Fuel

Mariano Martinez Peck
marianopeck@gmail.com
http://marianopeck.wordpress.com/

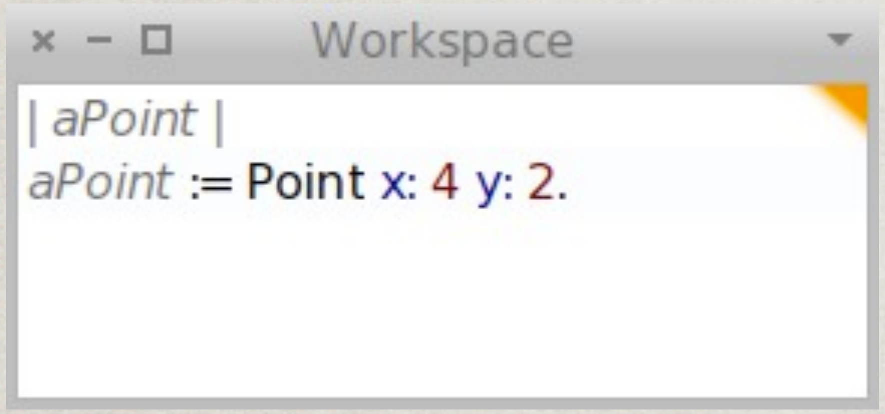
THANKS A LOT!!!

[|] EUROPEAN SMALLTALK USER GROUP

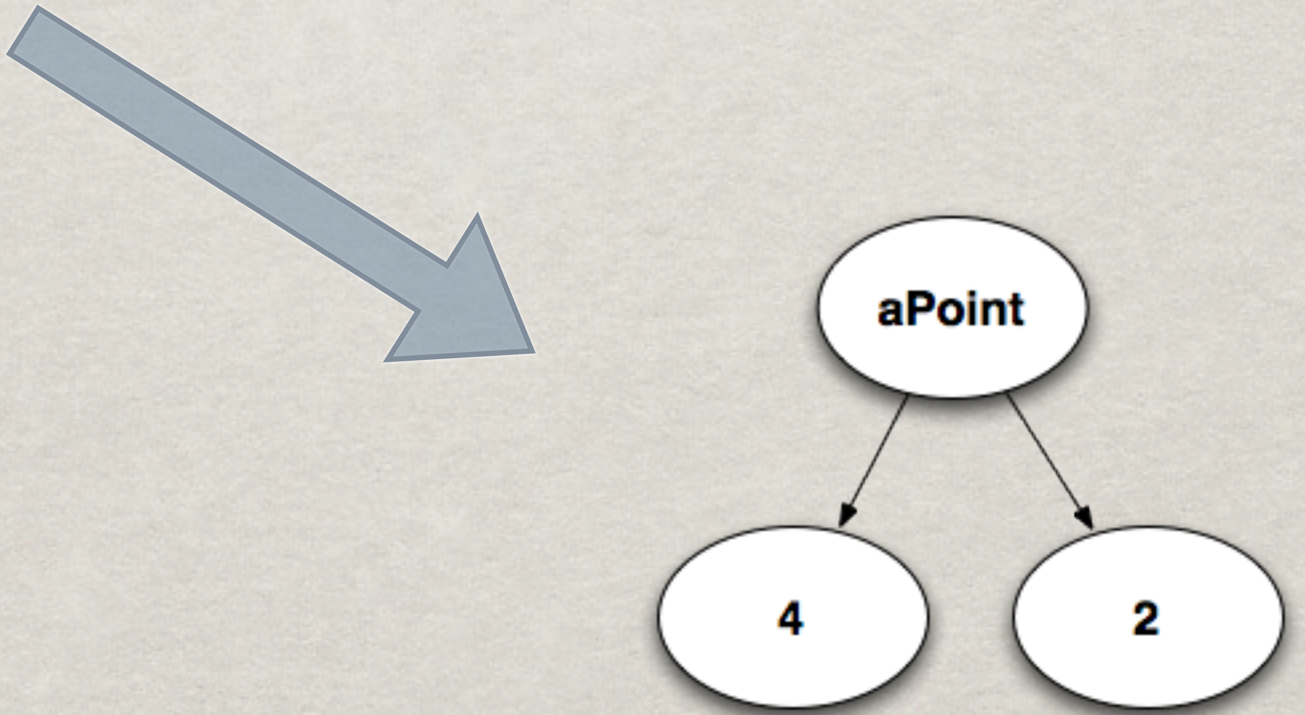
SummerTalk 2011

Student: Martin Dias
Mentor: Mariano Martínez Peck

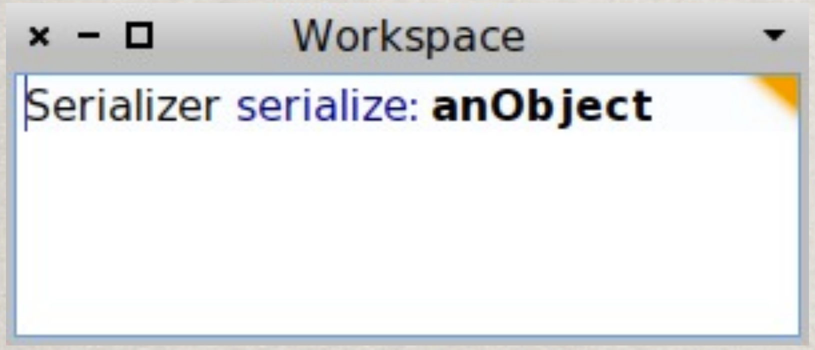
Object references



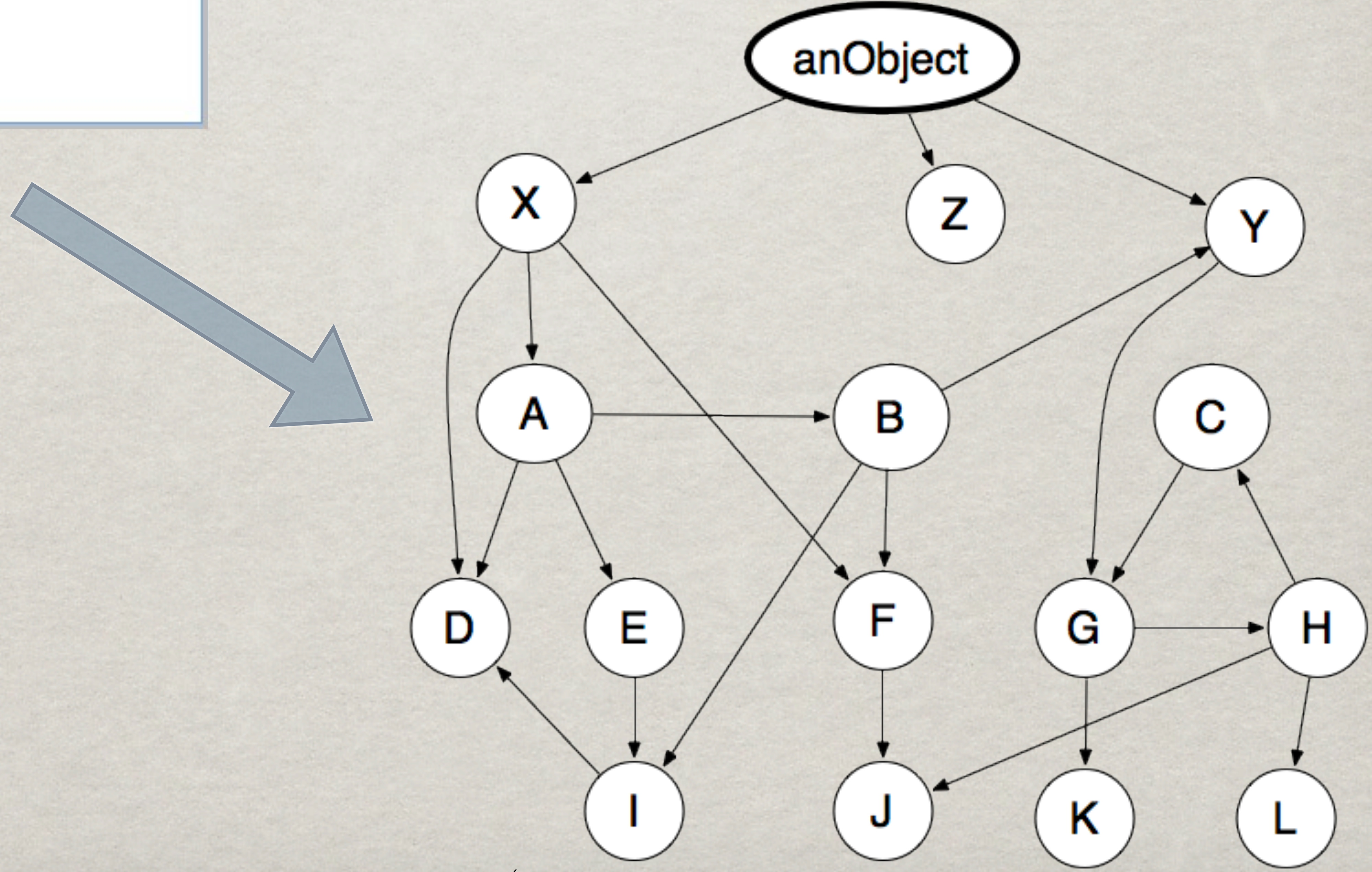
```
| aPoint |
aPoint := Point x: 4 y: 2.
```



Object graph



```
Serializer serialize: anObject
```

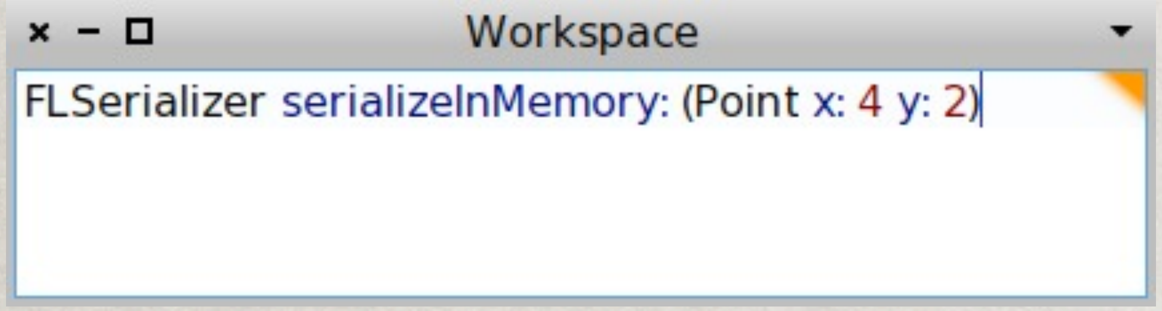


Fuel

An objet **graph** serializer.

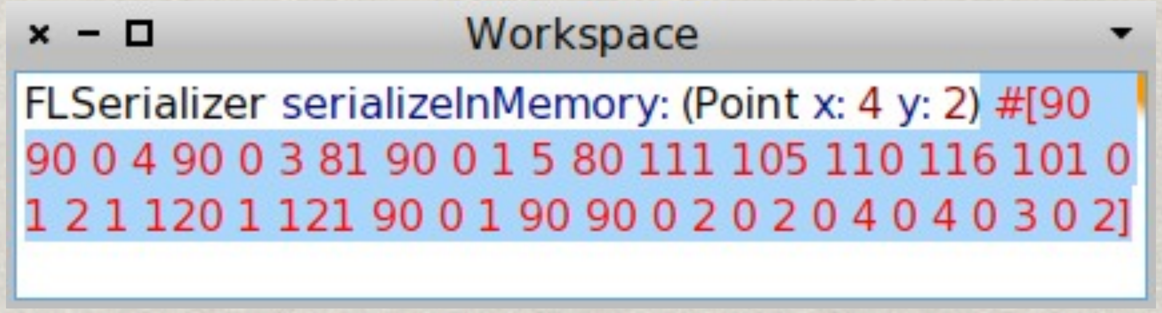
Serialize

Input: an object graph



```
FLSerializer serializeInMemory: (Point x: 4 y: 2)
```

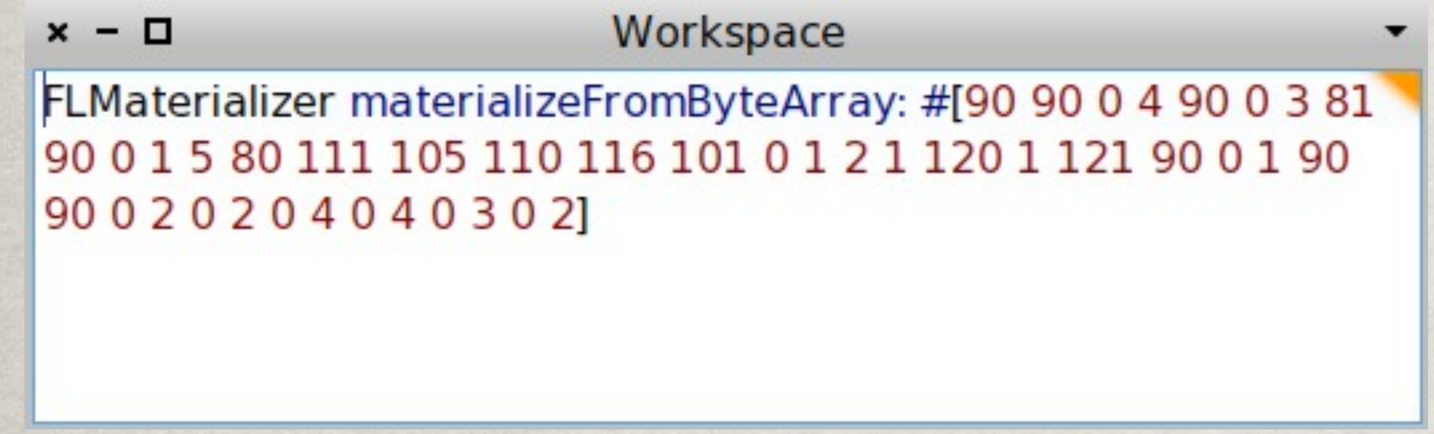
Output: stream of bytes



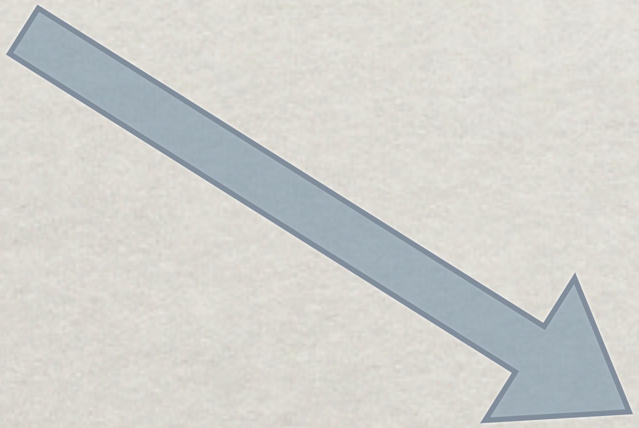
```
FLSerializer serializeInMemory: (Point x: 4 y: 2) #[90
90 0 4 90 0 3 81 90 0 1 5 80 111 105 110 116 101 0
1 2 1 120 1 121 90 0 1 90 90 0 2 0 2 0 4 0 4 0 3 0 2]
```

Materialize
(deserialize)

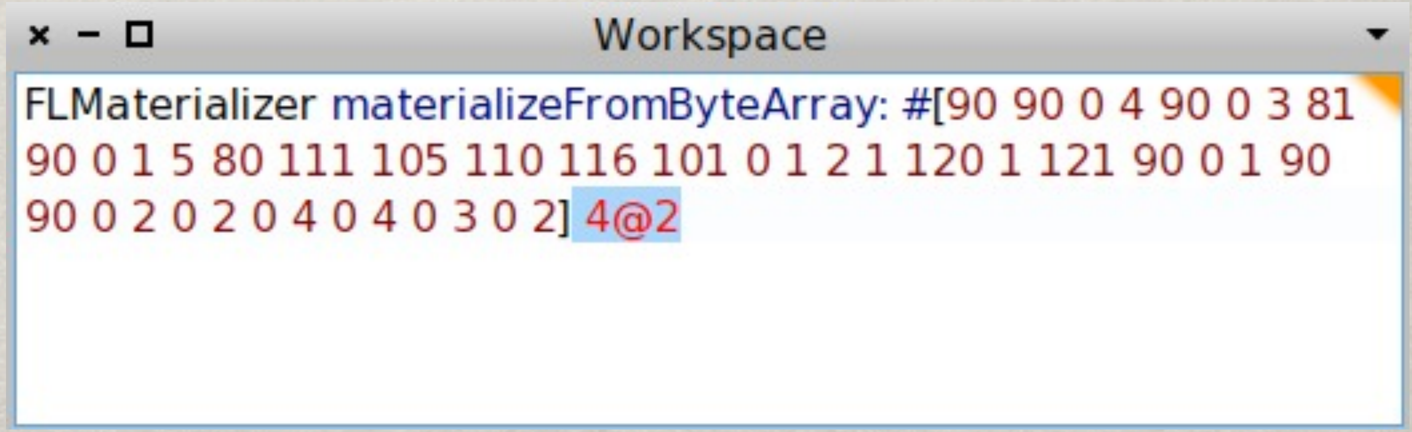
Input: stream of bytes



```
FLMaterializer materializeFromByteArray: #[90 90 0 4 90 0 3 81
90 0 1 5 80 111 105 110 116 101 0 1 2 1 120 1 121 90 0 1 90
90 0 2 0 2 0 4 0 4 0 3 0 2]
```

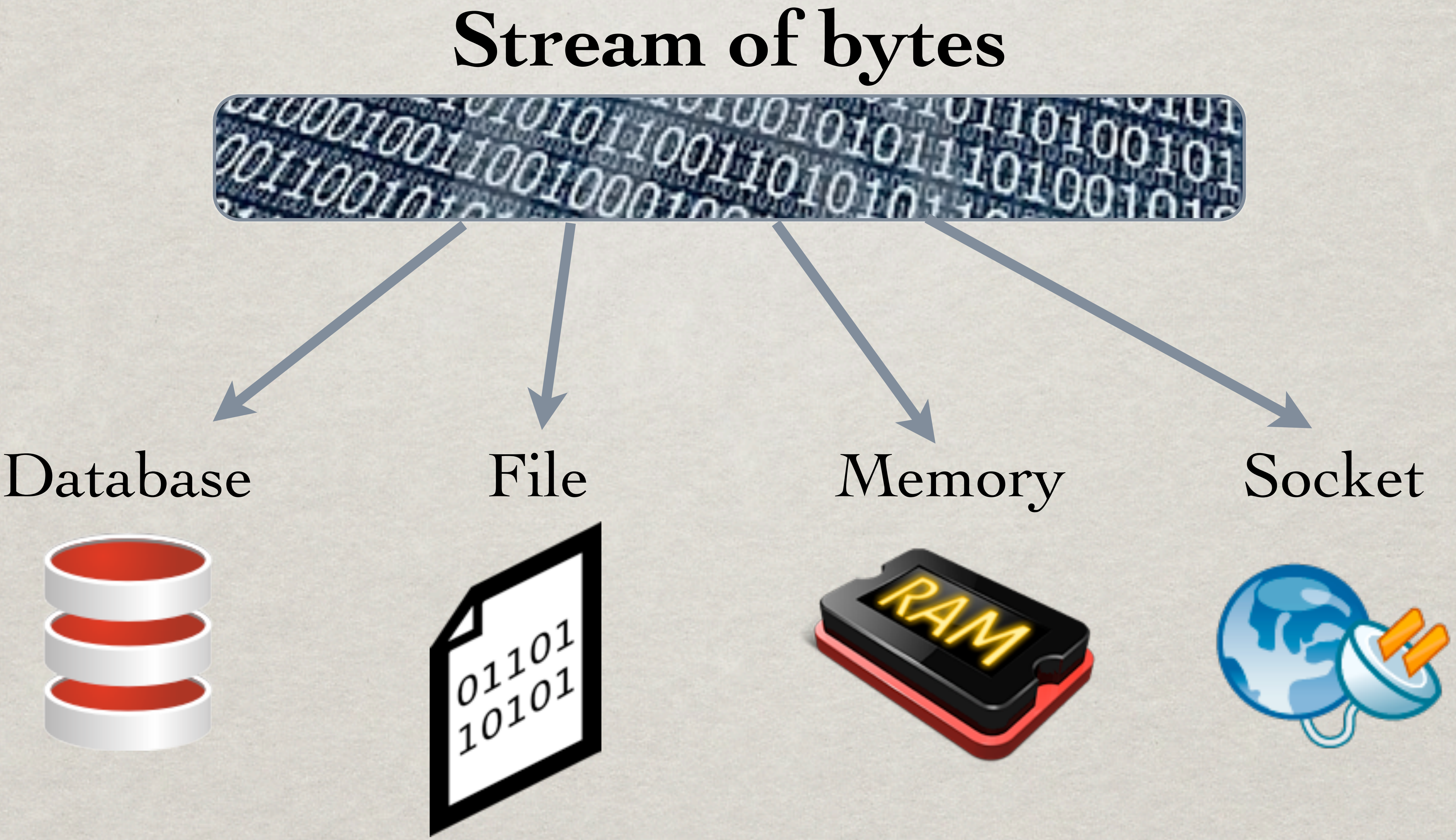


Output: an object graph



```
FLMaterializer materializeFromByteArray: #[90 90 0 4 90 0 3 81
90 0 1 5 80 111 105 110 116 101 0 1 2 1 120 1 121 90 0 1 90
90 0 2 0 2 0 4 0 4 0 3 0 2] 4@2
```

Once serialized...



Fuel's main goals

- Provide **fast** object serialization and materialization.
- Be flexible and easy to customize.
- Have a good OO design, well tested and benchmarked.
- No need of special support from the VM.
- Be a **general purpose** serializer.
- Allow tools to be built on top of Fuel.

Key features

- Fast serialization and materialization.
- Class reshape support.
- Serialization of any kind of object.
- Cycles support.
- Global objects references.
- Buffered writing.
- Support for some “hook methods”.

Key Characteristics

- **Pickle format**.
- Objects grouped in clusters.
- Analysis phase before writing.
- Stack over recursion.
- Two phases for writing instances and references.
- Iterative graph recreation.

Pickle format

Invest more time in serialization so that objects can then be materialized much faster.

Grouping objects in clusters

“Similar” objects (they share writing/loading information) are grouped together in clusters. The most common case, yet not the only one, takes place when a class is a cluster for its instances.

THÉ

Each jar has a specific type of element

Each jar has a specific type of element

Jars are in order

Each jar has a specific type of element

Jars are in order

Label:
- What's inside?
- How much?

Each jar has a specific type of element

Jars are in order

Label:
- What's inside?
- How much?

Different sizes and different amounts of elements

s/jar/cluster

Each jar has a specific type of element

Jars are in order

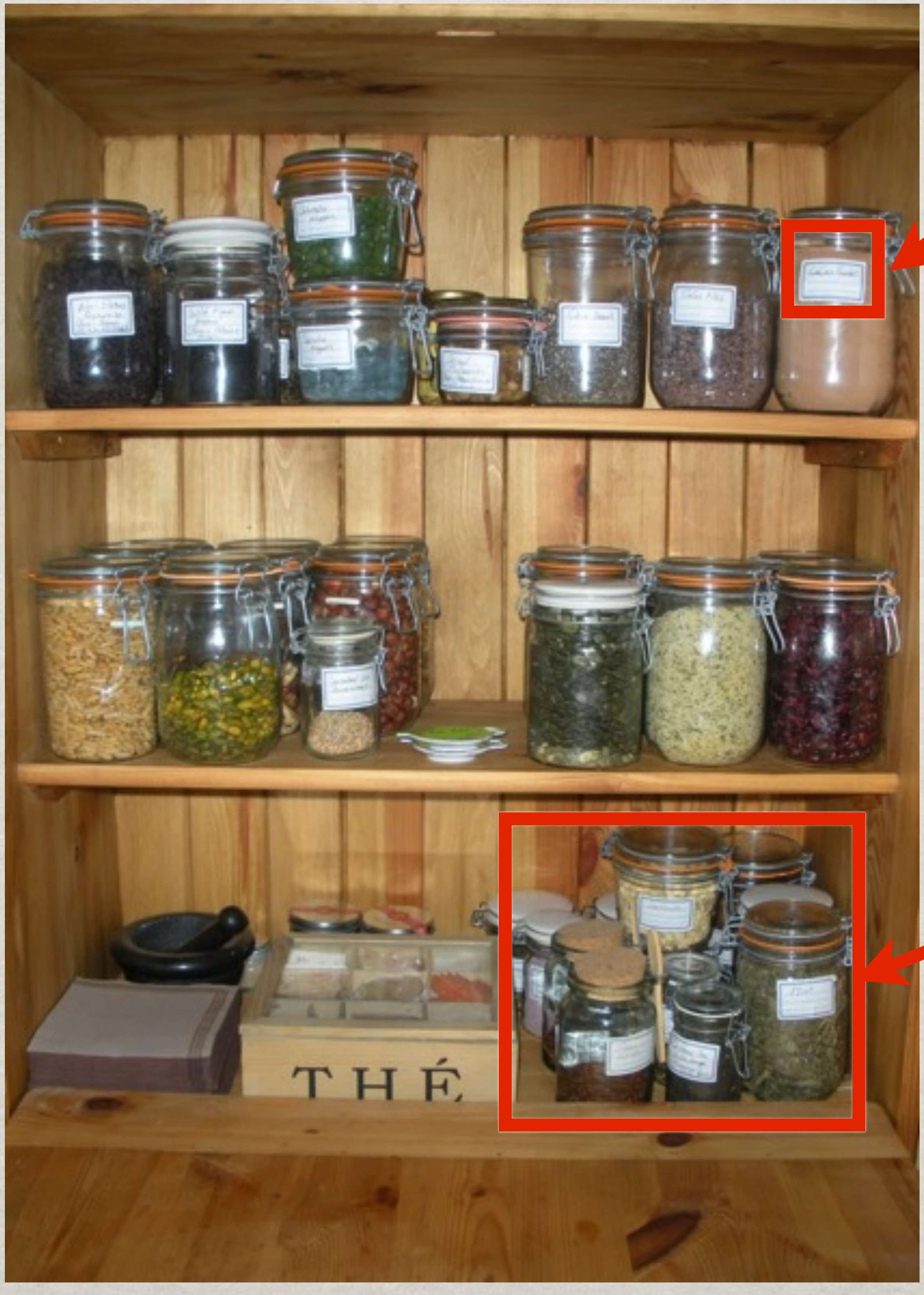
Label:
- What's inside?
- How much?

Different sizes and different amounts of elements

s/jar/cluster

Each jar has a specific type of element

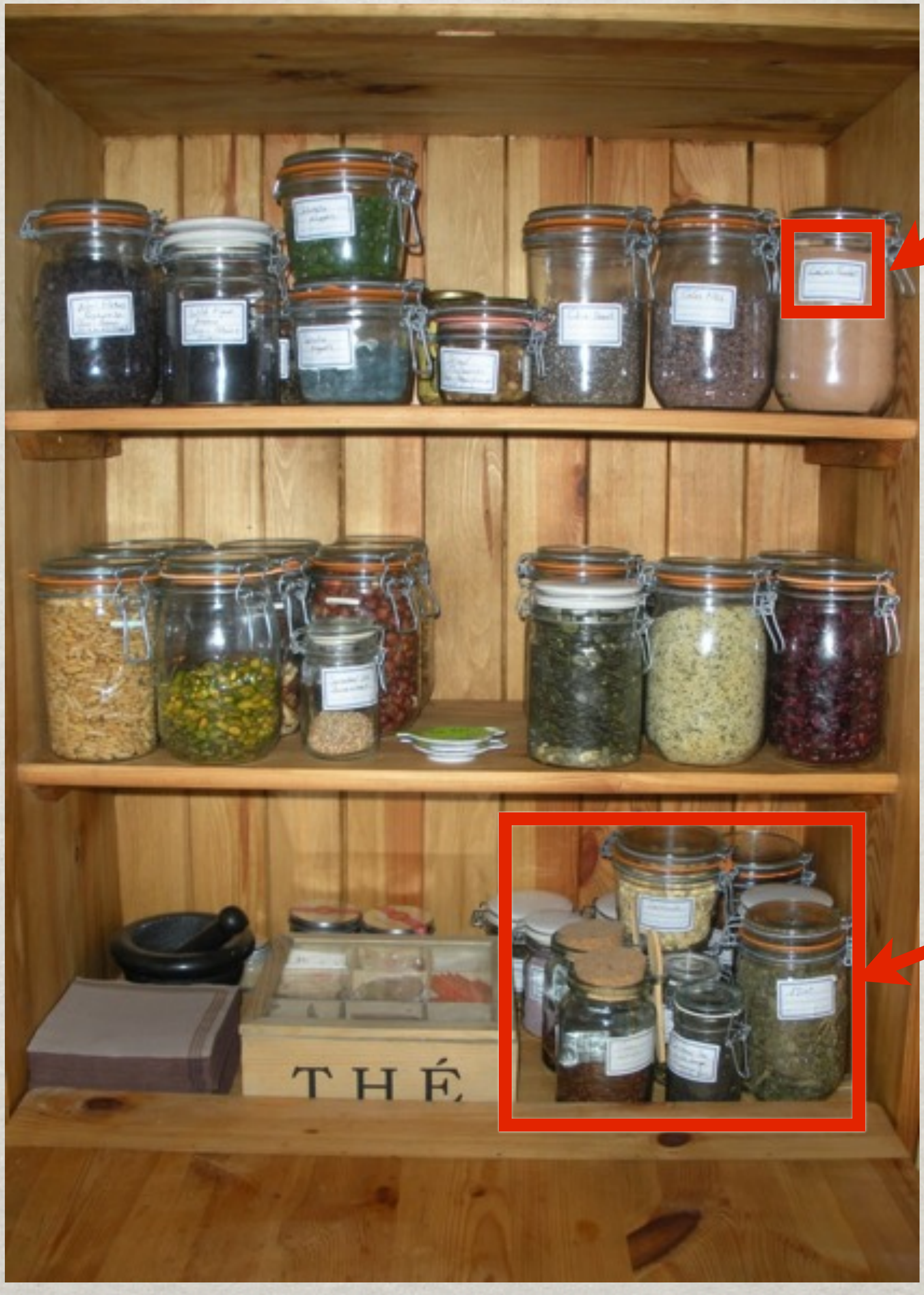
Jars are in order



Label:
- What's inside?
- How much?

Different sizes and different amounts of elements

s/jar/cluster



Each cluster has a specific type of object

Jars are in order

Label:
- What's inside?
- How much?

Different sizes and different amounts of elements

s/jar/cluster

Each cluster has a specific type of object

Jars are in order

Label:
- What's inside?
- How much?

Different sizes and different amounts of elements

s/jar/cluster

__Label__:
- What's inside?
- How much?

Each cluster has a specific type of object

Clusters are in order

Different sizes and different amounts of elements

s/jar/cluster

Each cluster has a specific type of object

Clusters are in order

<u>Label</u>:
- What's inside?
- How much?

Different sizes and different amounts of elements

s/jar/cluster

Each cluster has a specific type of object

Clusters are in order

Label:
- Cluster ID
- Amount of objects

Different sizes and different amounts of elements

s/jar/cluster

Each cluster has a specific type of object

Clusters are in order

<u>Label</u>:
- Cluster ID
- Amount of objects

Different sizes and different amounts of elements

s/jar/cluster

Each cluster has a specific type of object

Clusters are in order

Label:
- Cluster ID
- Amount of objects

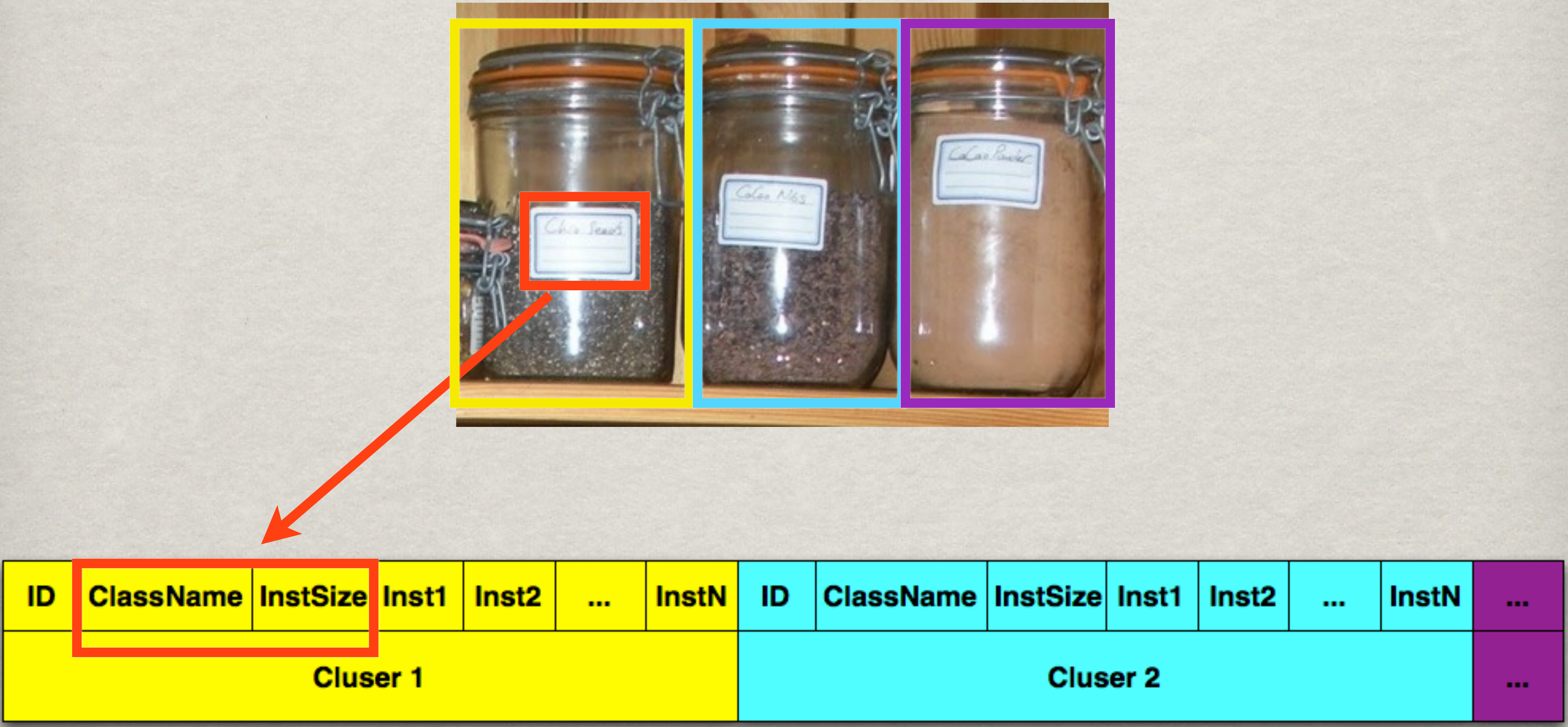
Different sizes and different amounts of objects

Pickle format basic

ID	ClassName	InstSize	Inst1	Inst2	...	InstN	ID	ClassName	InstSize	Inst1	Inst2	...	InstN	...
Cluser 1							Cluser 2							...

Stream

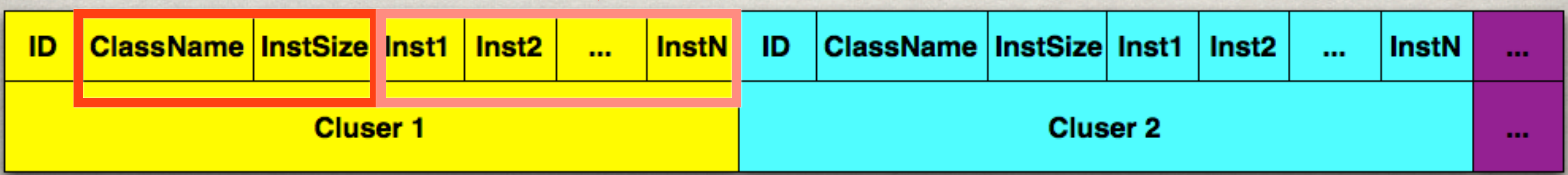
Pickle format basic



ID	ClassName	InstSize	Inst1	Inst2	...	InstN	ID	ClassName	InstSize	Inst1	Inst2	...	InstN	...
Cluser 1							Cluser 2							...

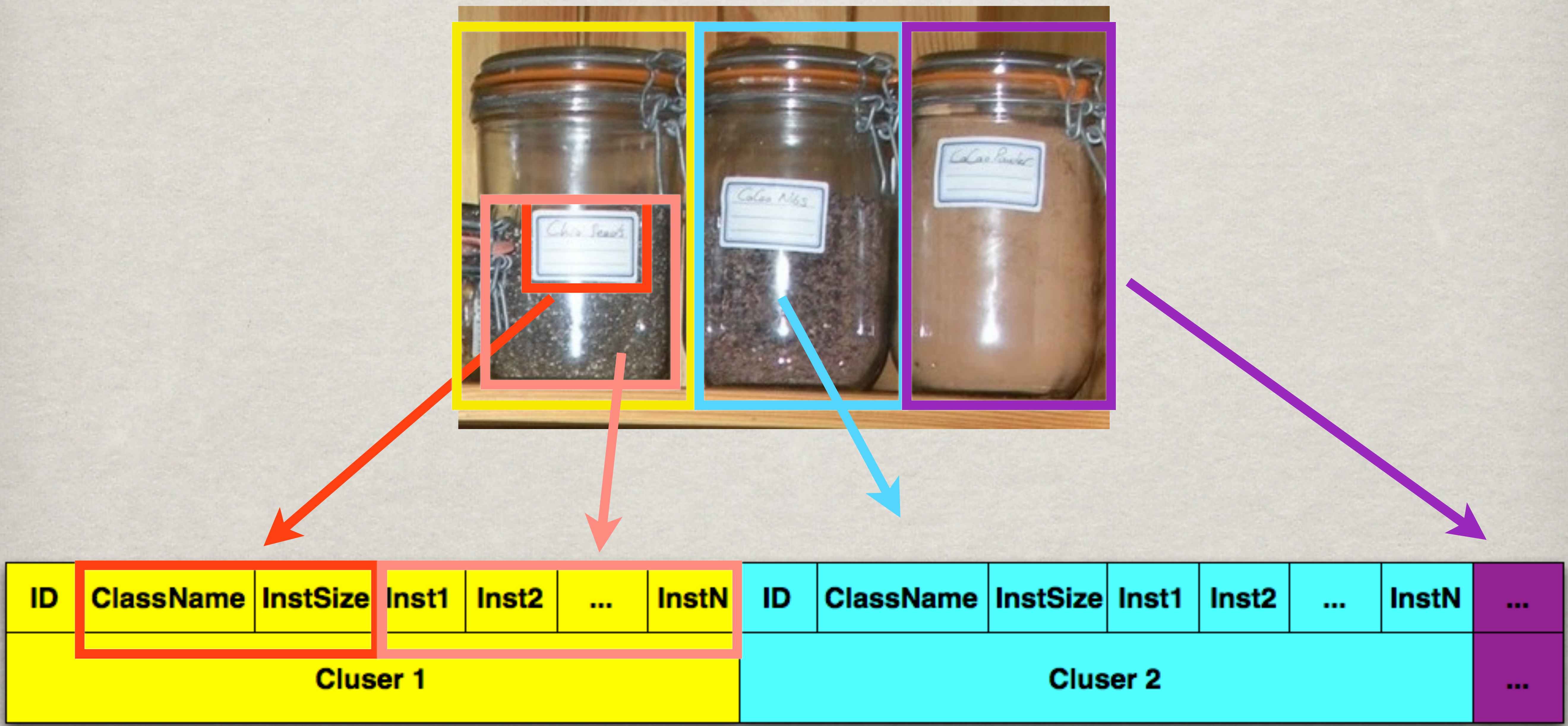
Stream

Pickle format basic



Stream

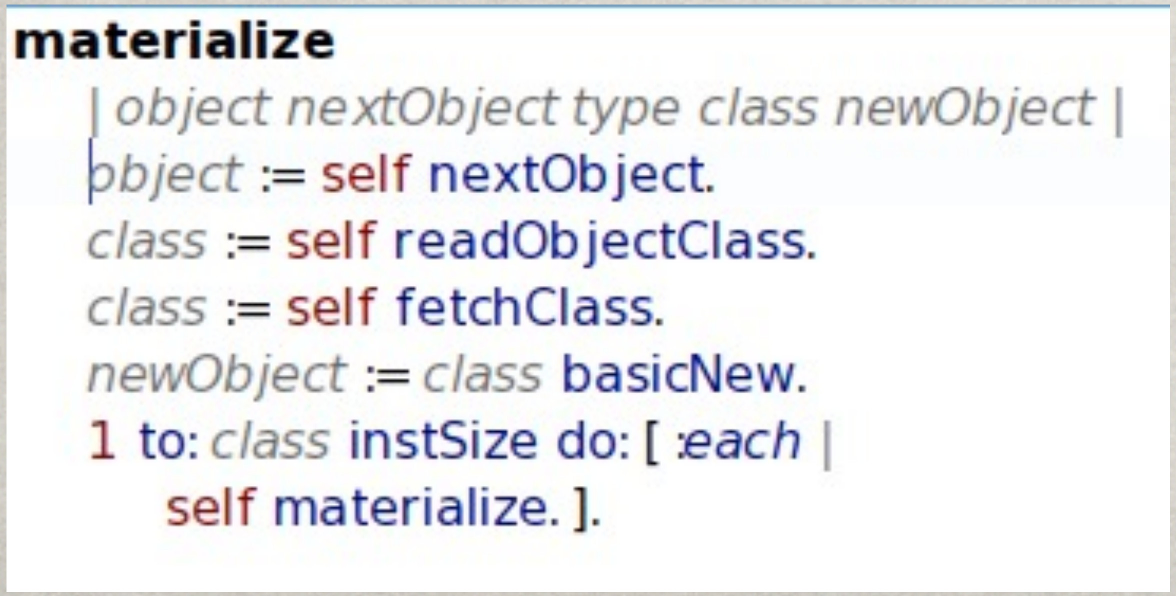
Pickle format basic



Stream

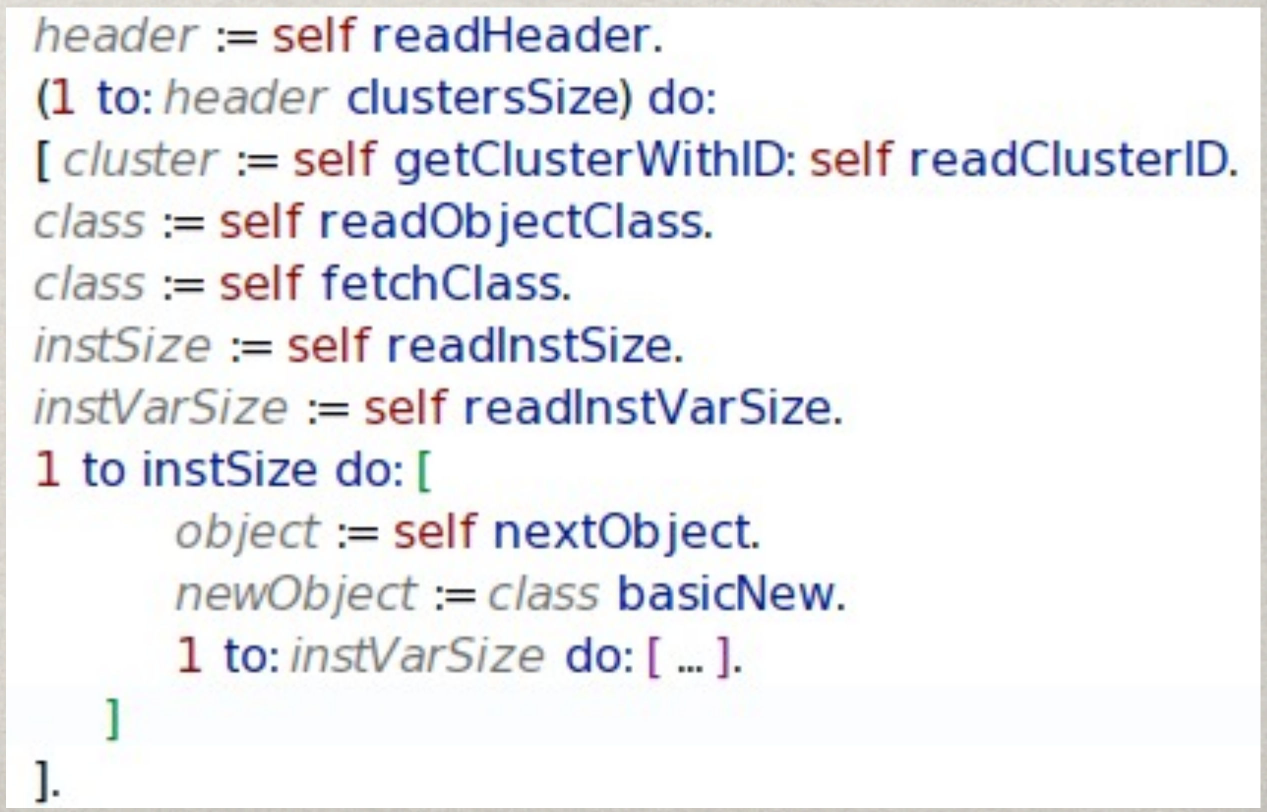
Why the pickle format is so fast in materialization?

Standard serializers



```
materialize
    | object nextObject type class newObject |
    object := self nextObject.
    class := self readObjectClass.
    class := self fetchClass.
    newObject := class basicNew.
    1 to: class instSize do: [ :each |
        self materialize. ].
```

Fuel pickle format

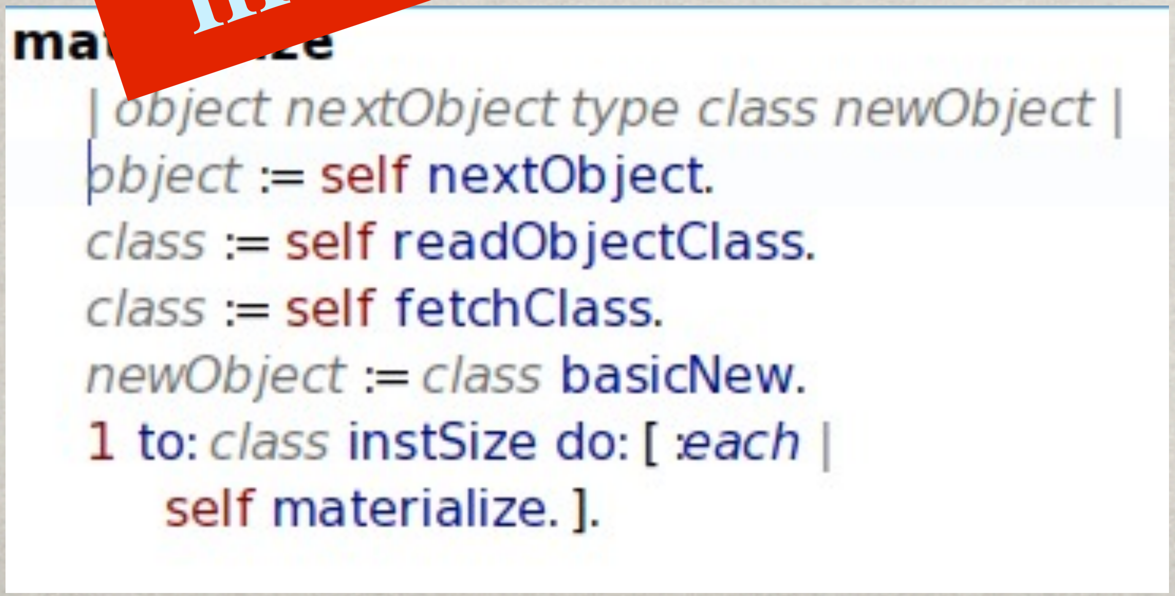


```
header := self readHeader.
(1 to: header clustersSize) do:
[ cluster := self getClusterWithID: self readClusterID.
class := self readObjectClass.
class := self fetchClass.
instSize := self readInstSize.
instVarSize := self readInstVarSize.
1 to instSize do: [
        object := self nextObject.
        newObject := class basicNew.
        1 to: instVarSize do: [ ... ].
    ]
].
```

Why the pickle format is so fast in materialization?

Standard serializers

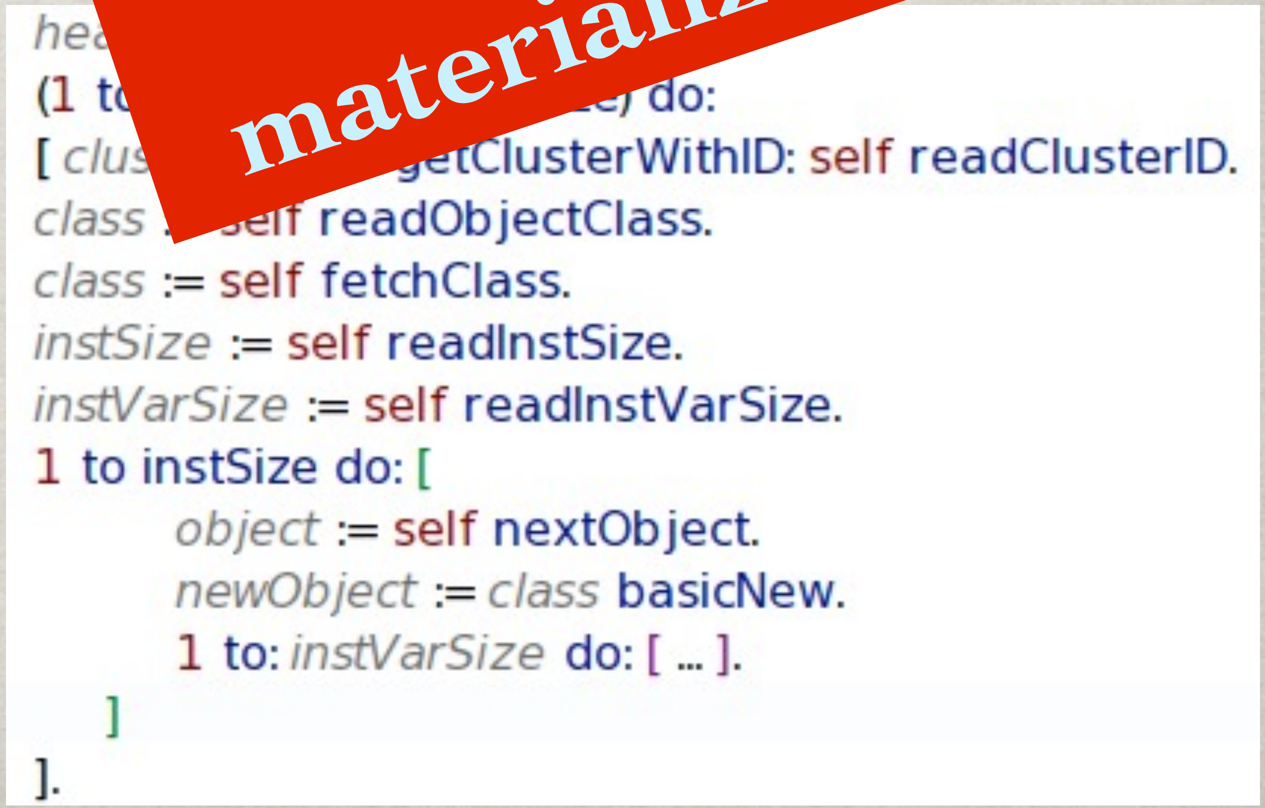
Recursive materialization



```
ma    ze
    | object nextObject type class newObject |
    object := self nextObject.
    class := self readObjectClass.
    class := self fetchClass.
    newObject := class basicNew.
    1 to: class instSize do: [ :each |
        self materialize. ].
```

Fuel pickle format

Iterative materialization

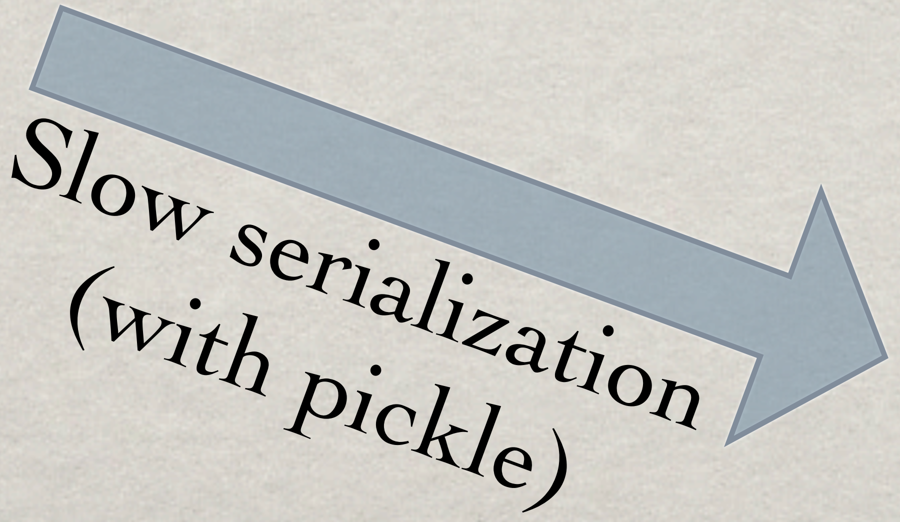


```
hea
(1 to                  e) do:
[ clus           getClusterWithID: self readClusterID.
class            self readObjectClass.
class := self fetchClass.
instSize := self readInstSize.
instVarSize := self readInstVarSize.
1 to instSize do: [
        object := self nextObject.
        newObject := class basicNew.
        1 to: instVarSize do: [ ... ].
    ]
].
```

Pickle advantages

- Batch/Bulk/Iterative materialization.
- Efficient since types are stored and fetch only once.
- Fast because at materialization we know the size of everything.
- The generated stream is smaller.
- More next....

There is no silver bullet...

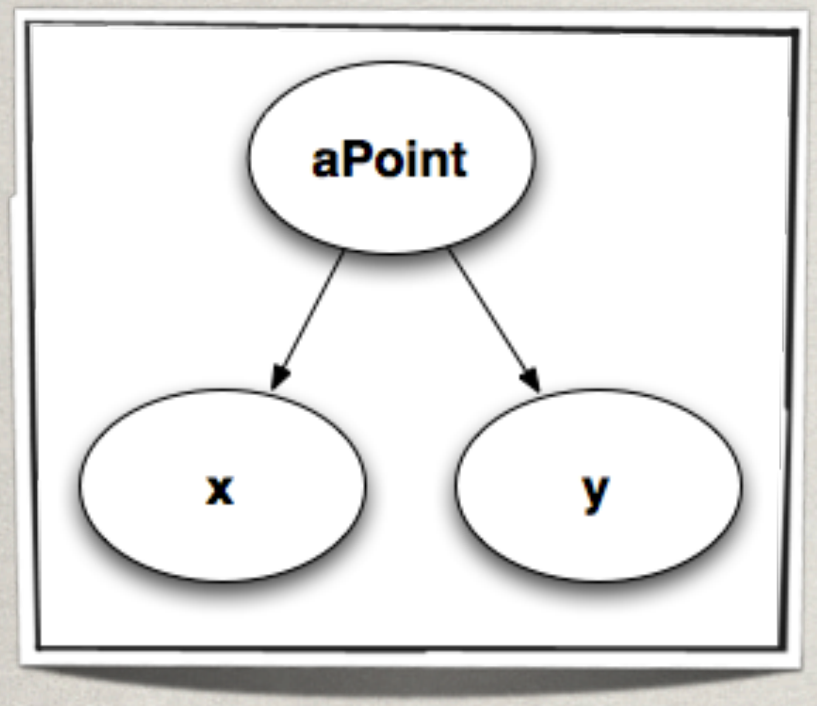


Fuel requires

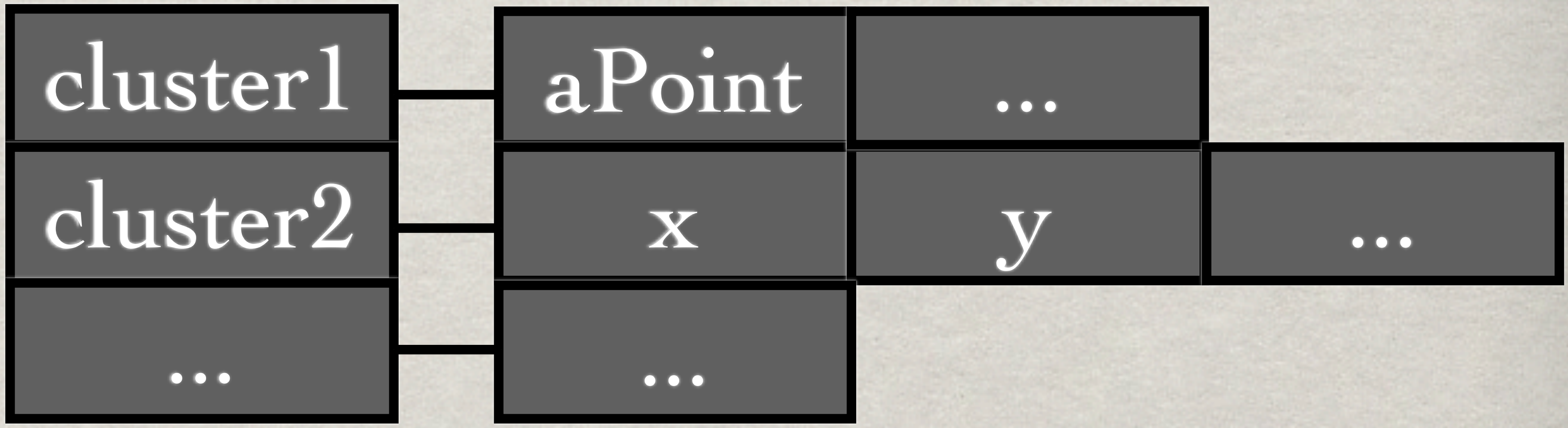
- Traversing the object graph.
- Mapping each object to a specific cluster.

This is done in a phase before serialization called “Analysis”.

Analysis phase

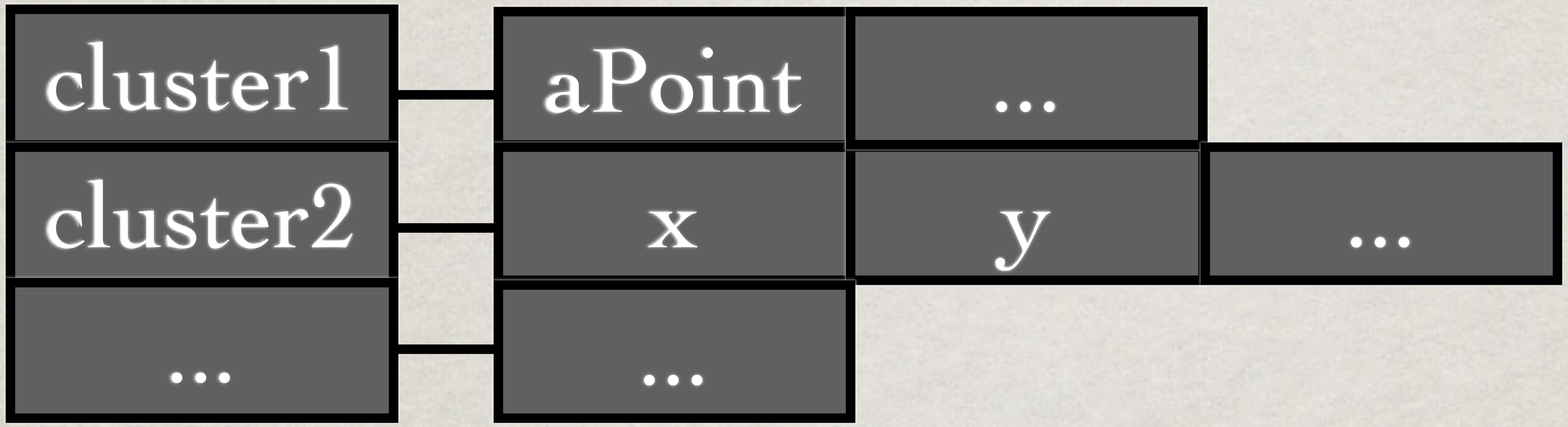


1) Traverse (#fuelAccept: aVisitor)
2) Fill dictionary (#fuelSerializer)

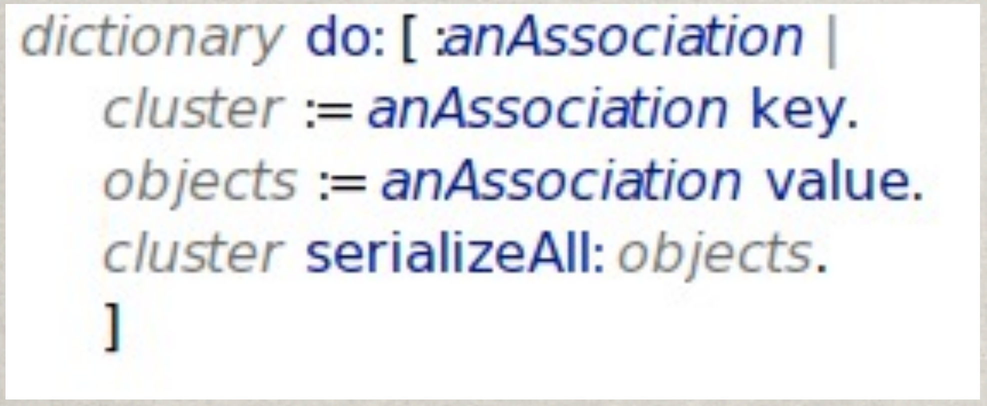


key (a cluster) value (a set)

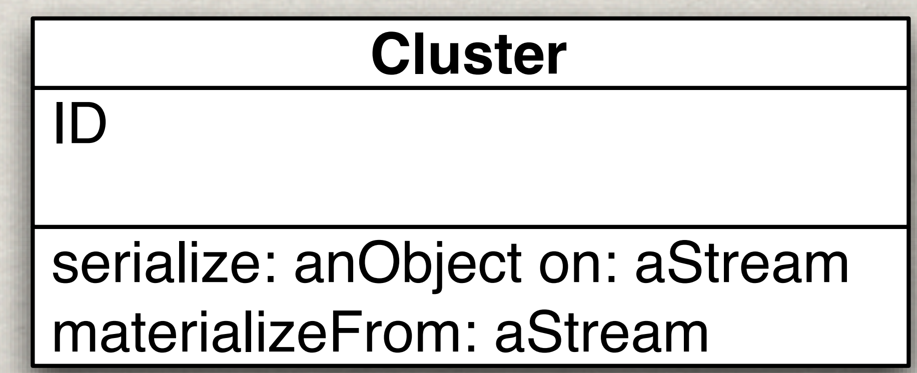
Serialization



key (a cluster) value (a set)

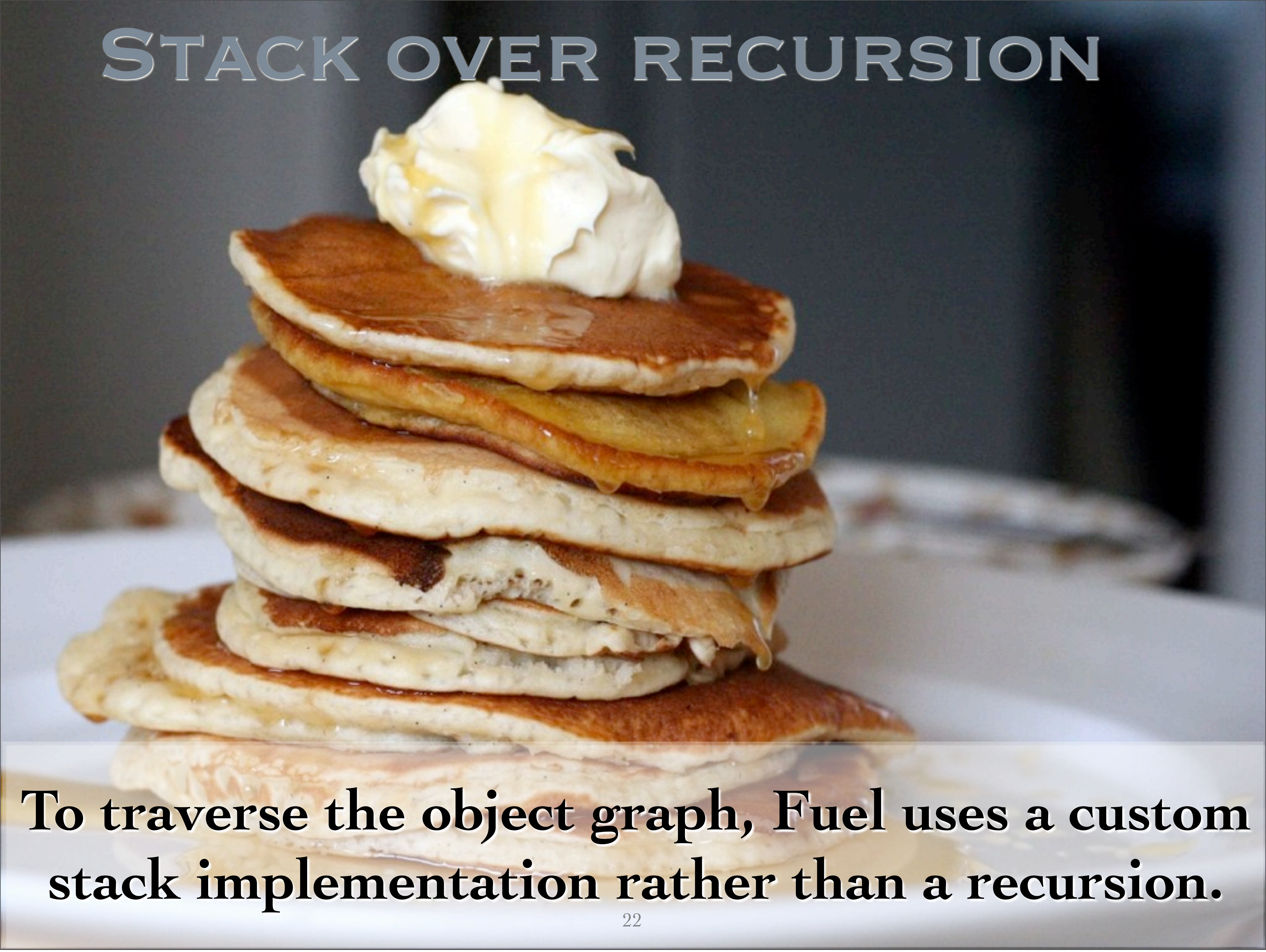


```
dictionary do: [ :anAssociation |
    cluster := anAssociation key.
    objects := anAssociation value.
    cluster serializeAll: objects.
    ]
```



A cluster defines how its objects are serialized and materialized.

Stack over recursion



To traverse the object graph, Fuel uses a custom stack implementation rather than a recursion.

Basic steps

Serialization

1. Analyze.
2. Serialize header.
3. Serialize instances.
4. Serialize references.
5. Serialize root.

Materialization

1. Materialize header.
2. Materialize instances.
3. Materialize references.
4. Materialize root.

Fuel for software
(so far)

- Moose export utility.
- SandstoneDB persistence.
- Pier kernel persistence.
- Newspeak language.
- Marea (my own research project!).

Future Work

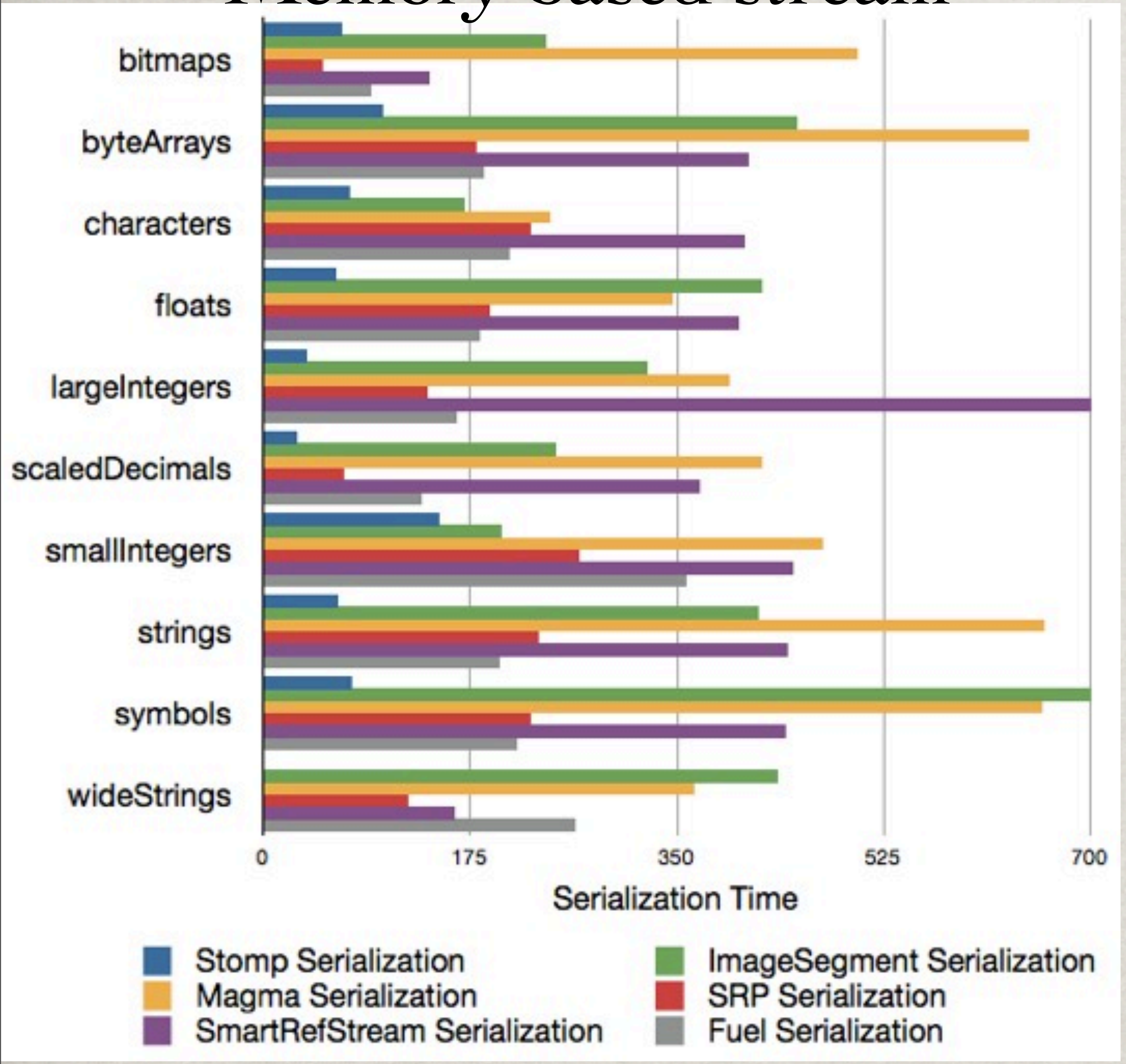
- Continue efforts on performance optimization.
- Create a tool for loading class and trait packages.
- Support user-defined Singletons.
- Fast statistics/brief info extraction of a stored graph.
- Partial loading of a stored graph.

Future work 2

- Enable to deploy serialization and materialization behavior independently.
- Support object replacement for serialization and materialization.
- Allow cycle detections to be disabled.
- Partial loading.

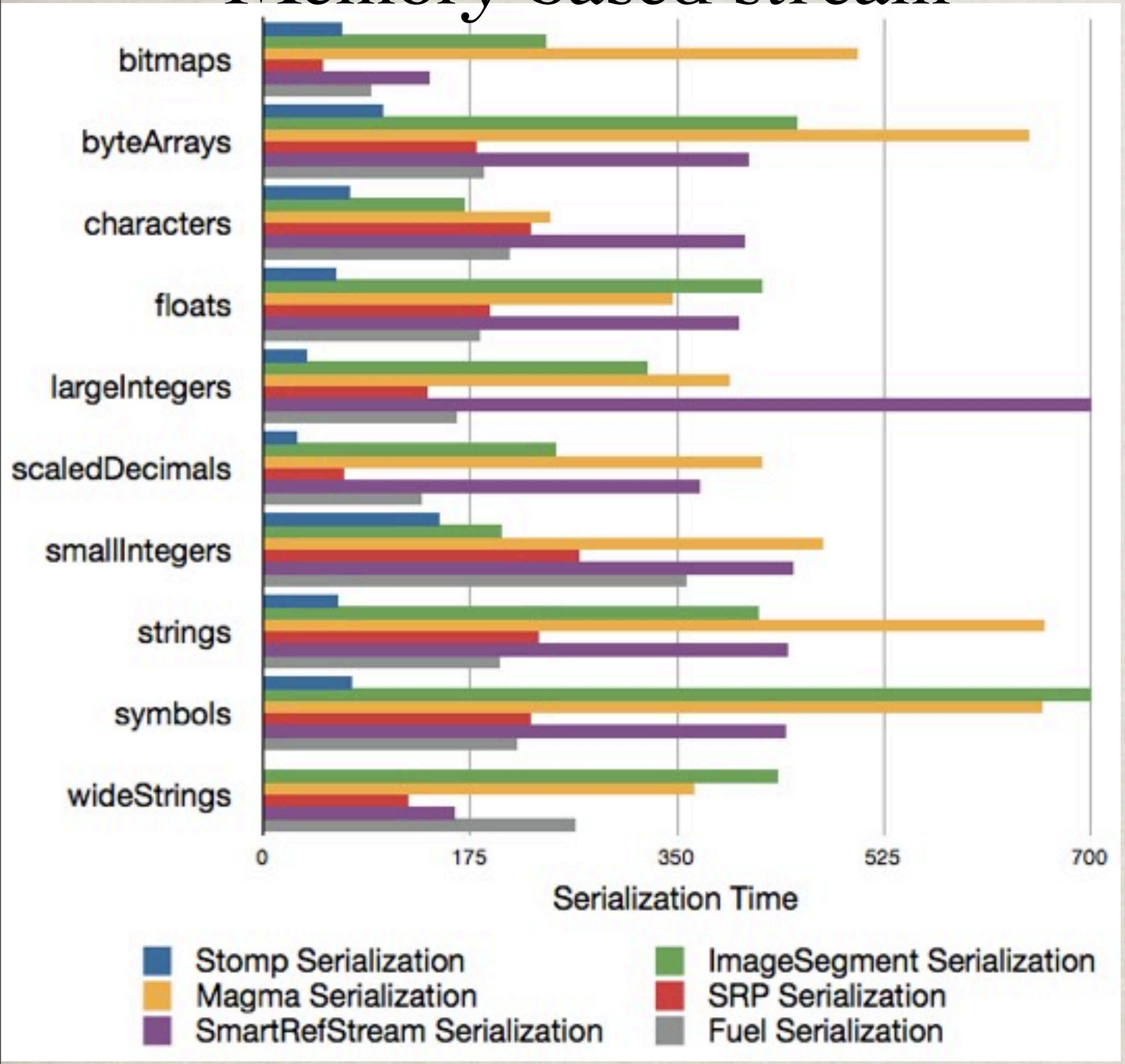
Serialization of primitive objects

Memory based stream



Serialization of primitive objects

Memory based stream



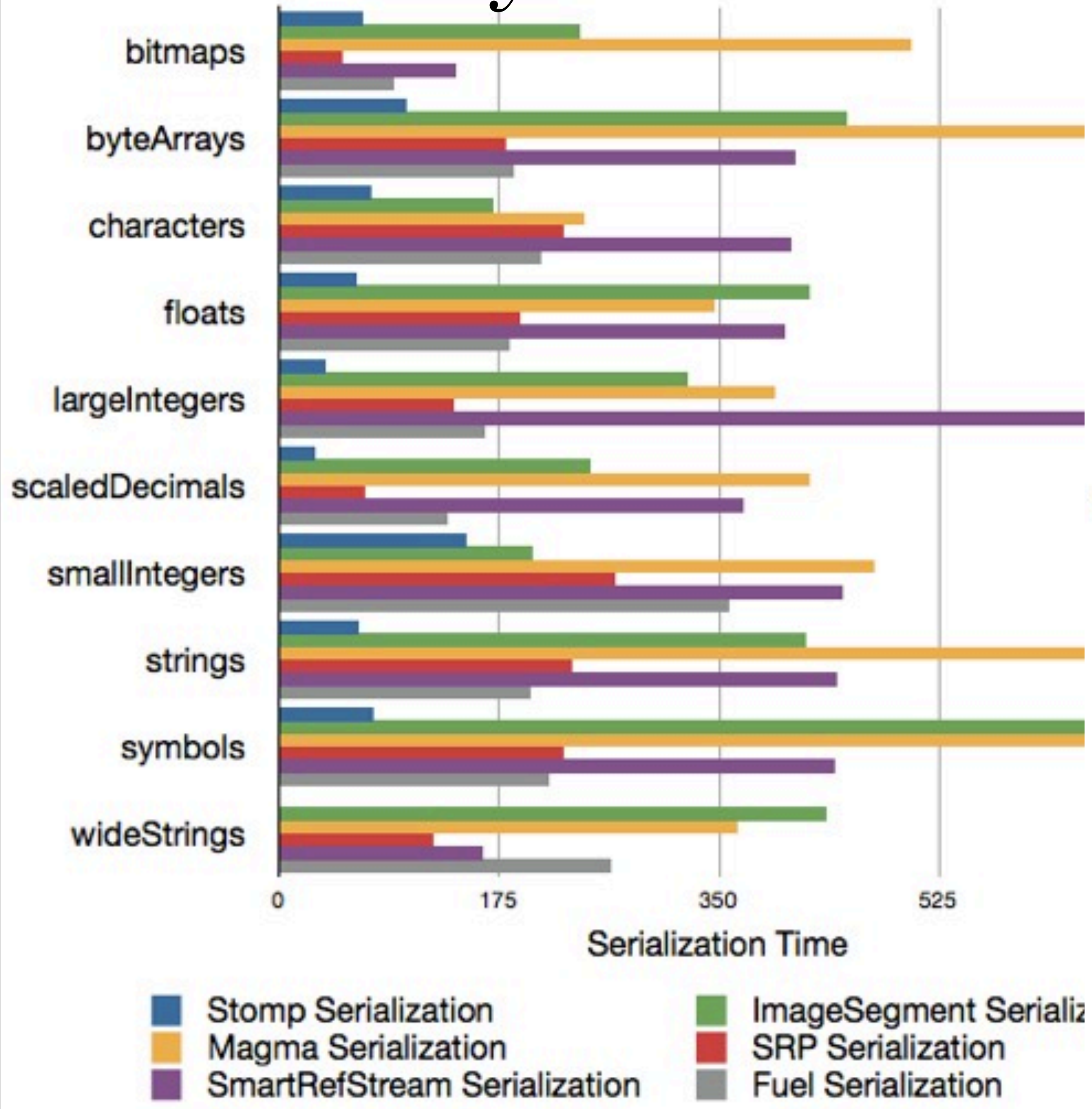
StOMP

SRP

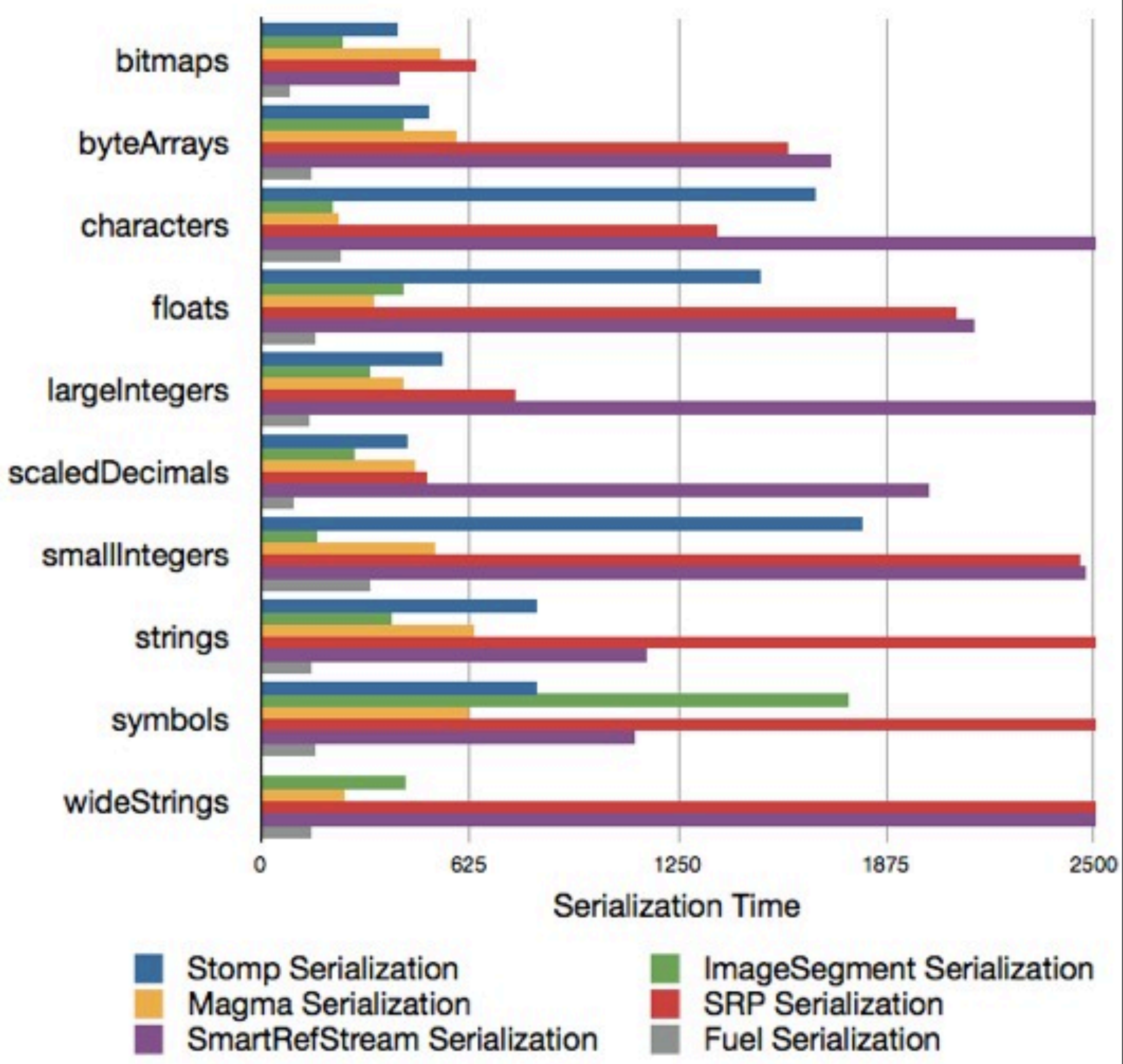
Fuel

Serialization of primitive objects

Memory based stream



File based stream



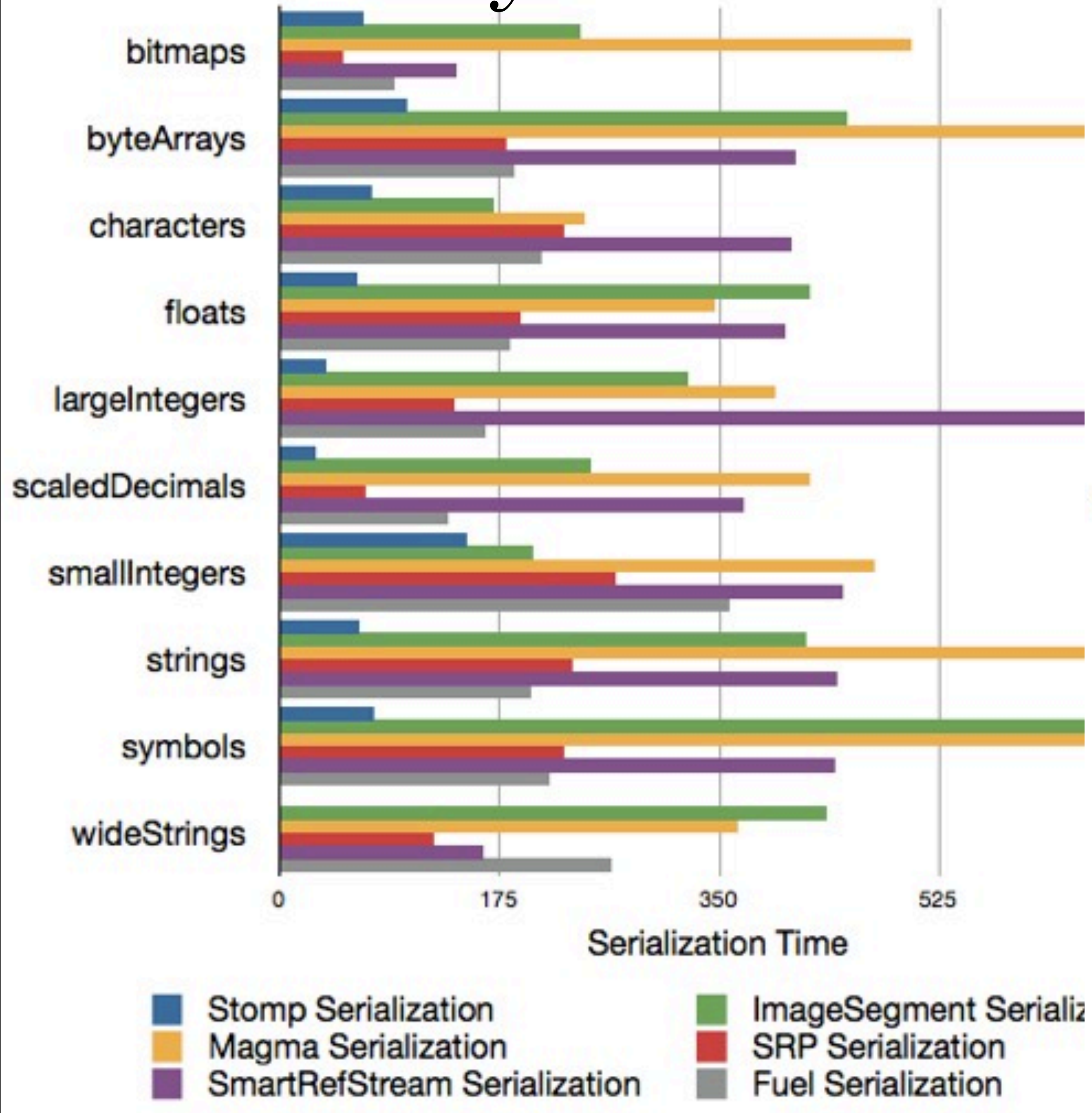
StOMP

SRP

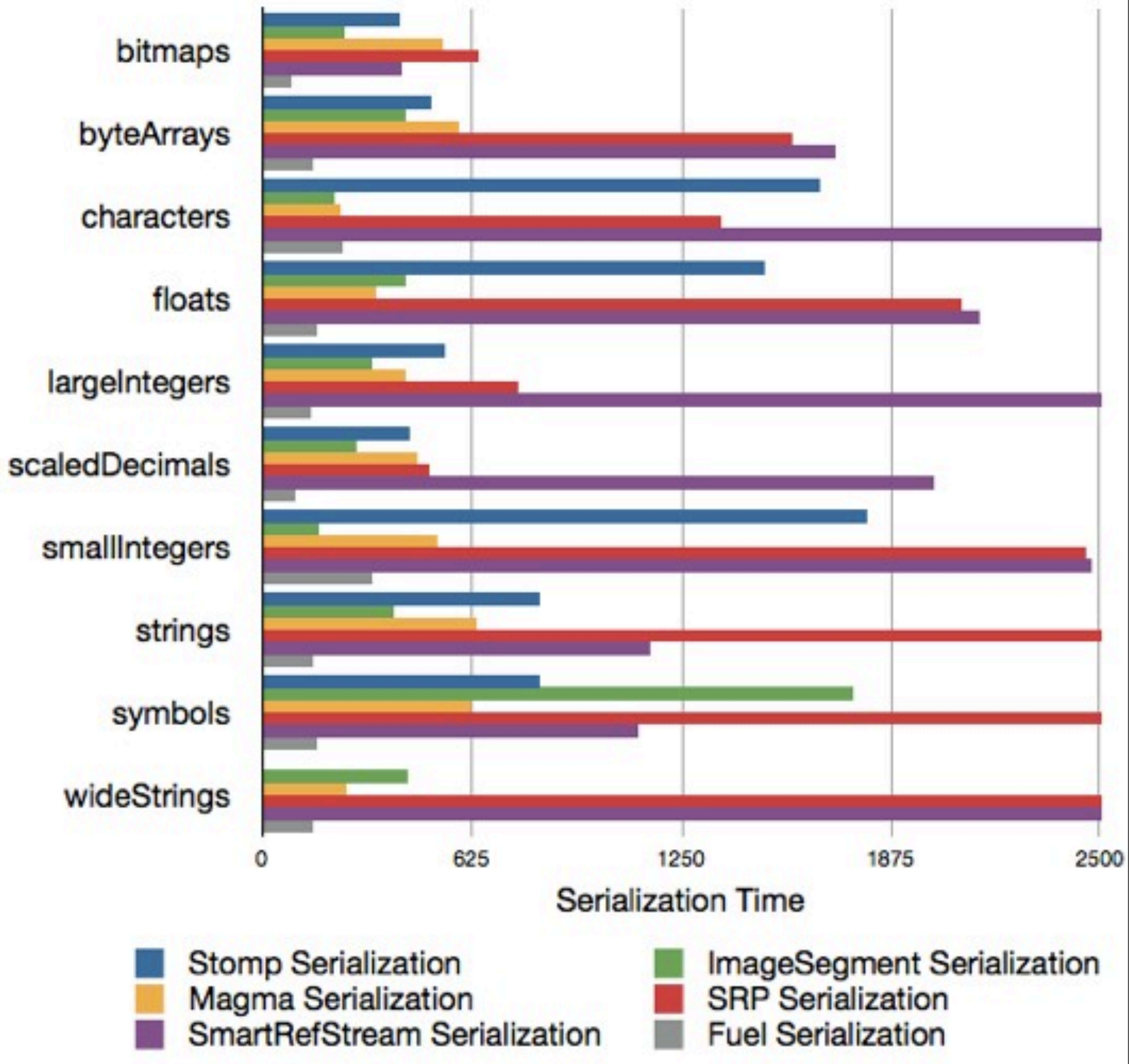
Fuel

Serialization of primitive objects

Memory based stream



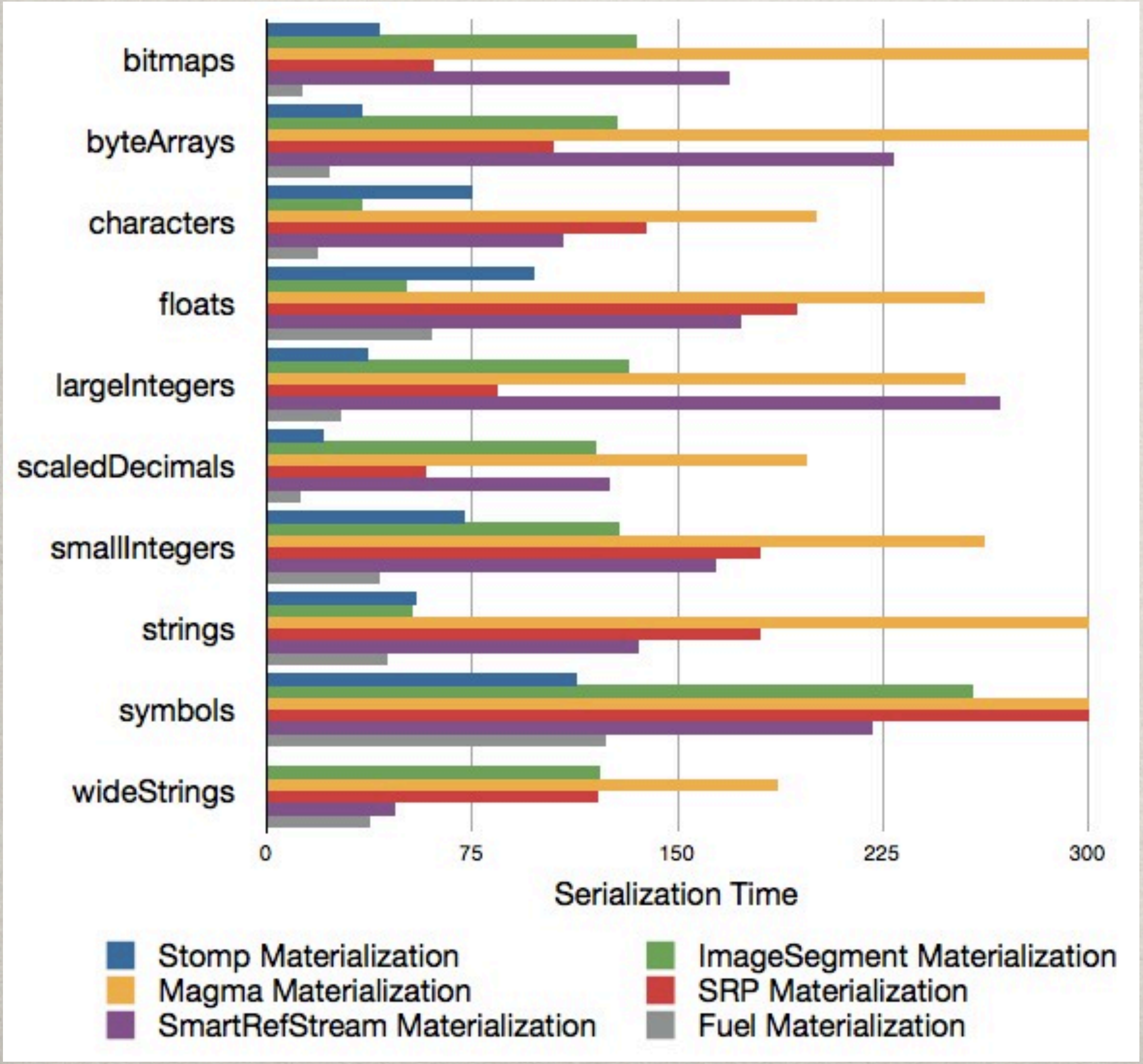
File based stream



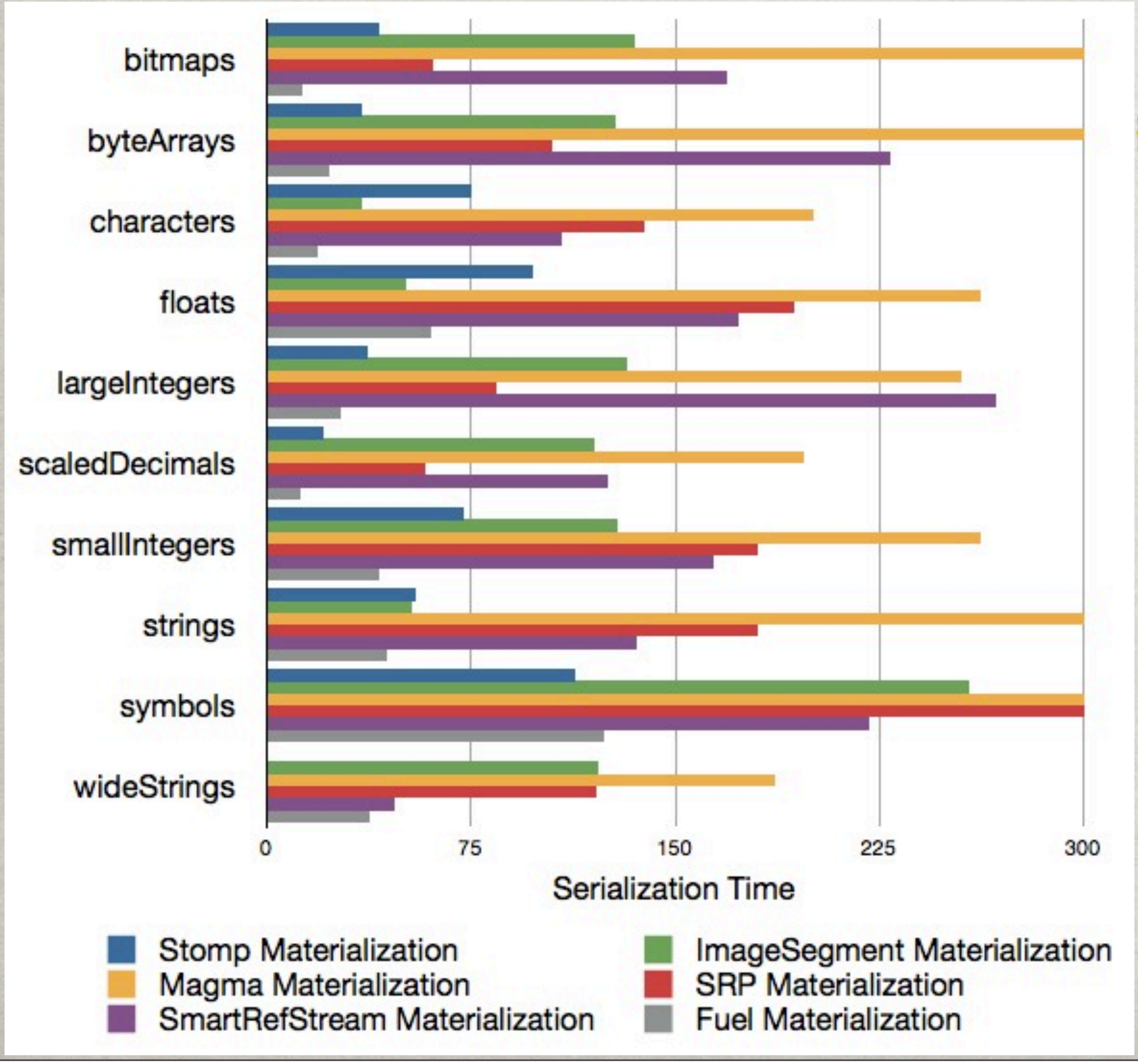
1. StOMP
2. SRP
3. Fuel

1. Fuel
2. SRP
3. StOMP

Materialization of primitive objects



Materialization of primitive objects

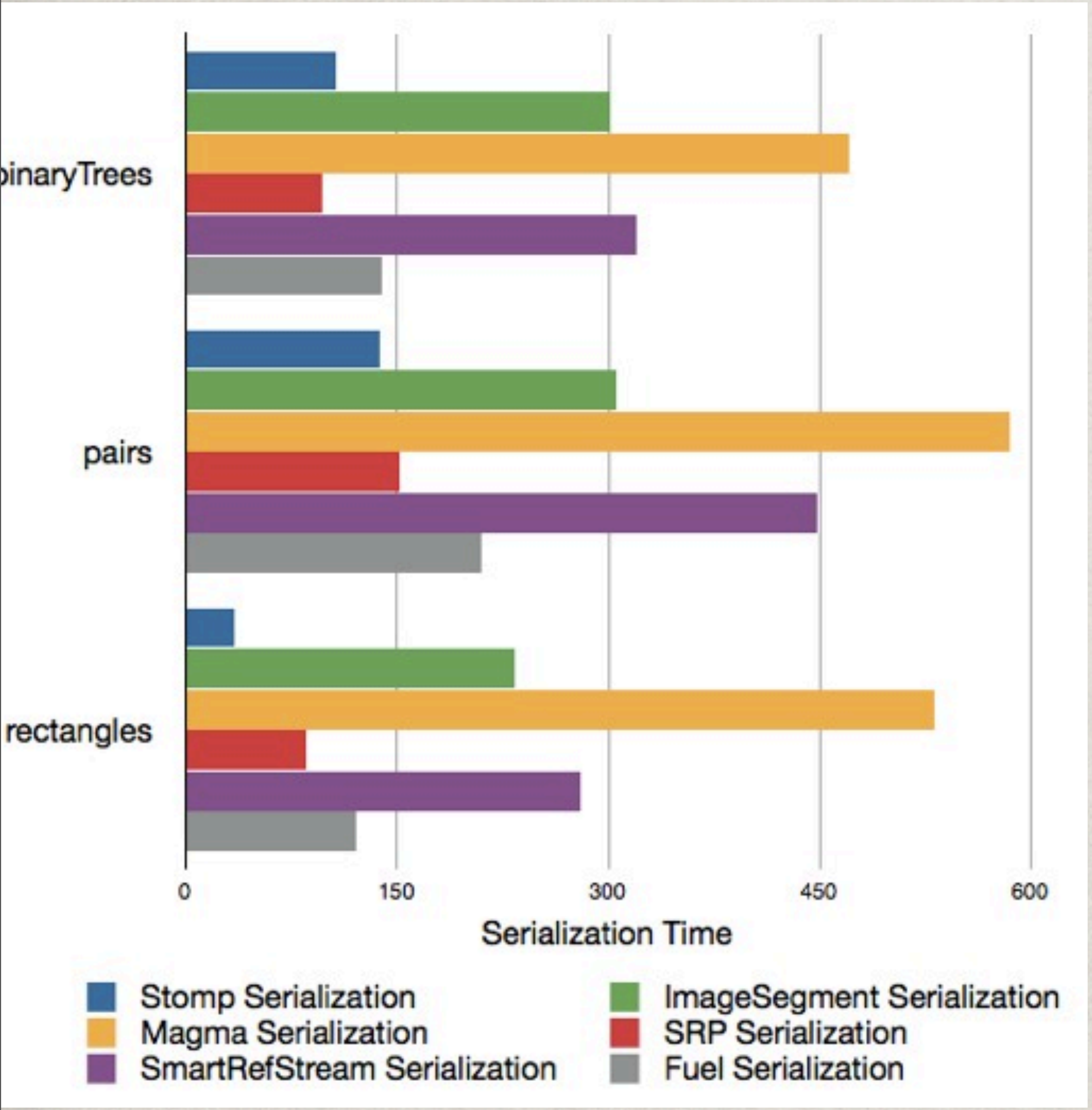


Fuel

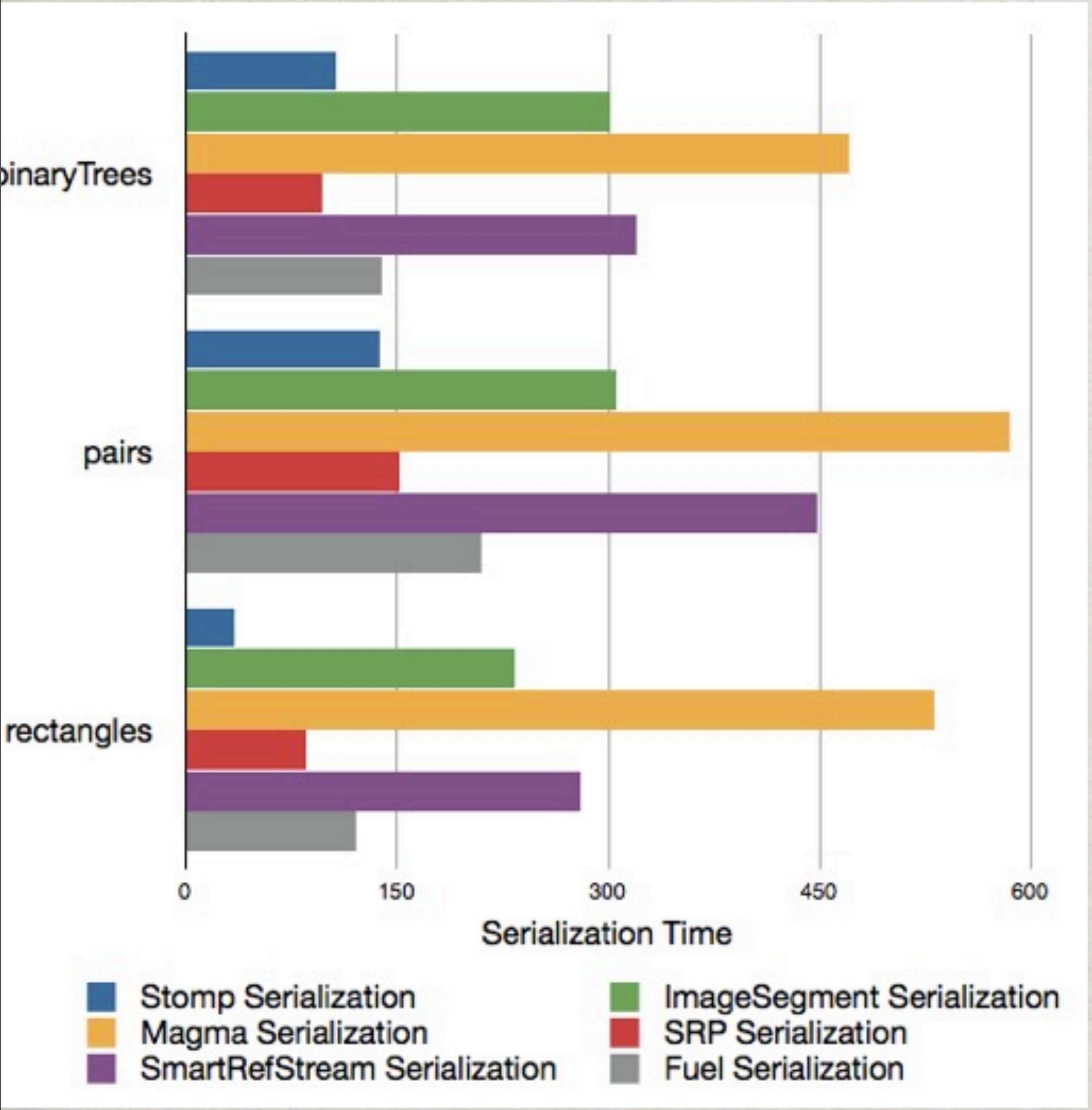
StOMP

ImageSegment

Non primitive objects



Non primitive objects

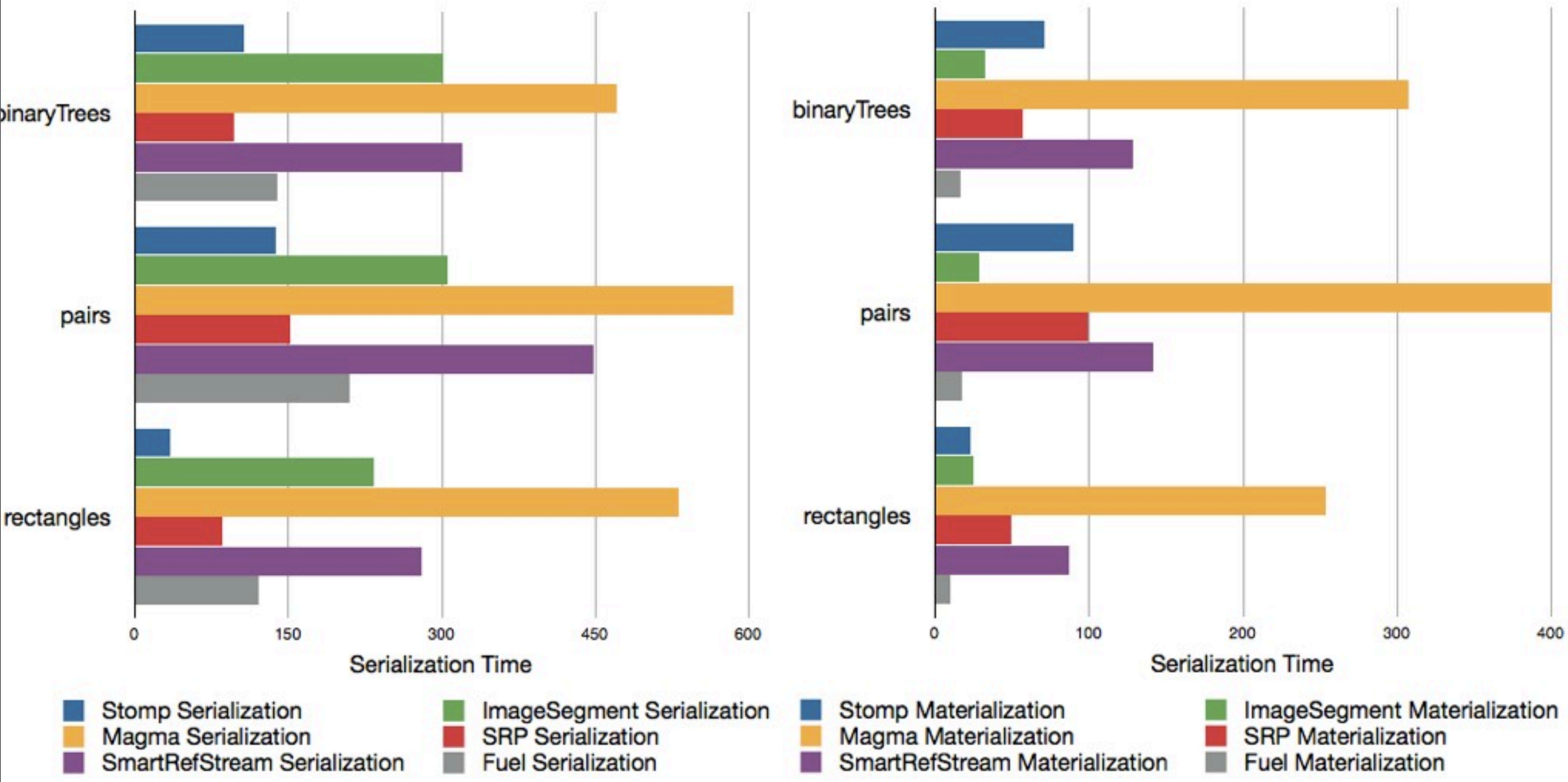


StOMP

SRP

Fuel

Non primitive objects

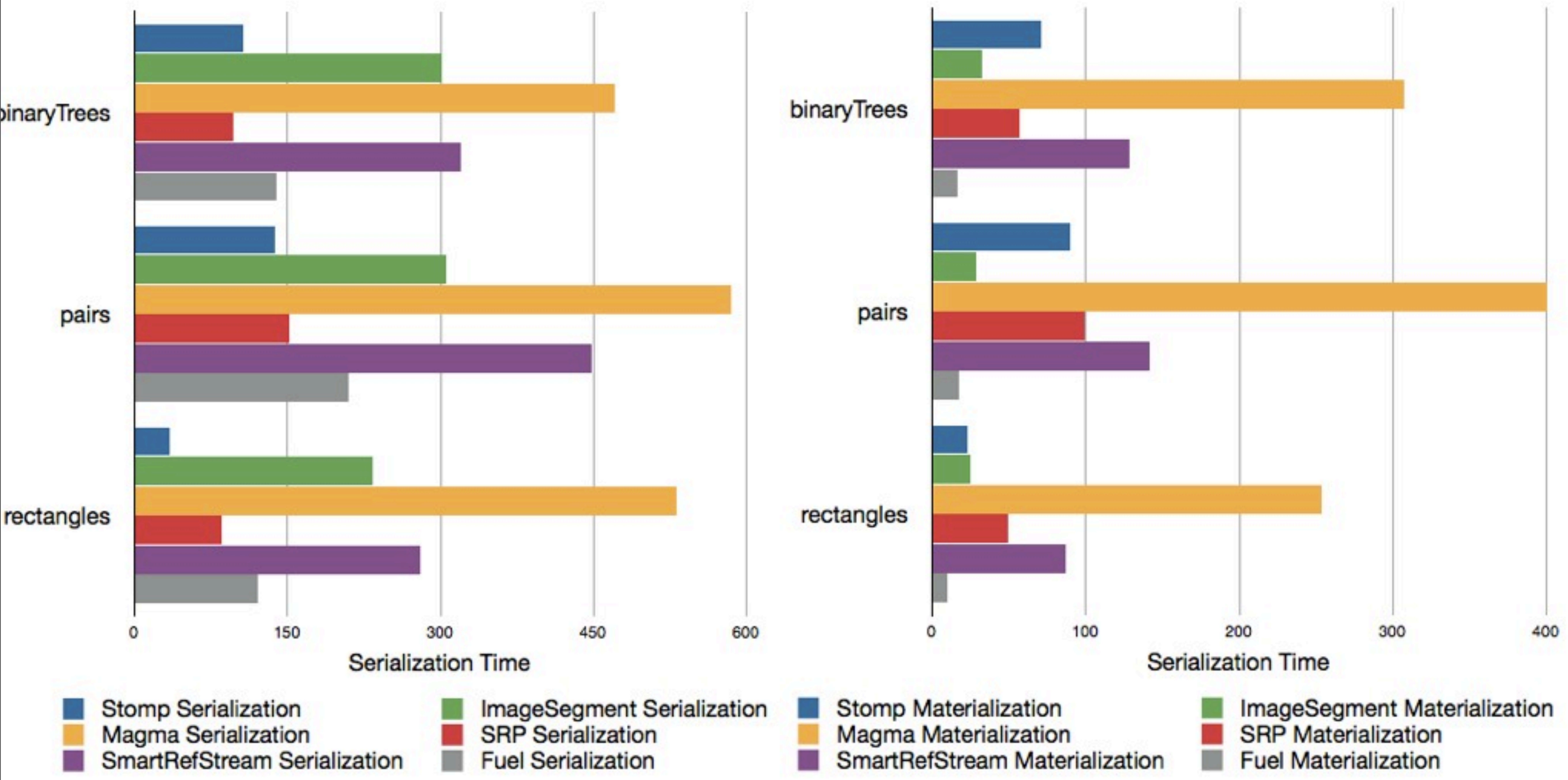


StOMP

SRP

Fuel

Non primitive objects



StOMP

SRP

Fuel

Fuel

ImageSegment

StOMP

Links

- Website: http://rmod.lille.inria.fr/web/pier/software/Fuel
- Issue tracker: http://code.google.com/p/fuel
- Source repository: http://www.squeaksource.com/Fuel
- Continuous integration server: https://pharo-ic.lille.inria.fr/hudson/job/Fuel/

Conclusion for us

Excellent performance without special support from VM and good OO design.

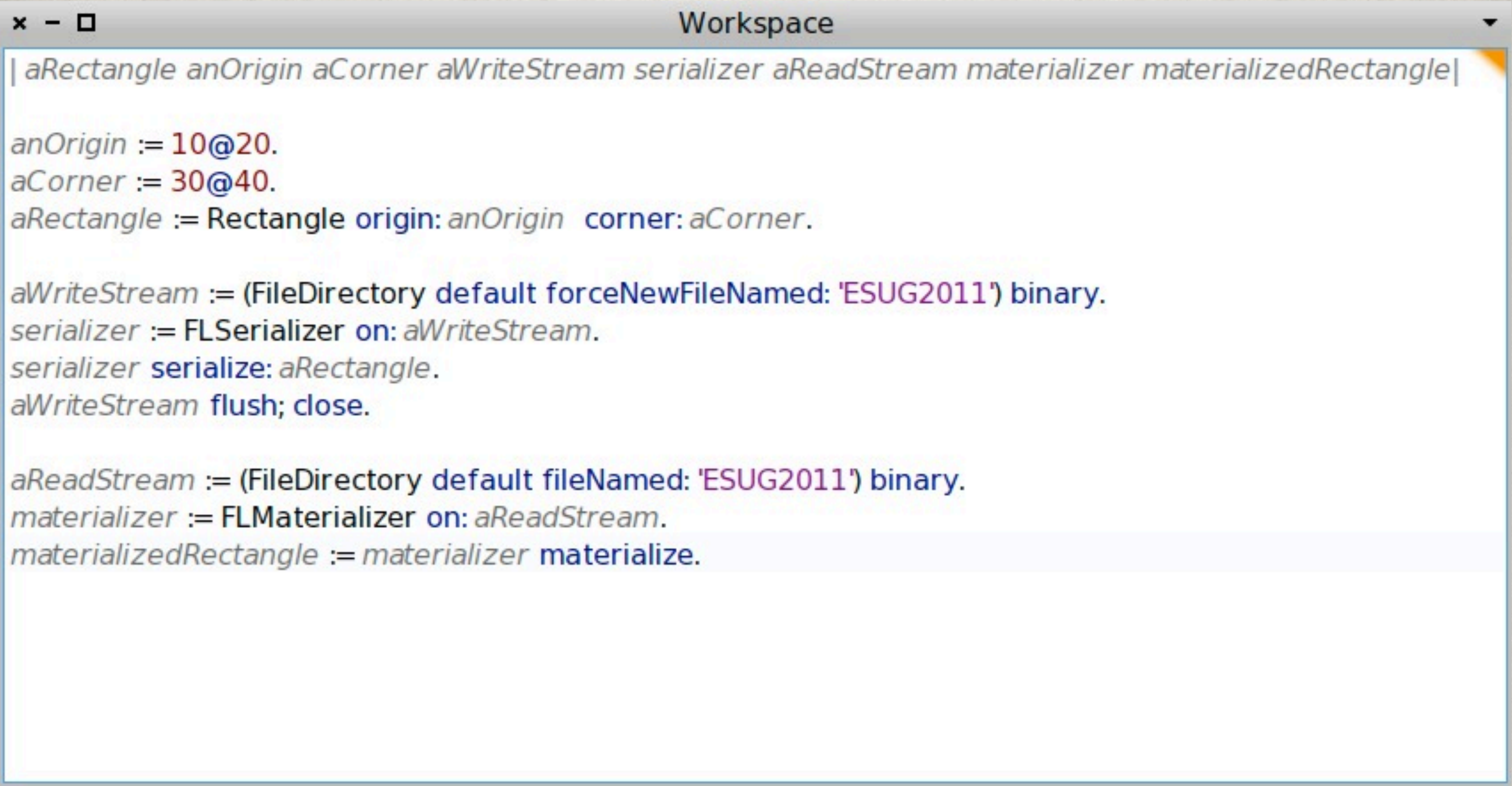
Conclusion for YOU

Fuel is a vehicle. It is infrastructure. You can build cool stuff on top of it.

Thanks!

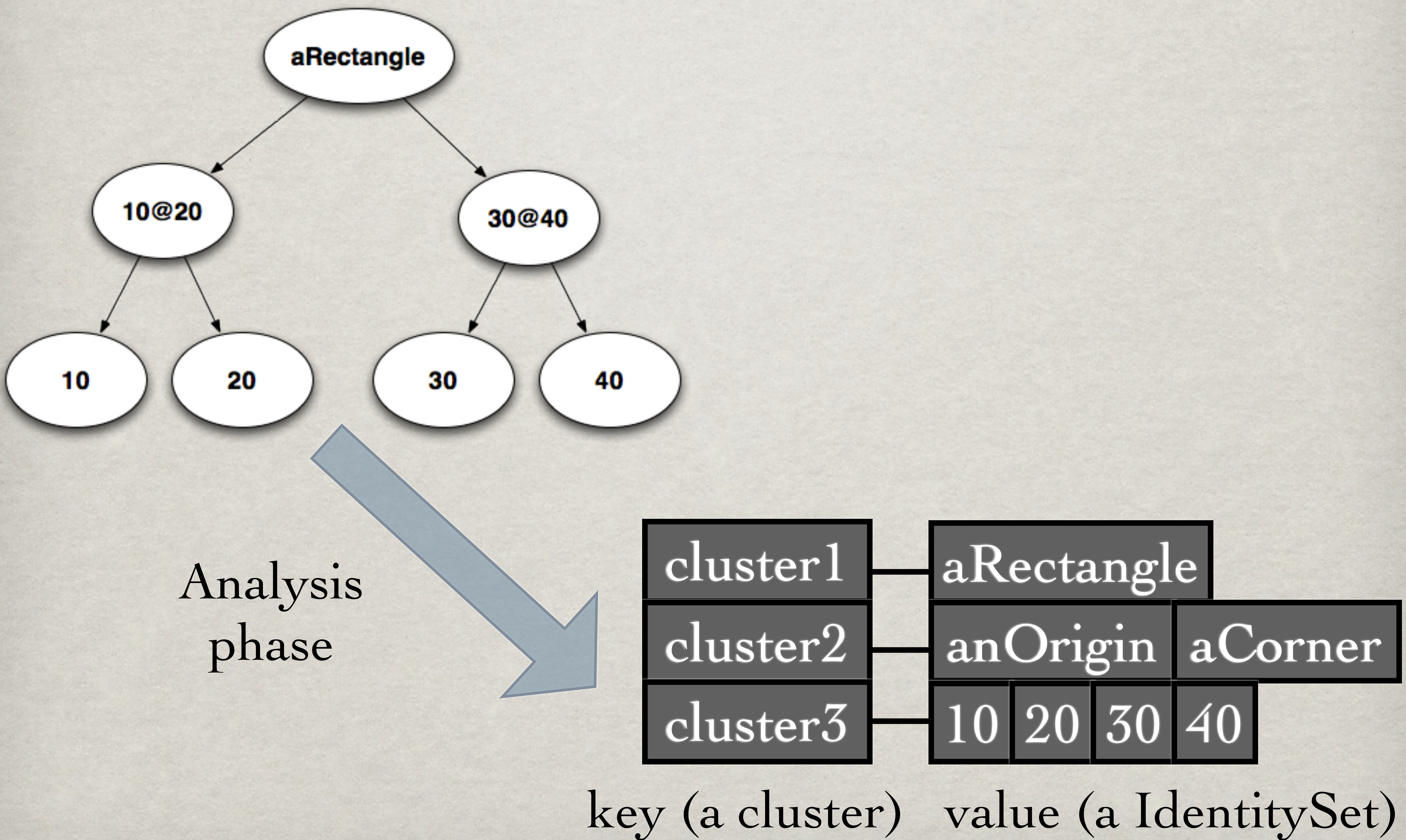
Mariano Martinez Peck
marianopeck@gmail.com
http://marianopeck.wordpress.com/

Concrete example

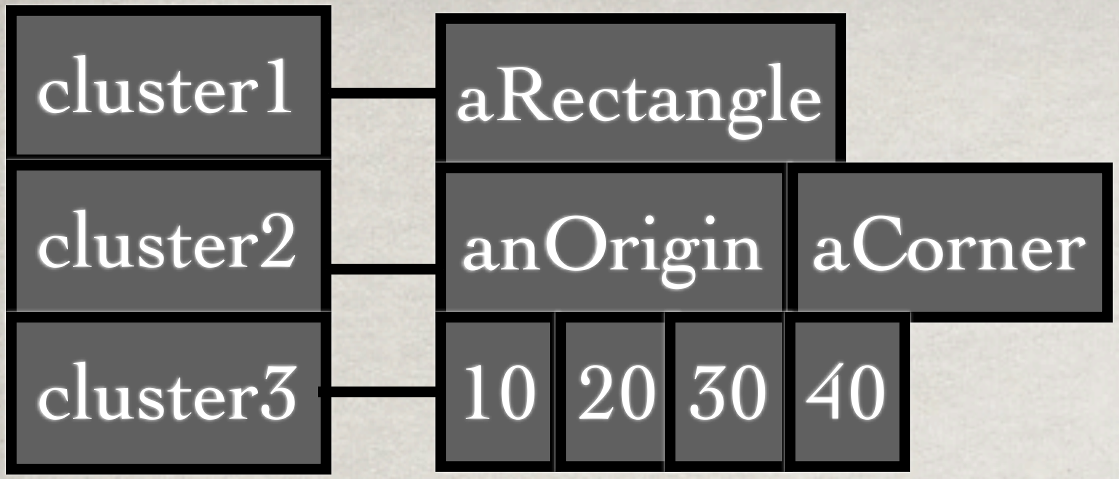


```
| aRectangle anOrigin aCorner aWriteStream serializer aReadStream materializer materializedRectangle|

anOrigin := 10@20.
aCorner := 30@40.
aRectangle := Rectangle origin: anOrigin  corner: aCorner.

aWriteStream := (FileDirectory default forceNewFileNamed: 'ESUG2011') binary.
serializer := FLSerializer on: aWriteStream.
serializer serialize: aRectangle.
aWriteStream flush; close.

aReadStream := (FileDirectory default fileNamed: 'ESUG2011') binary.
materializer := FLMaterializer on: aReadStream.
materializedRectangle := materializer materialize.
```

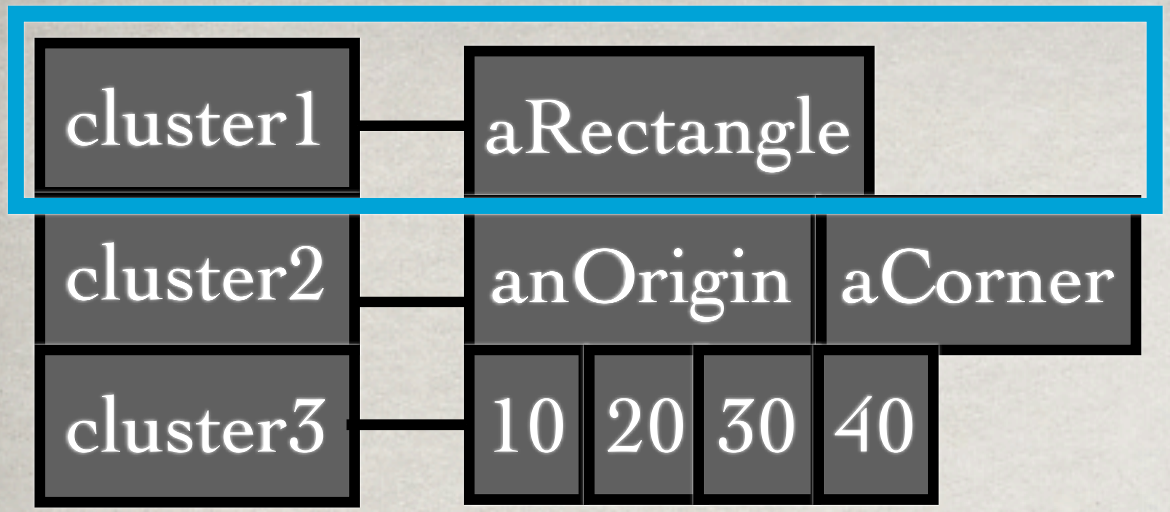
Analysis phase



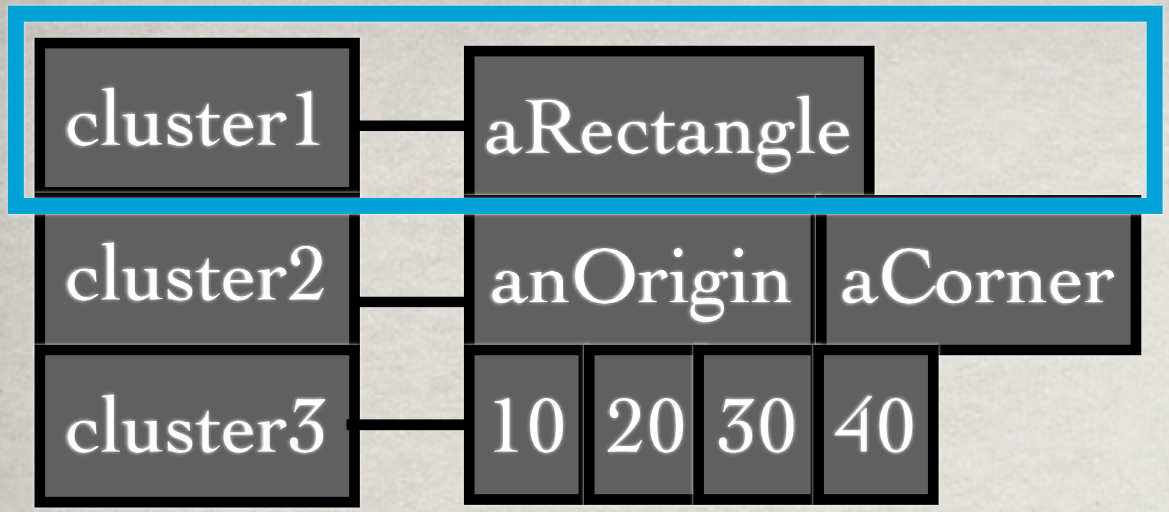
Serialization instances phase

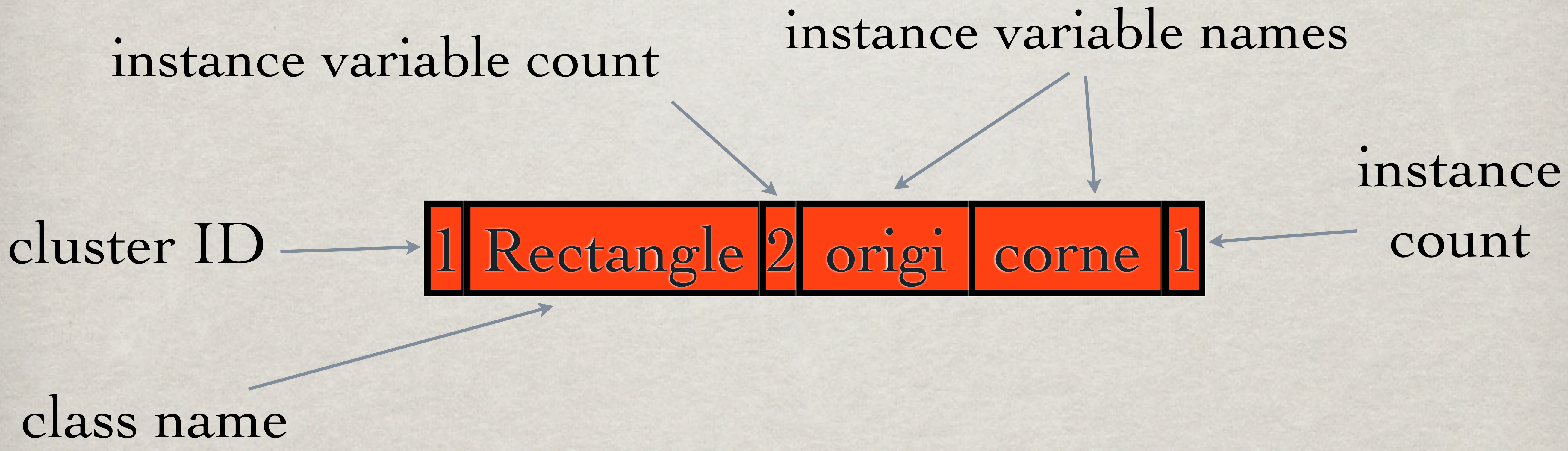


Serialization
instances
phase

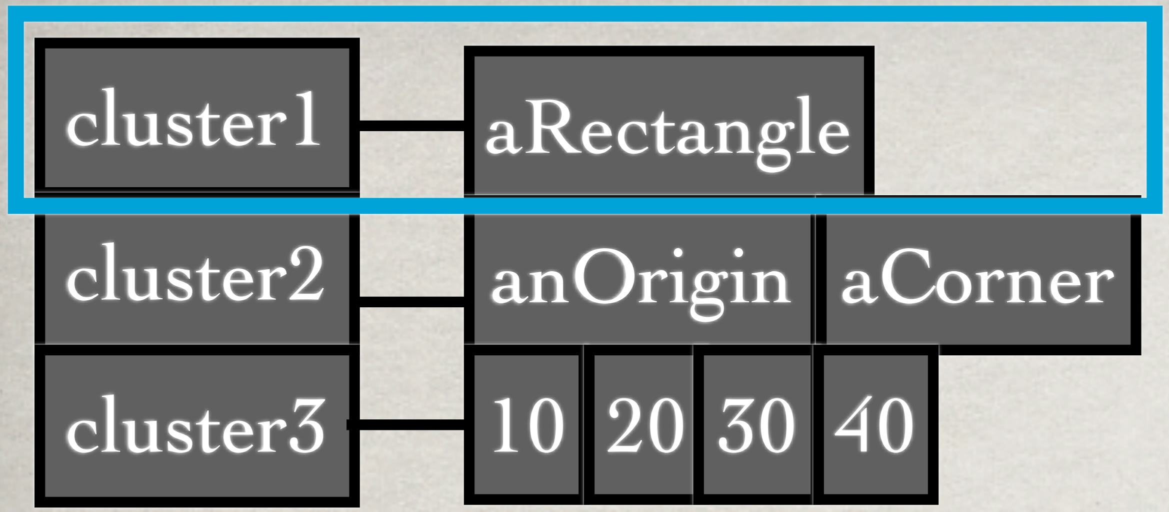


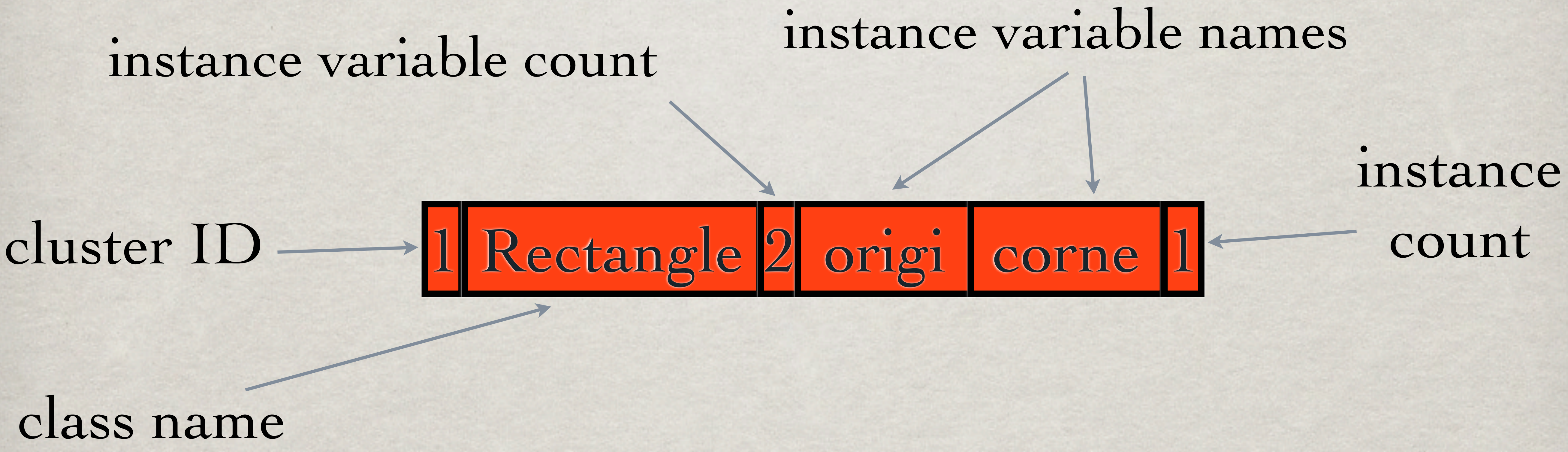
Serialization instances phase

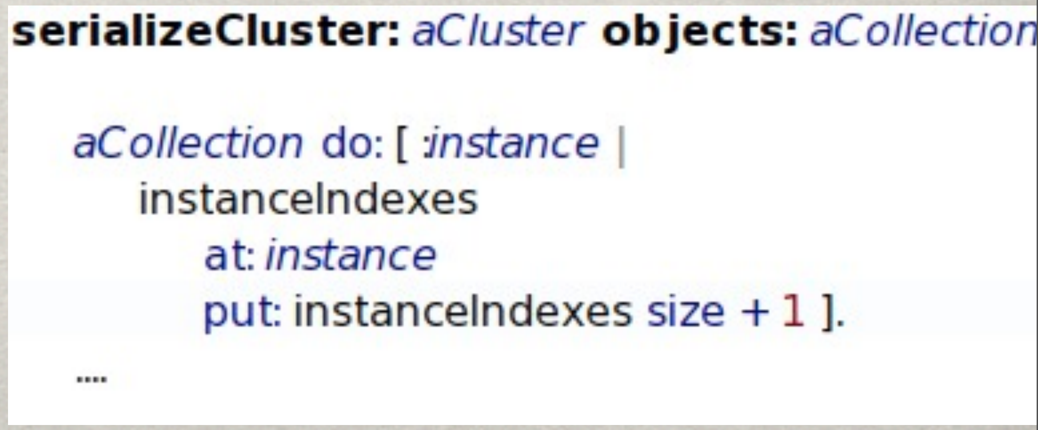




Serialization instances phase

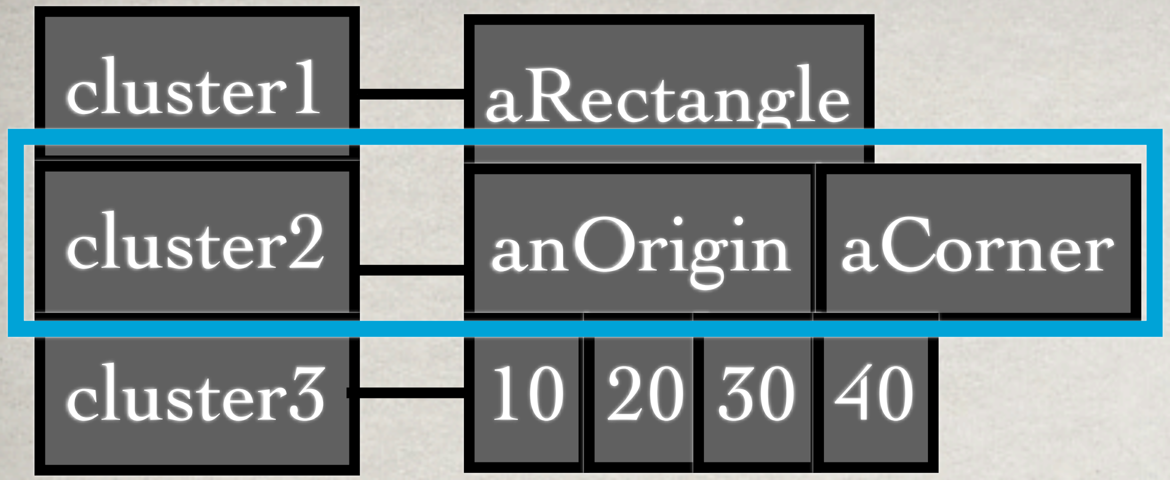


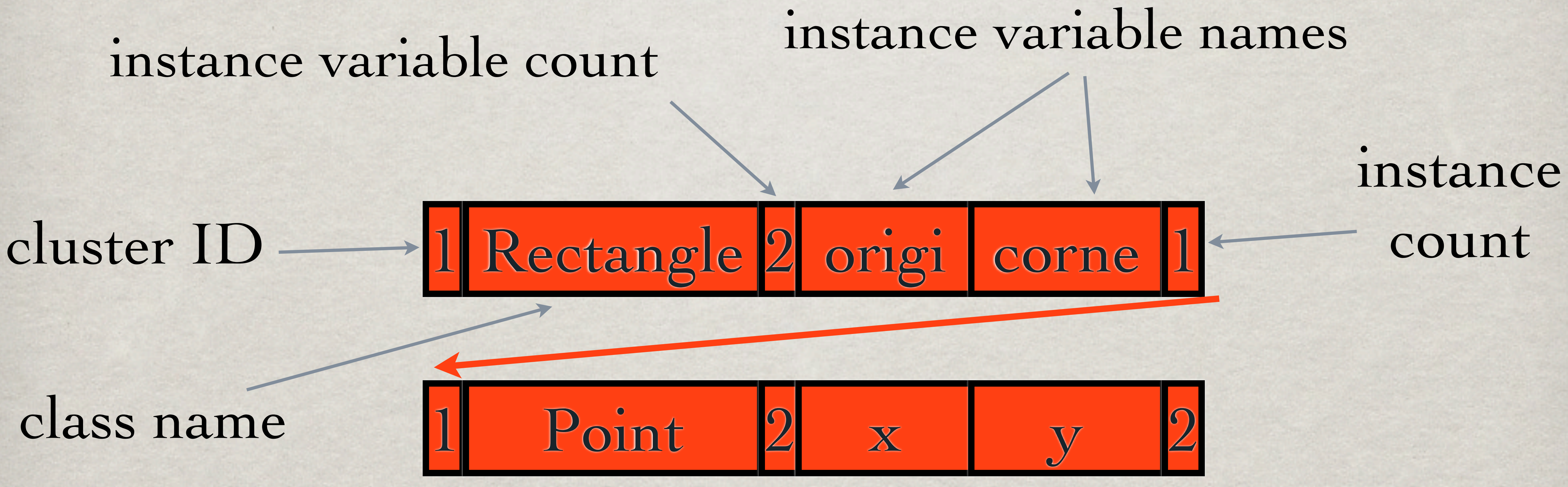


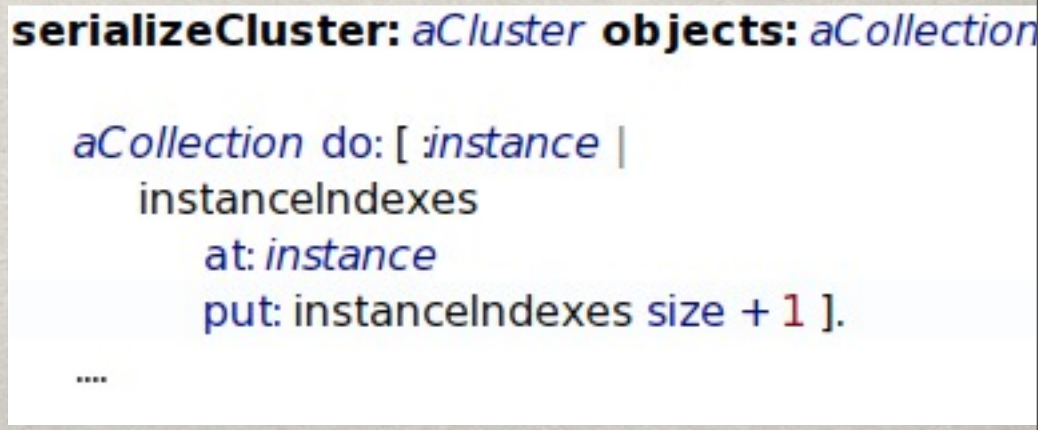


```
serializeCluster: aCluster objects: aCollection

    aCollection do: [ :instance |
        instanceIndexes
            at: instance
            put: instanceIndexes size + 1 ].
    ....
```

Serialization instances phase

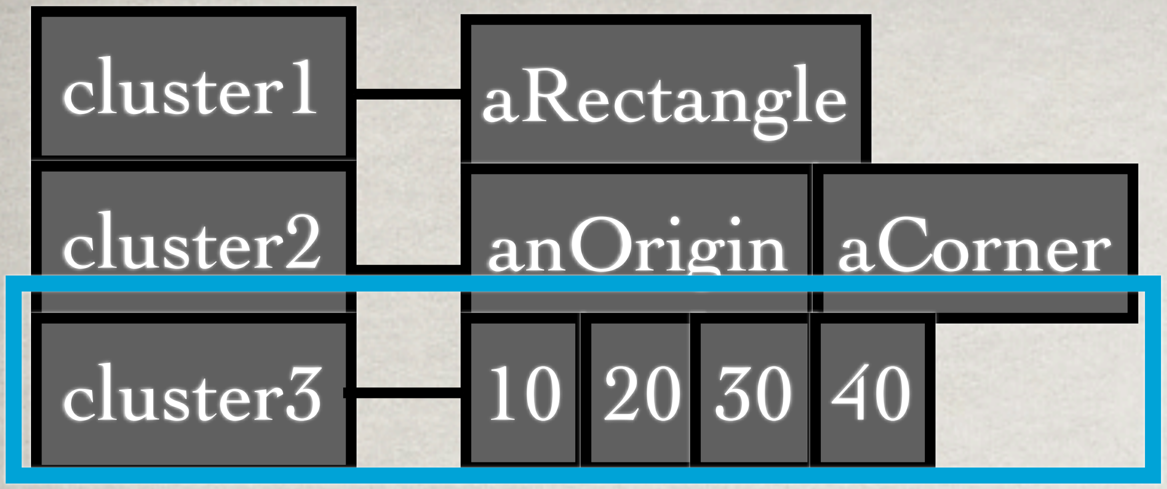


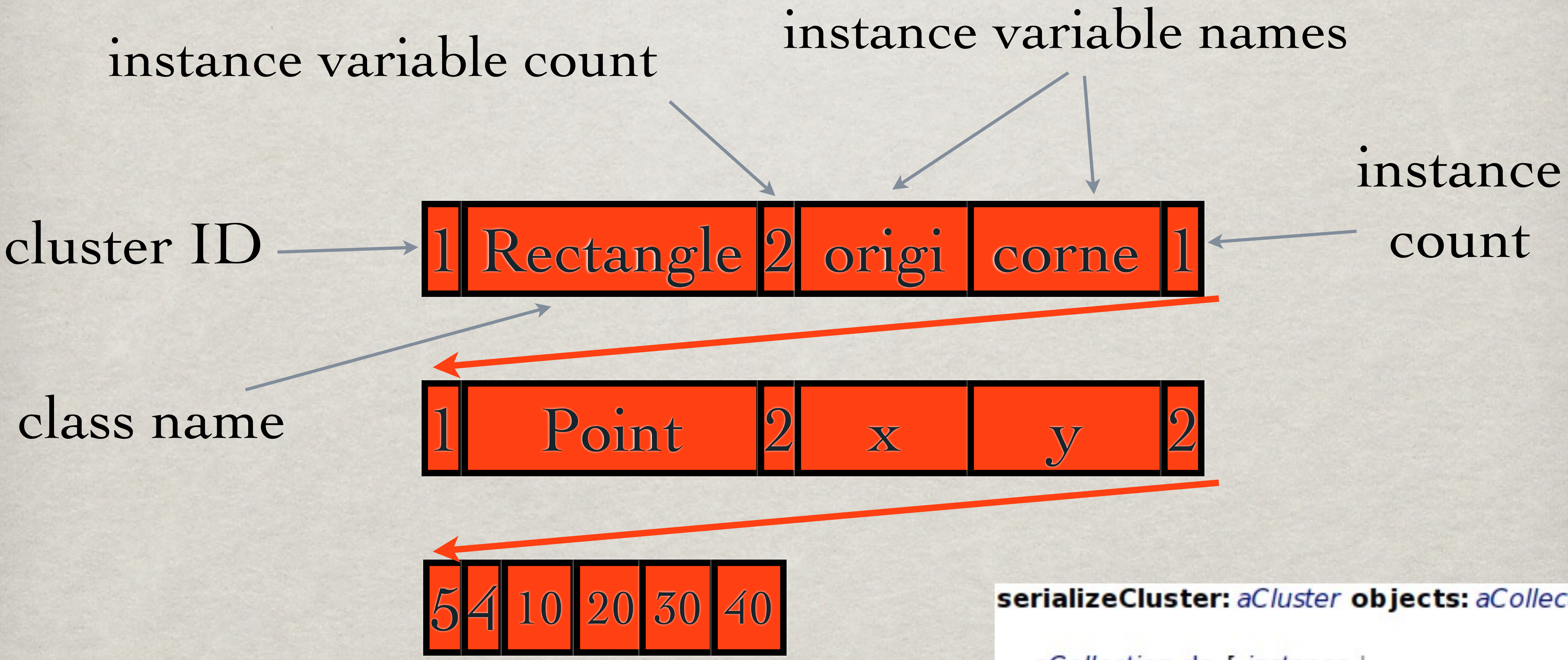


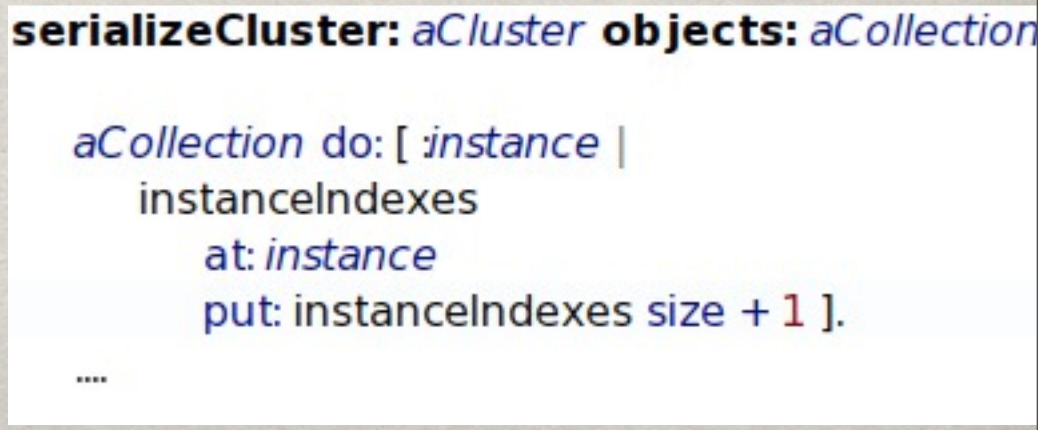


```
serializeCluster: aCluster objects: aCollection

    aCollection do: [ :instance |
        instanceIndexes
            at: instance
            put: instanceIndexes size + 1 ].
    ....
```

Serialization instances phase

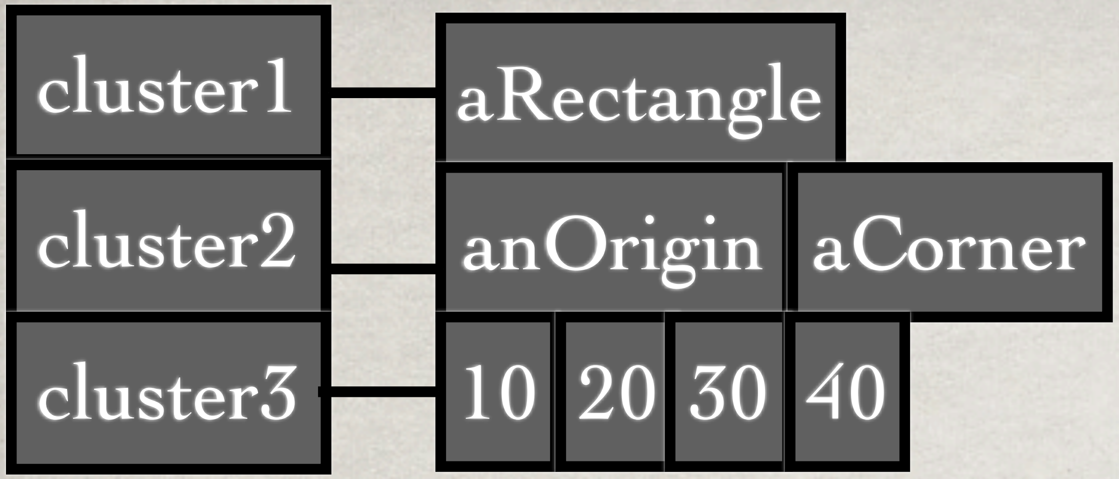


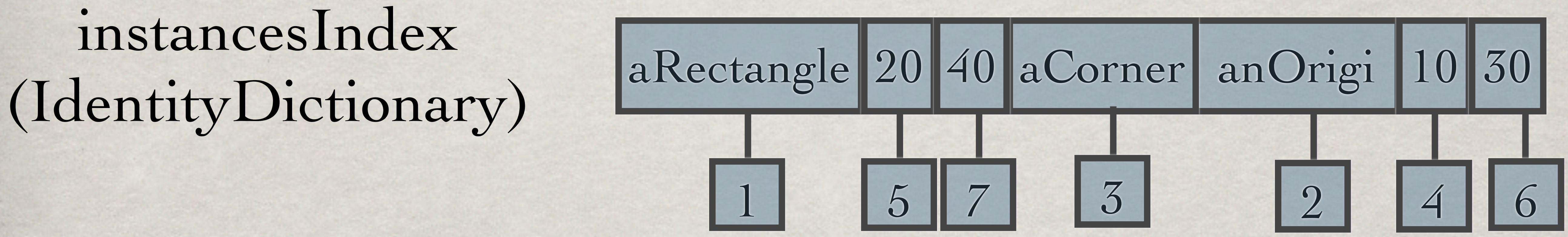


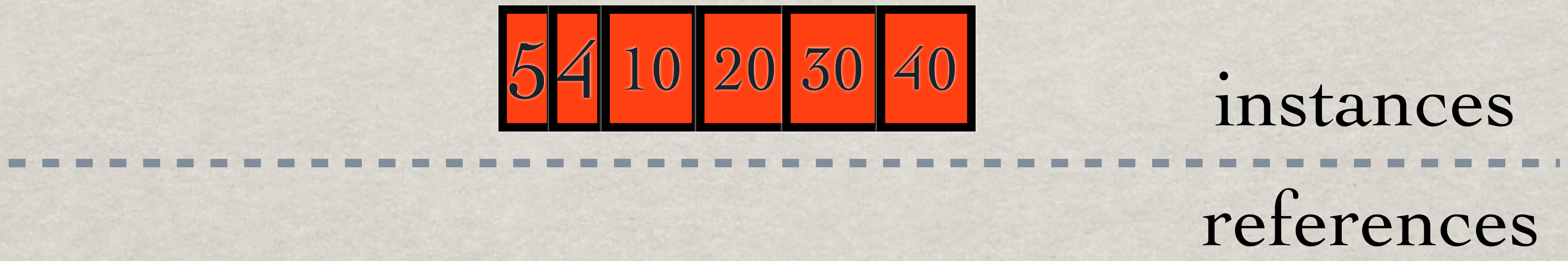


```
serializeCluster: aCluster objects: aCollection

    aCollection do: [ :instance |
        instanceIndexes
            at: instance
            put: instanceIndexes size + 1 ].
    ....
```

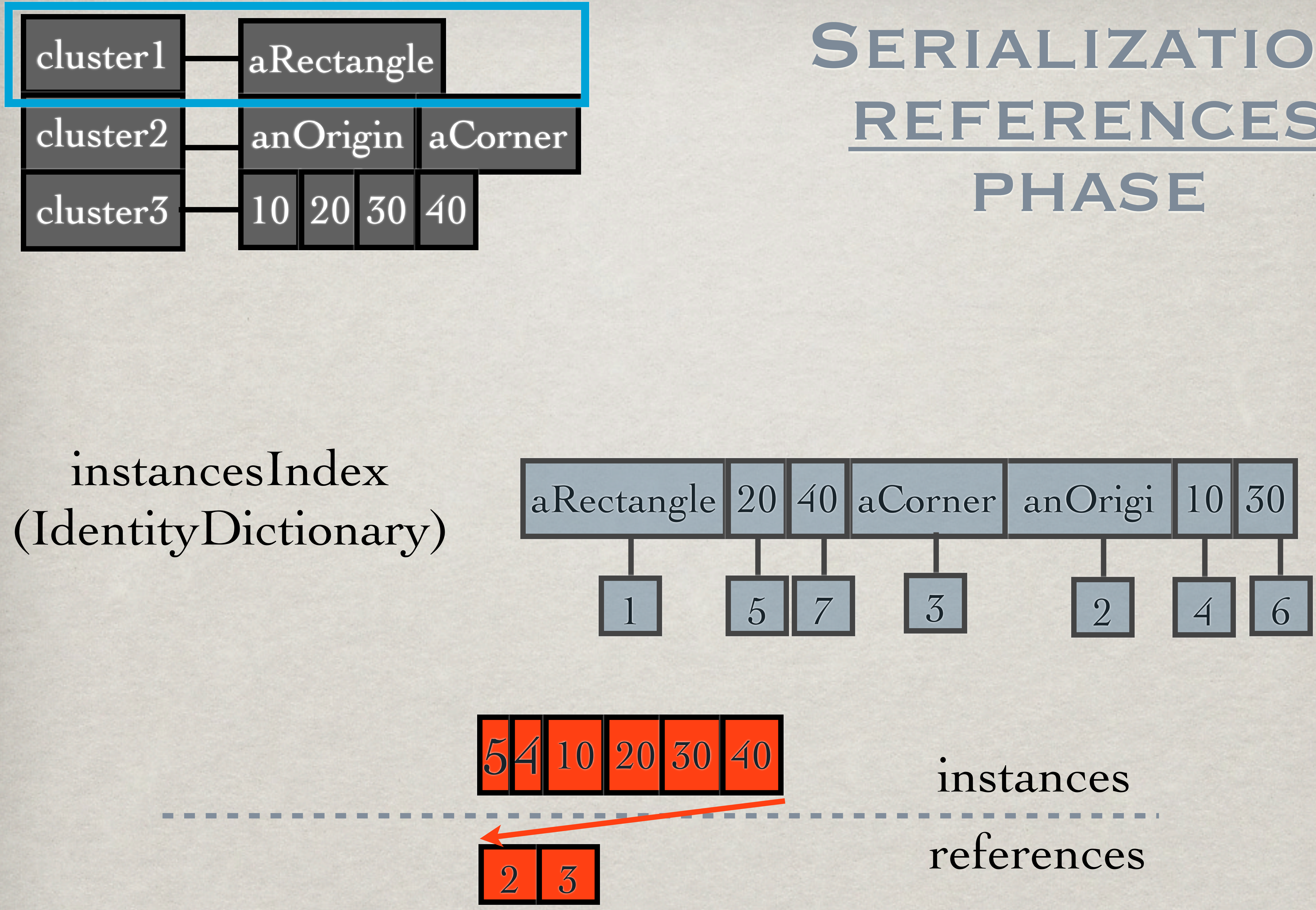
Serialization references phase



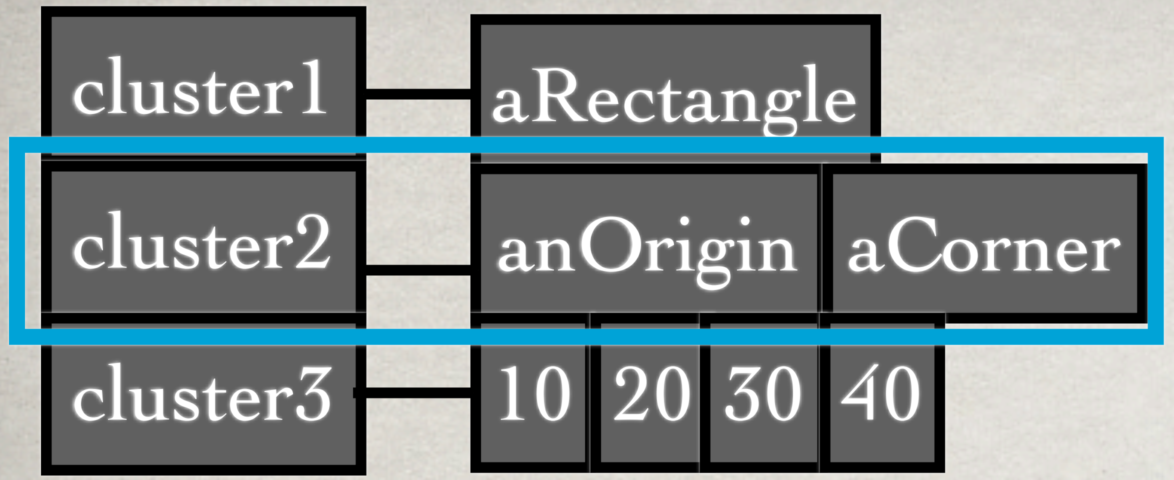




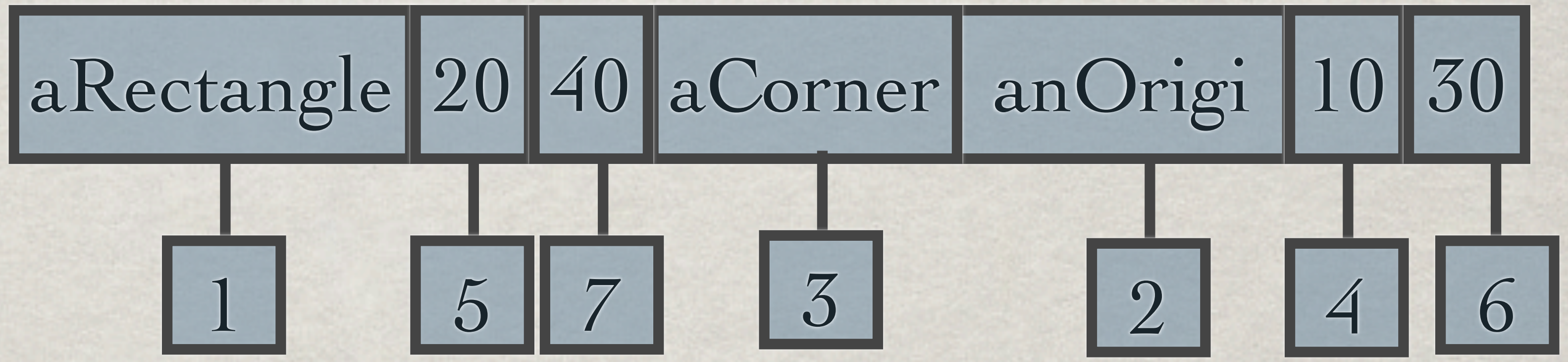
Serialization references phase

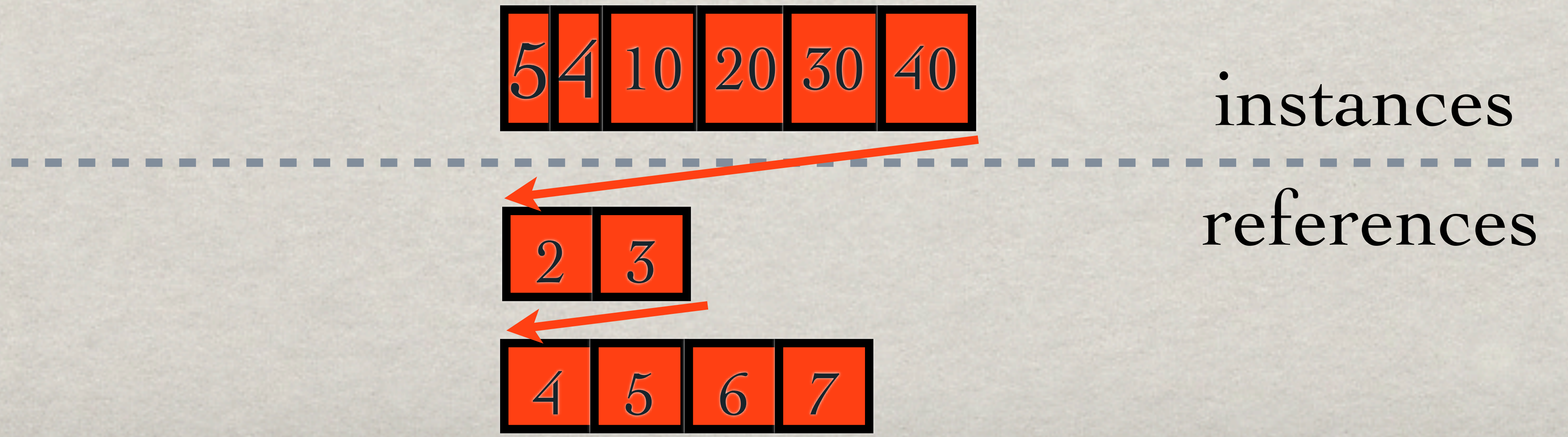


SERIALIZATION
REFERENCES PHASE

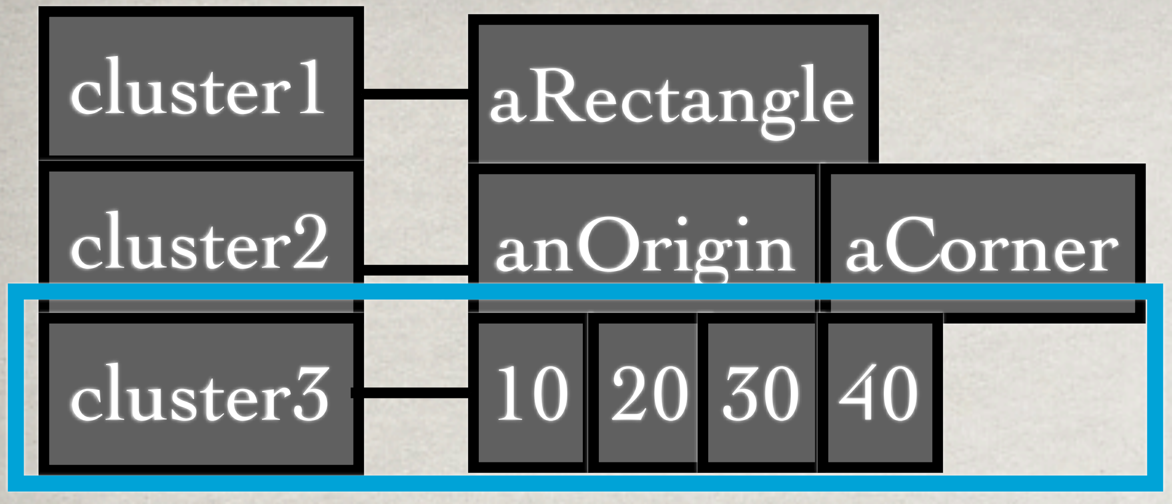


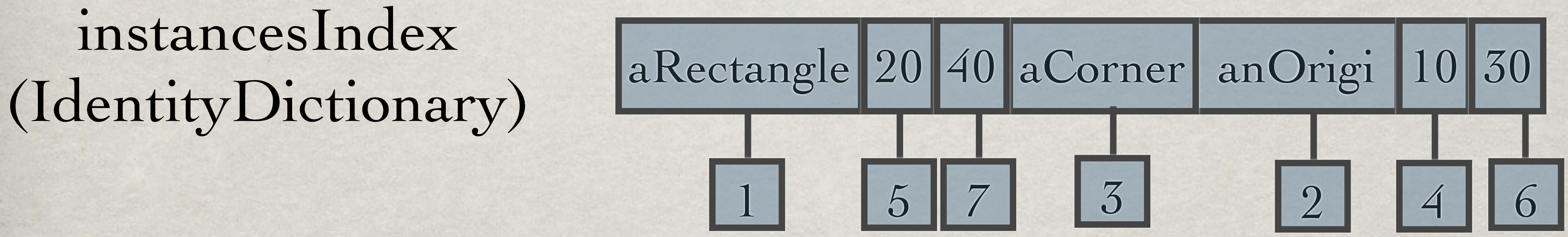
instancesIndex
(IdentityDictionary)

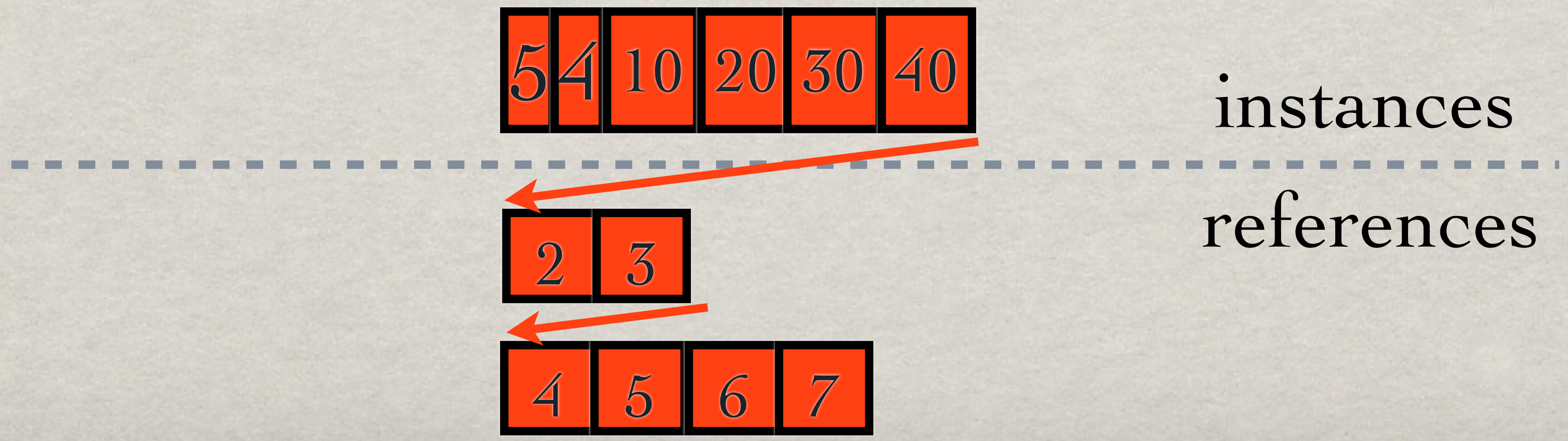




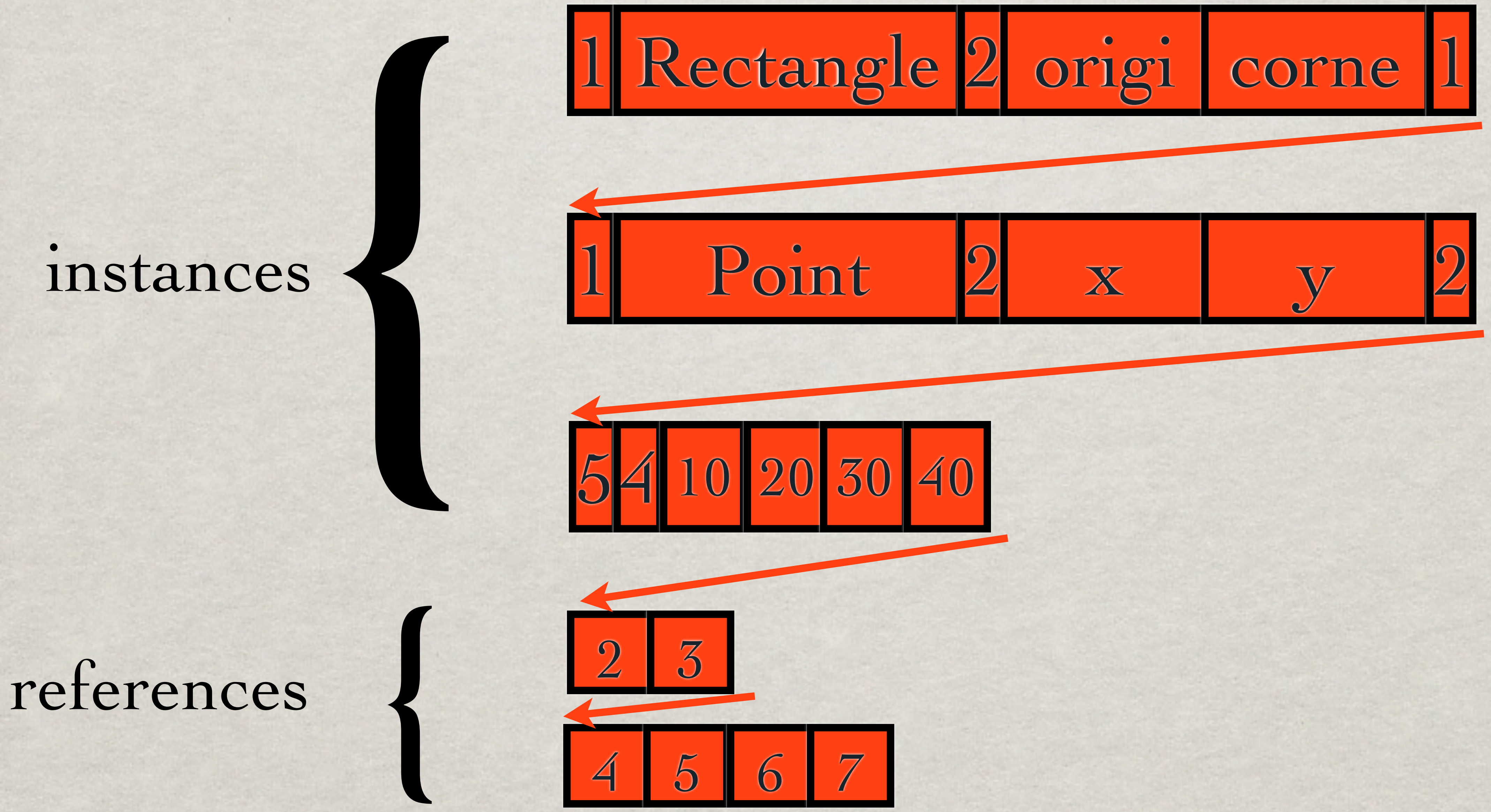
Serialization references phase



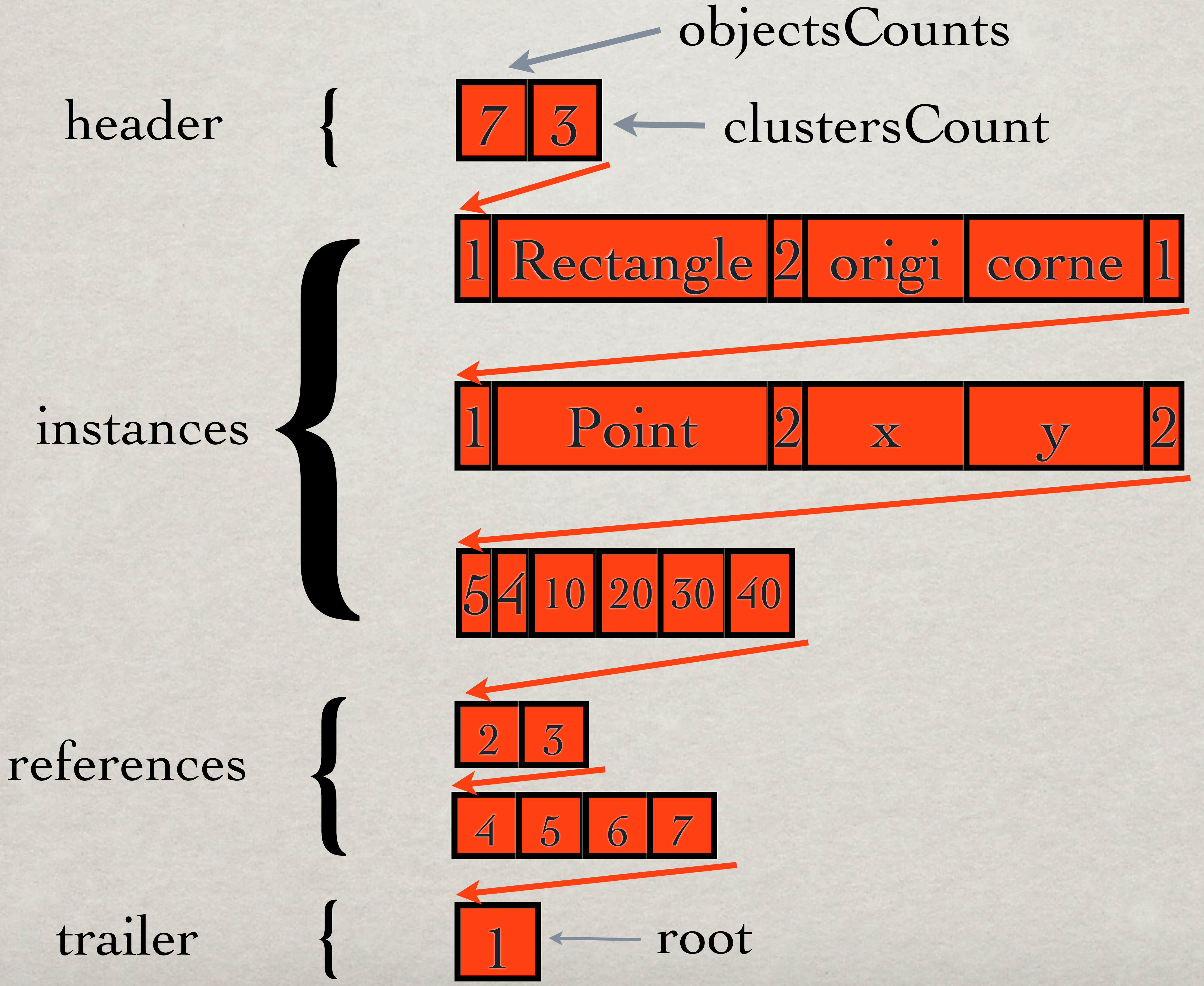




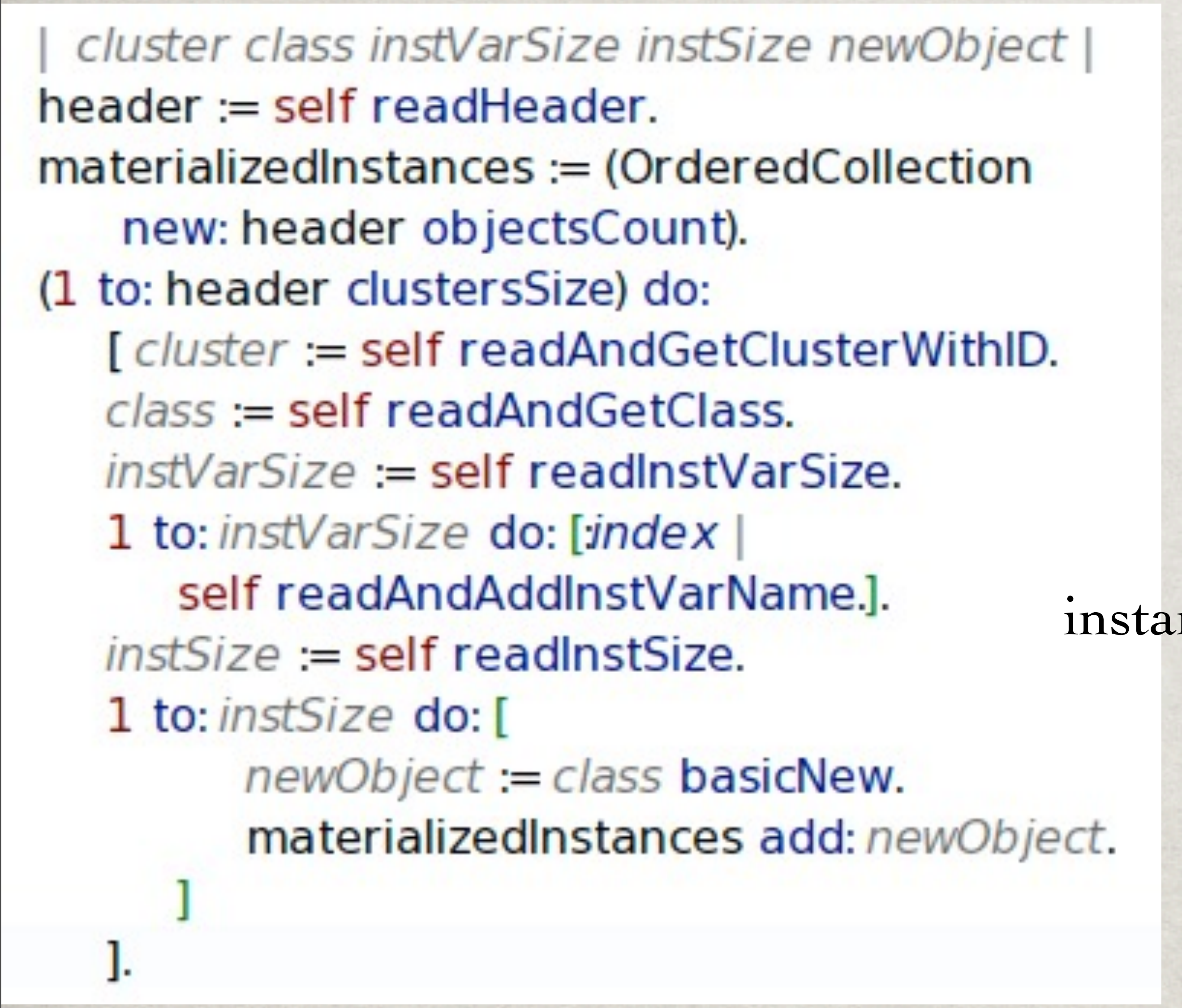
FINAL STREAM



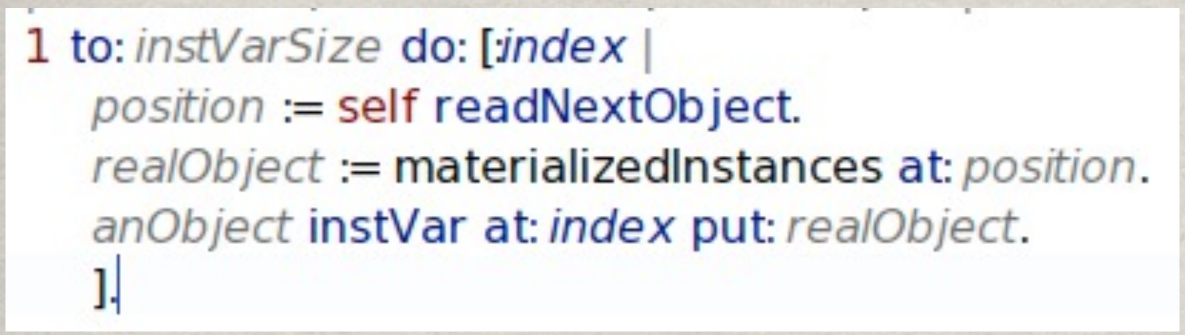
FINAL STREAM



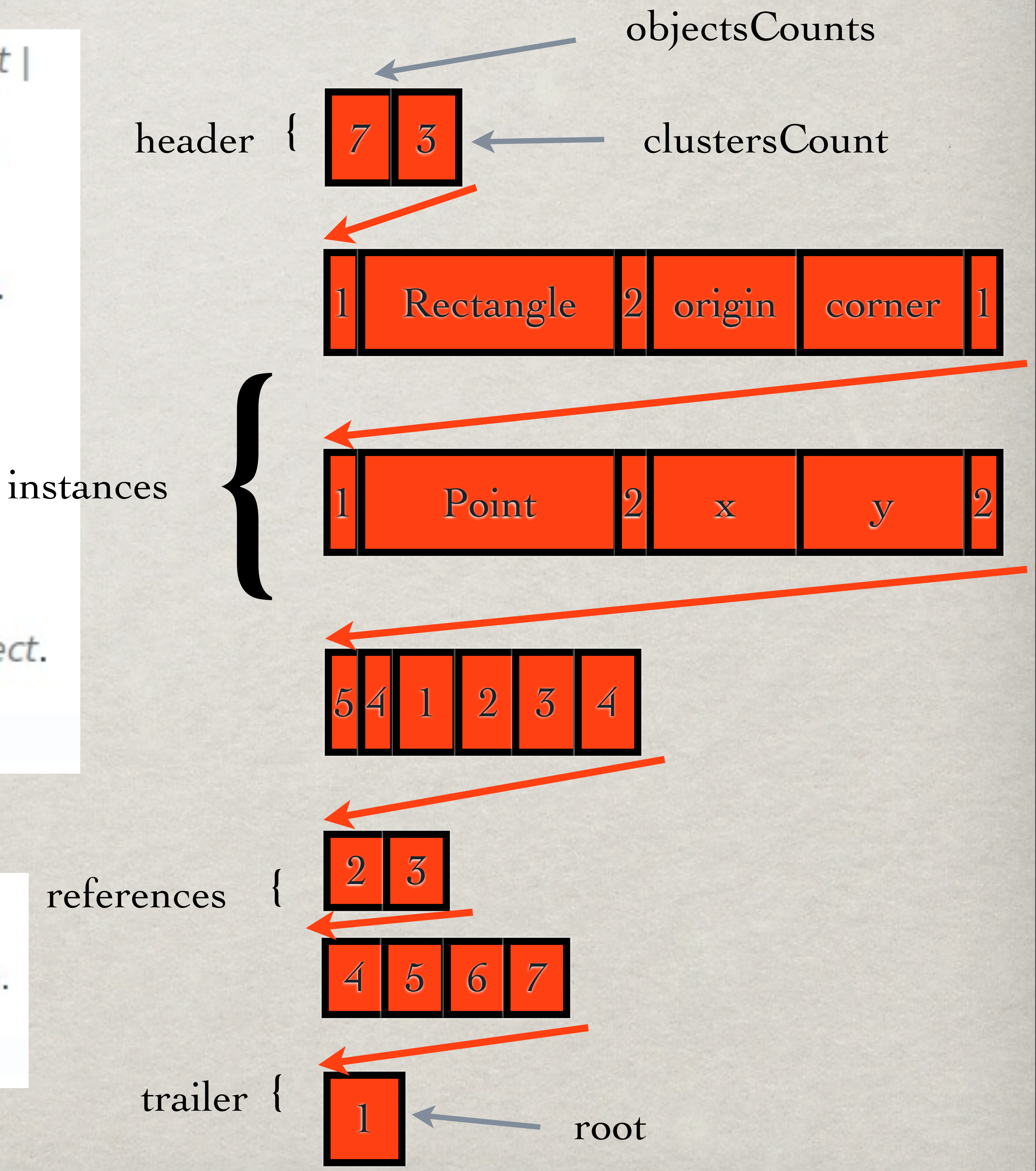
Materialization



```
| cluster class instVarSize instSize newObject |
header := self readHeader.
materializedInstances := (OrderedCollection
     new: header objectsCount).
(1 to: header clustersSize) do:
    [ cluster := self readAndGetClusterWithID.
    class := self readAndGetClass.
    instVarSize := self readInstVarSize.
    1 to: instVarSize do: [:index |
        self readAndAddInstVarName.].
    instSize := self readInstSize.
    1 to: instSize do: [
            newObject := class basicNew.
            materializedInstances add: newObject.
        ]
    ].
```



```
1 to: instVarSize do: [:index |
   position := self readNextObject.
   realObject := materializedInstances at: position.
   anObject instVar at: index put: realObject.
   ].
```



objectsCounts
header
7
3
clustersCount
1
Rectangle
2
origin
corner
1
instances
1
Point
2
x
y
2
5
4
1
2
3
4
references
2
3
4
5
6
7
trailer
1
root