THANKS A LOT!!!

SummerTalk 2011

Student: Martin Dias
Mentor: Mariano Martínez Peck
Object references

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

Diagram:
- aPoint
  - 4
  - 2
An objet graph serializer.
Serialize

Input: an object graph

Output: stream of bytes
**Materialize**

(deserialize)

Input: stream of bytes

Output: an object graph
Once serialized...

Stream of bytes

Database

File

Memory

Socket
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”.

Monday, August 22, 2011
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.
Invest more time in serialization so that objects can then be materialized much faster.
“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.
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. Different sizes and different amounts of elements.

Label:
- What's inside?
- How much?
Each jar has a specific type of element

Jars are in order

Different sizes and different amounts of elements

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.
Each cluster has a specific type of object

- What’s inside?
- How much?

Different sizes and different amounts of elements

Jars are in order

s/jar/cluster
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

Clusters are in order

Label:
- What’s inside?
- How much?

Different sizes and different amounts of elements
s/jar/cluster

- 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.

Label:
- Cluster ID
- Amount of objects

Different sizes and different amounts of elements.
Each cluster has a specific type of object

Clusters are in order

Different sizes and different amounts of elements

Label:
- Cluster ID
- Amount of objects

s/jar/cluster

Monday, August 22, 2011
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

Stream
Pickle format basic

Stream

Cluster 1

Cluster 2

ID   ClassName  InstSize  Inst1  Inst2  ...  InstN

ID   ClassName  InstSize  Inst1  Inst2  ...  InstN

...
### Pickle Format Basic

#### Stream

<table>
<thead>
<tr>
<th>ID</th>
<th>ClassName</th>
<th>InstSize</th>
<th>Inst1</th>
<th>Inst2</th>
<th>...</th>
<th>InstN</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Cluster 1

<table>
<thead>
<tr>
<th>ID</th>
<th>ClassName</th>
<th>InstSize</th>
<th>Inst1</th>
<th>Inst2</th>
<th>...</th>
<th>InstN</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Cluster 2

<table>
<thead>
<tr>
<th>ID</th>
<th>ClassName</th>
<th>InstSize</th>
<th>Inst1</th>
<th>Inst2</th>
<th>...</th>
<th>InstN</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

...
Why the pickle format is so fast in materialization?

<table>
<thead>
<tr>
<th>Standard serializers</th>
<th>Fuel pickle format</th>
</tr>
</thead>
</table>

```plaintext
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. ]. |

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?

<table>
<thead>
<tr>
<th>Standard serializers</th>
<th>Fuel pickle format</th>
</tr>
</thead>
<tbody>
<tr>
<td>Recursive materialization</td>
<td>Iterative materialization</td>
</tr>
</tbody>
</table>

```plaintext
Recursive materialization

```

```plaintext
Iterative materialization

```
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...

Fast serialization (without pickle)

Slow serialization (with pickle)

Monday, August 22, 2011
Fuel requires

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

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

Monday, August 22, 2011
Analysis Phase

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

key (a cluster) | value (a set)
---|---
cluster1 | aPoint | ...
cluster2 | x | y | ...
... | ... | ...
A cluster defines how its objects are serialized and materialized.

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

- bitmaps
- byteArrays
- characters
- floats
- largeIntegers
- scaledDecimals
- smallIntegers
- strings
- symbols
- wideStrings

Serialization Time

- Stomp Serialization
- Magma Serialization
- ImageSegment Serialization
- SRP Serialization
- SmartRefStream Serialization
- Fuel Serialization
Serialization of primitive objects

Memory based stream

- bitmaps
- byteArrays
- characters
- floats
- largeIntegers
- scaledDecimals
- smallIntegers
- strings
- symbols
- wideStrings

Serialization Time

0 175 350 525

File based stream

- bitmaps
- byteArrays
- characters
- floats
- largeIntegers
- scaledDecimals
- smallIntegers
- strings
- symbols
- wideStrings

Serialization Time

0 625 1250 1875 2500

StOMP
SRP
Fuel
Serialization of primitive objects

Memory based stream

File based stream

- bitmaps
- byteArray
- characters
- floats
- largeIntegers
- scaledDecimals
- smallIntegers
- strings
- symbols
- wideStrings

Serialization Time

- Stomp Serialization
- Magma Serialization
- SmartRefStream Serialization
- SRP Serialization
- Fuel Serialization

Monday, August 22, 2011
Materialization of primitive objects

Monday, August 22, 2011
Materialization of primitive objects

![Bar Chart]

- bitmaps
- byteArrays
- characters
- floats
- largeIntegers
- scaledDecimals
- smallIntegers
- strings
- symbols
- wideStrings

Serialization Time

- Stomp Materialization
- Magma Materialization
- ImageSegment Materialization
- SRP Materialization
- SmartRefStream Materialization
- Fuel Materialization

Monday, August 22, 2011
Non primitive objects

![Graph showing serialization times for different types of objects: binaryTrees, pairs, rectangles. The x-axis represents serialization time, and the y-axis represents different serialization methods such as Stomp Serialization, Magma Serialization, ImageSegment Serialization, SRP Serialization, SmartRefStream Serialization, and Fuel Serialization.](image)
Non primitive objects

![Bar chart showing serialization time for different objects and serialization methods: Stomp, Magma, ImageSegment, SRP, and SmartRefStream. The x-axis represents serialization time in milliseconds (0-600), and the y-axis represents the objects: binaryTrees, pairs, and rectangles. The chart compares the performance of StOMP, SRP, and Fuel serialization methods.]
Non primitive objects

- binaryTrees
- pairs
- rectangles

StOMP
SRP
Fuel

Monday, August 22, 2011
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://pharomic.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
anOrigin := 10@20.
aCorner := 30@40.

aRectangle := Rectangle origin: anOrigin corner: aCorner.

serializer := FLSerializer on: aWriteStream.
serializer serialize: aRectangle.
aWriteStream flush; close.

materializer := FLMaterializer on: aReadStream.
materializedRectangle := materializer materialize.
Analysis phase

key (a cluster) value (a IdentitySet)
Serialization

instances

phase

cluster1
aRectangle

cluster2
anOrigin aCorner

cluster3
10 20 30 40
<table>
<thead>
<tr>
<th>cluster ID</th>
<th>instance variable count</th>
<th>instance variable names</th>
<th>class name</th>
</tr>
</thead>
<tbody>
<tr>
<td>cluster1</td>
<td>aRectangle</td>
<td></td>
<td></td>
</tr>
<tr>
<td>cluster2</td>
<td>anOrigin aCorner</td>
<td></td>
<td></td>
</tr>
<tr>
<td>cluster3</td>
<td>10 20 30 40</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Serialization Instances Phase**

- Instance variable names:
  - Rectangle
  - origi
  - corne
  - 1

Monday, August 22, 2011
Serialization

instances

phase

cluster1

aRectangle

cluster2

anOrigin aCorner

cluster3

10 20 30 40

instance variable count

instance variable names

cluster ID

class name

1 Rectangle 2 origi corne 1

instance count

serializeCluster: aCluster
objects: aCollection

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

....
Serialization instances phase

instance variable count

cluster ID

class name

1 Rectangle 2 origi corne 1

serializeCluster: aCluster objects: aCollection

aCollection do: [ instance |
  instancelIndexes
  at: instance
  put: instancelIndexes size + 1 ].
Serialization instances phase

instance variable count

cluster ID

class name

instance variable names

instance count

serializeCluster: aCluster objects: aCollection

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

....

Monday, August 22, 2011
Serialization

references

phase

instancesIndex
(IdentityDictionary)

instances
references
Serialization references phase

instancesIndex (IdentityDictionary)

cluster1
- aRectangle

cluster2
- anOrigin
- aCorner

cluster3
- 10
- 20
- 30
- 40

Serialization references
Serialization References Phase

instancesIndex (IdentityDictionary)

```
cluster1  aRectangle
cluster2  anOrigin aCorner
cluster3  10 20 30 40
```

```
instances
1
2
3
4
5
6
7
```

```
Serialization references phase
```

```
1
2
3
4
5
6
7
```

```
aRectangle  20 40
            1
            5
            7
            3
            2
            4
            6
```

```
10 20 30 40
```

```
5 4
10
20
30
40
```
Serialization references phase

instancesIndex (IdentityDictionary)

Serialization
references
phase

Monday, August 22, 2011
FINAL STREAM

instances

{ 
1 Rectangle 2 origi corne 1
1 Point 2 x y 2

5 4 10 20 30 40

2 3

4 5 6 7

references

{ }
FINAL STREAM

header

\{ 

\begin{array}{c}
\text{7} \quad \text{3} \\
\end{array}

\}

\begin{array}{c}
\text{1} \quad \text{Rectangle} \quad \text{2} \quad \text{original} \quad \text{corner} \quad \text{1} \\
\end{array}

instances

\{ 

\begin{array}{c}
\text{1} \quad \text{Point} \quad \text{2} \quad x \quad y \quad \text{2} \\
\end{array}

\}

\begin{array}{c}
5 \quad 4 \quad 10 \quad 20 \quad 30 \quad 40 \\
\end{array}

references

\{ 

\begin{array}{c}
\text{2} \quad \text{3} \\
\text{4} \quad \text{5} \quad \text{6} \quad \text{7} \\
\end{array}

\}

trailers

\{ 

\begin{array}{c}
\text{1} \\
\end{array}

\}

\begin{array}{c}
\text{root} \\
\end{array}

\text{objectsCounts}

\text{clustersCount}
Materialization

```ruby
| cluster class instVarSize instSize newObject |
| header := self readHeader. |
materializedInstances := (OrderedCollection new: header objectsCountsCount).
(1 to: header clustersCountSize) 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. |
]|
```