TEXAS A&M UNIVERSITY

SERIALIZATION

Vijayasenthil VC

Venkatesh Krishnamoorthy

Aparna Rajeev

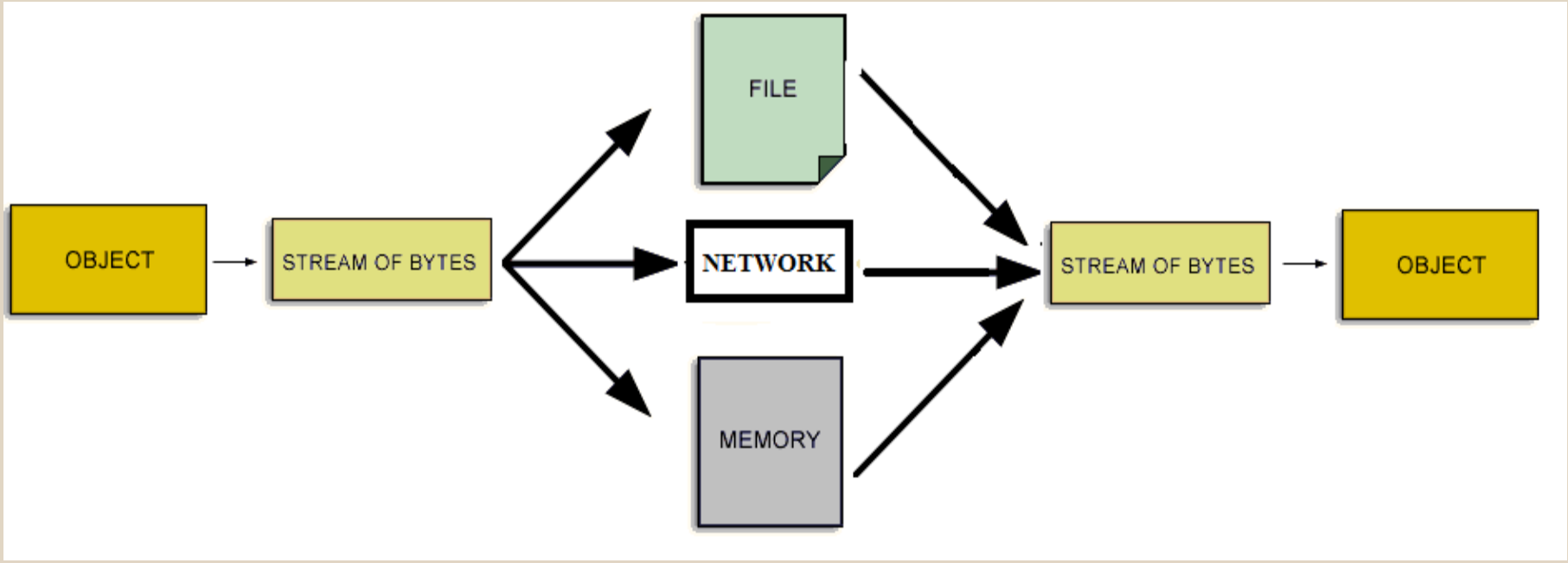
TEXAS A&M UNIVERSITY

Overview

- What is Serialization
- About the Project
- Requirements
- Serialization in other languages
- Features supported
- Interfaces
- Detailed Design with examples
- Discussions
- Future work & References

What is serialization all about

- **Serialization** is the process of converting a data structure or object into a format that can be "*stored*" (for example, in a file or memory buffer, or transmitted across network connection link) and "*resurrected*" later in the same or another computer environment.



Few Applications

- Can send objects while doing RPC
- For Stack ripping in event driven programming
- Any client server system which needs to communicate objects can use this.
- Any place where construction of a data structure after some analysis takes lot of time so it is preferable to save the data structure as it is.
- Eg: The indexed (say disk file indexer) data structures like tries, ternary search trees can be saved and restored instead of indexing every time.
- Can detect changes in time-varying data

About the project

What we did:

- Generate self-identifying stream from C++ objects
- It works independent of byte ordering/machine dependency.
- Currently no member names stored. So saving and retrieval should be done in the same order
- Stream extraction is supported so as to allow network transfer

What we couldn't do:

- Pointers to primitive types except char*
- Virtual pointers

Features supported

- C++ primitive types
- User Defined Types
- Pointers of User Defined Types and Char*
- Arrays. But we need ‘n’
- Self-Identifying Streams
- STL container and adapters
- Complex object graphs
- Inheritance

Requirements

- What goes where
  - Memory Layout/Data Members of the user defined type.

```
Class tree_node
{
   int value;
   string color;
   tree_node *left;
   tree_node * right;
}
```

Serialization in other languages

- No need to modify the classes which need to be serialized
- Underlying framework abstracts object layout from memory layout
- **Reflection**: ability of an object-oriented language to generate objects that describes classes
- Eg:
  - ActionScript
  - JavaScript
  - Java (implements Serializable)
  - C# ([Serializable] attribute in the class)

Serializer Interfaces

```
Class Serializer
{
protected:
        stringstream s;
        map<void*,int> ser_map;
public:
        template <class T> void     save_object(const T& t,
                                    const string name="dummy");
        template <class T> void     save_object(const T*& t,
                                    const string name="dummy");
        template <class T> void     save_array(const T* ptr , int n, const
                                    string name = "dummy");
        const stringstream&         get_stream( );
};
```

Data Structures

- Why stringstream
  - It gives the exact functionality we require
    - Read/write various data types from/to buffer/stream.
  - Eliminates platform dependent issues
  - Less bugs
- Why maps
  - To identify redundant pointer inclusions and avoid loops

Deserializer Interfaces

```
Class Deserializer
{
        map<void*,int> deser_map;
public:
        Deserializer (const stringstream& s);
        const string&                   get_name ( );
        template <class T> void         load_object (T& t);
         template <class T> void        load_object (T*& t);
        int         get_array_size( );
        template <class T> void         load_array (T*& ptr);

};
```

Support for Arrays

Design

- User Defined Types should implement two functions namely “serialize” and “deserialize”.

```
void serialize (Serializer &s){
        s<<value<<color<<left<<right;
}
void deserialize (Deserializer &d){
        d>>value>>color>>left>>right;
}
```

Maintain order

Design - Pointers and Arrays

- To restore pointers, classes can optionally have a *static* function to allocate memory.
    - Eg: *static tree_node* allocate_memory( );*
  - This way we support custom memory management.
  - If allocate_memory function is not provided by the user, we use default "*new*" to allocate memory.
  - If a pointer points to an array of objects, he should use save_array else data will be lost.

  template <class T> void save_array (const T ptr , int n, string name);*
- **Pointers Assumption**:
  - No advantage in storing single pointer of basic types. So no serialization support for them except char*
  - Ask the length before storing Arrays
    - *int get_array_size( );*
    - *template <class T> void load_array (T*& ptr);*

Sample Serializable class

```
Class  tree_node {
    friend class Serializer;
    friend class Deserializer;

    int             value;
    string          color;
    tree_node *     left,
    tree_node*      right;

    void serialize(Serializer &s){
        s<<value<<color<<left<<right;
    }
    void deserialize (Deserializer &s){
        s>>value>>color>>left>>right;
    }
    static tree_node * allocate_memory();
}
```

To make serialize/deserialize private`

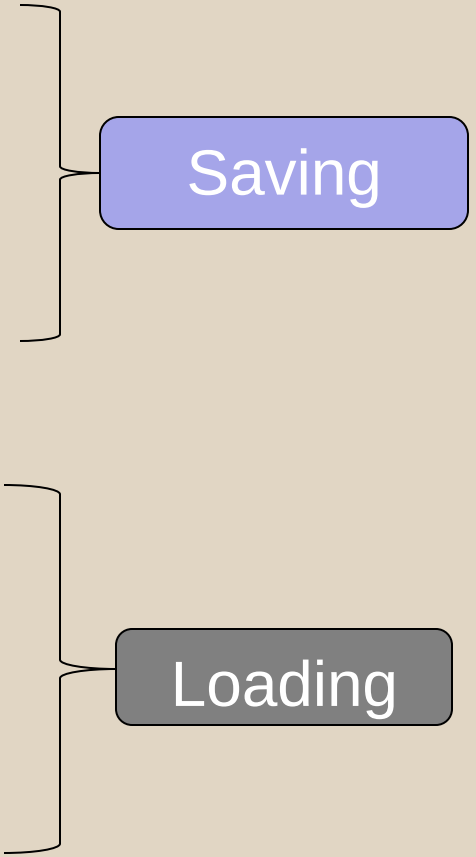
Custom memory management support

Saving/Loading

```
int main()
{
    Student st;
    Serializer ser;
    ser.save_object(st, "Student");
    string_stream str = ser.get_stream();
    /*Write to file / send over network*/

    Student st1;
    Deserializer dz(streamFromFile);
    string objName = dz.get_name( );
    /*Map "name" to object creation*/

    dz.load_object(st1);
}
```



Example: Pointers and Arrays

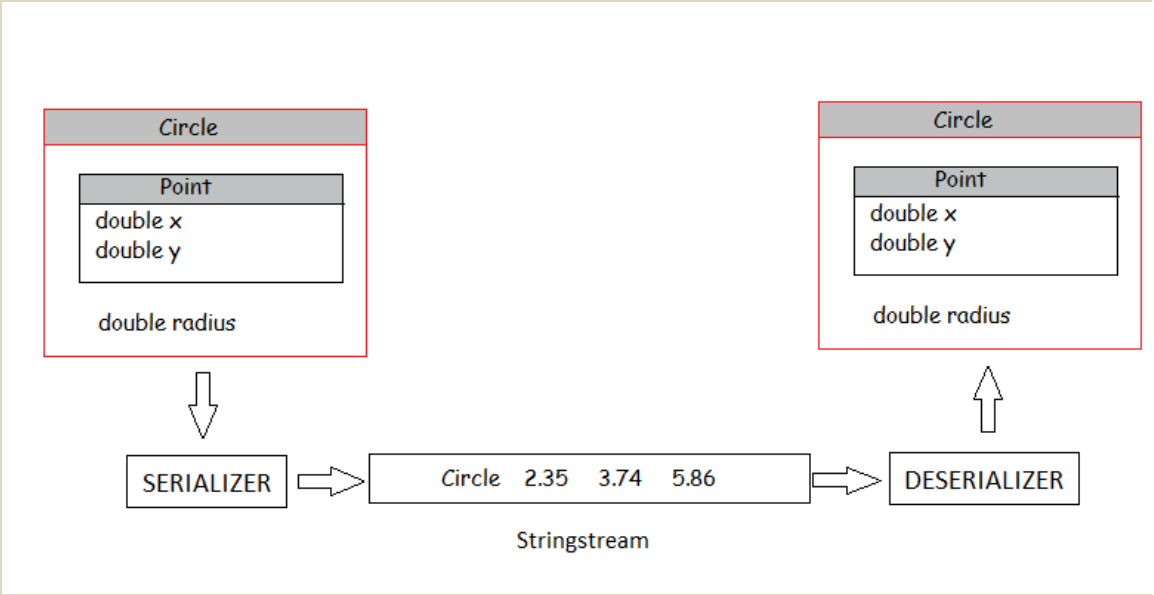
```
int fn(){
    Student* obj;
    ser.save_object(obj, "Student");
    Student st1;
    Deserializer dz(stmFromFile);
    string objName = dz.get_name( );
    /*Map "name" to object creation*/
    dz.load_object(st1);
}
```

```
int fn(){
    Student obj[10]; // fill later
    ser.save_array(obj, 10, "Student");
    Student* st1;
    Deserializer dz(stmFromFile);
    string objName = dz.get_name( );
    int length = get_array_size();
    st1 = new student [10];
    dz.load_array(st1);
}
```

Containment

- If an object contains another object we (de)serialize it by recursively calling (de)serialize function of the objects contained in it.

```
Class Circle
{
  Point p;
  double radius;
  void serialize(Serilaizer & sr){
     sr << p <<radius;
   }
  void deserialize(Deserilaizer &dz){
     dz >> p >>radius;
   }
 }
```



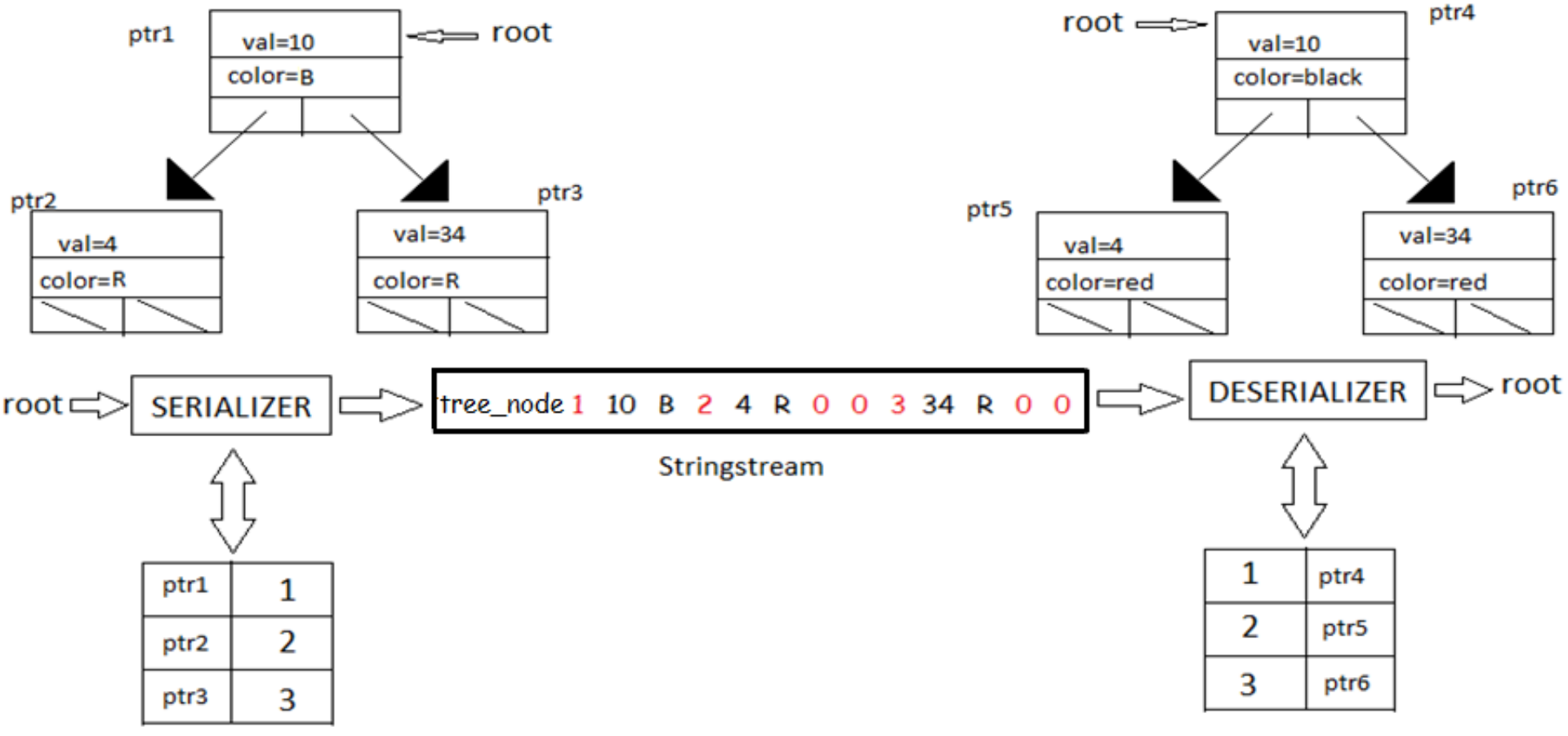
Inheritance

- No virtual pointers support
- Start with derived class object. And typecast it to base class object and push it. Base class::serialize will be called.

```
Class Square : public Shape
{
        int side;
        void serialize (Serializer & sr) {
                sr << *(static_cast<shape *>(this)) <<side;
        }
}
```

As a result “Shape::serialize“is called.

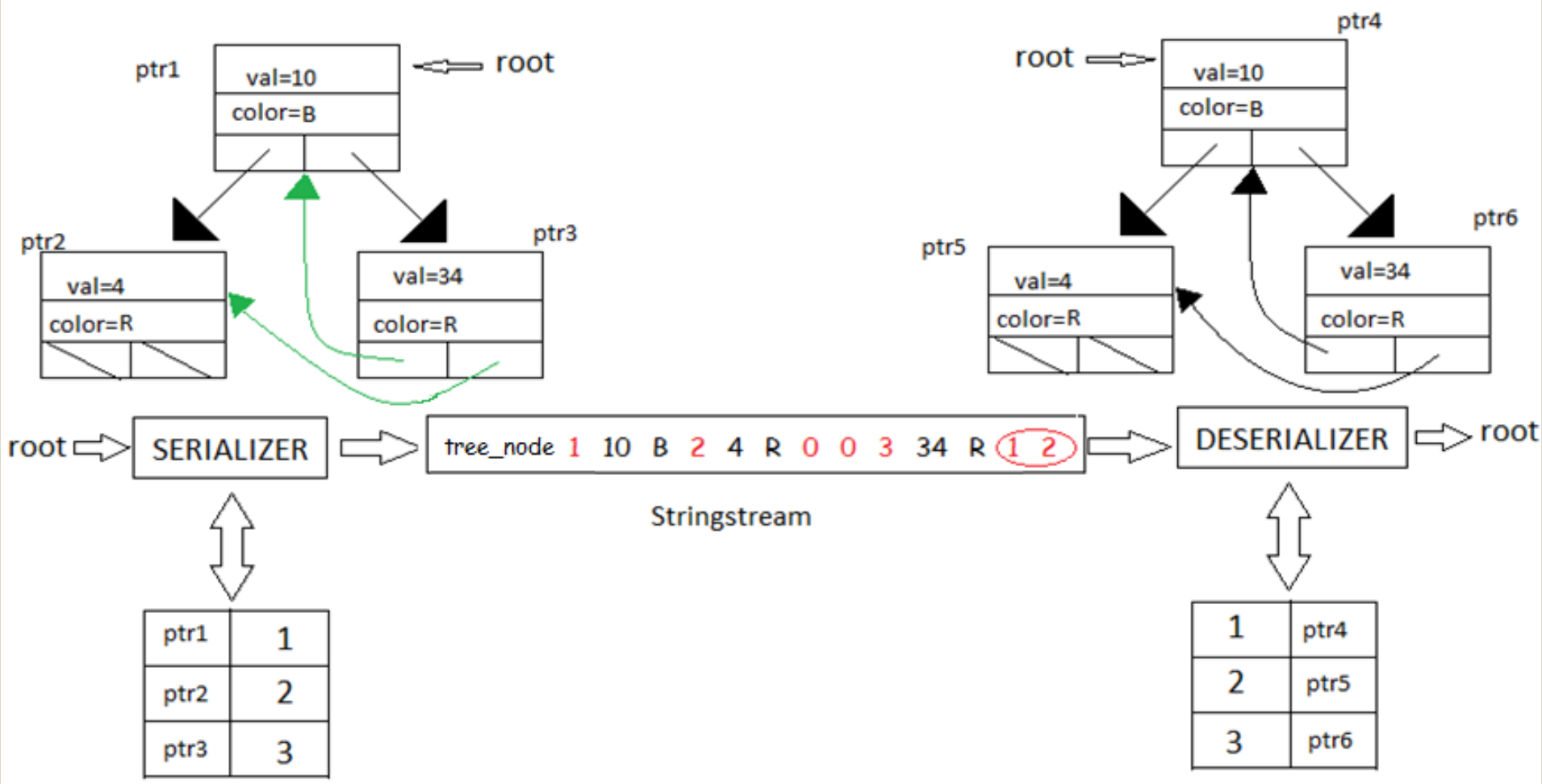
Serializing a Tree



Cycles and Joins

- In order to avoid repeated storage of objects and infinite loops we use a map that has the address as the key and a unique identifier as the value.
- This allows us to check whether a particular pointer has been visited.
- For Eg. the tree_node object would be serialized as
- 1 value color 2 3

Trees with cycles



Discussion: Virtual pointer support

- Issue(s)
  - Hard to know which class the base pointer points to.
- Thoughts:
  - virtualize serialize/deserialize functions.
  - Serializer saves the derived class's ID
  - Before reconstructing, the client asks the ID and constructs appropriate derived class object
  - No way to differentiate normal obj call and ptr call. So we will end up adding IDs always.
  - Can solve this with special serialization function to be invoked from pointers

Discussion: Support for Legacy Classes

- Develop adaptors and serialize it

```
Class Legacy
{
    int val; string name;
    public:
        int getVal();
        void setVal();
        string getName();
        void setName(string n);
};
```

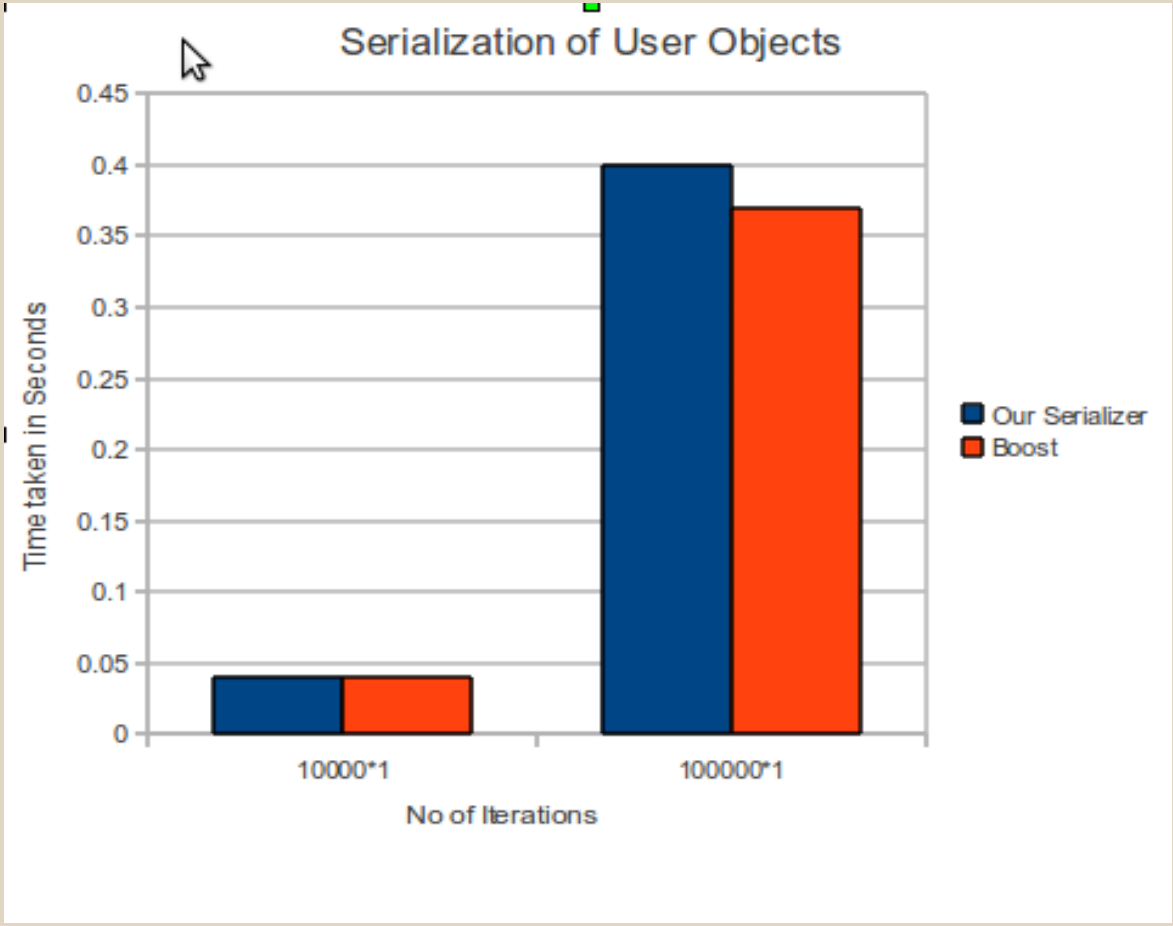
```
Class Adaptor : public Legacy
{
    void serialize(Serializer& sz) {
        sz << getVal() << getName ();
    }
    void deserialize(Deserializer& dz) {
        int v; string n;
        dz >> v >> n;
        setVal(v); setName(n)
    }
}
```

Discussion : Alternate Designs

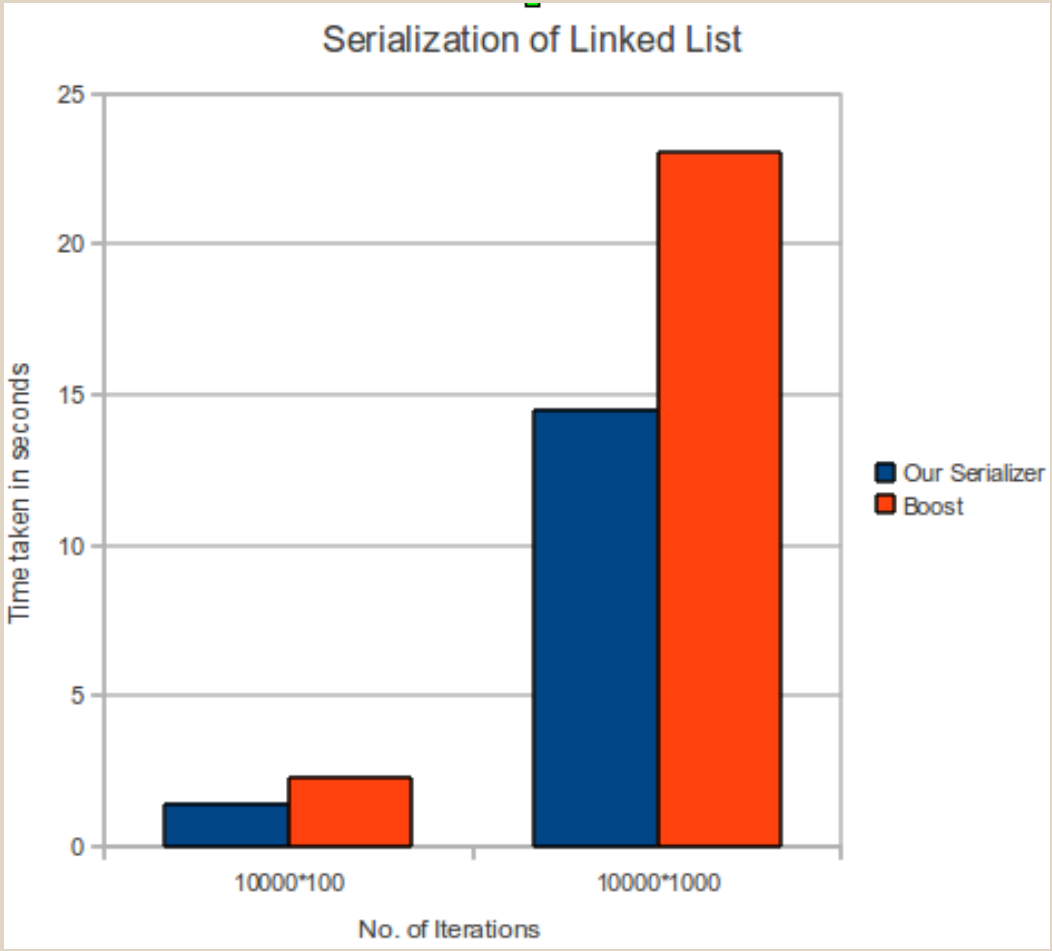
- Memory Dump
  - Cannot work reliably for all architectures
  - Pointers doesn't work
- Member and Offset recording
  - Create a shadow class keeping a list of members. Provide interfaces to create this. Eg: register class, addmember etc.
  - Record the offsets.
  - Access the members via offsets.

Performance(1)

Comparision with boost::textarchive



Performance(2)



TEXAS A&M UNIVERSITY

Future work

- XML format support
- Data compression
- Support for virtual pointers

References

- http://www.parashift.com/c++-faq-lite/serialization.html
- http://www2.research.att.com/~bs/C++0xFAQ.html#default
- http://www.clear-objects.com/Object-Serialization/
- http://java.sun.com/developer/technicalArticles/ALT/Reflection/
- http://www.boost.org/doc/libs/1_46_1/libs/serialization/doc/tutorial.html

QUESTIONS??

Test Code

```
testObj obj(31, 9.73, "simple_string", 'z', 998);
std::string filename(boost::archive::tmpdir());
filename += "/demofile.txt";
{
        std::ofstream ofs(filename.c_str());
        boost::archive::text_oarchive oa(ofs);    // save the schedule
        for(int i=0;i< 10000; i++)
                oa << obj;              //oa << obj1;
}
testObj obj2;
std::ifstream ifs(filename.c_str());
boost::archive::text_iarchive ia(ifs);
for(int i=0;i< 10000; i++)
        ia >> obj;
```

```
class testObj{
        friend class boost::serialization::access;
        int i;
        float j;
        string s;
        char c;
        long l;
}
```