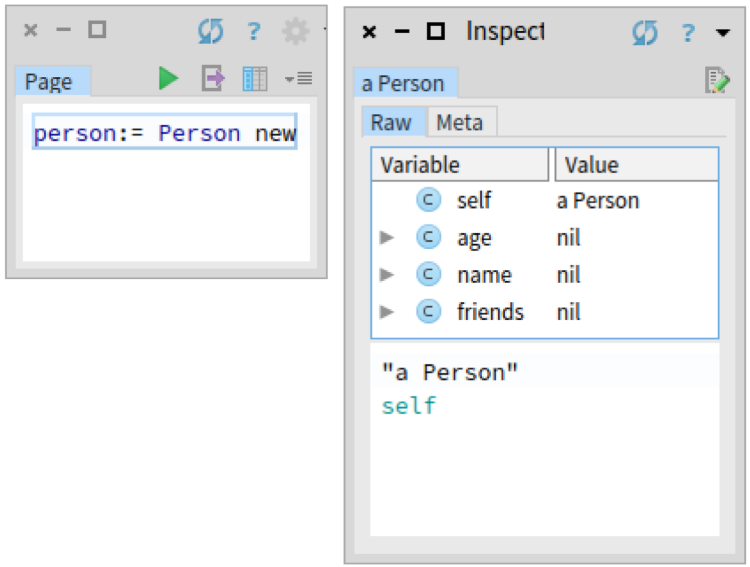


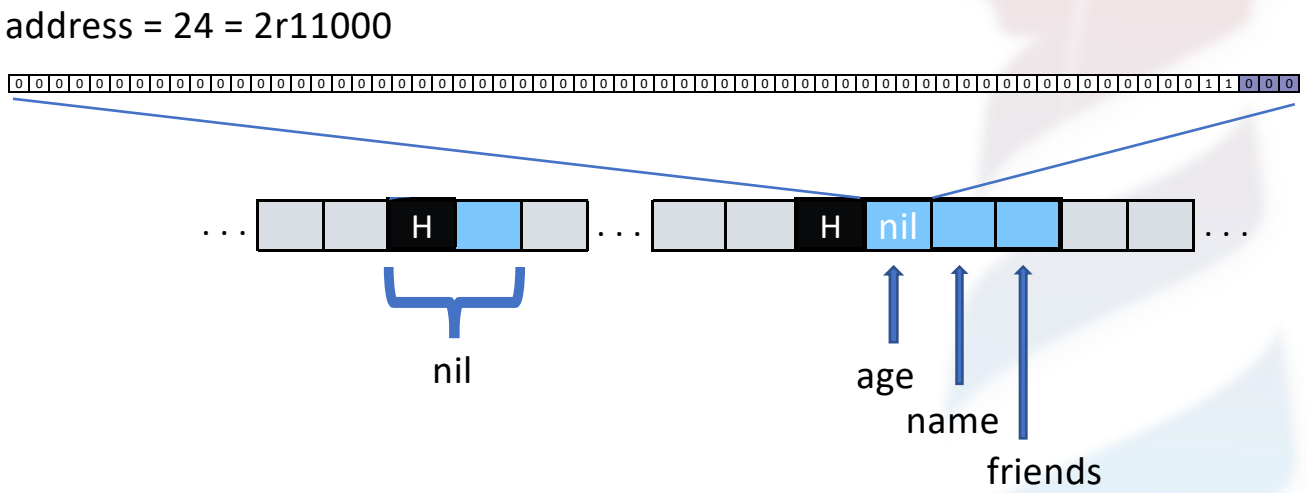
a Person object in memory: **Fixed Layout**





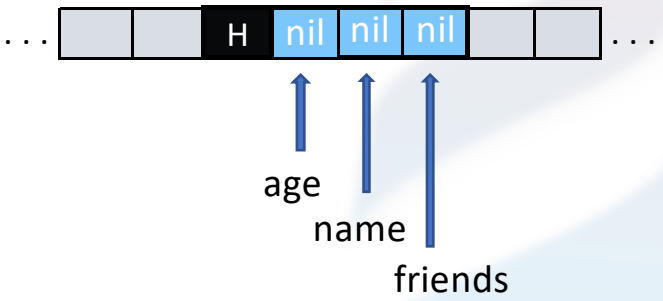
a Person object in memory: **Fixed Layout**



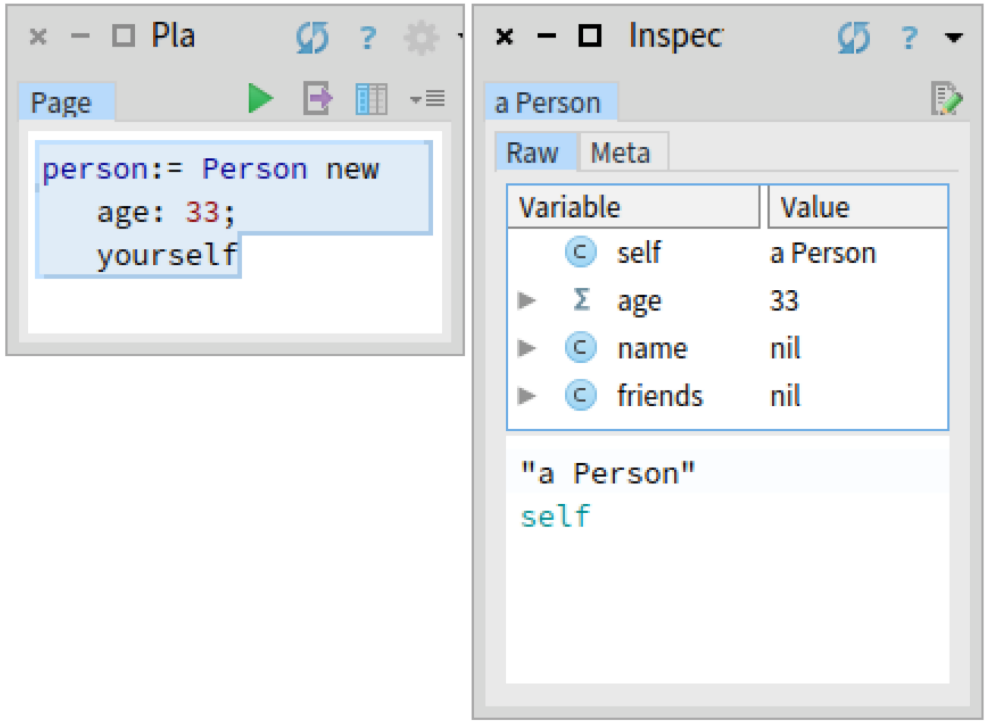


a Person object in memory: **Fixed Layout**

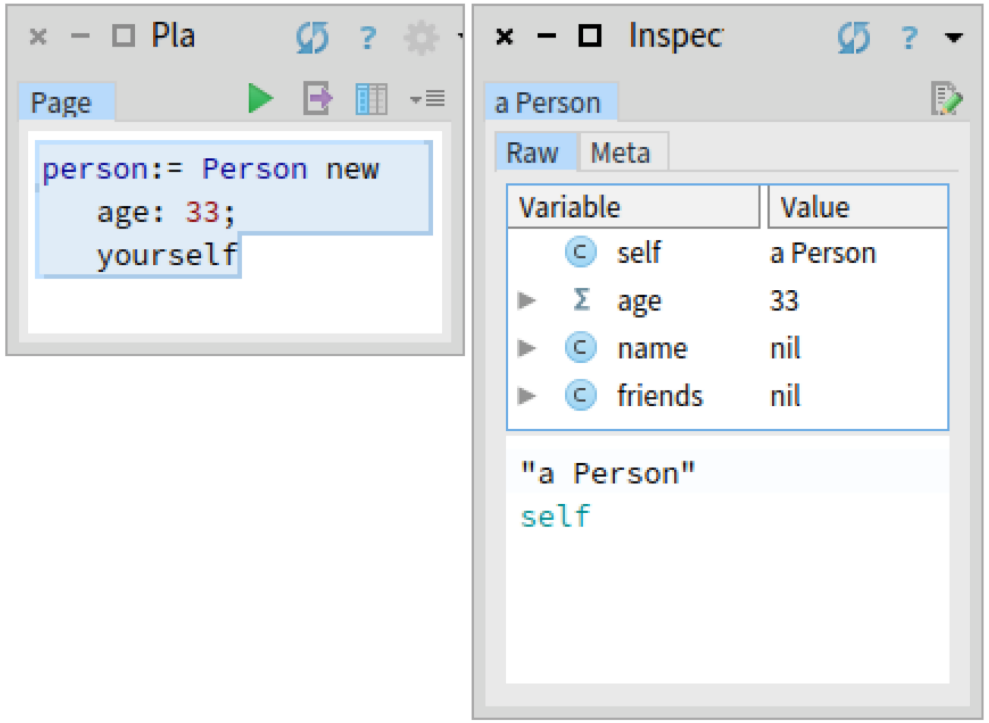
- Each slot is an **instance variable**
- Objects created sending the message **new**
- Number of slots is **constant** and defined by the class
- Each slot is initialized with a valid value: in this case with ‘nil’
- Slots contain **Oop** (ordinary object pointer)
  - **references** (address of another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

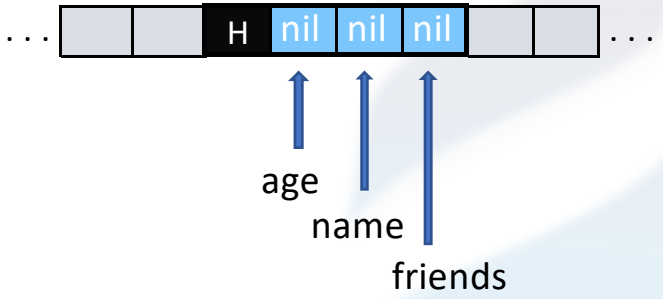


Setting the **age**

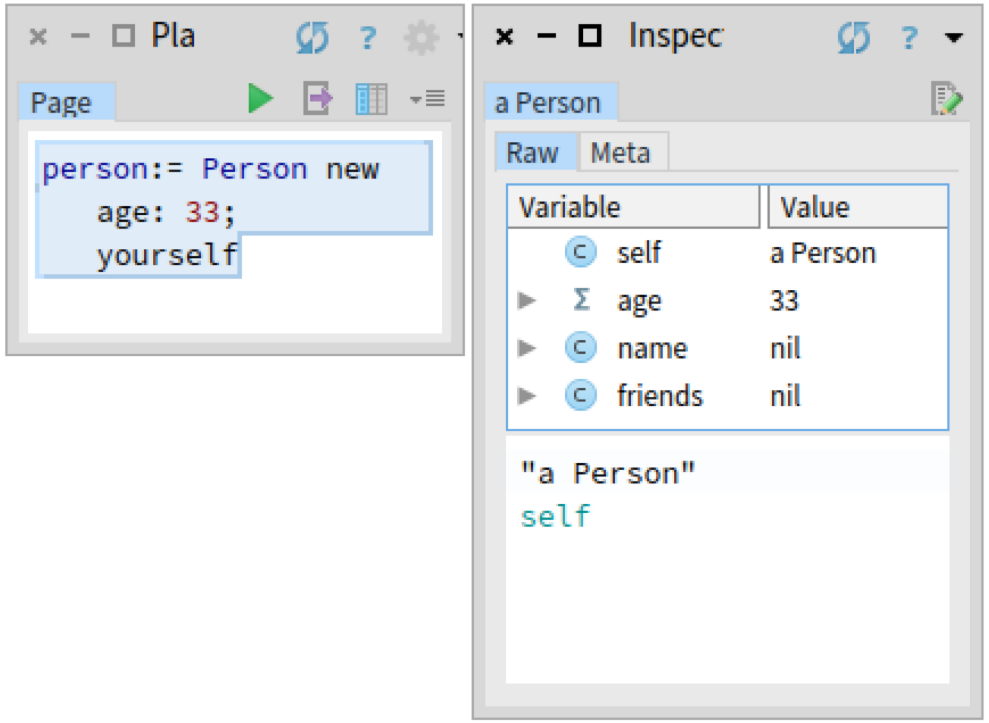


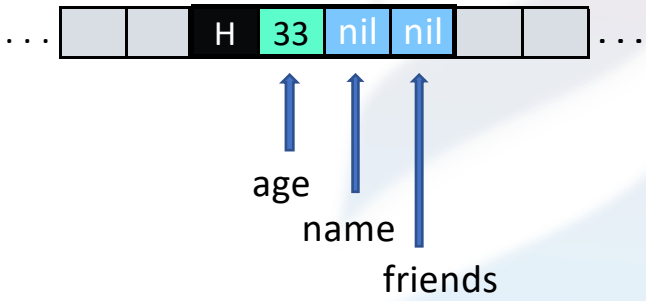
Setting the **age**



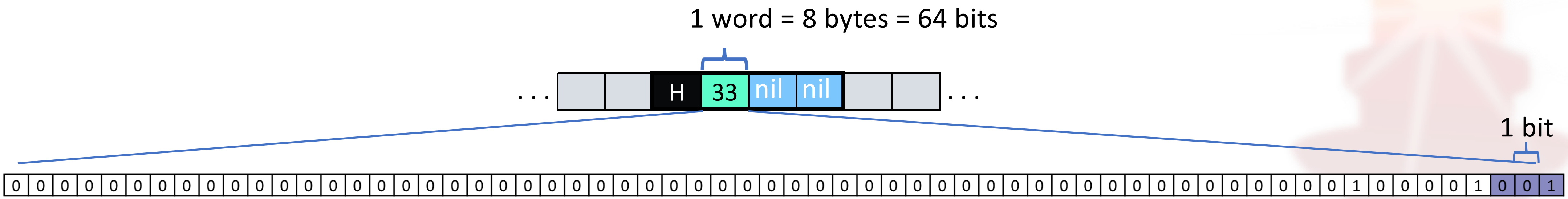


Setting the **age**





Setting the age: **Immediate layout**

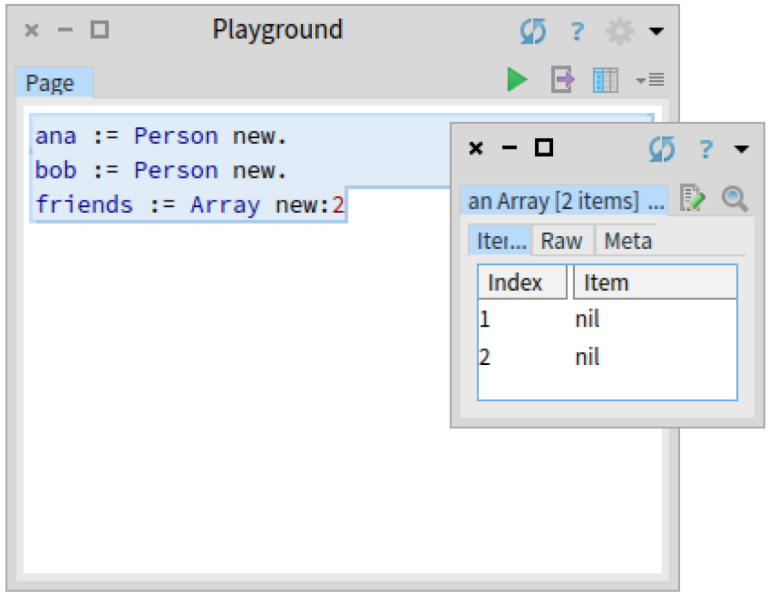


Tag saved in the 3 least significant bits:

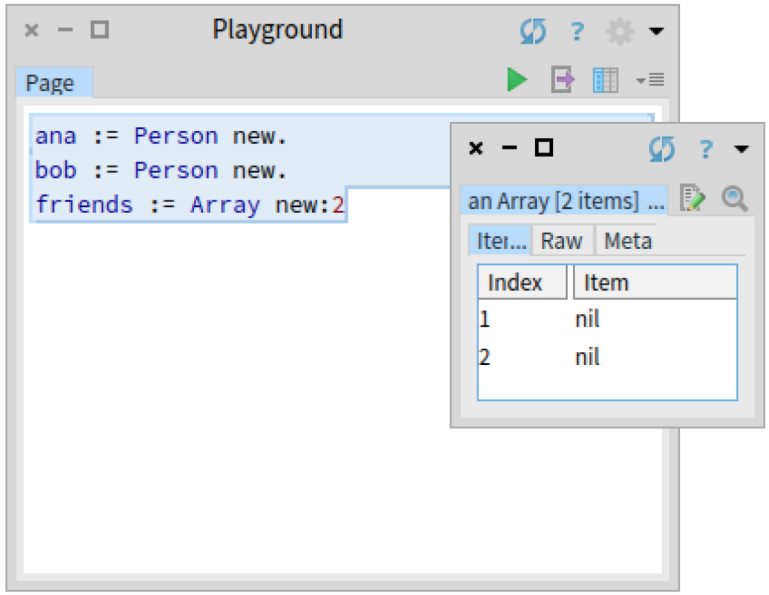
- 001 -> SmallInteger
- 010 -> Character
- 100 -> SmallFloat64 (only in 64 bits)

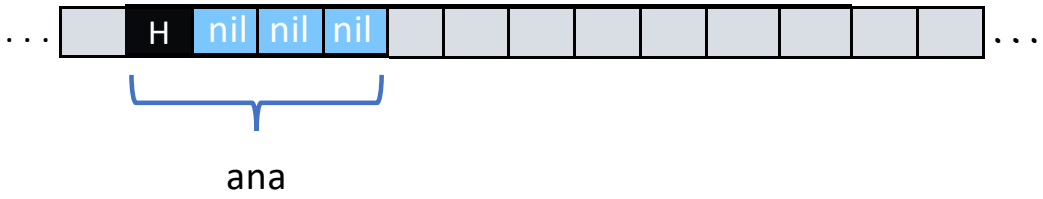
Immediates are a **memory optimization**!

Setting the **friends**

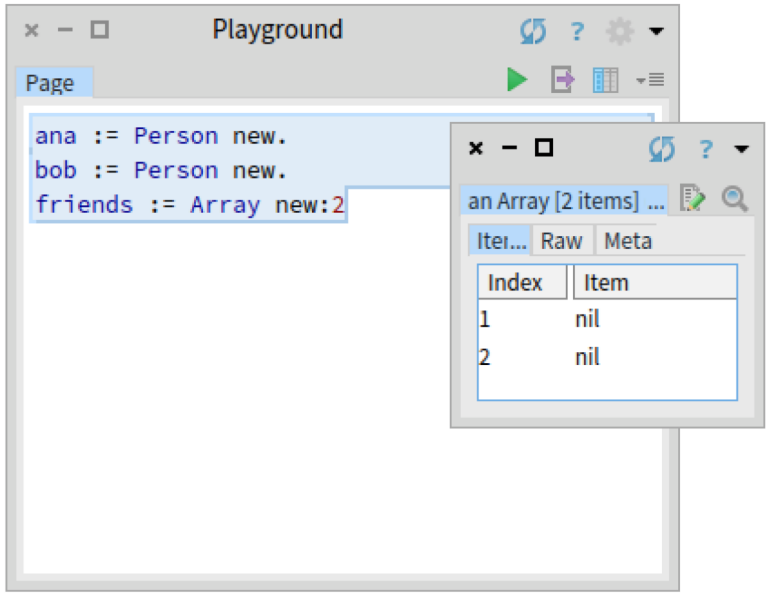


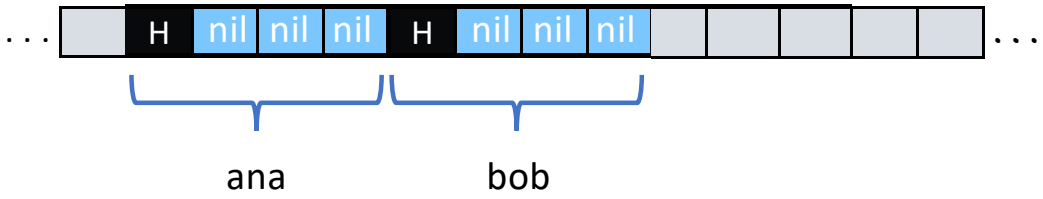
Setting the **friends**



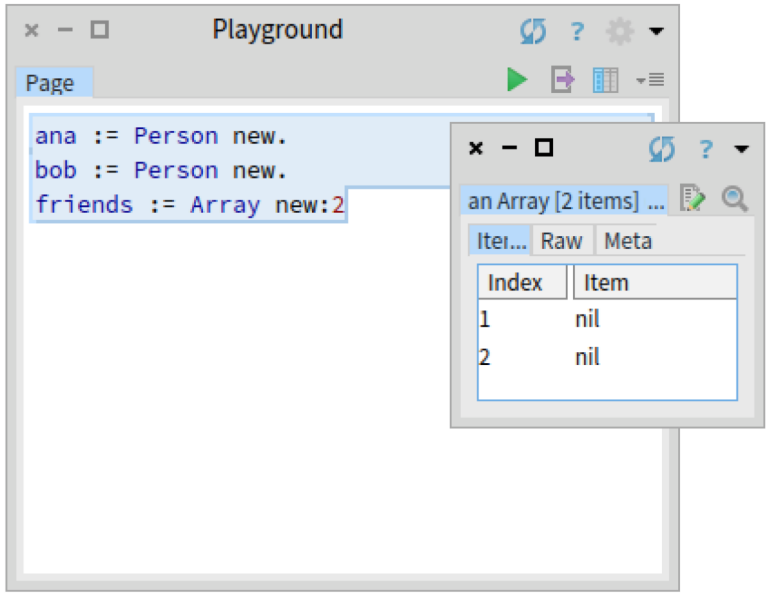


Setting the **friends**

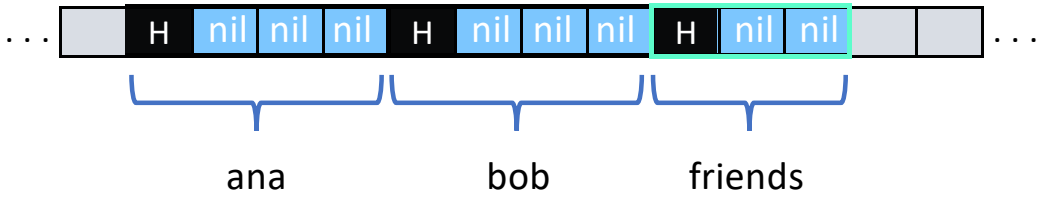




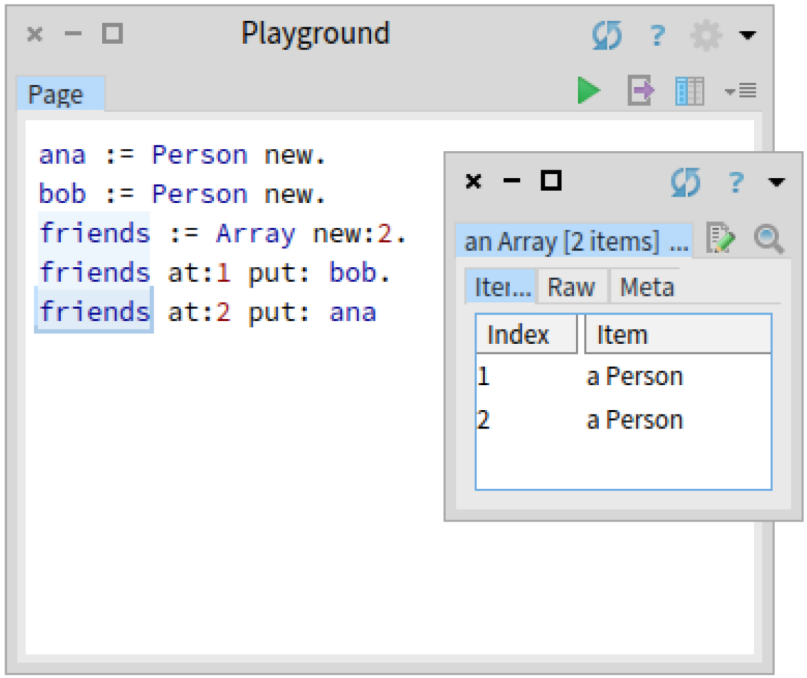
Setting the friends: Indexable Pointer layout



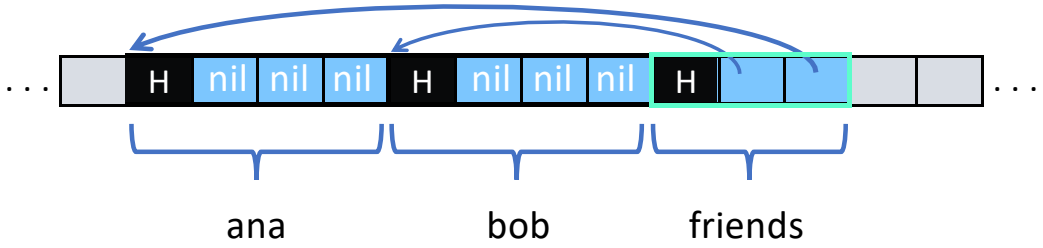
- **Variable number of slots**
  - Defined dynamically at the moment of **allocation**
- Object created by sending the message **new:**
- Objects **can't grow or shrink**!
- Size of slots is 1 word
- Slots contain **Oop**



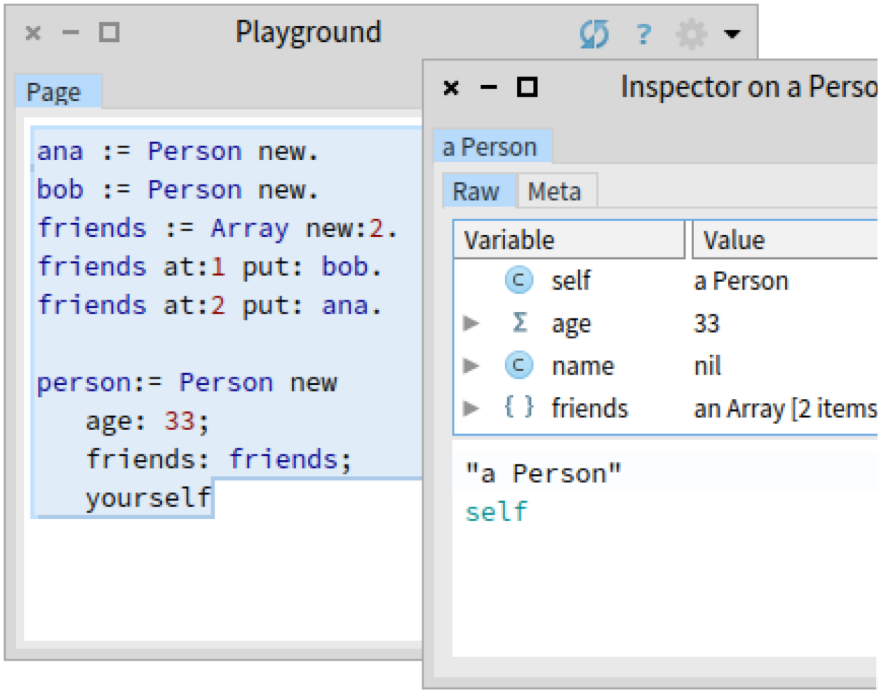
Setting the friends: Indexable Pointer layout



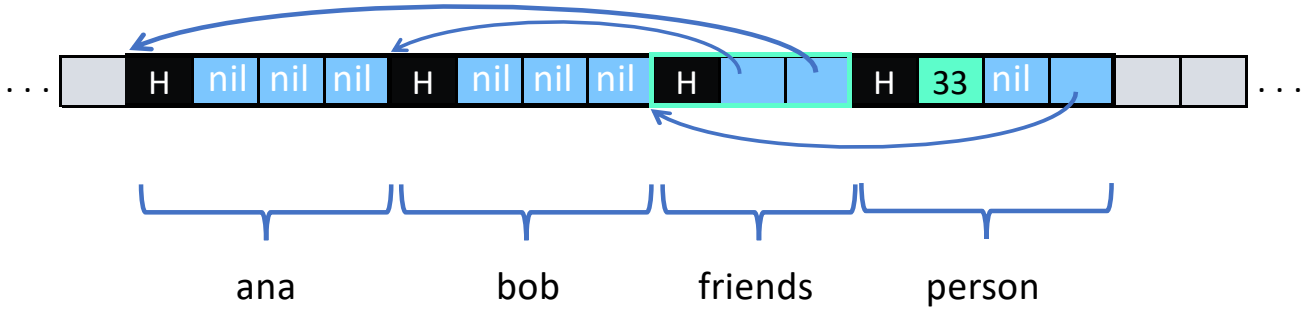
- **Variable number of slots**
  - Defined dynamically at the moment of **allocation**
- Object created by sending the message **new:**
- Objects **can't grow or shrink**!
- Size of slots is 1 word
- Slots contain **Oop**



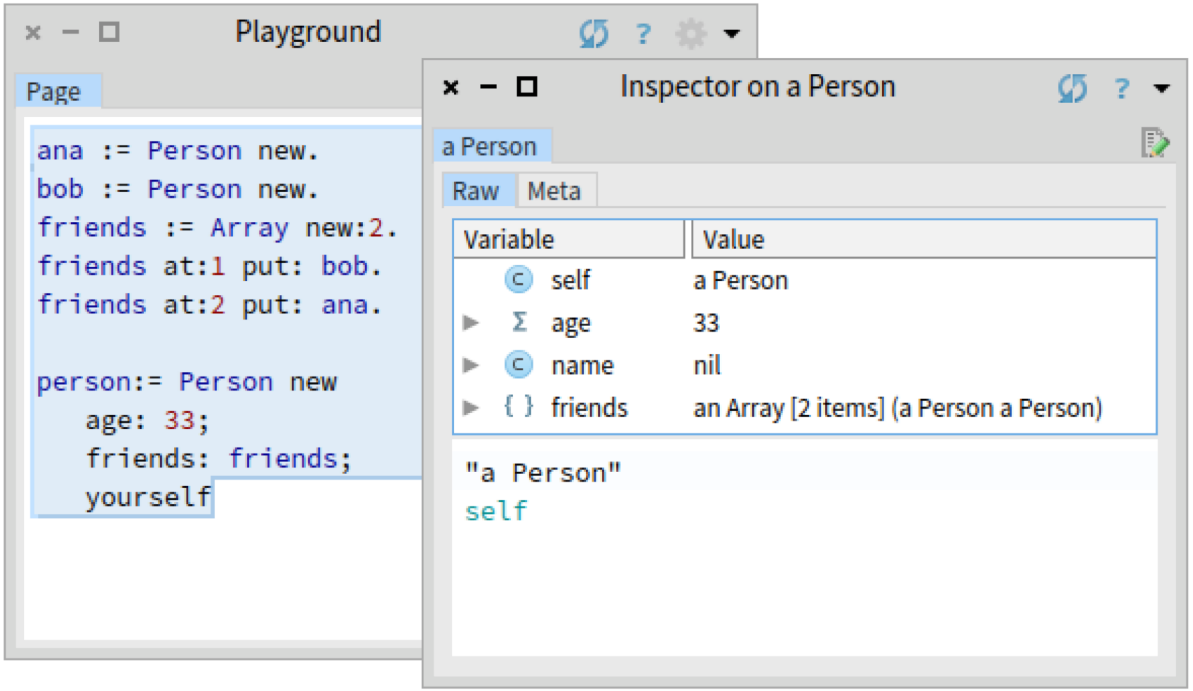
Setting the friends: Indexable Pointer layout

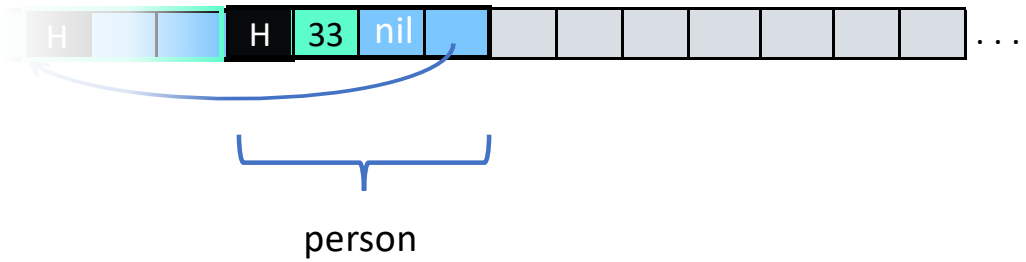


- **Variable number of slots**
  - Defined dynamically at the moment of **allocation**
- Object created by sending the message **new:**
- Objects **can't grow or shrink**!
- Size of slots is 1 word
- Slots contain **Oop**

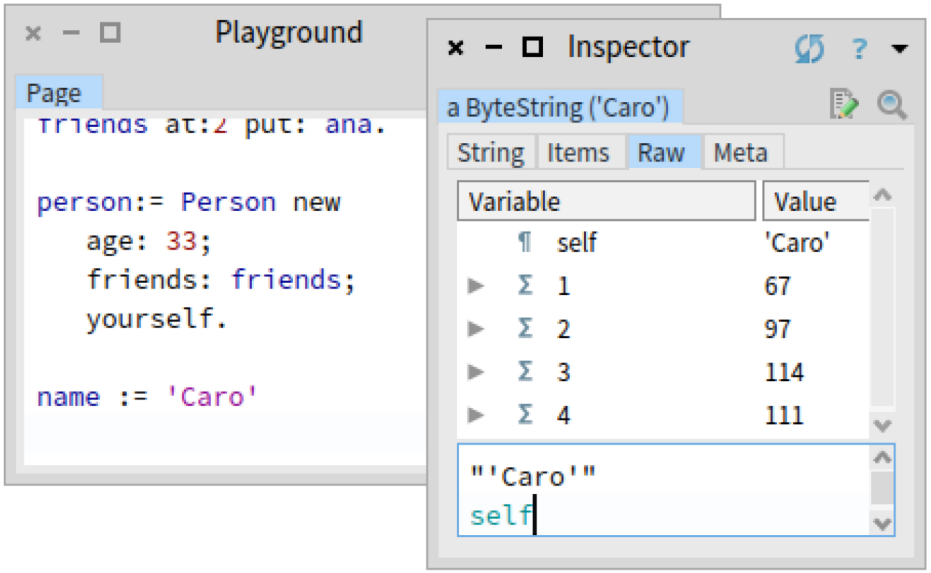


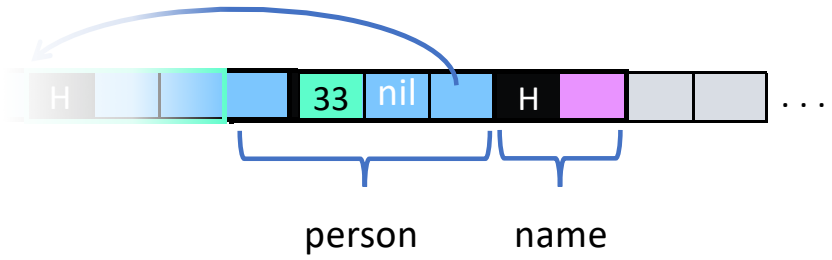
Setting the **name**



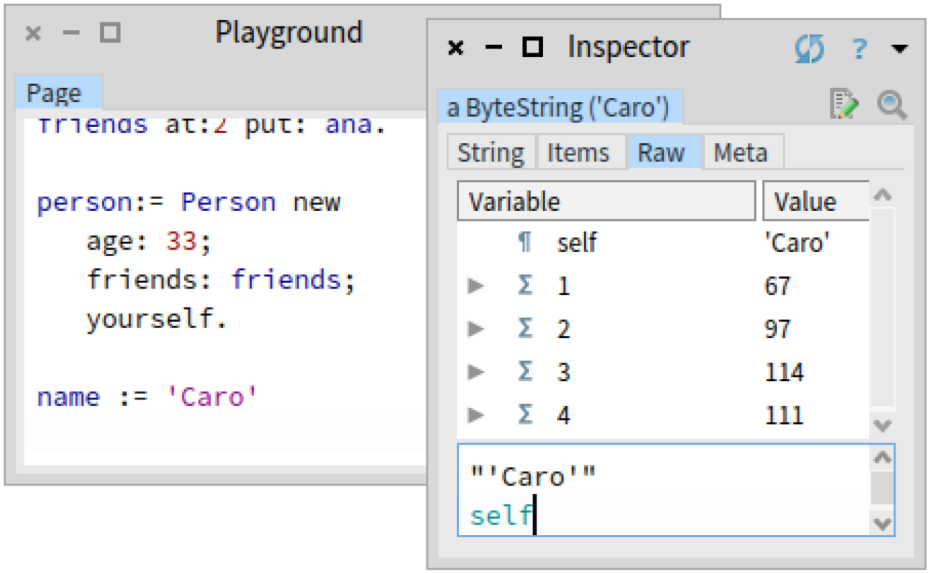


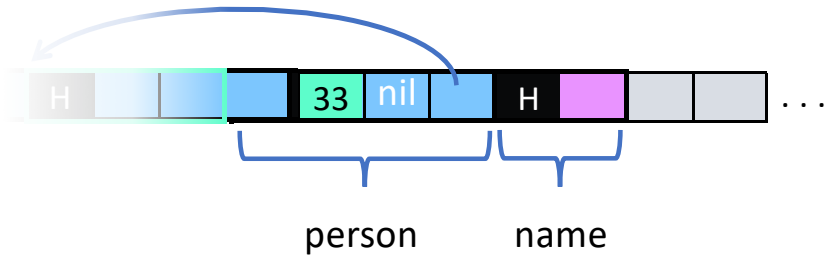
Setting the **name**





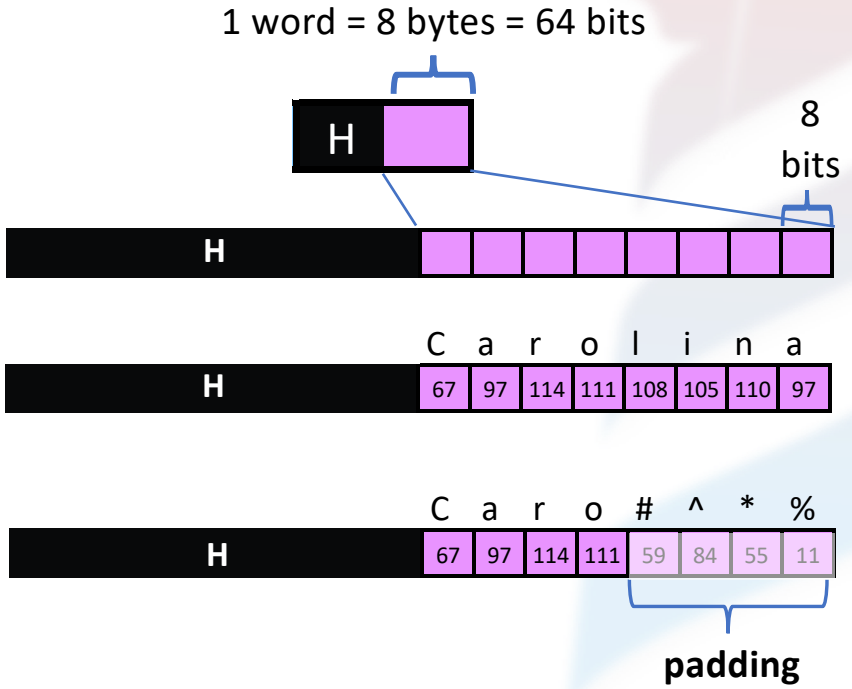
Setting the name: **Indexable Byte layout**





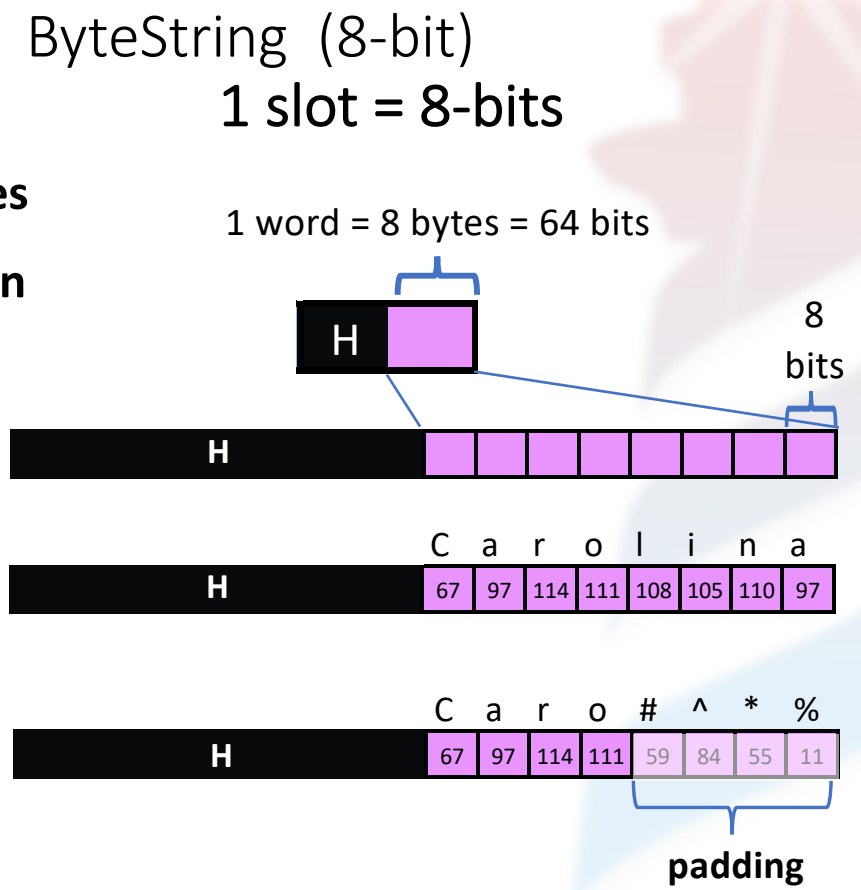
ByteString (8-bit)

1 slot = 8-bits



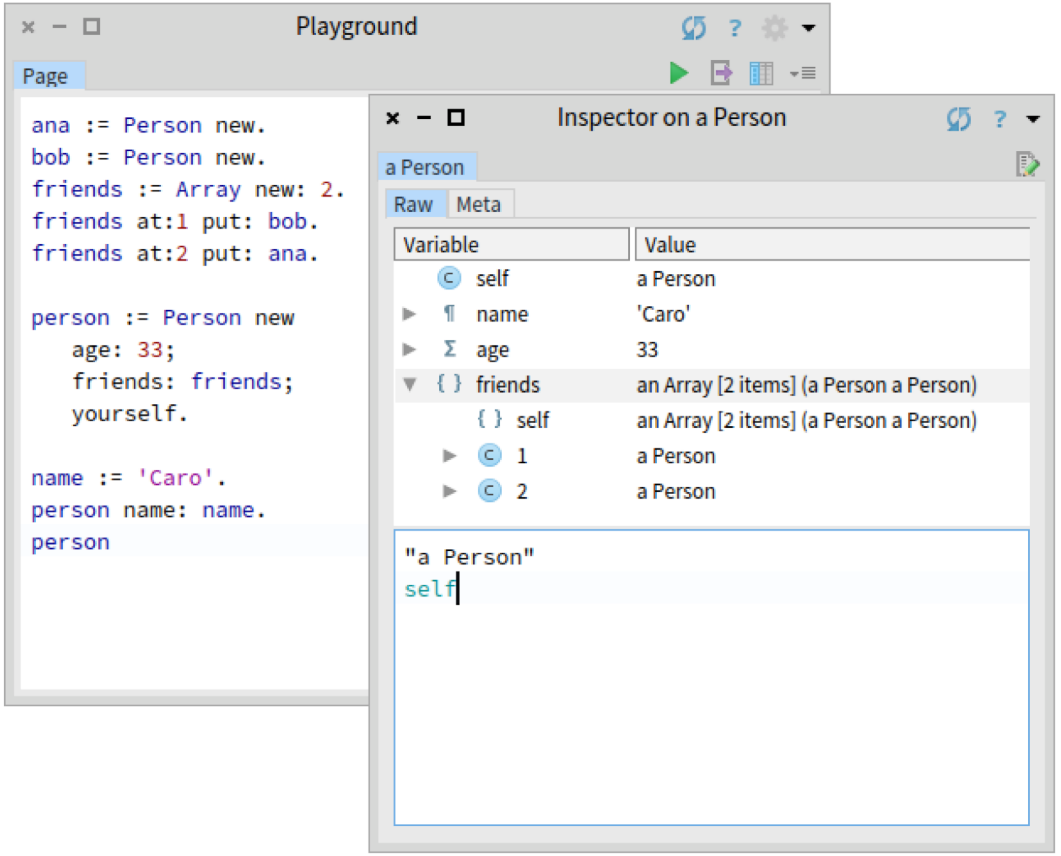
Setting the name: **Indexable Byte layout**

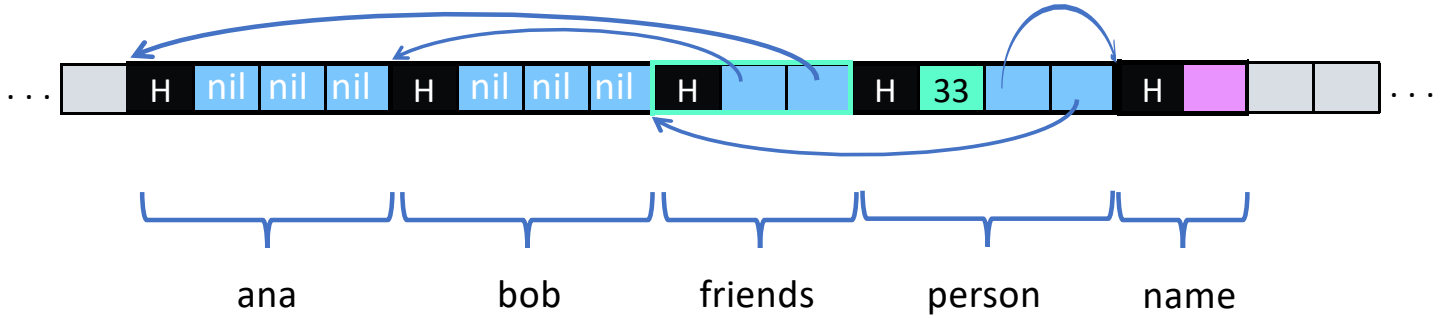
- **Variable number of slots**
  - Slots **do not contain Oop**, they contain **raw bytes**
  - Defined dynamically at the moment of **allocation**
  - Created by sending the message **new:**
  - Objects **can't grow or shrink**!
  - **Size of slots depends on the object's layout**
    - 8-bit (eg: ByteArray)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



Finally!
A Person in memory

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

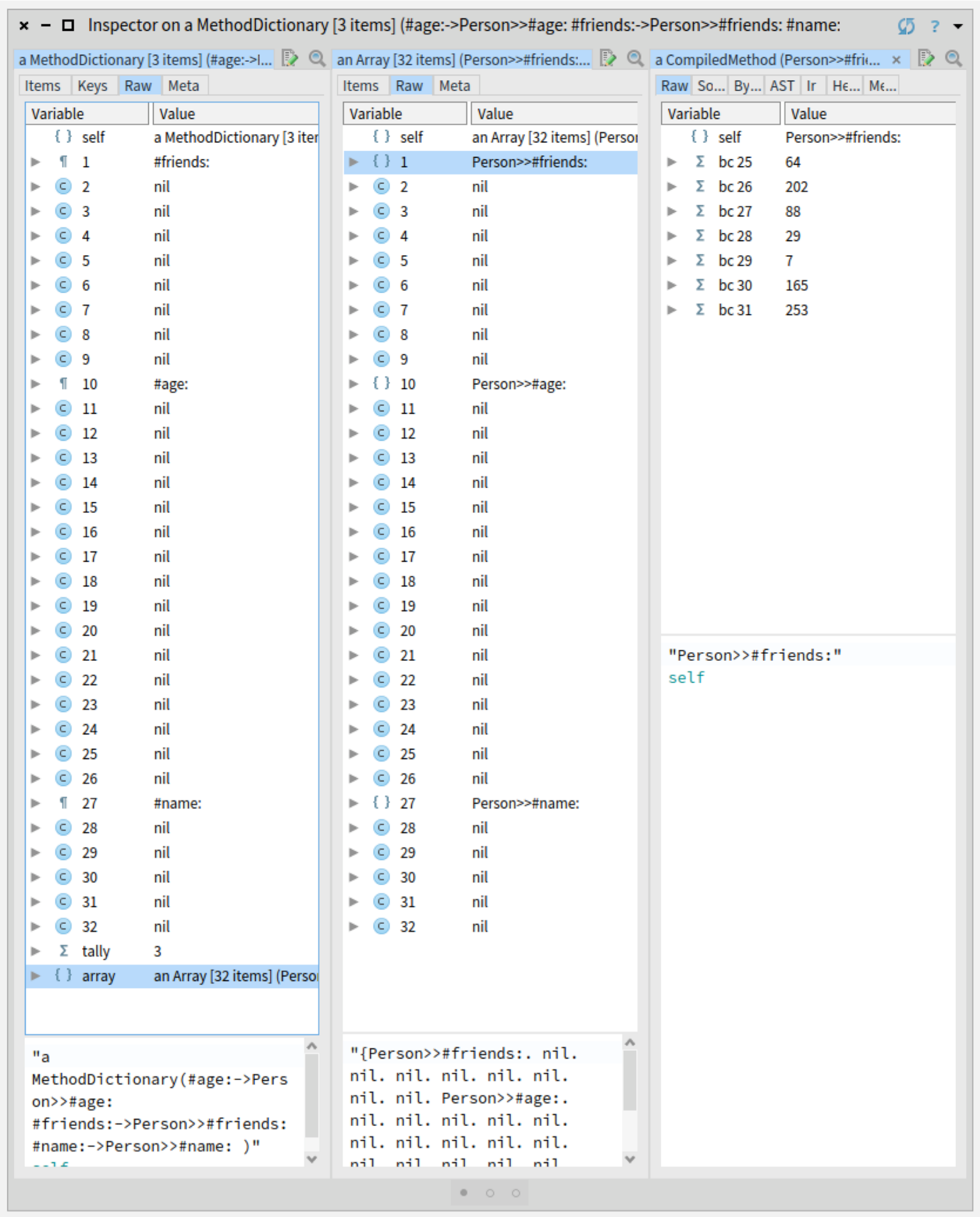




Special layout: **Indexable with Instance Variables layout**

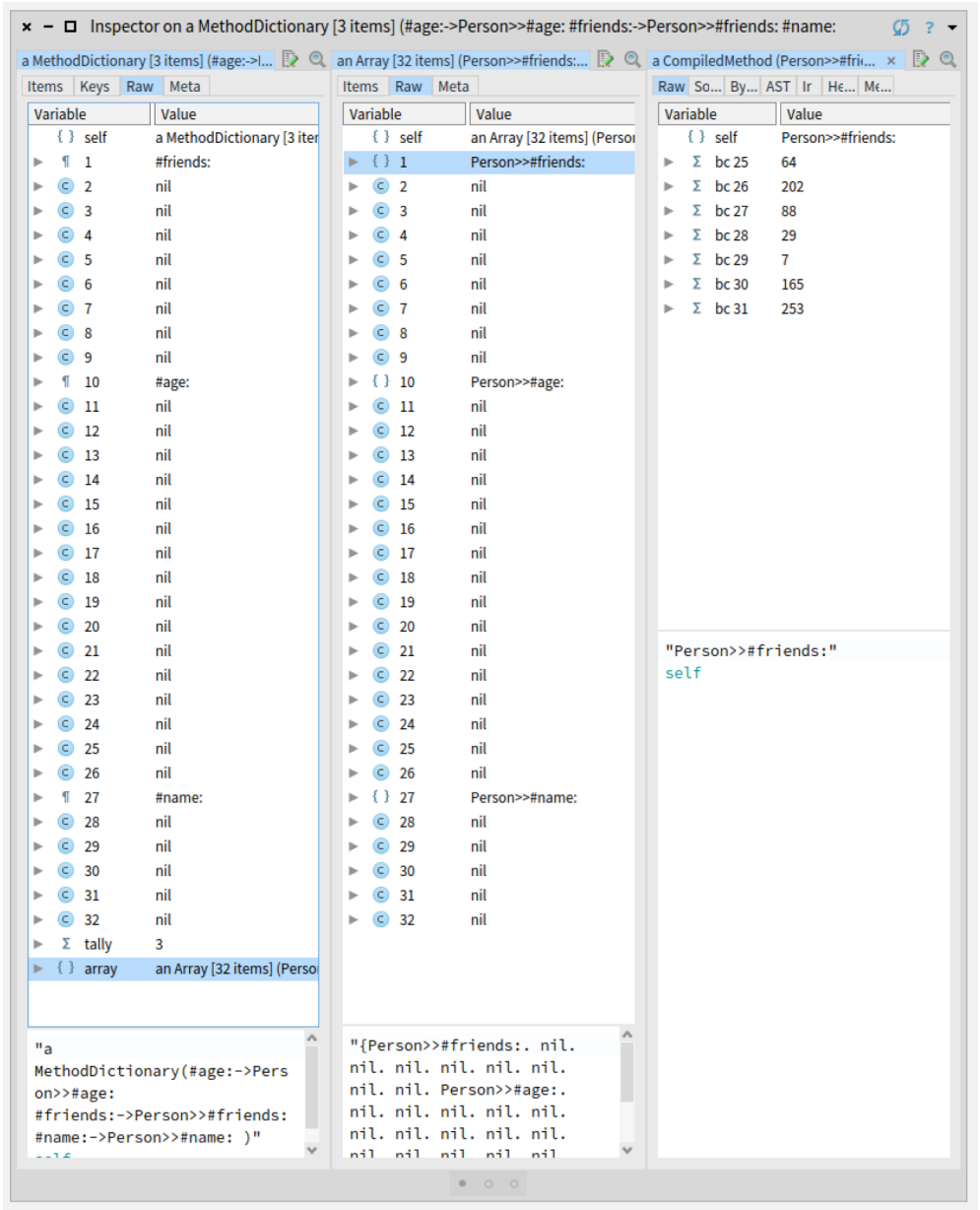
Examples:

- Context
- CompiledMethod
- **MethodDictionary**



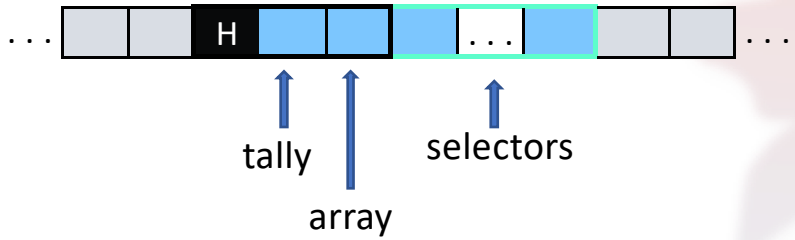
Indexable with Instance Variables layout

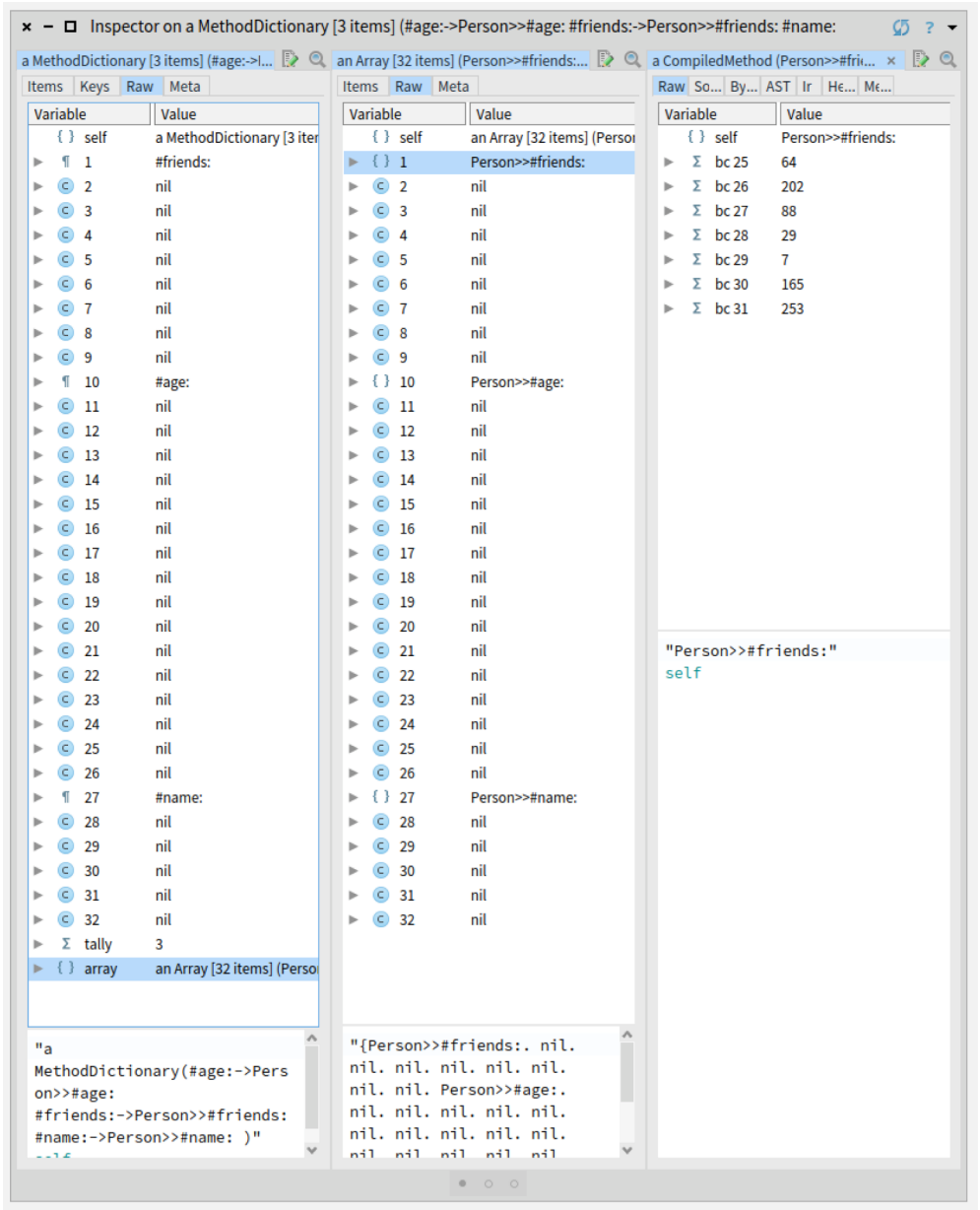
Example: MethodDictionary



Indexable with Instance Variables layout

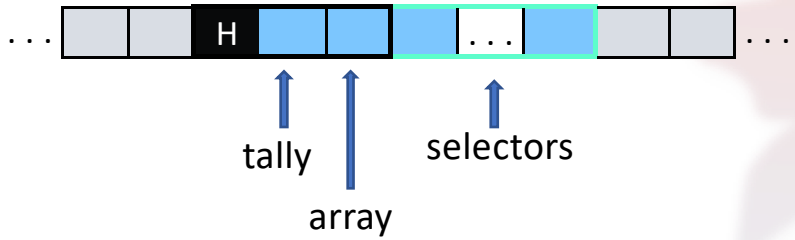
Example: MethodDictionary





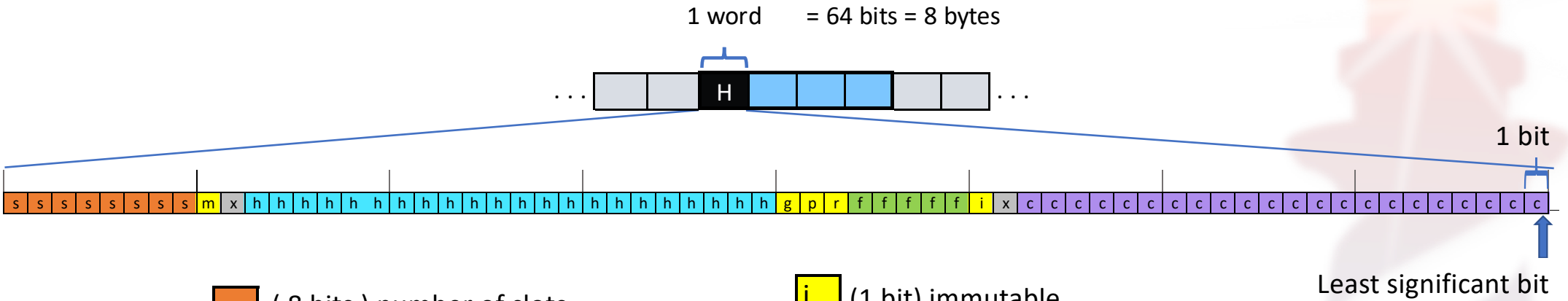
Indexable with Instance Variables layout

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

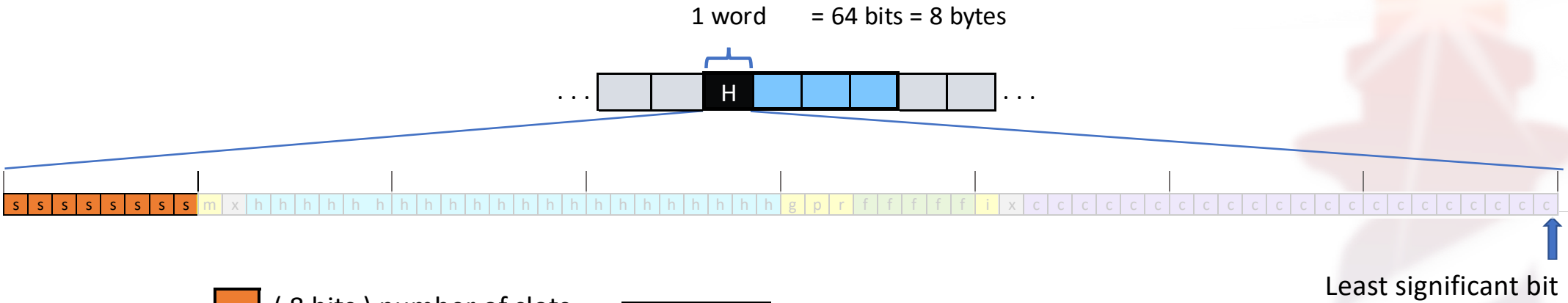
The Header

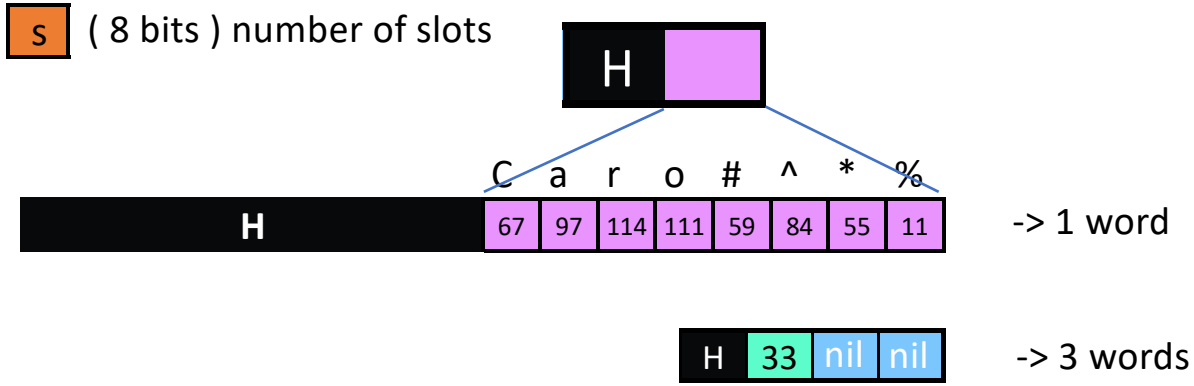


- s (8 bits) number of slots
- h (22 bits) identity hash
- f (5 bits) format
- c (22 bits) class index

- i (1 bit) immutable
- m (1 bit) marked
- p (1 bit) pinned
- g (1 bit) grey
- r (1 bit) remembered
- x unused

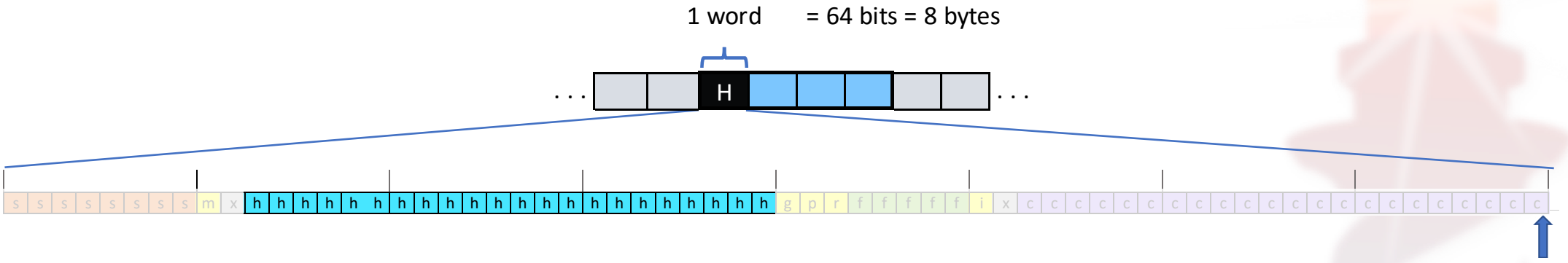
The Header: **number of Slots** (number of **words**)





- 0 <= number of slots < 255
- When number of slots is 255, the object is a Big Object

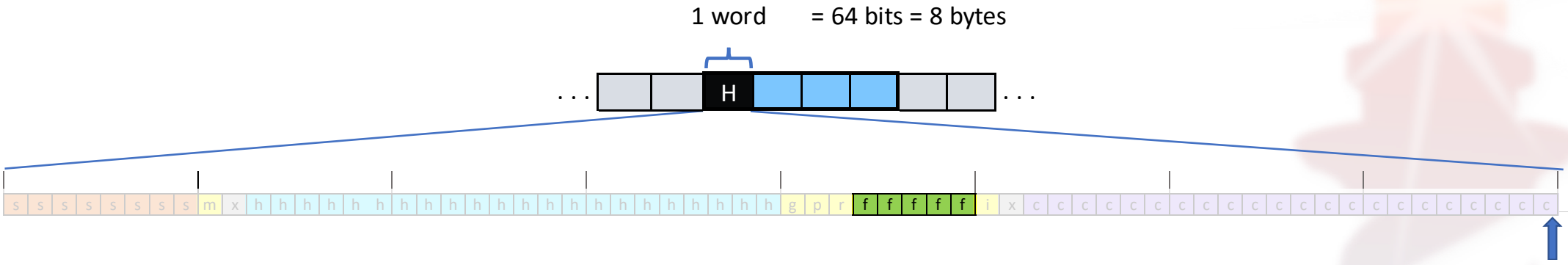
The Header: **identity hash**



h (22 bits) identity hash

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

The Header: **format**

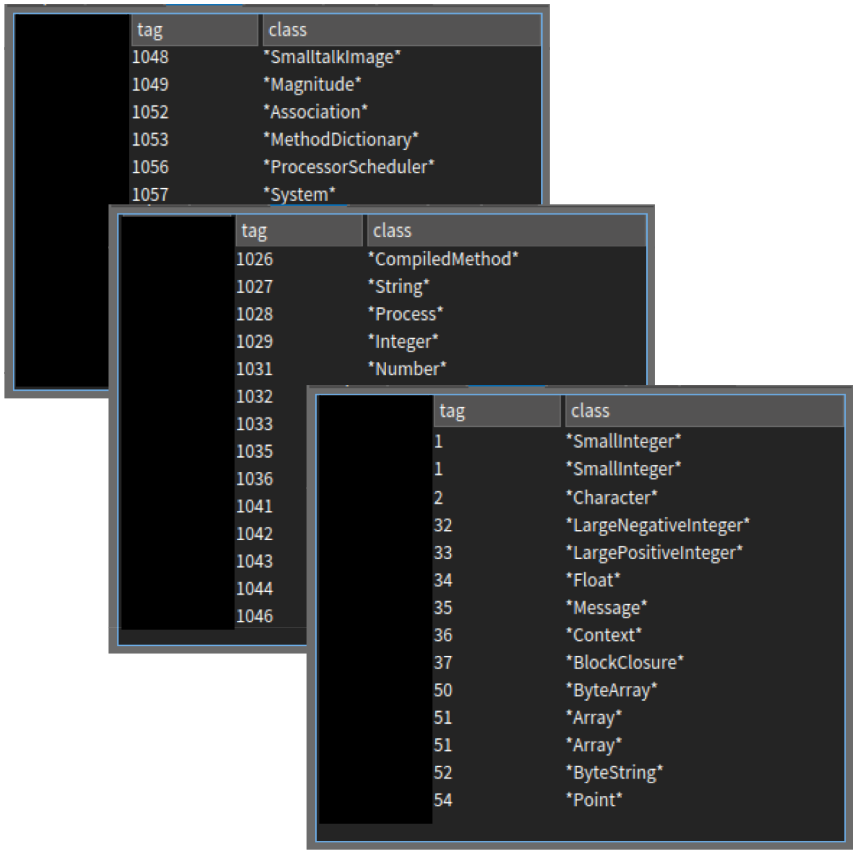


f (5 bits) format

- 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

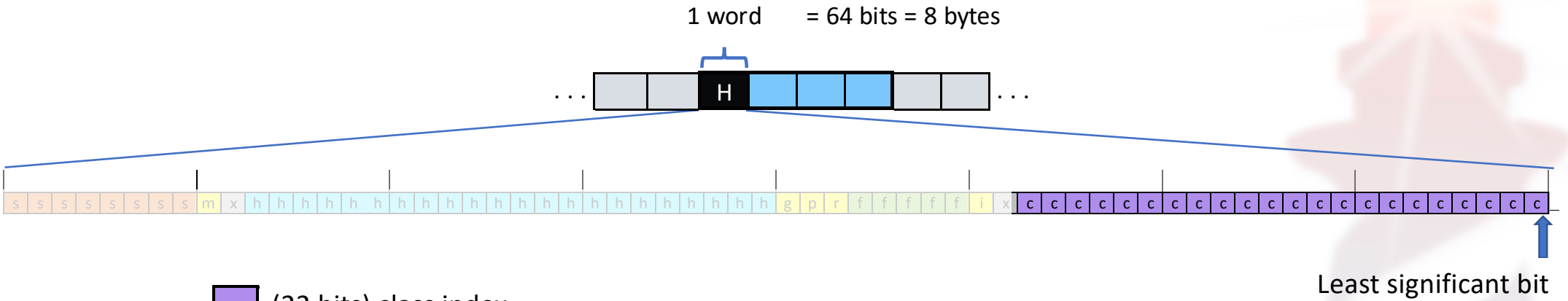


tag	class
1048	*SmalltalkImage*
1049	*Magnitude*
1052	*Association*
1053	*MethodDictionary*
1056	*ProcessorScheduler*
1057	*System*

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*

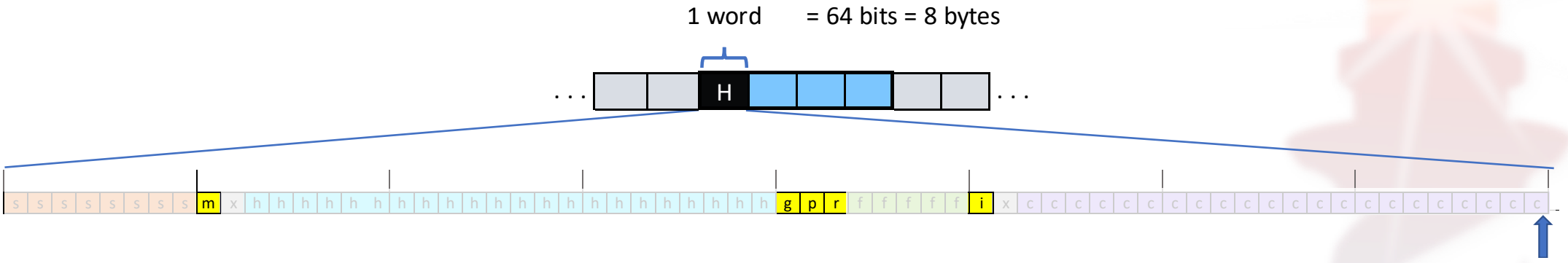
The Header: **class index**



c (22 bits) class index

- Stores the index of the class in the class table (hidden from the image side)
  - This is an optimization -> we don't save class references in objects
- <12 bits> <10 bits>
  - 12 bits for storing the page index
  - 10 bits for storing the index of the class in the page (1024 entries per page)
  - Approx 4 millions of classes maximum
- The identity hash of a Class object is its class index

The Header: **GC flags**



i	(1 bit) immutable
m	(1 bit) marked
p	(1 bit) pinned
g	(1 bit) grey
r	(1 bit) remembered

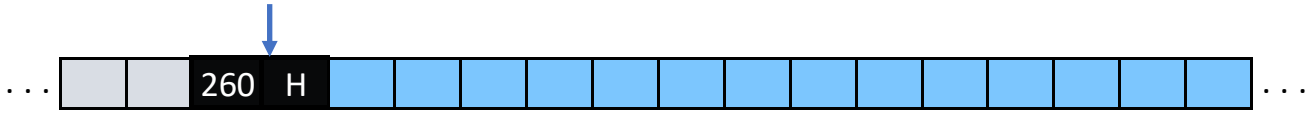
- Reserved for Garbage Collection algorithms

Big objects

- Objects with more than 254 slots

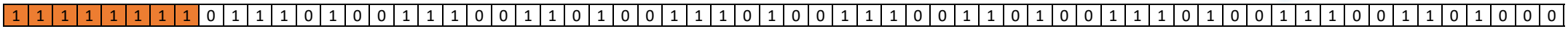
Example: Object with 260 instance variables

- 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

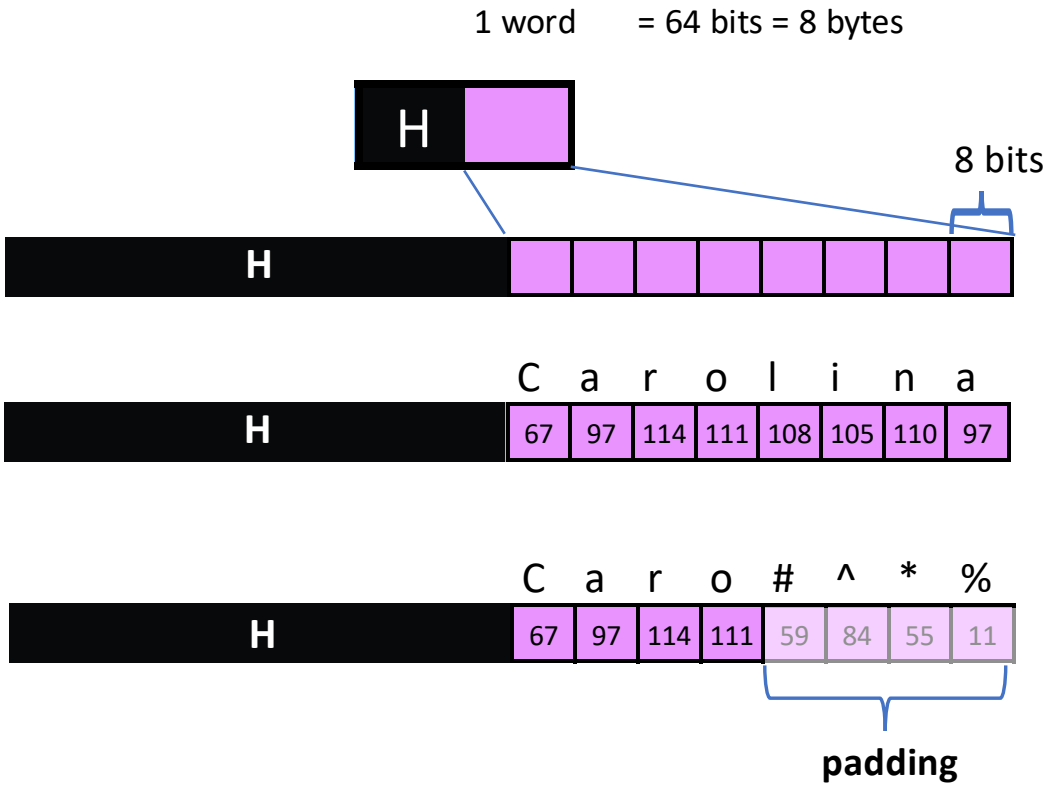
- Layouts:
  - Fixed
  - Immediate
  - Indexable Pointer
  - Indexable Byte
  - Indexable with Instance Variables
- Object
  - Header
  - Class Table
  - Big Objects

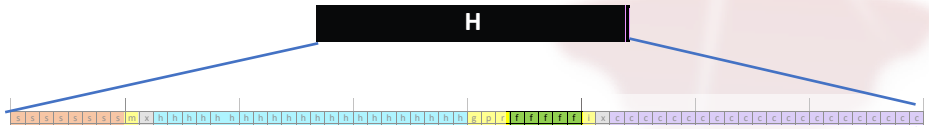
Carolina Hernández Phillips

RMod Team

More about the padding in: **Indexable Byte layout**

Example: ByteString => 8 bit indexable opaque layout



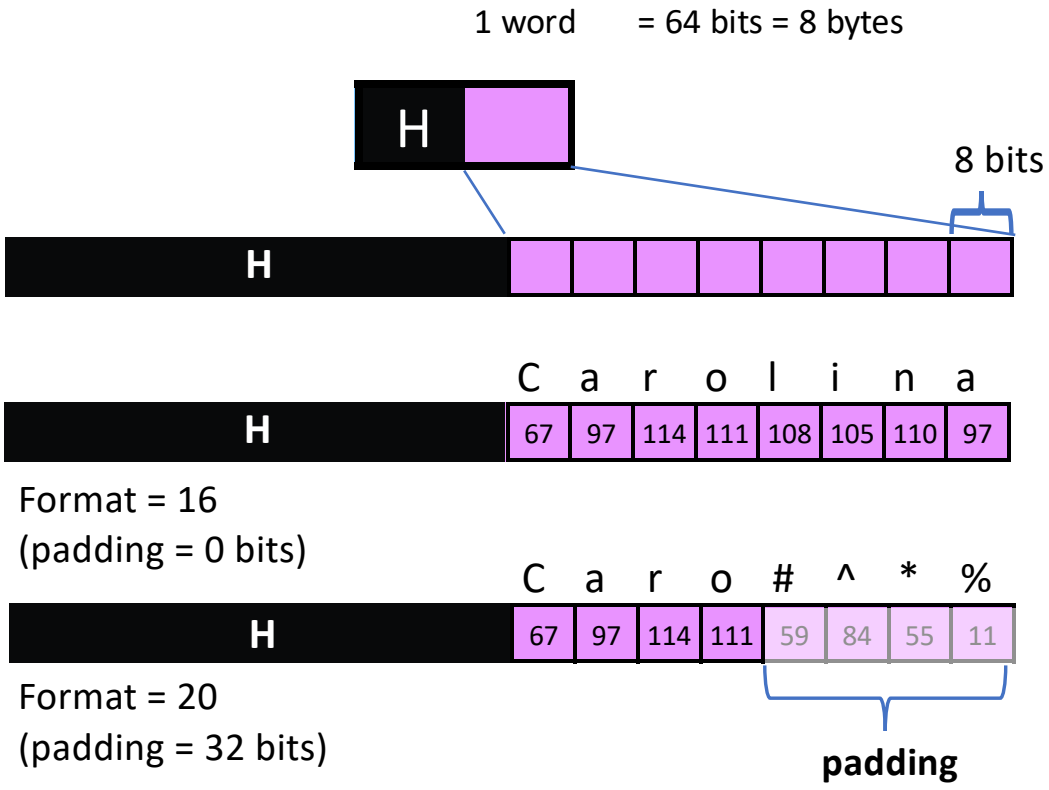


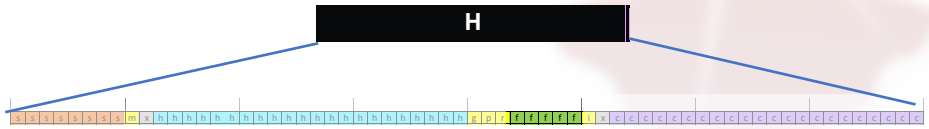
16-23 Object **format** 8 bits indexable opaque layout

- 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

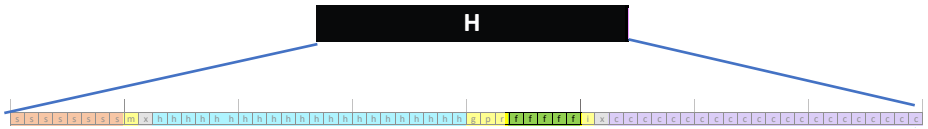




16-23 Object format 8 bits indexable opaque layout

- 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**



Object **formats**

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**