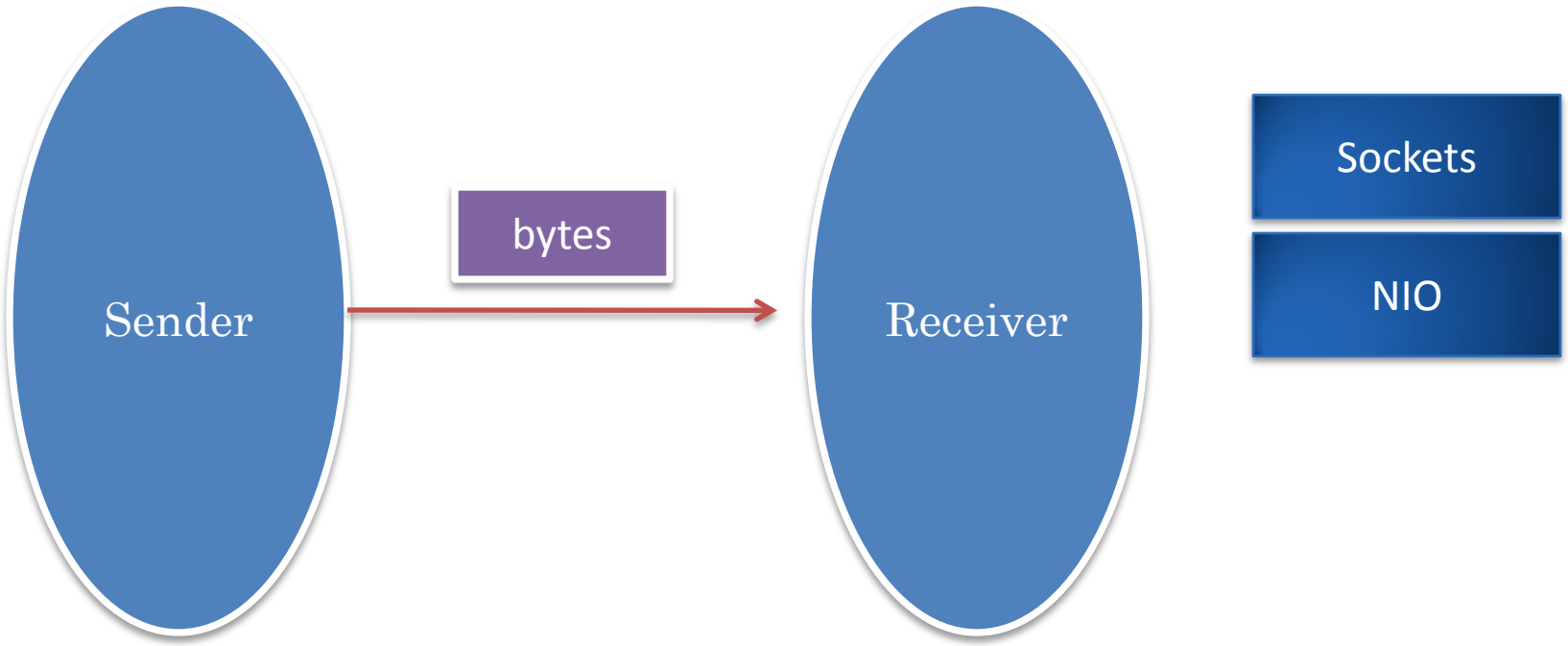
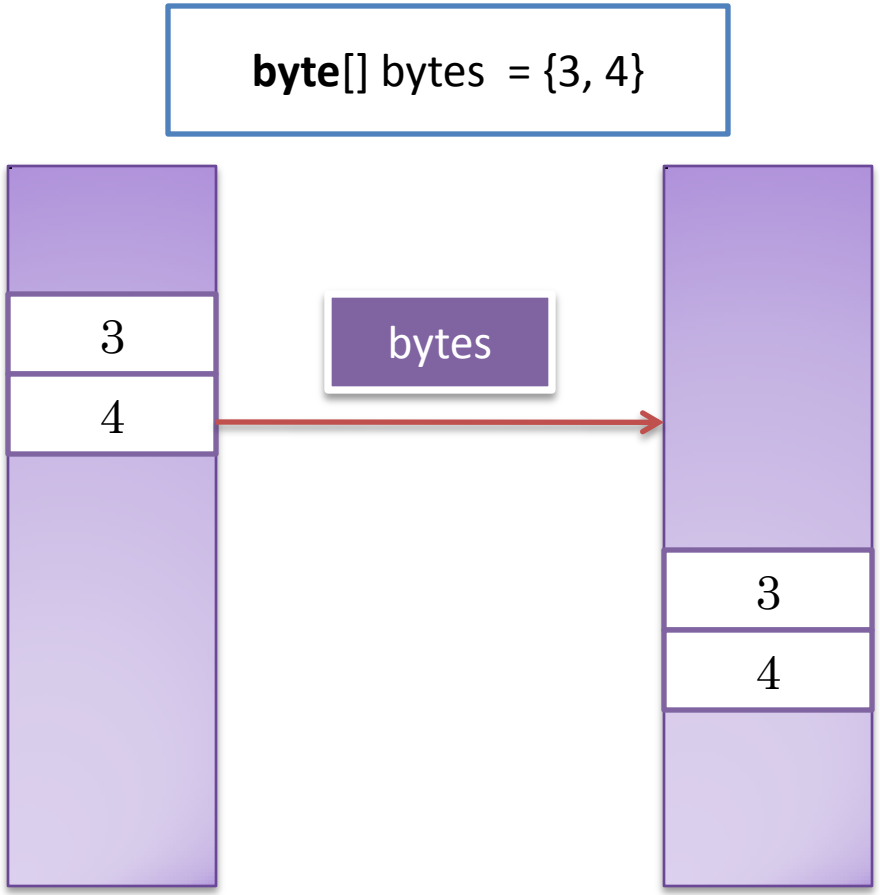
Java Object Communication

Instructor: Prasun Dewan (FB 150, dewan@unc.edu)

Communication so far: Byte Communication

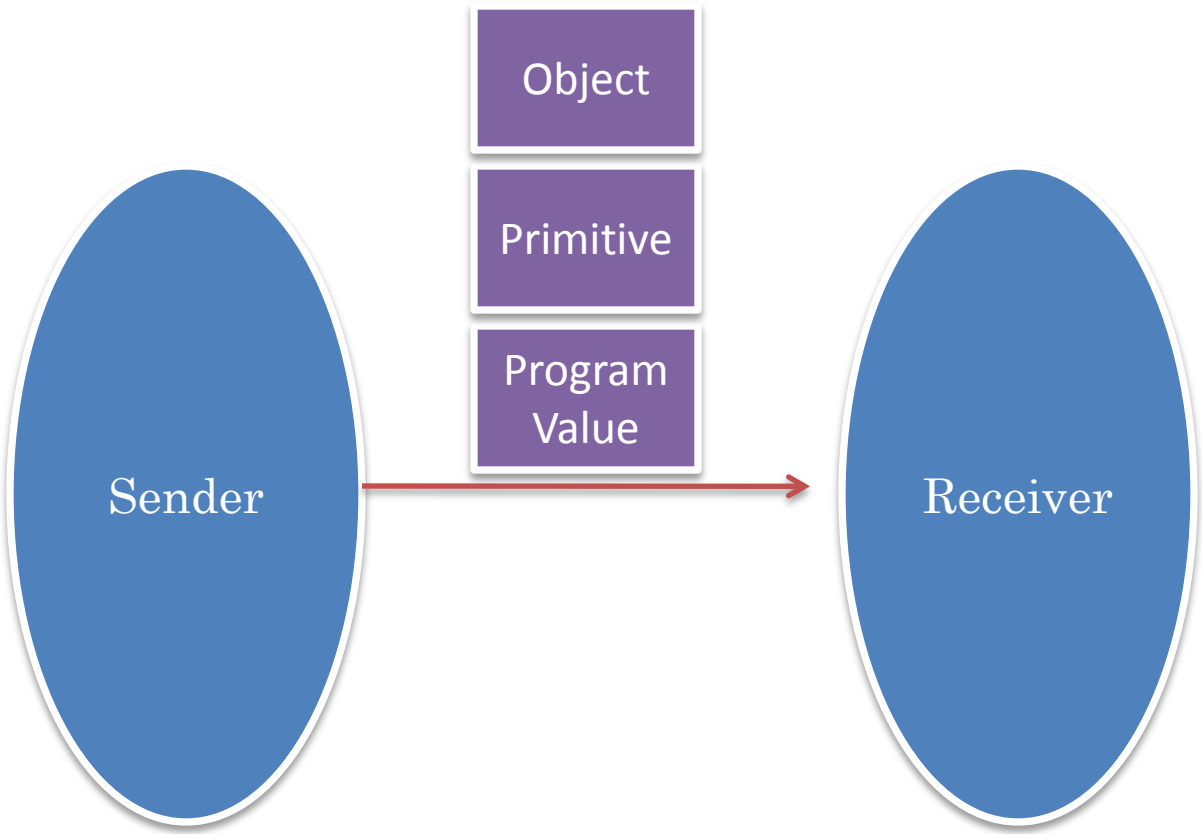


Byte Communication



The receiving site creates in its memory a copy of a block of memory at the sending site

Object Communication: Values of Multiple Types



Object communication even though communicating data structures that include objects and primitive values (Data structure communication)

Motivation and application?

Motivation for Non Byte Communication

Originally, RPC was implemented directly on byte communication

Java developed idea of object data communication

Remote procedure call (RMI)

Blocking stream object communication (Object Stream,)

Blocking byte communication (Sockets)

Separate layer can imply more flexibility

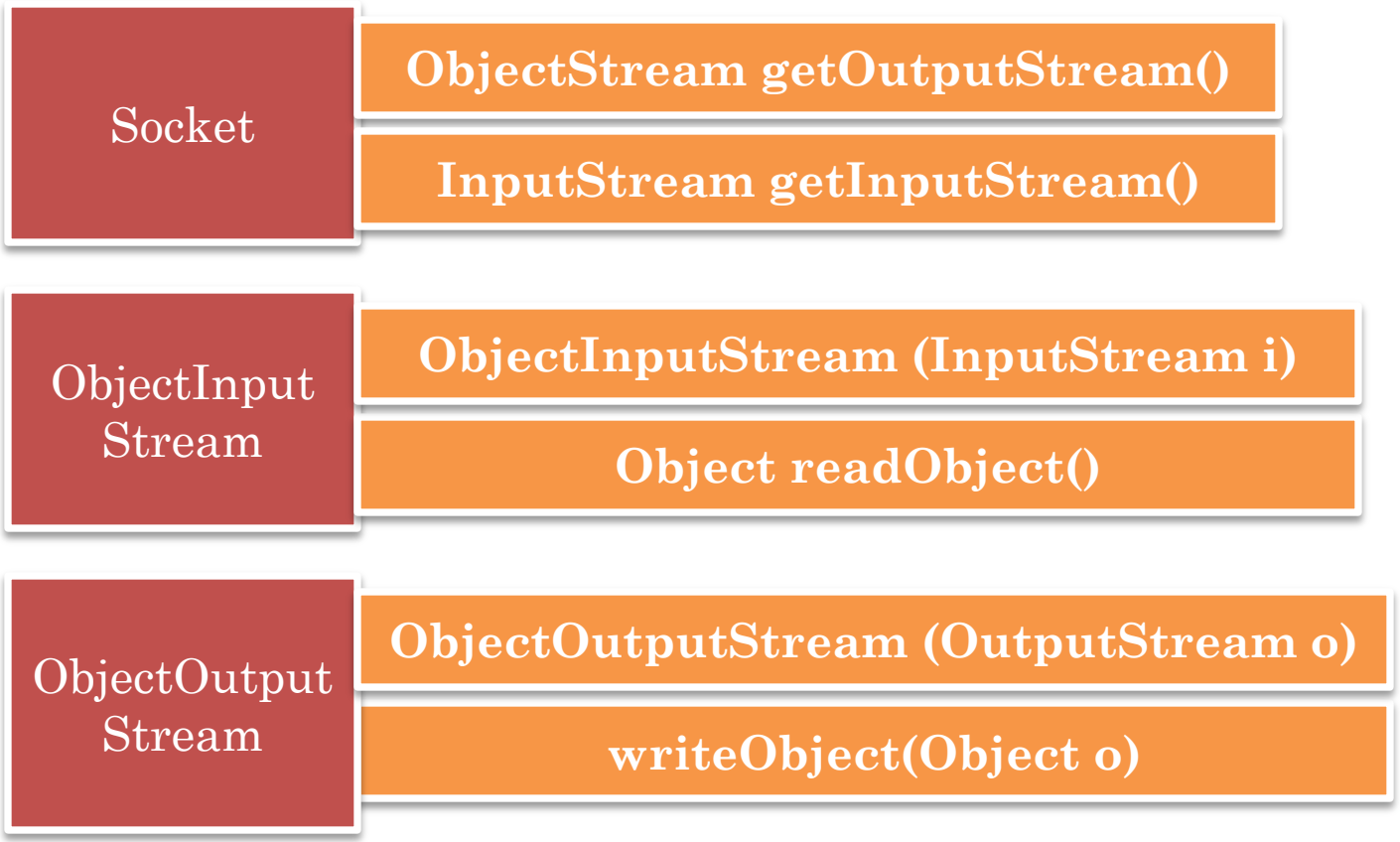
Data Communication Implemented by Who

Infrastructure (Language, Library)

Programmer (of Communicated Objects)

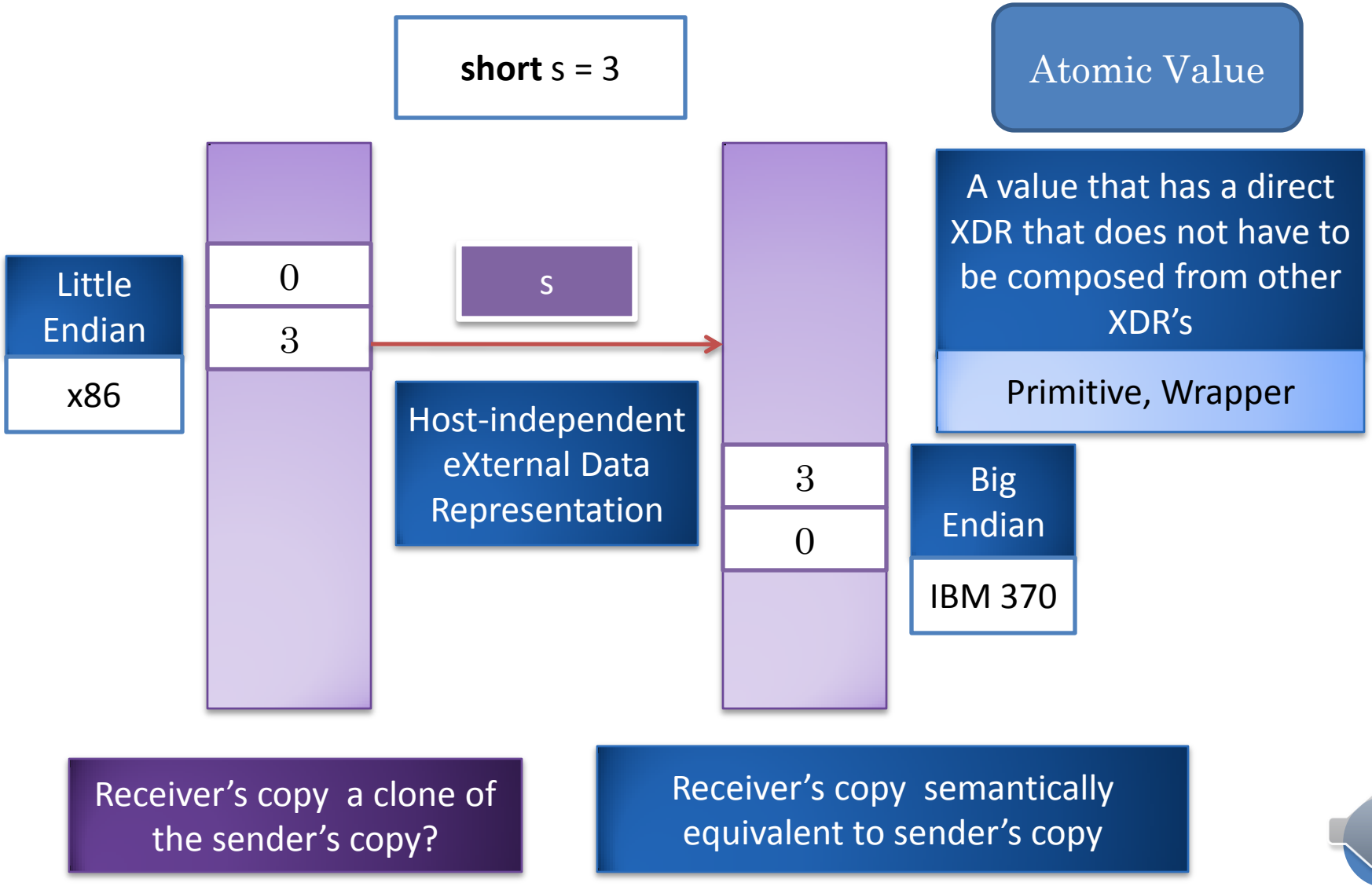
Both (Programmer code called by Infrastructure)

Java Object Output and Input Streams

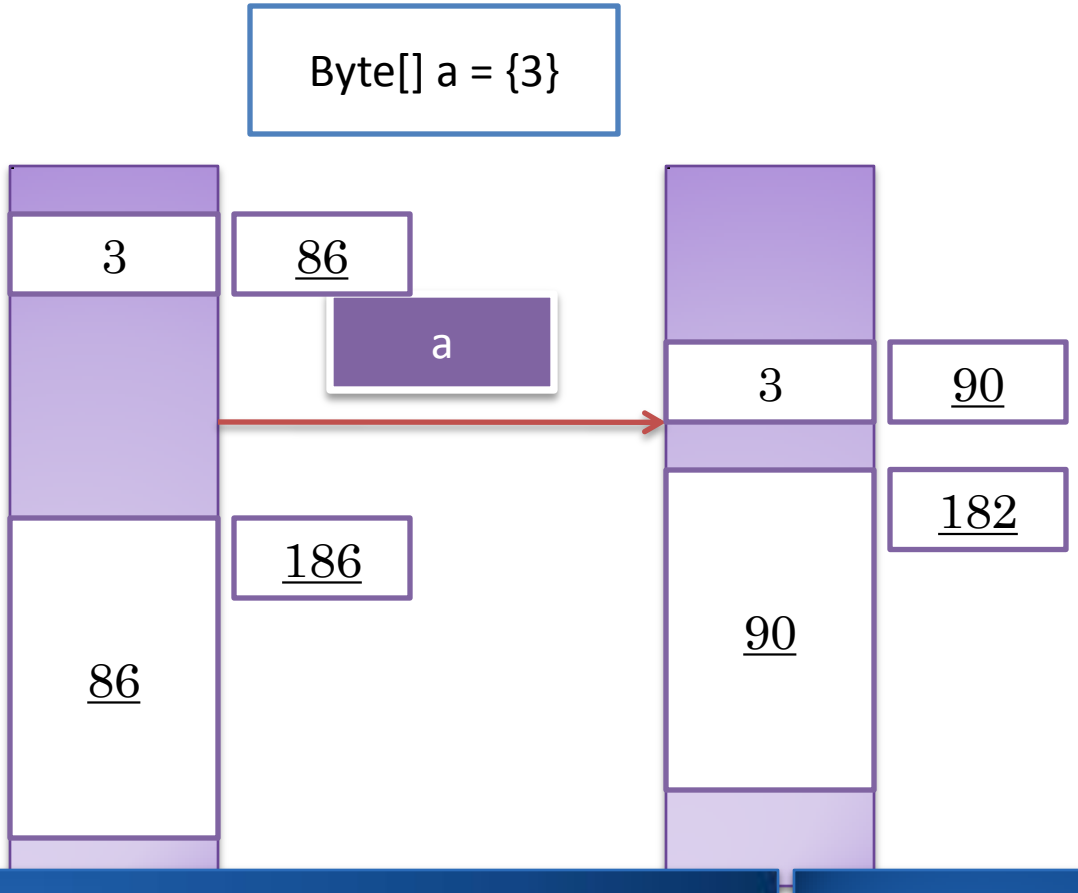



```
ObjectOutputStream socketOut = new ObjectOutputStream(socket.getOutputStream());
socketOut.writeObject(3);
ObjectInputStream socketIn  = ObjectOutputStream(socket.getInputStream());
Integer readVal = (Integer) socketIn.readObject();
```

Object Communication and External Data Representation (XDR)



Structure Communication: Serialized XDR



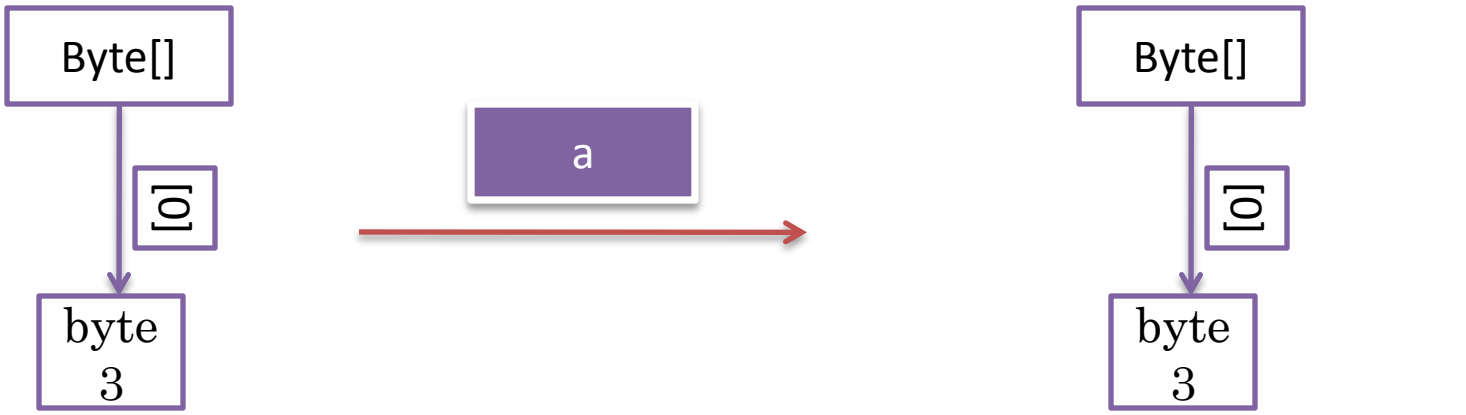
Serialization: gathering a program data structure possibly scattered in memory into a serial byte sequence (containing XDRs of primitive values)

Deserialization: Reconstructing byte sequence into equivalent data structure, possibly scattering the value

Sending Graphs

Byte[] a = {3}

Arrows to primitive and object component



Relationship between sent and received object?

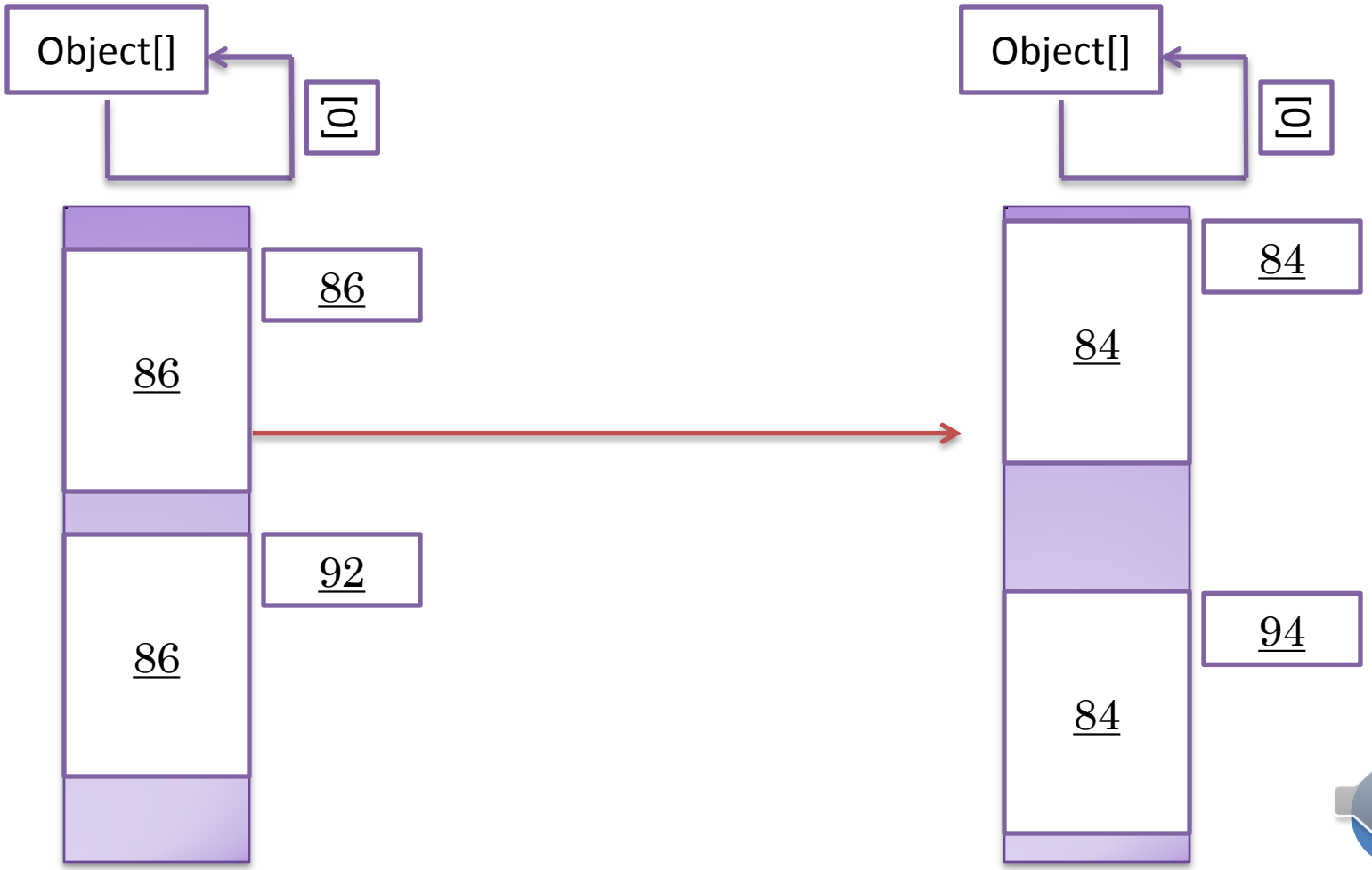
Received data structure is isomorphic to the sender's data structure

And is of the same type

Corresponding leaf primitive values represent the same abstract value

Non Tree Structures

```
Object[] objects = new Object[1];
objects[0] = objects
```



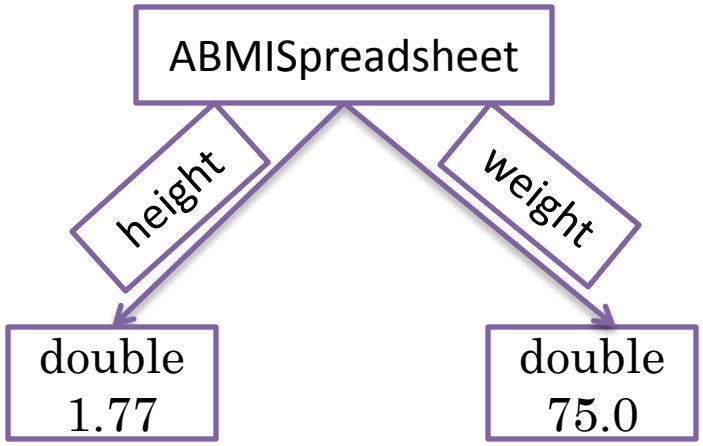
Programmer-Defined Class: ABMISpreadsheet

```
public class ABMISpreadsheet implements BMISpreadsheet {
  double height = 1.77;
  double weight = 75;
  public double getWeight() {return weight;}
  public void setWeight(double newWeight) {
    weight = newWeight;
  }
  public double getHeight() {return height;}
  public void setHeight(double newHeight) {height = newHeight;}
  public double getBMI() {return weight/(height*height);}
}
```

Structure?

Physical Structure

BMISpreadsheet b = **new** ABMISpreadsheet()



Every object serializable?

Physical structure defined by instance variables stored in memory

Internal nodes labeled by class name

Each component of an object is a value stored in an instance variable declared in class

Leaf nodes labeled type and value

Location-Dependent Objects

Requirement: Receiver's copy semantically equivalent to sender's copy

System.in	Socket s	File f	TextField t

Several objects have location-dependent semantics

The OS or some other external software such as window system keeps state that define full semantics

The external software may not exist at the other end, may not be known to serialization system, and may not be willing or able to create equivalent state

e.g. file, text field, socket, System.in

Not All Objects are Serializable

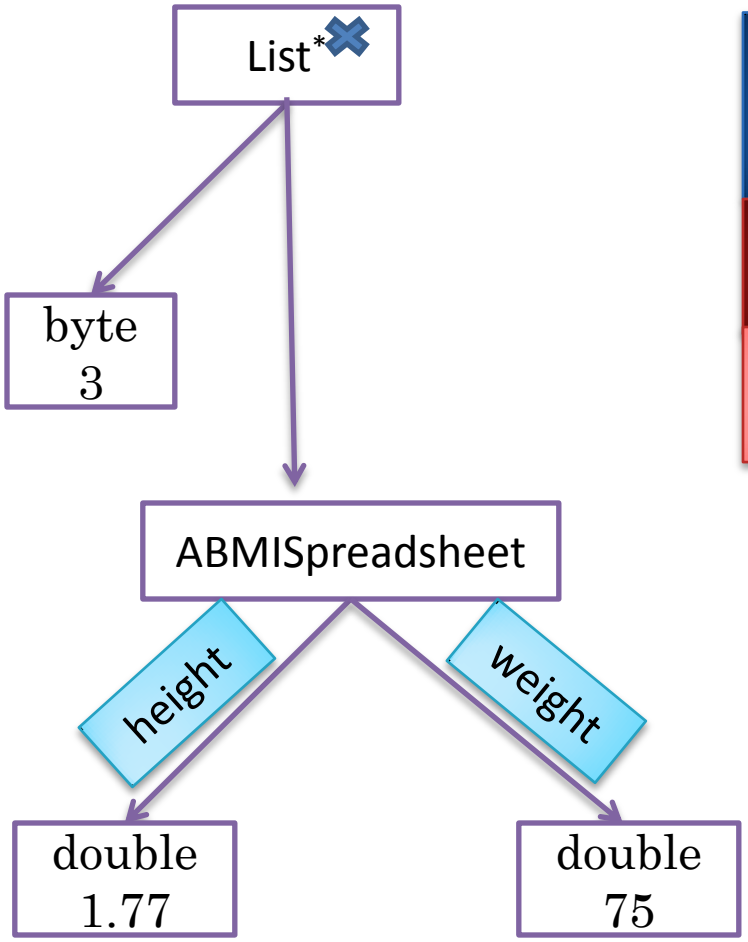
Sending an isomorphic copy does not guarantee semantic equivalence

How to tell some generic serialization system that some type is not serializable?

Programmer used some directive

Ideally with language support

Shallow Tagging

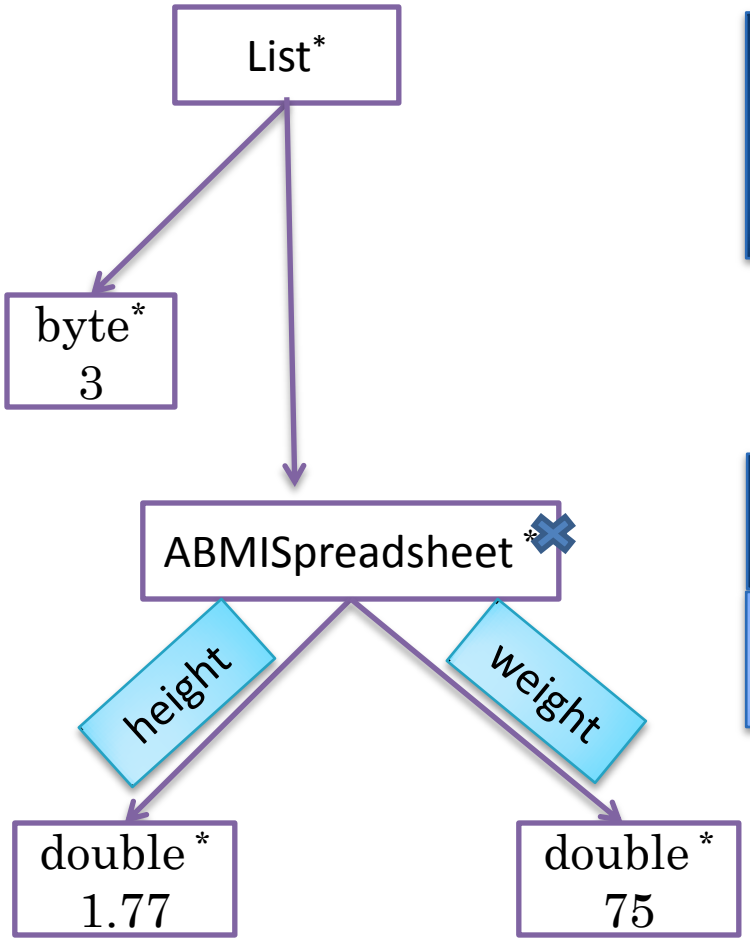


An object is serialized only if it is tagged as serializable

Structure is often determined at runtime

Can add a serializable or non serializable component

Deep Tagging



An object is sesrialized only if every component of its (logical or physical) structure is tagged as serializable

Not serializable even though top object is Serializable

Runtime exception

Interface Based Tag

Object O is Serializable if O instance of Serializable

Serializable is a an empty interface – has no methods

Any class implements it if it says so

BMISpreadsheet

```
import java.io.Serializable;
public interface BMISpreadsheet extends Serializable {
  double getWeight();
  void setWeight(double newWeight);
  double getHeight();
  void setHeight(double newHeight);
  double getBMI();
}
```

No Changes to ABMISpreadsheet

```
public class ABMISpreadsheet implements BMISpreadsheet {
  double height = 1.77;
  double weight = 75;
  public double getWeight() {return weight;}
  public void setWeight(double newWeight) {
    weight = newWeight;
  }
  public double getHeight() {return height;}
  public void setHeight(double newHeight) {height = newHeight;}
  public double getBMI() {return weight/(height*height);}
}
```

Inherited Tag

```
public class ADistributedBMISpreadsheet extends ABMISpreadsheet {
     Socket peer;
     public Socket getSocket();
    ….

}
```

Instance of ADistributedBMISpreadsheet is instance of Serializable

Fundamental Problem with Inherited Tag

```
import java.io.Serializable;
public interface BMISpreadsheet extends Serializable {
  double getWeight();
  void setWeight(double newWeight);
  double getHeight();
  void setHeight(double newHeight);
  double getBMI();
}
```

When a type is created, can guarantee behavior of only instances of that type

Not of instances of yet to be created subtypes

C# Attributes

```
[Serializable]
public class ABMISpreadsheet:BMISpreadsheet {
    …
}
```

An object is serializable if its class holds the non inheritable Serializable attribute

Semi-Serialized Structures?

```
public class ADistributedBMISpreadsheet extends ABMISpreadsheet {
     Socket peer;
     public Socket getSocket();
    ….

}
```

Send structure without socket?

Socket may be constructed from serializable components such as host name and port

Semi-Serialized Structures

Send part of the logical structure to another site?

Certain parts may be derived from other parts

Not sending these parts more efficient

These parts may not be serializable (e.g. some socket)

Derived Structure in BMI Example?

```
public class AnotherBMISpreadsheet implements BMISpreadsheet{
  double height = 1.77;
  double weight = 75;
  double bmi = calculateBMI();
  public double getHeight() {return height;}
  public void setHeight(double newHeight) {
    height = newHeight;
    bmi = calculateBMI();
  }
  public double getWeight() {return weight;}
  public void setWeight(double newWeight) {
     weight = newWeight;
     bmi = calculateBMI();
  }
  public double getBMI() { return bmi;}
  double calculateBMI() {
     return weight/(height*height);
}
```

How does infrastructure know what is derived?

Annotations, keywords, attributes can be used to tag

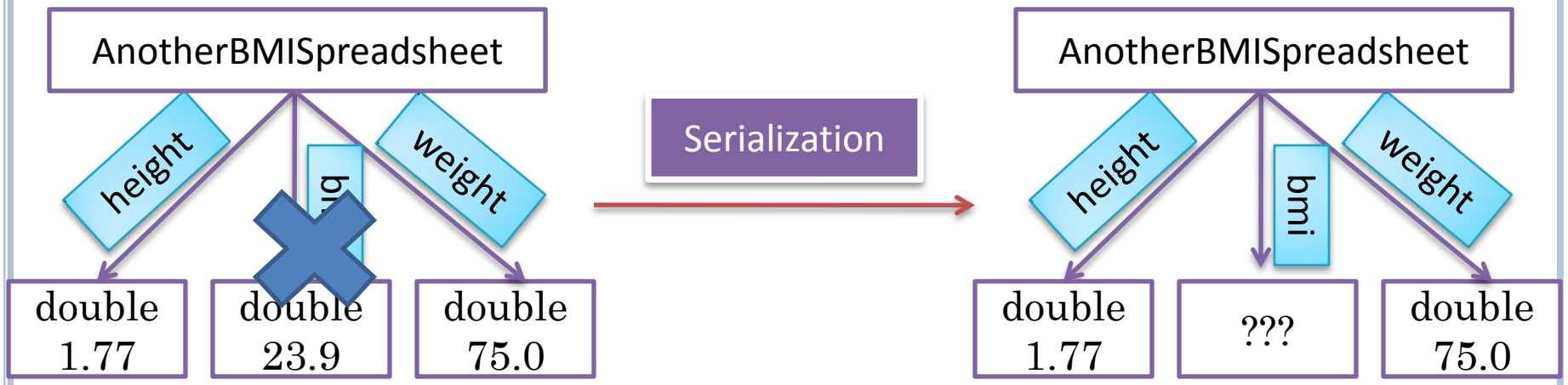
Java Transient Keyword Tag for Physical

```
public class AnotherBMISpreadsheet implements BMISpreadsheet{
  double height = 1.77;
  double weight = 75;
  transient double bmi = calculateBMI();
  public double getHeight() {return height;}
  public void setHeight(double newHeight) {
    height = newHeight;
    bmi = calculateBMI();
  }
  public double getWeight() {return weight;}
  public void setWeight(double newWeight) {
     weight = newWeight;
     bmi = calculateBMI();
  }
  public double getBMI() { return bmi;}
  double calculateBMI() {
     return weight/(height*height);
  }
}
```

Logical?

Partially Serialized Objects

BMISpreadsheet b = **new** AnotherBMISpreadsheet()



Constructors usually used to initialize variables

Both transient and non transient

How to initialize only transient when infrastructure is deserializing?

Initializing Method called by Programmer

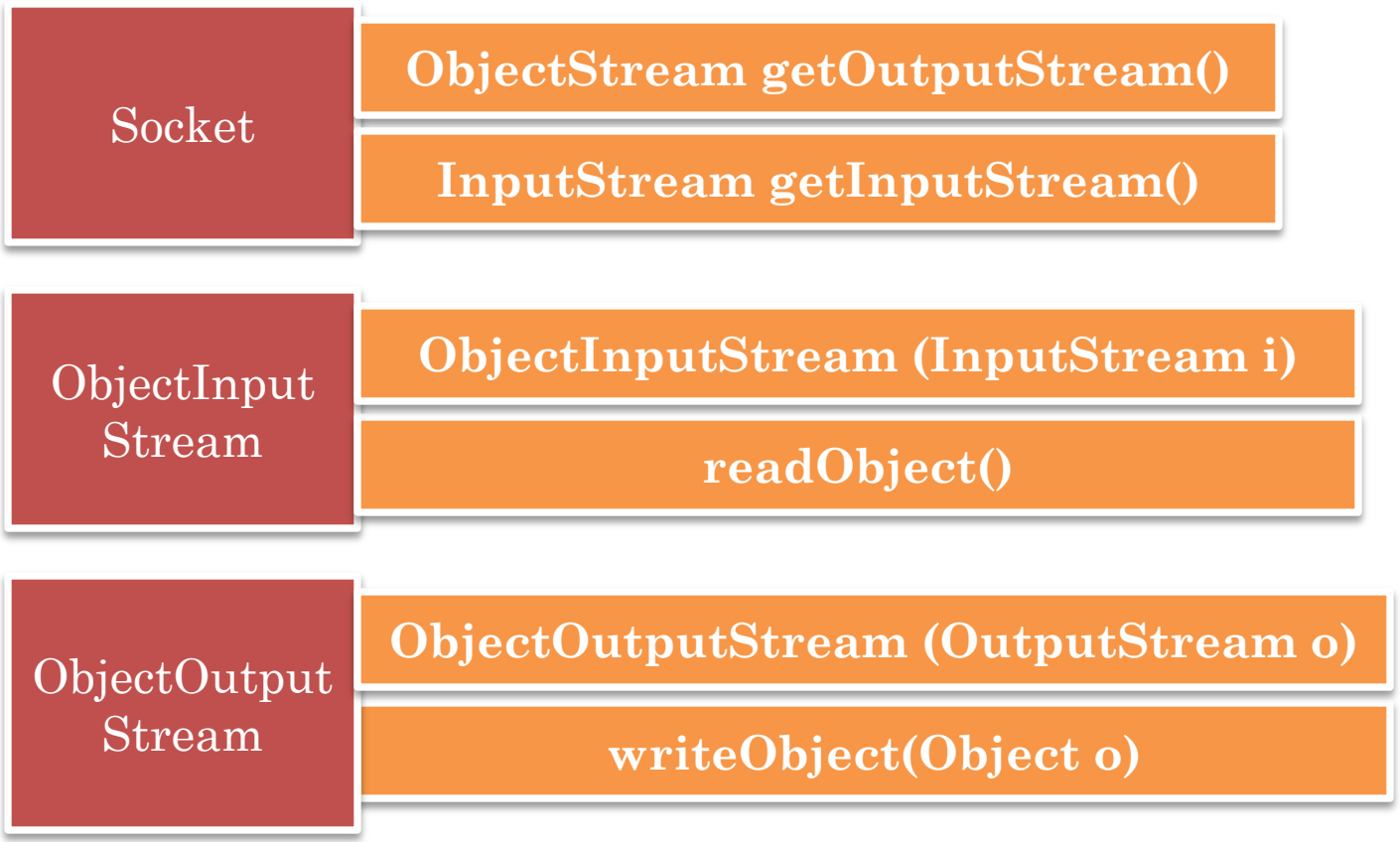
```
public class AnotherBMISpreadsheet implements BMISpreadsheet{
  double height = 1.77;
  double weight = 75;
  transient double bmi = calculateBMI();
  public double getHeight() {return height;}
  public void setHeight(double newHeight) {
    height = newHeight;
    bmi = calculateBMI();
  }
  public double getWeight() {return weight;}
  public void setWeight(double newWeight) {
     weight = newWeight;
     bmi = calculateBMI();
  }
  public double getBMI() { return bmi;}
  double calculateBMI() {
     return weight/(height*height);
  }
  public void initSerializedObject() {
    bmi = calculateBMI();
   }
```

Extensibility General Goals and Architecture

A class can select which subobjects are serialized to serialize it

Infrastructure determines if object has been visited before and adds type names if necessary

Java Object Output and Input Streams




```
ObjectOutputStream socketOut = new ObjectOutputStream(socket.getOutputStream());
socketOut.writeObject("hello world");
```

Serializable Pattern based Functions

Serializable

private readObject(ObjectInputStream s)

private writeObject(ObjectOutputStream s)

These methods called if they exist

Also used to initialize transients structures

ObjectInput Stream

defaultReadObject()

ObjectOutput Stream

defaultWriteObject()

Default methods (de)serialize non transients

Class without Custom Serialization

```
public class AnotherBMISpreadsheet implements BMISpreadsheet{
  double height = 1.77;
  double weight = 75;
  transient double bmi = calculateBMI();
  public double getHeight() {return height;}
  public void setHeight(double newHeight) {
    height = newHeight;
    bmi = calculateBMI();
  }
  public double getWeight() {return weight;}
  public void setWeight(double newWeight) {
     weight = newWeight;
     bmi = calculateBMI();
  }
  public double getBMI() { return bmi;}
  double calculateBMI() {
     return weight/(height*height);
  }
  public void initSerializedObject() {
    bmi = calculateBMI();
   }
```

Programmer-Defined readObject

```
private void readObject(ObjectInputStream stream){
   try {
     stream.defaultReadObject();
     initSerializedObject();
   } catch (Exception e) {
     e.printStackTrace();
   }
}
```

Class does not have a writeObject

Implicit System-Defined Write Object

```
  private void writeObject(ObjectInputStream stream)
           throws IOException{
    stream.defaultWriteObject();

  }
}
```

Default behavior of Java

Extending Class With Custom Serialization

```
public class ANamedBMISpreadsheet extends
        AnotherBMISpreadsheet implements NamedBMISpreadsheet{
  public static final int HIGH_BMI = 27;
  transient boolean isOverWeight = calculateOverWeight();
  String name = "";
  boolean calculateOverWeight() {return bmi > HIGH_BMI;}
  public boolean isOverWeight() {
    return isOverWeight;
   }
  public String getName() {return name;}
  public void setName(String newValue) {
    name = newValue;
  }
  public void initSerializedObject() {
    super.initSerializedObject();
    isOverWeight = calculateOverWeight();
  }
```

Extending Class With Custom Serialization

```
  private void readObject(ObjectInputStream stream){
     try {
       stream.defaultReadObject();
     } catch (Exception e) {
        e.printStackTrace();
     }
   }
}
```

defaultReadObject() reads declared variables

No need to call **super**.readObject()

Java calls explicit or implicit readObject() on each super class

Cannot call super() : non private methods not recognized

Would cause multiple reads

Method not really needed as it is the default readObject()

Missing ReadObject

AnotherBMISpreadsheet

```
private void readObject(ObjectInputStream stream){
  try {
    stream.defaultReadObject();
    initSerializedObject();
  } catch (Exception e) {
    e.printStackTrace();
  }
}
```

Stream Corrupted

ANa dBMISpreadsheet

```
  private void readObj (ObjectInputStream stream){
     try {
       stream.defaultReadObject();
     } catch (Exception e) {
        e.printStackTrace();
     }
   }
}
```

Externalizable Interface based Functions

Externalizable

- **readExternal (ObjectIn)**
- **writeExternal(ObjectOut)**

These methods must exist if interface implemented

ObjectIn

- **Object readObject(Primitive)(Object (Primitive))**

ObjectOut

- **writeObject(Primitive)(Object(Primitive))**

No default read and write of Object

Externalizable use in ABMISPreadsheet

```
public void readExternal(ObjectInput in) {
   try {
     height = in.readDouble();
     weight = in.readDouble();
     initSerializedObject();
   } catch (Exception e) {e.printStackTrace();}
}
public void writeExternal(ObjectOutput out)  {
   try {
     out.writeDouble(height);
     out.writeDouble(weight);
   } catch (Exception e) {e.printStackTrace();}
}
```

Both public interface methods must be implemented and no default read or write methods

Extending Class With Custom Serialization

```
public void readExternal(ObjectInput in) {
  try {
    super.readExternal(in);
    name= in.readLine();
 } catch (Exception e) {e.printStackTrace();}
}
public void writeExternal(ObjectOutput out)  {
  try {
    super.writeExternal(out);
    out.write(name.getBytes());
  } catch (Exception e) {e.printStackTrace();}
}
```

Can and must call super on public member

AnObjectHistory

```
public class AnObjectHistory<ElementType> implements
         ObjectHistory<ElementType
  public final int MAX_SIZE = 50;
  Object[] contents = new Object[MAX_SIZE];
  int size = 0;
  public int size() {return size;}
  public ElementType get(int index) {
    return (ElementType) contents[index];
  }
  boolean isFull() {return size == MAX_SIZE;}
  public void add(ElementType element) {
  if (isFull())
    System.out.println("Adding item to a full history");
  else {
    contents[size] = element;
    size++;
   }
}
```

Serialize how?

AnObjectHistory

```
public class AnObjectHistory<ElementType> implements
         ObjectHistory<ElementType
  transient public final int MAX_SIZE = 50;
  transient Object[] contents = new Object[MAX_SIZE];
  int size = 0;
  public int size() {return size;}
  public ElementType get(int index) {
    return (ElementType) contents[index];
  }
  boolean isFull() {return size == MAX_SIZE;}
  public void add(ElementType element) {
  if (isFull())
    System.out.println("Adding item to a full history");
  else {
    contents[size] = element;
    size++;
   }
}
```

AnObjectHistory (Initialization and Extension)

```
  private void writeObject(ObjectOutputStream stream) {
    try {
        stream.defaultWriteObject( );
        for (int i = 0; i < size; i++)
            stream.writeObject(contents[i]);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
  private void readObject(ObjectInputStream stream) {
    try {
      stream.defaultReadObject();
      contents = new Object[MAX_SIZE];
      for (int i = 0; i < size; i++)
        contents[i] = stream.readObject();
      } catch (Exception e) {
        e.printStackTrace();
      }
    }
}
```

Size written

Size read

Object not revisited

```
ObjectHistory objectHistory = new AnObjectHistory();
objectHistory.add(objectHistory);
```

AnObjectHistory (Initialization and Extension)

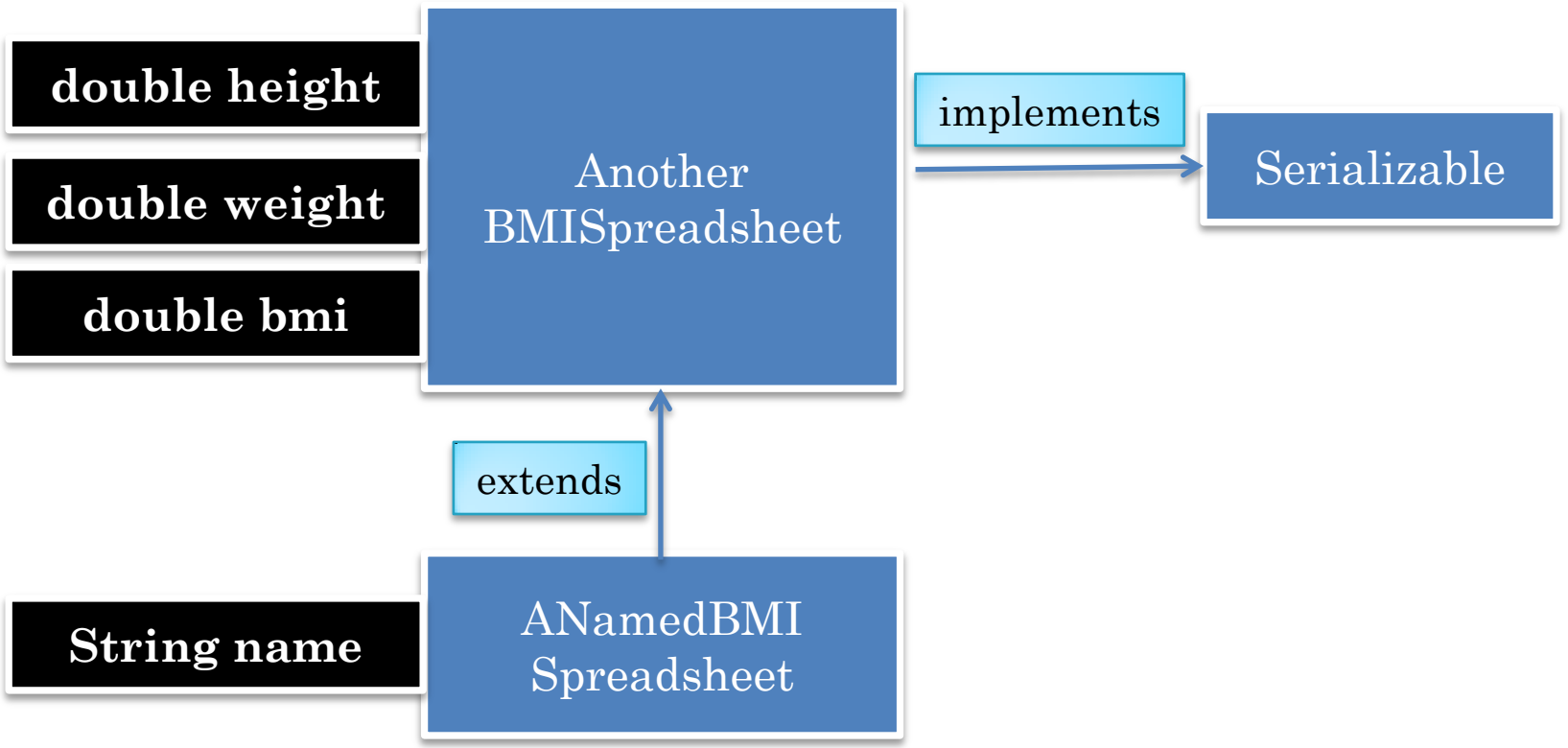
```
public void writeExternal(ObjectOutput out) throws IOException {
  out.writeInt(size);
  for (int i = 0; i < size; i++)
    out.writeObject(contents[i]);

  }
}
public void readExternal(ObjectInput in) throws IOException,
ClassNotFoundException {
  contents = new Object[MAX_SIZE];
  size = in.readInt();
  for (int i = 0; i < size; i++)
     contents[i] = in.readObject();

}
```

Serializable vs. Externalizable

```
private void readObject(ObjectInputStream stream){
  try {
    stream.defaultReadObject();
    initSerializedObject();
  } catch (Exception e) {e.printStackTrace();}
}
```

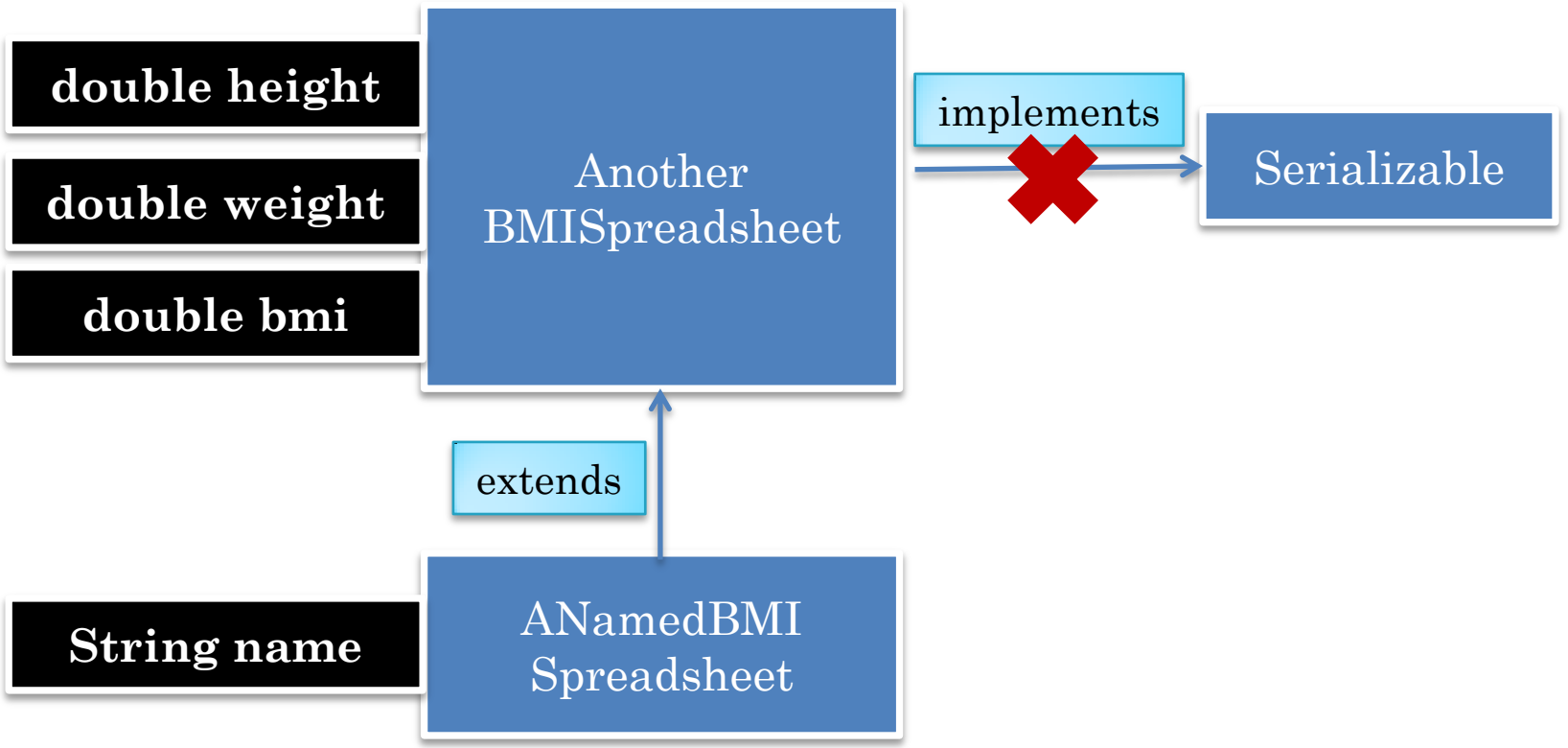
```
public void readExternal(ObjectInput in) {
   try {
     height = in.readDouble();
     weight = in.readDouble();
     initSerializedObject();
   } catch (Exception e) {e.printStackTrace();}
}
public void writeExternal(ObjectOutput out)   {
   try {
     out.writeDouble(height);
     out.writeDouble(weight);
   } catch (Exception e) {e.printStackTrace();}
}
```

Inheritance and Serializable



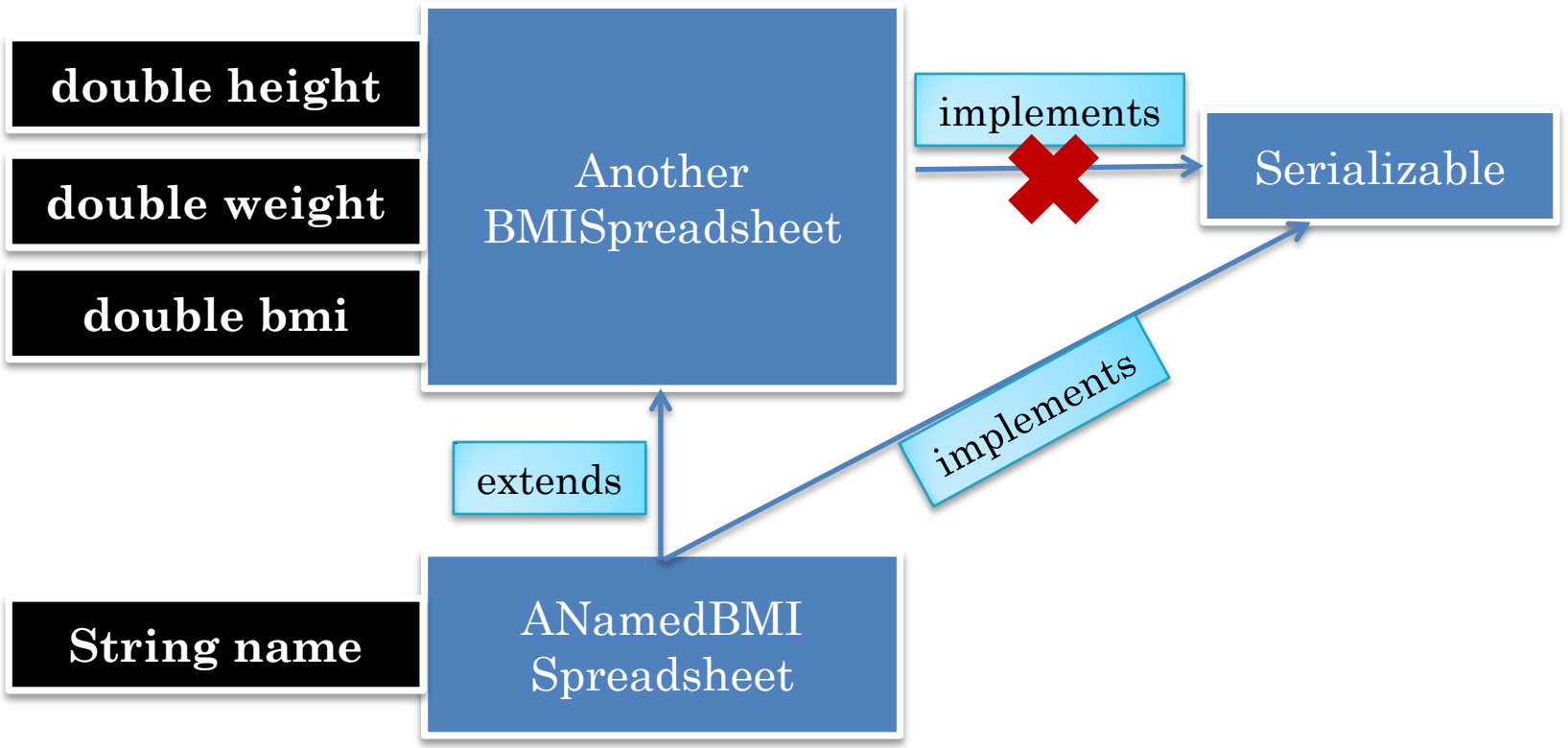
If object is instance of Serializable all of its non transient variables are serialized (by default)

Inheritance and Serializable



Not Serializable exception when serializing instance of either class

Inheritance and Serializable



Variables of super classes that do not implement serializabe not serialized

Serialization looks at implements rather than instance of relation

Rule

If object is instance of Serializable all of its non transient variables are serialized (by default)

If object is instance of Serializable all of its non transient variables declared in classes that implement Serializable are serialized

Serializable vs. Externalizable

Serializable	Externalizable
Private Custom Methods	Public Custom Methods
Can make mistake in method signature	Must implement both methods
Can invoke default methods	No default methods
Can only initialize transients	Serialize both transient and non transients
Sends field names, field types and declaring super classes	Sends only values explicitly written by programmer
Less efficient	Cannot read data without class at receiver's site

Extra Slides

Coupling of Transients Initialization and Extensibility

Extensibility and initialization of transients is coupled

Often they are done together

Only one set of abstraction learnt

Disadvantages?

Coupling of Initialization and Extensibility

Serializable: App must be aware of and call defaultReadObject()

```
priva
  try
    stream.defaultReadObject();
    initSerializedObject();
  } catch (Exception e) {e.printStackTrace();}
}
```

Both: App must know about input stream and deserialization exception handling

```
publ
  try {
    height = in.readDouble();
    weight = in.readDouble();
  }
}
```

Externalizable: App must do complete serialization and deserialization

```
public void writeExternal(ObjectOutput out)  {
  try {
    out.writeDouble(height);
  } c
}
```

Separating interface (method signatures) for extensibility and transient initialization does not have this problem

Coupling of Processing and Extended Serialization

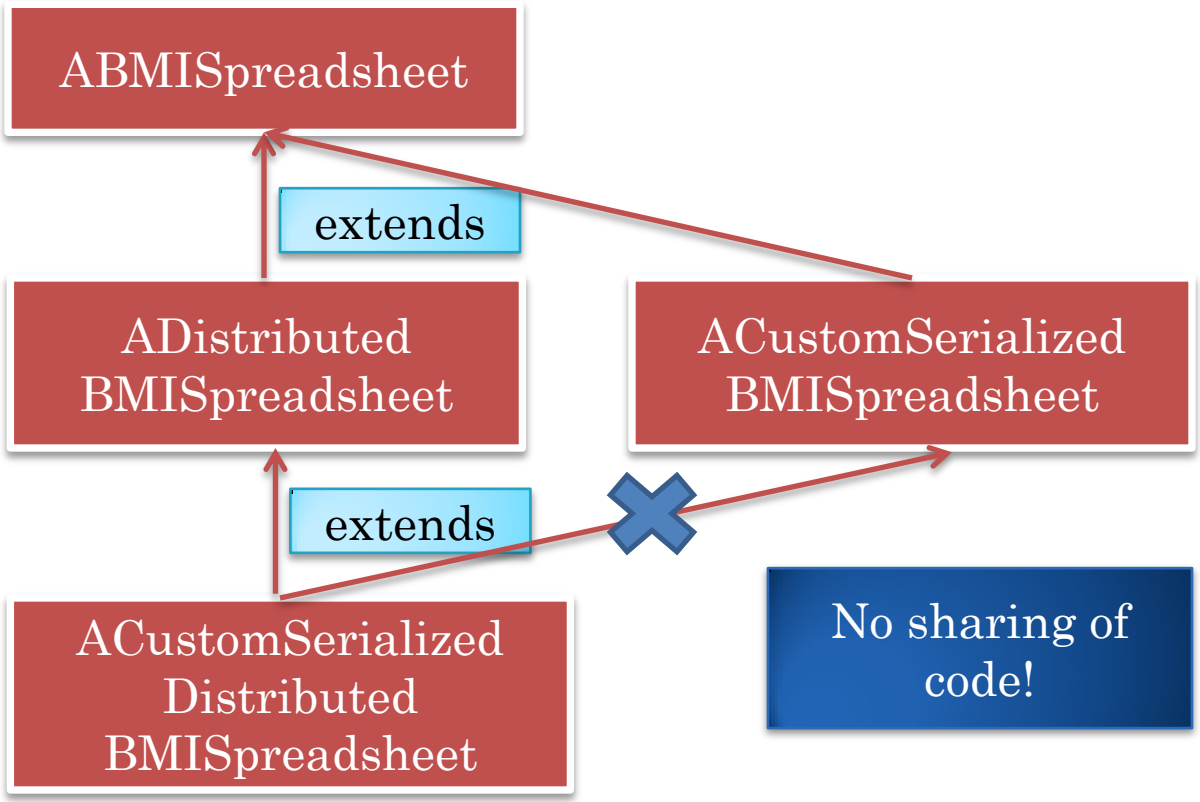
Serialization extension must be done by the class being serialized

Allows extension to easily and efficiently access physical structure (without reflection)

Inheriting Internal Serializer

Cannot add serializer for class without source code

Must inherit and change references and not reuse code



Coupling of Processing and Extended Serialization

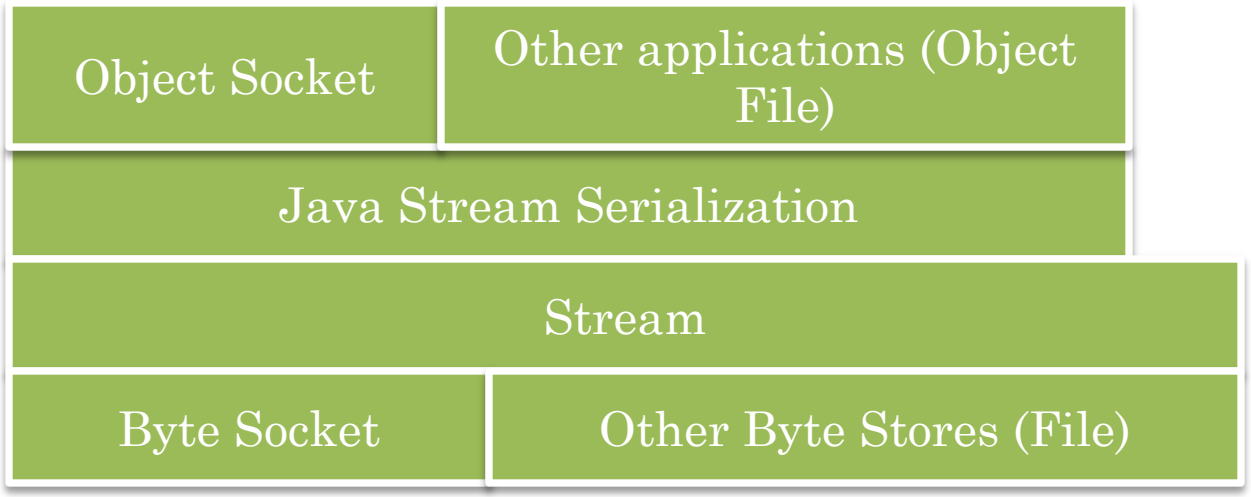
Cannot have multiple serializers for different contexts

Text vs Binary, Logical vs. Physical, Mobile vs. Desktop, Single Language vs. Multiple Language

Binary serializer of physical structures that can be extended by internal routines

Text serializer of logical structures that is driven by serialization directives but cannot be extended

Java Serialization and Object Communication



No (direct) serialization support for NIO