Fuel

a fast and flexible object deployment tool

Intro

Goal in a nutshell

Store **objects** and restore them **fast**.

The **store** time is not the focus.

It will work with any kind of objects, but its main focus is on storing **classes** and **methods**.

Other goals

- Have a minimal restore package (to work with Seed kernels).
- Be flexible and configurable so that it could be useful for more low-level kind of storing (Mariano's Marea).
- Also other tools could use it for storing their objects (Moose?)

Other goals (II)

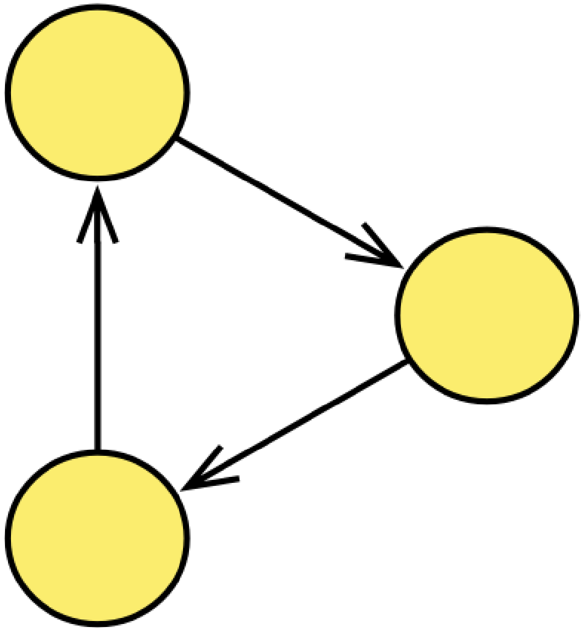
- Allow other ways of reading the stream:
  - Partial loading: sometimes we don't want all the stored objects.
  - Brief info extraction: it would be nice to do a 'light' read of the stream, to extract some information but not restoring the objects.

Problems

Identity: reference vs. creation

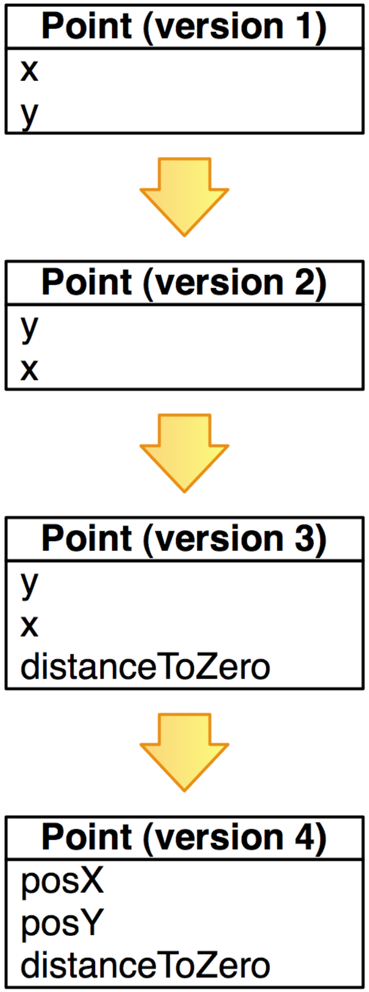
- Some objects in the graph shouldn't be stored, but only put a reference to them.
- Simple examples:
  - A reference to Transcript
  - A reference to a system class like Integer
- But there are examples much more hard to detect and also restore the reference later:
  - A reference to a 'user defined' singleton instance.

Cycles



- When traversing the graph is necessary to check for **cycles**.
- This check has a high cost, but only affects on **storing** time.

Class shape changing



Point (version 1)
x
y

Point (version 2)
y
x

Point (version 3)
y
x
distanceToZero

Point (version 4)
posX
posY
distanceToZero

When loading an object, it can happen that its class has changed. So we could:

- Have some automatic tolerance.
- Allow the user to solve conflicts easily.

Portability

- Between **tool** versions:
  - When restoring from a stream, we should look what version was used to store it, and then act in consequence.
- Between **image** versions:
  - Suppose that CompiledMethod implementation changes. We should be able to adapt the original format, or at least detect a problem and throw an error.
- Between **dialects**:
  - Is not a goal for us.

Minimal loading

- Fuel would be used for binary loading of packages in a **Seed** image.
  - Maybe in absence of a **Compiler**.
- In order to achieve this minimality, the packages for **reading** should be **independent** of the ones for writing.

Known implementations

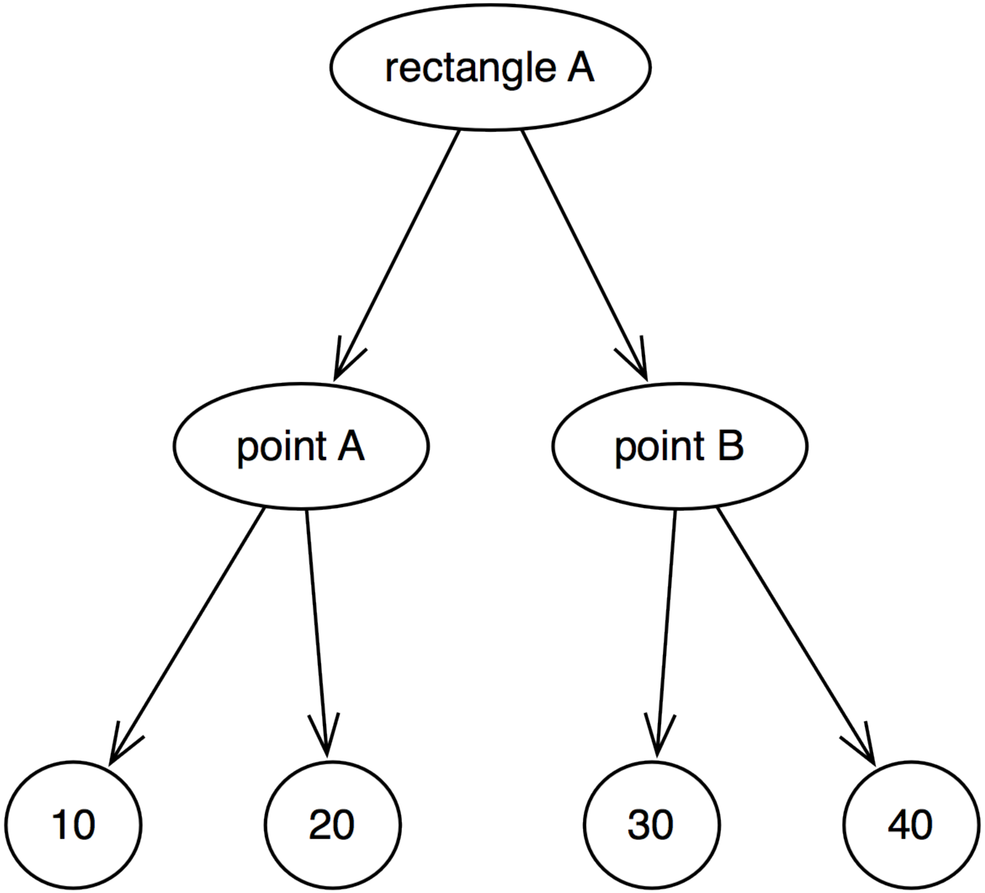
- Parcels (VW)
- ReferenceStream / SmartRefStream
- Magma
- Monticello2
- SRP (VW)
- BOSS (VW)
- ObjectDumper (GNU Smalltalk)
- SIXX

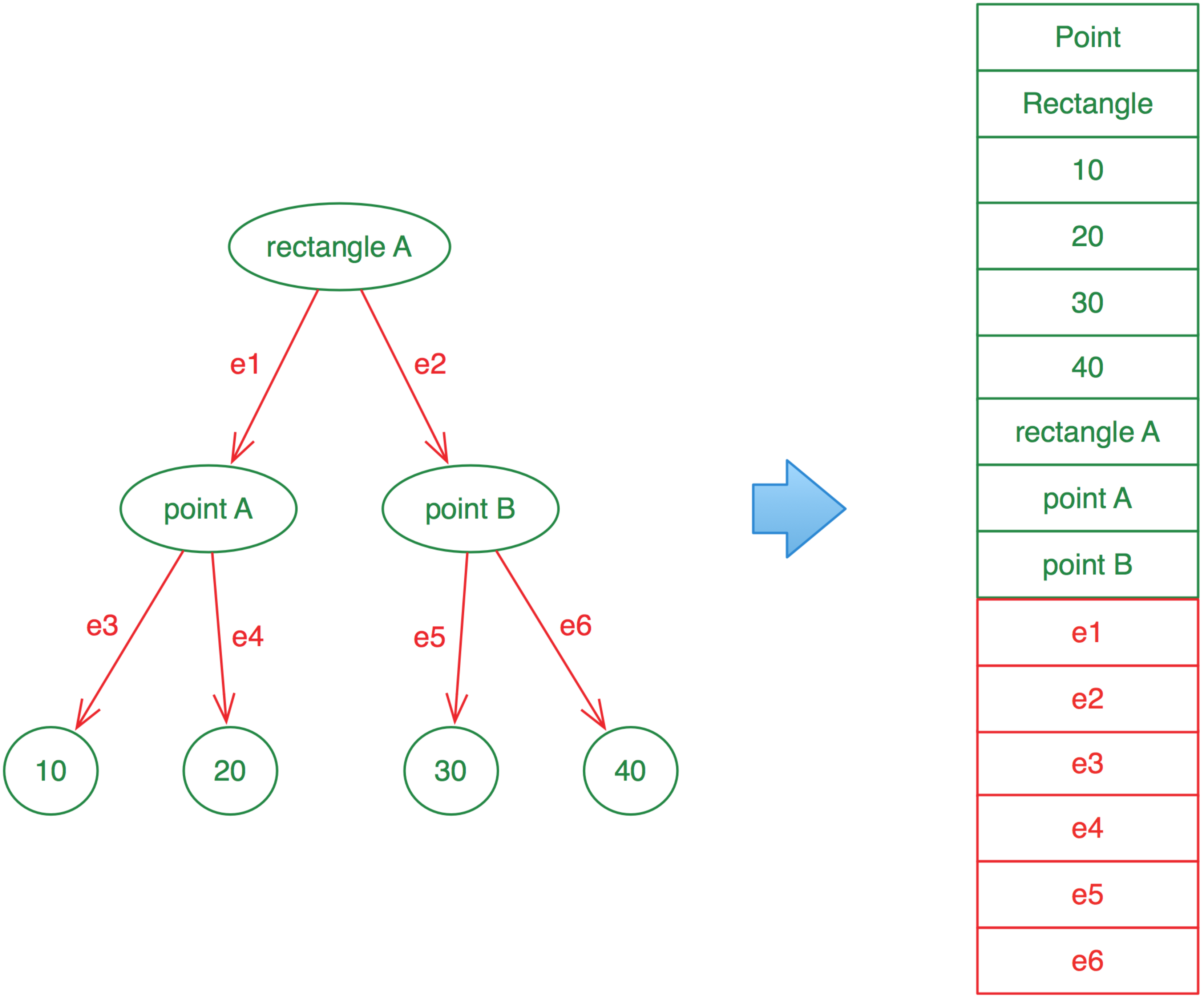
Our solution

(idea taken from VW's Parcels)

Principles

- Store **iteratively** the object graph, in **two parts**:
  - Instances (nodes)
  - References (edges)
- **Group** the objects and write them in such way to have **very fast loading**.

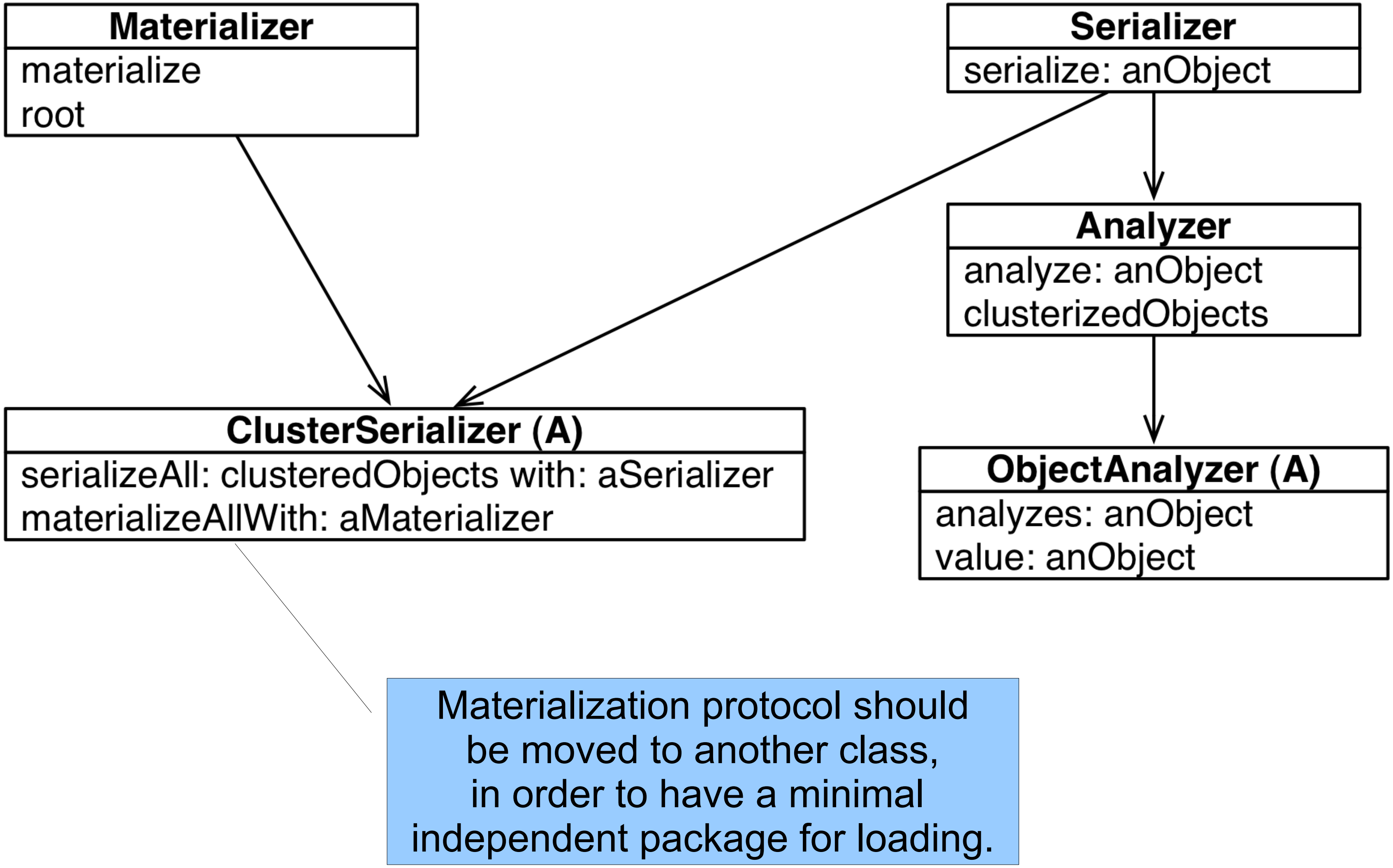




Point
Rectangle
10
20
30
40
rectangle A
point A
point B
e1
e2
e3
e4
e5
e6

Current implementation

Main classes



Materializer

materialize

root

Serializer

serialize: anObject

Analyzer

analyze: anObject

clusterizedObjects

ClusterSerializer (A)

serializeAll: clusteredObjects with: aSerializer

materializeAllWith: aMaterializer

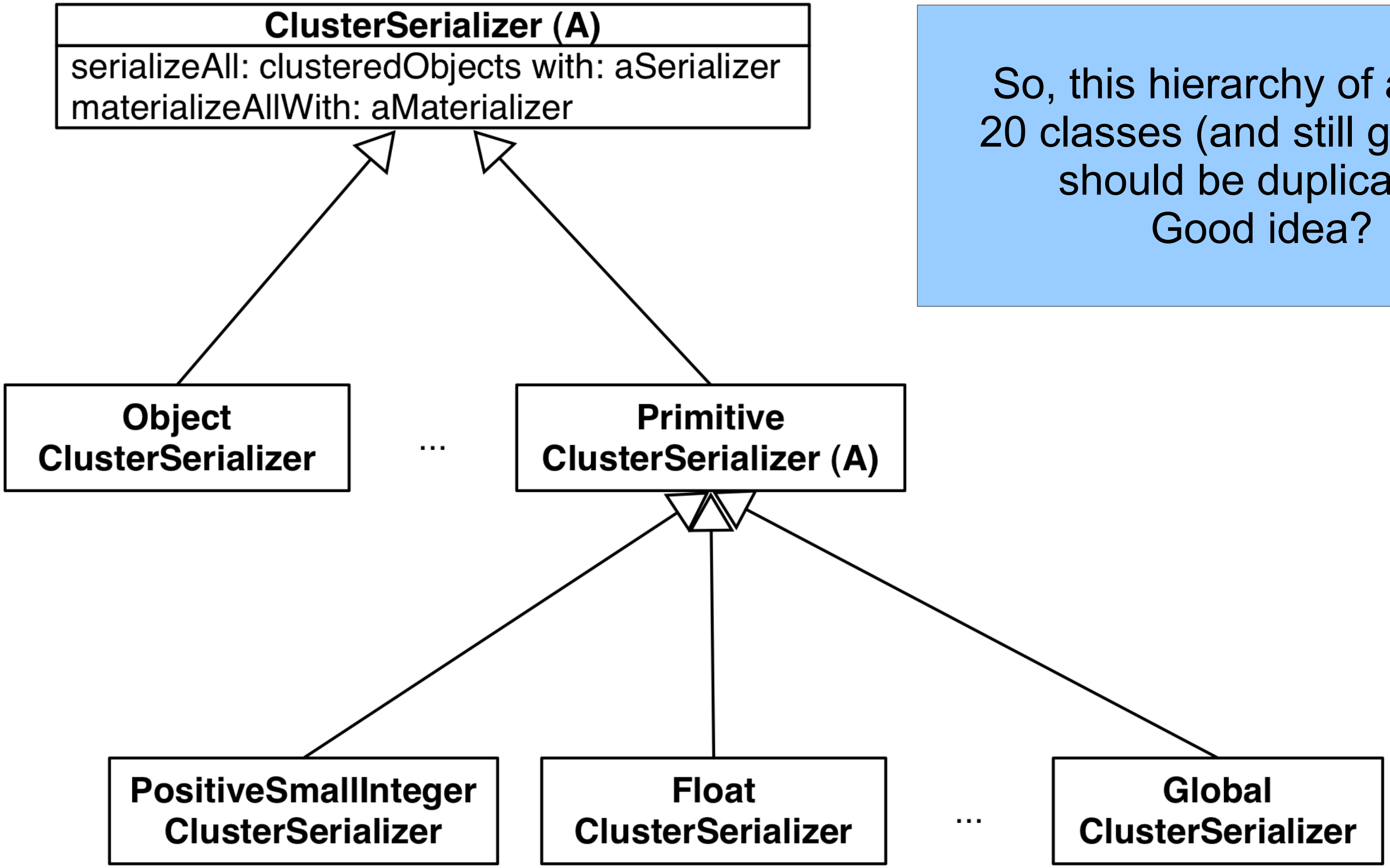
ObjectAnalyzer (A)

analyzes: anObject

value: anObject

Materialization protocol should
be moved to another class,
in order to have a minimal
independent package for loading.

ClusterSerializer hierarchy



ClusterSerializer (A)

serializeAll: clusteredObjects with: aSerializer
materializeAllWith: aMaterializer

Object ClusterSerializer

...

Primitive ClusterSerializer (A)

PositiveSmallInteger ClusterSerializer

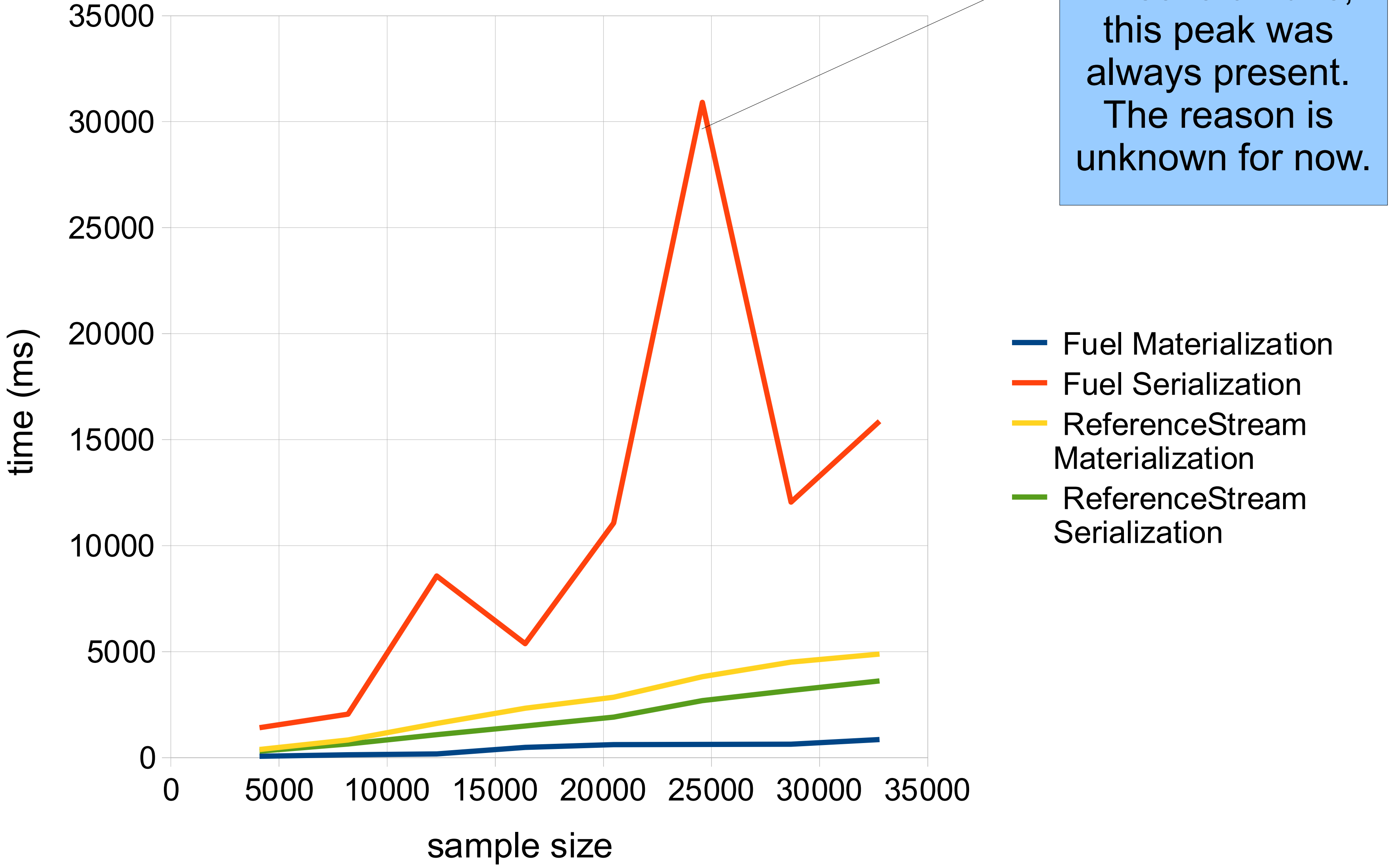
Float ClusterSerializer

...

Global ClusterSerializer

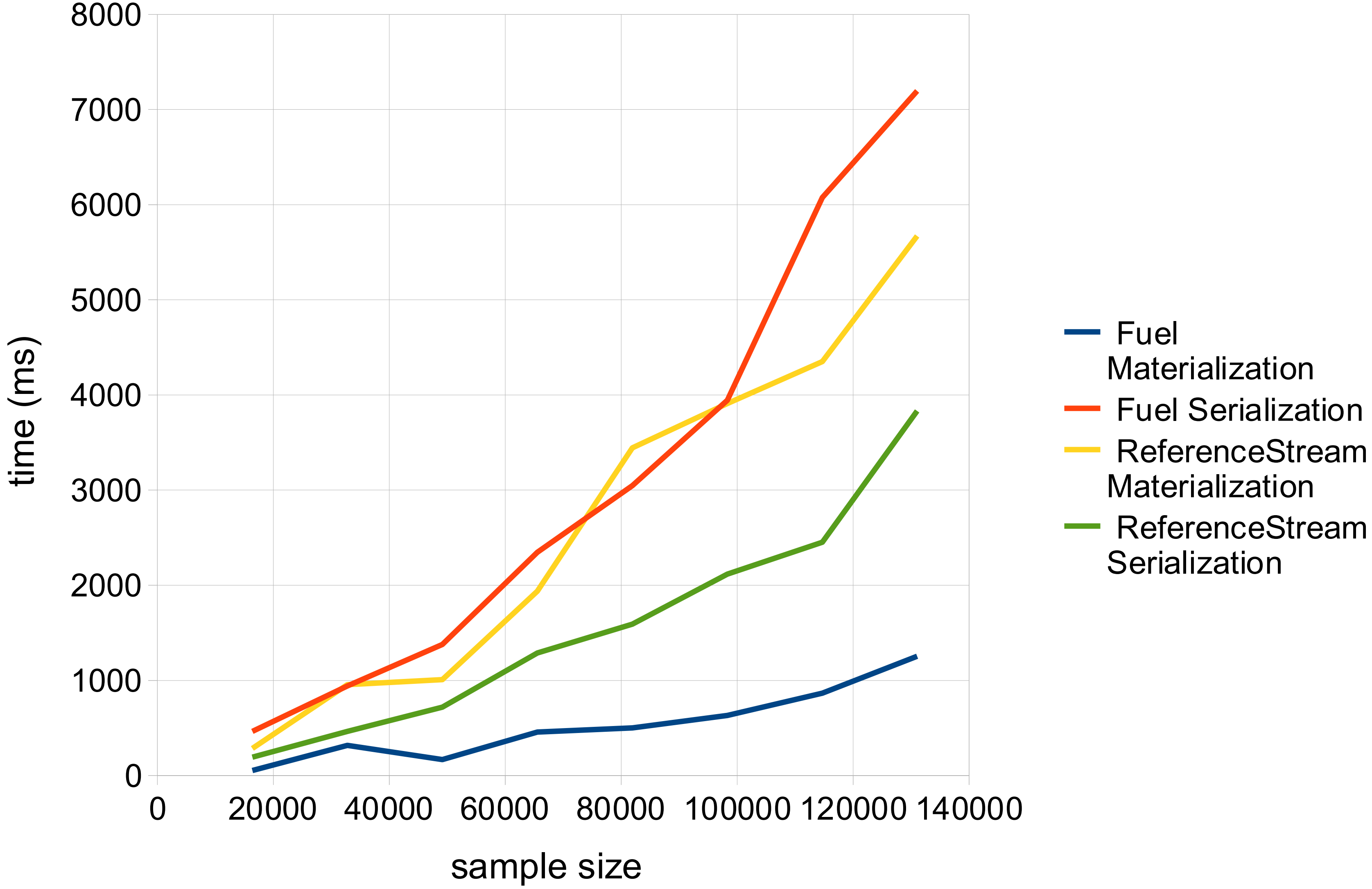
So, this hierarchy of around
20 classes (and still growing)
should be duplicated.
Good idea?

Rectangles

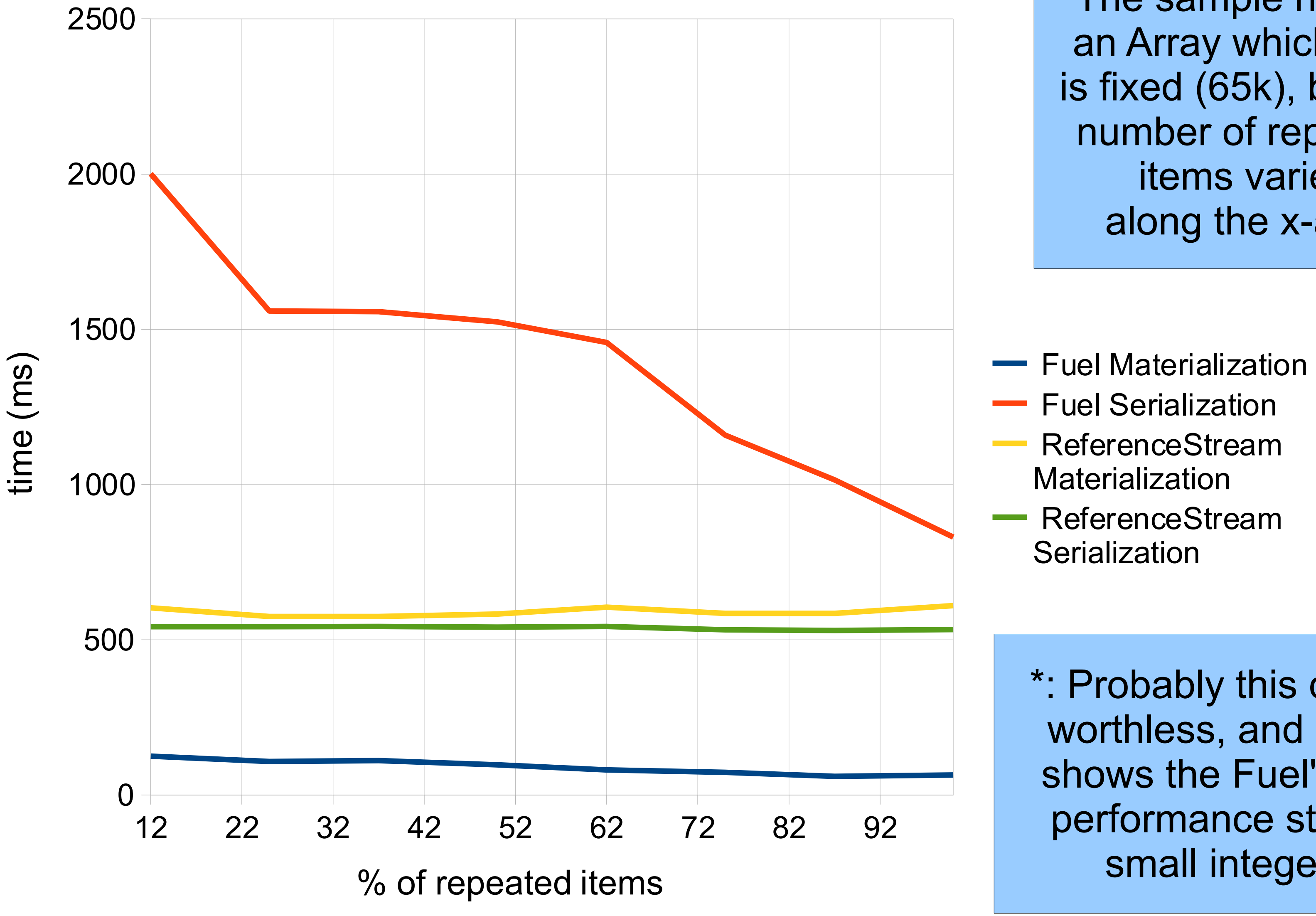


In several runs, this peak was always present. The reason is unknown for now.

Strings


8000
7000
6000
5000
4000
3000
2000
1000
0
time (ms)
0
20000
40000
60000
80000
100000
120000
140000
sample size
Fuel Materialization
Fuel Serialization
ReferenceStream Materialization
ReferenceStream Serialization

Array with repeated integers*



The sample here is an Array which size is fixed (65k), but the number of repeated items varies along the x-axis.

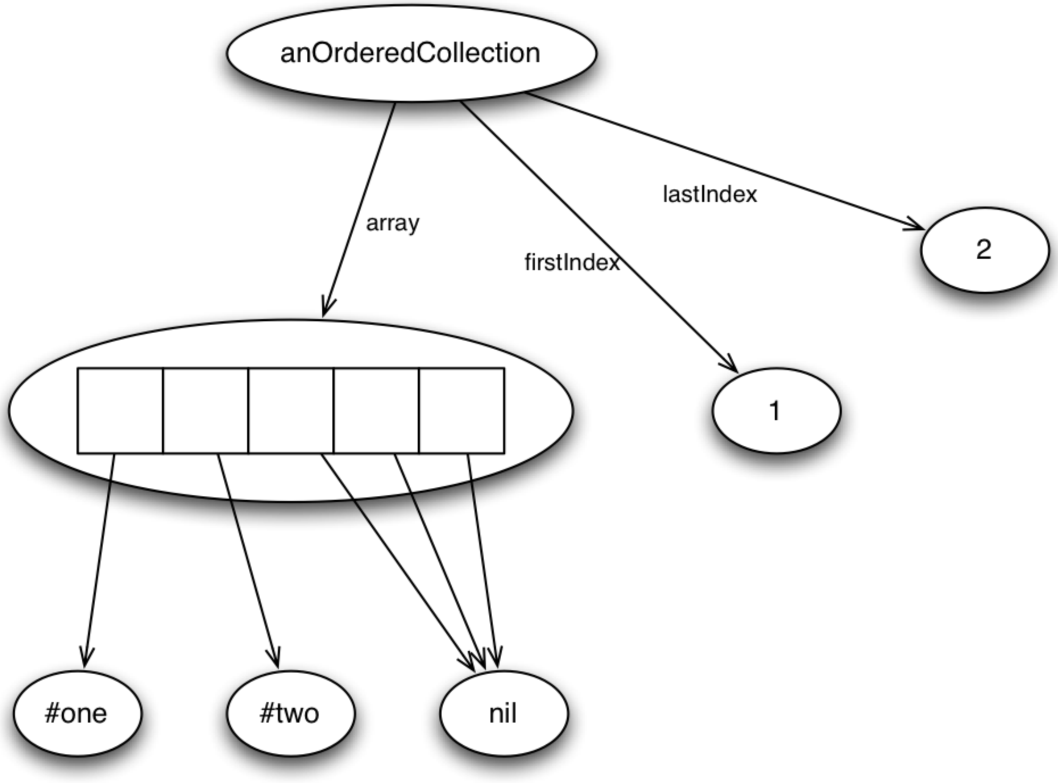
*: Probably this chart is worthless, and it only shows the Fuel's bad performance storing small integers.

Next steps

- Fuel portability between its versions.
- Store a full class (and a trait). Now it only stores a reference to a class (which has to be present in the image).
- Class shape changing.
- Isolate reading from storing?
- Optimize times.

Discussion

Storing with too much detail?



An Ordered Collection is stored with its internal representation

- Not portable
- Slow?

Storing with too much detail?

A compiled method is stored with its full internal representation.

- Avoids using compiler
  - Faster
  - Minimal
- Could be a security problem

Summary

- Problems
  - Identity: reference vs. creation
  - Cycles
  - Class shape changing
  - Portability
  - Minimal loading
- Our solution
- Current implementation + benchmarks
- Discussion