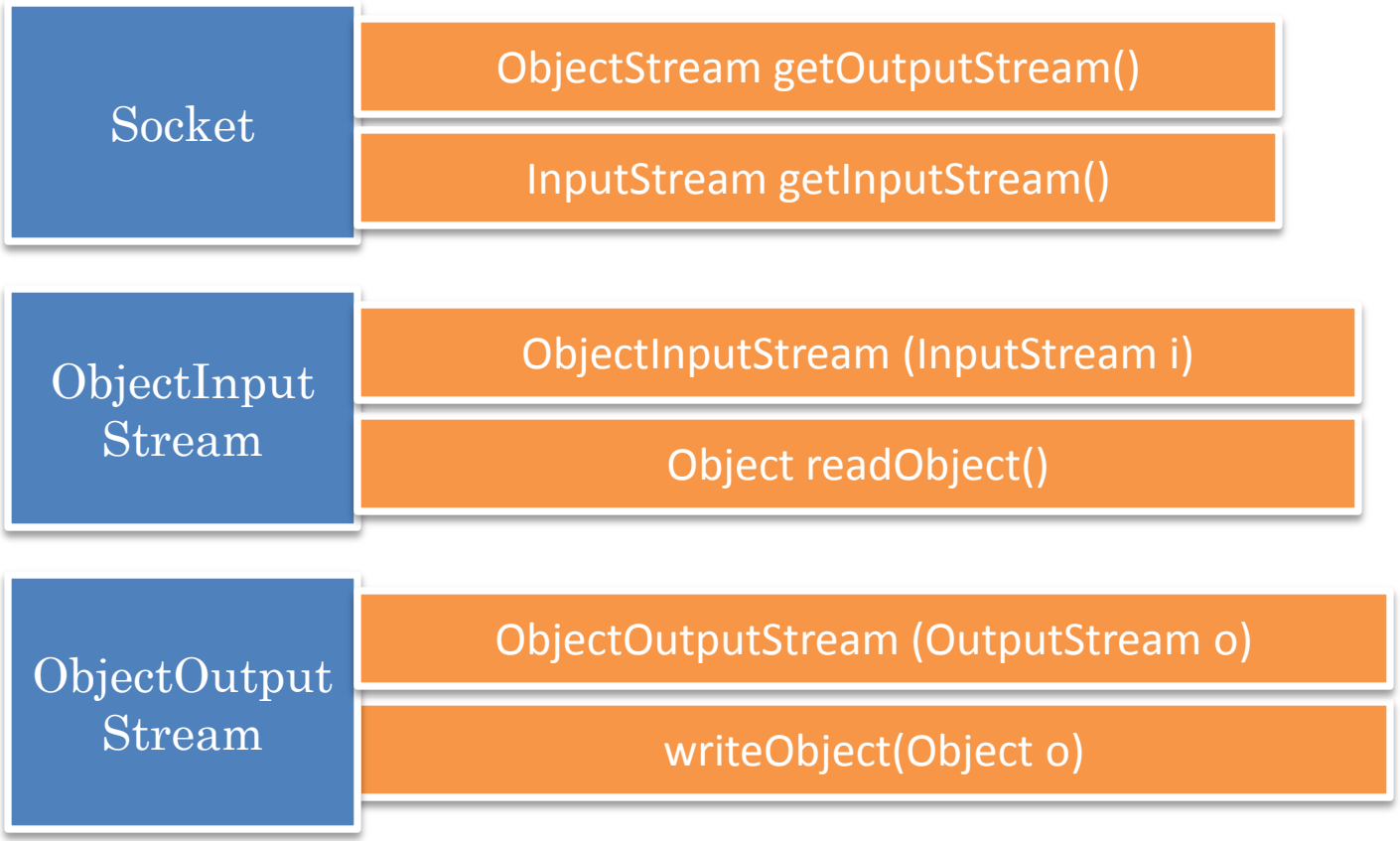
Issues in Design and Implementation of Serialization

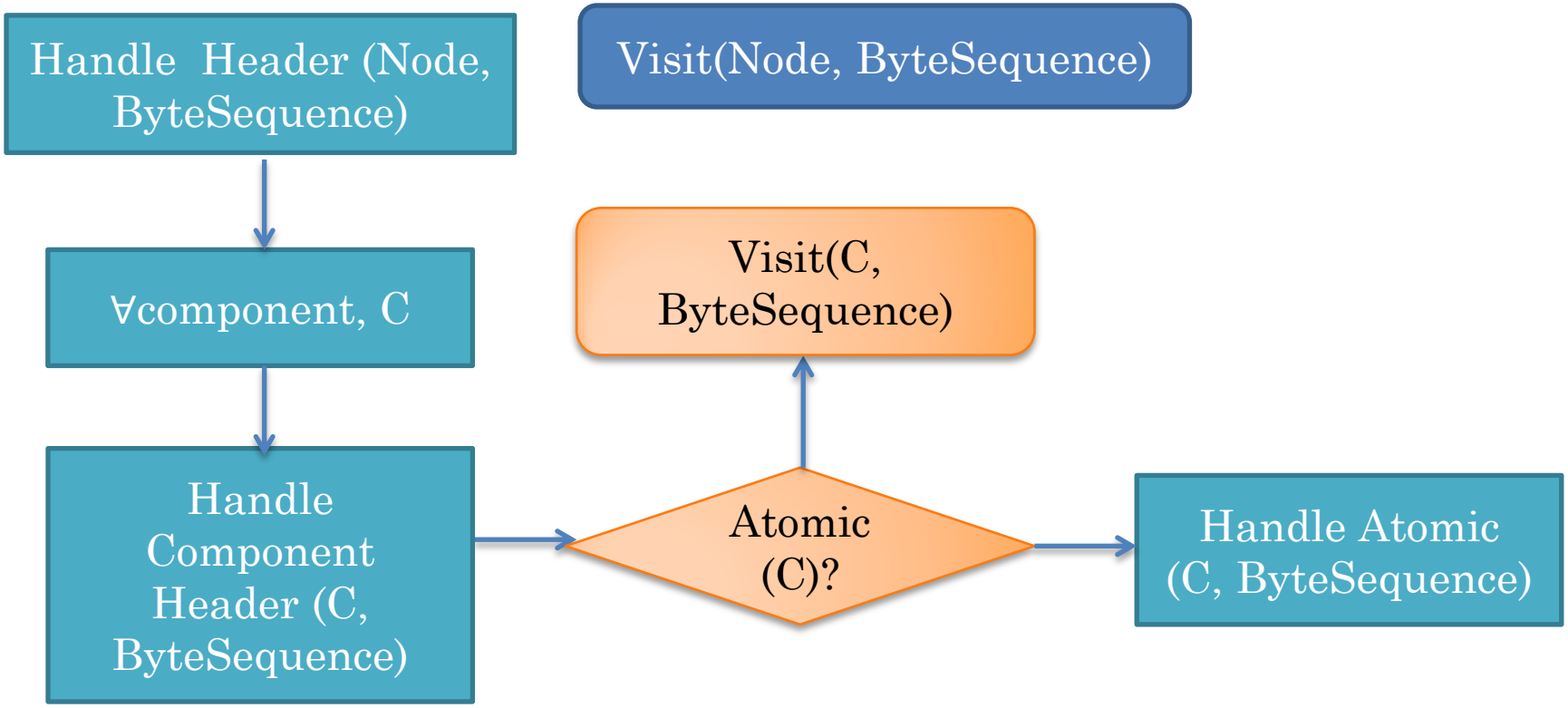
Instructor: Prasun Dewan (FB 150, dewan@unc.edu)

Java Object Output and Input Streams

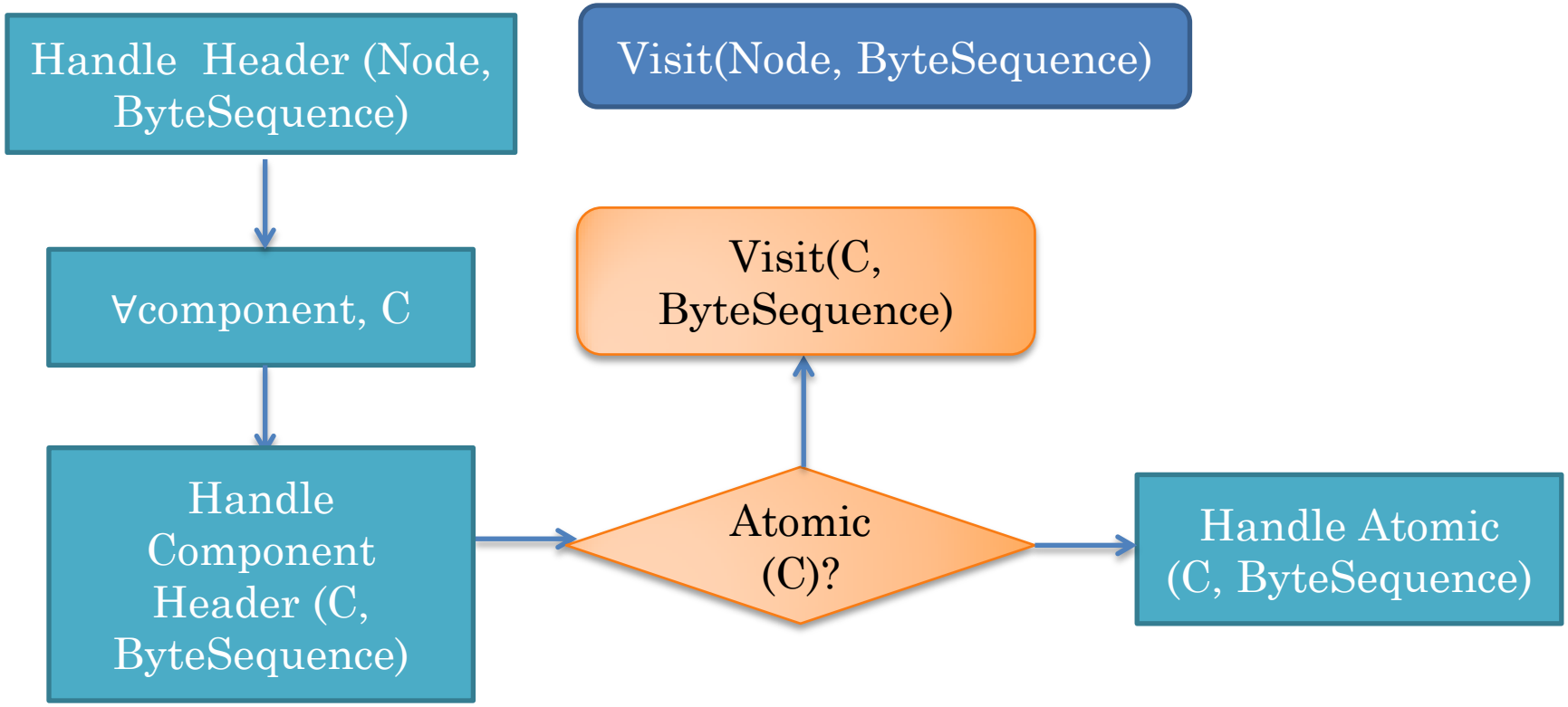



```
ObjectOutputStream socketOut = new ObjectOutputStream(socket.getOutputStream());
socketOut.writeObject(list);
ObjectInputStream socketIn  = ObjectOutputStream(socket.getInputStream());
list readVal = (List) socketIn.readObject();
```

Basic Serialization



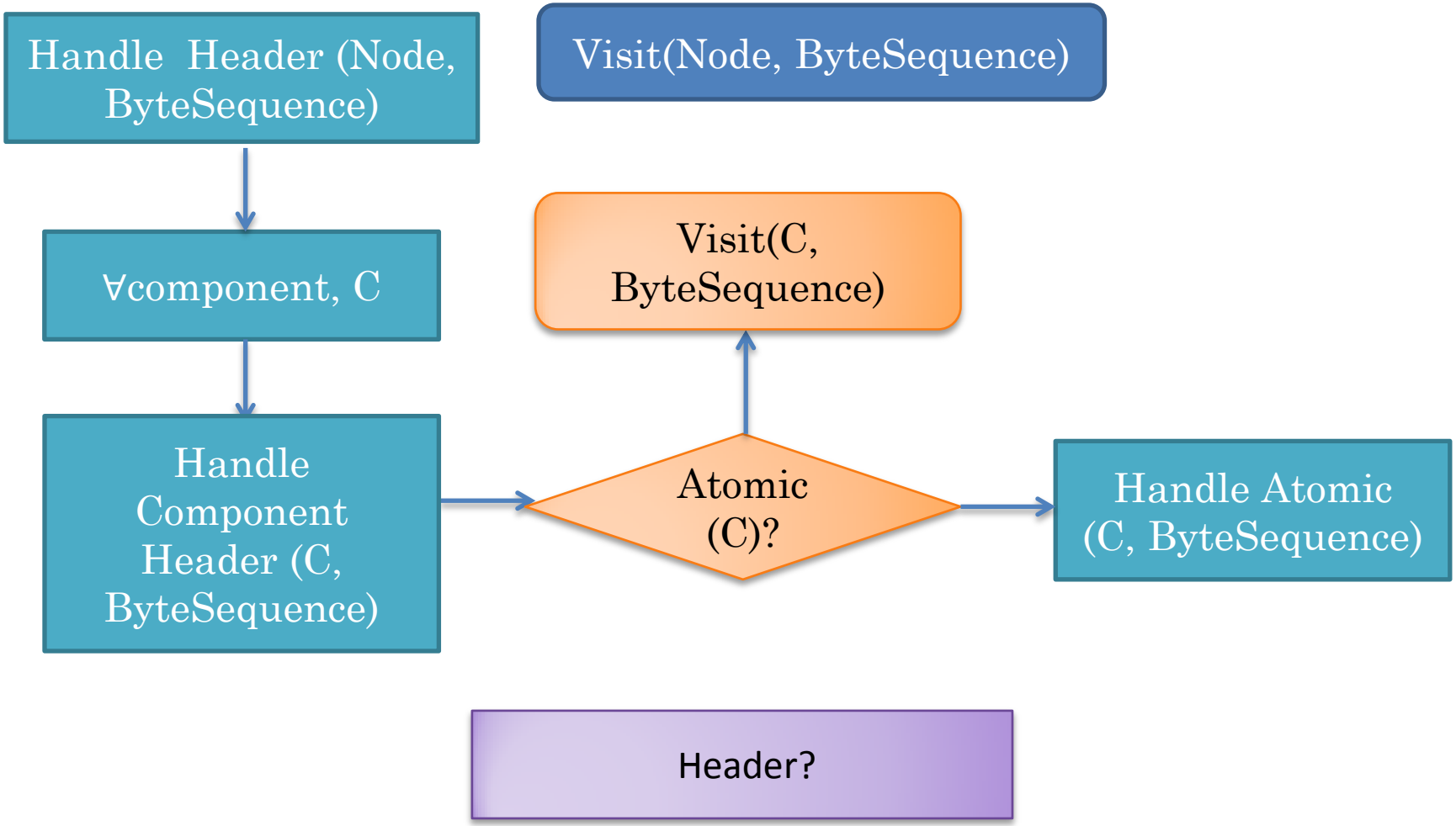
Serialization vs. Deserialization



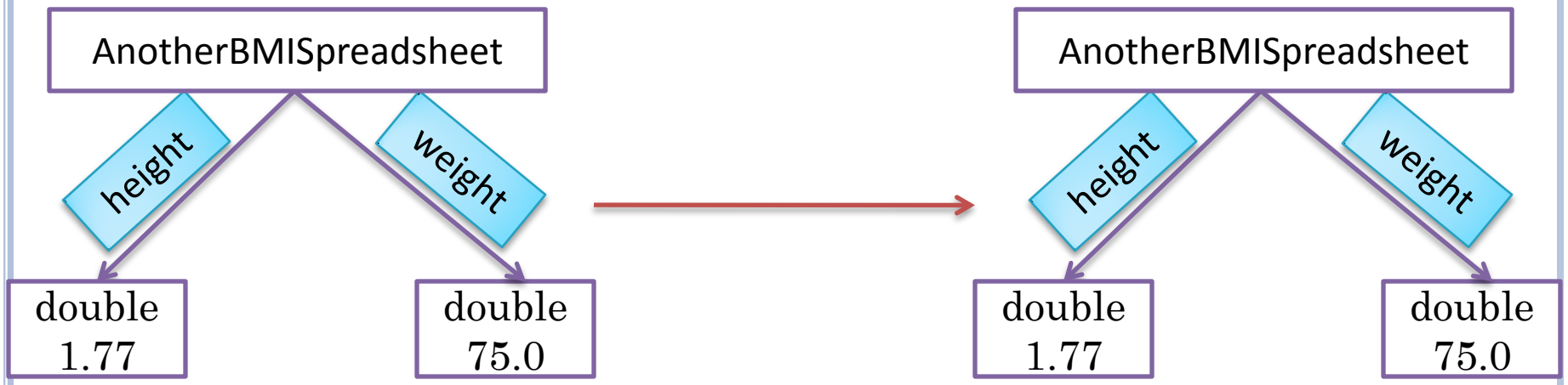
Assuming that ByteSequence is a Reference Parameter

Visit would write or read the value depending on the mode

Basic Serialization



Implementing Serialization



The other party must be able to create an instance of the appropriate type based on the originator's type

Nature of Serialized Message

Type ID can be string or some other well-defined identifier

Type is optional

When is Type ID Needed

Type needed when other party does not know Atomic type or class of received value

Port or application may constrain values to expected types

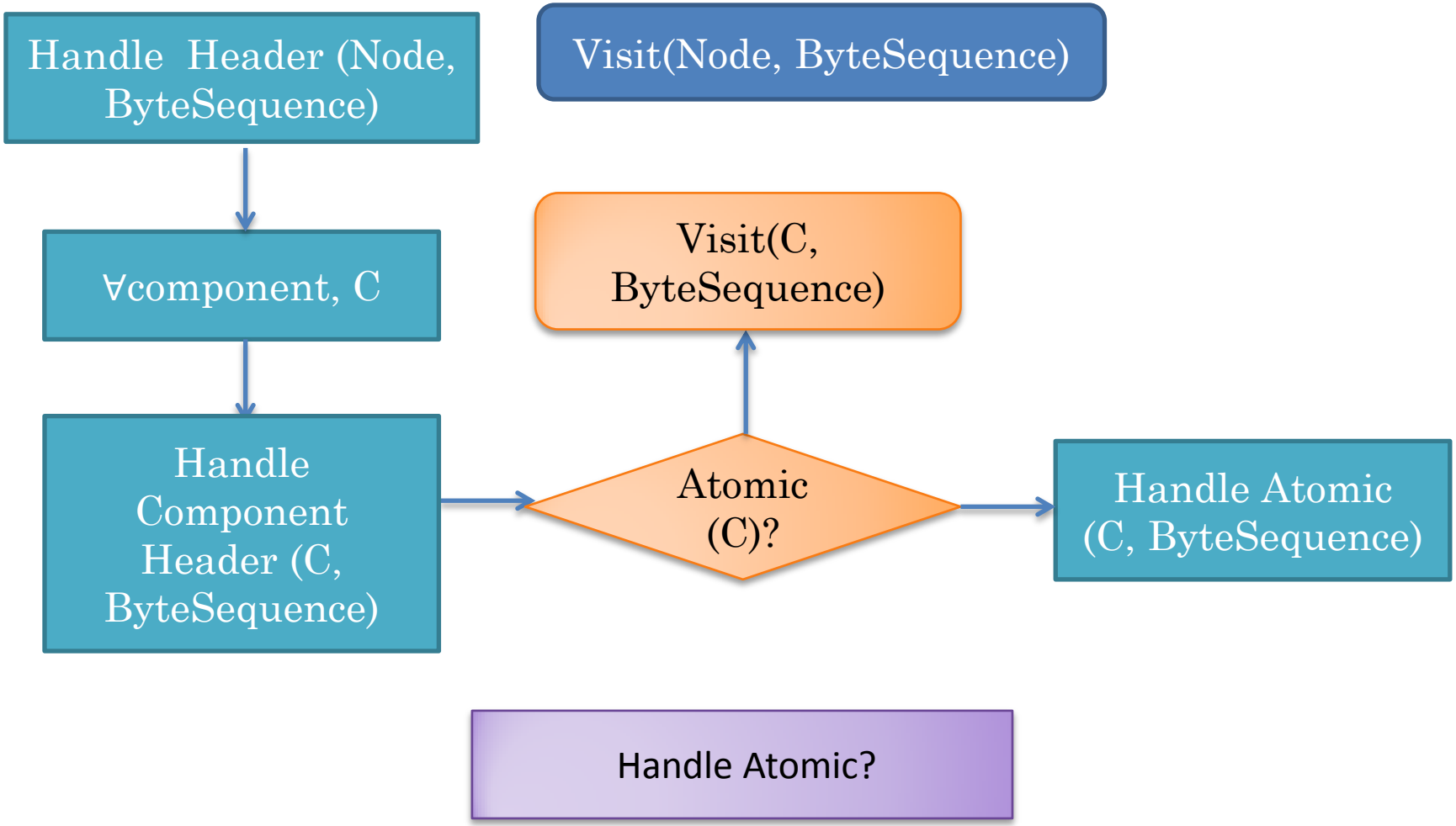
In remote assignment or procedure call, types well defined

```
send load(2.2)
```

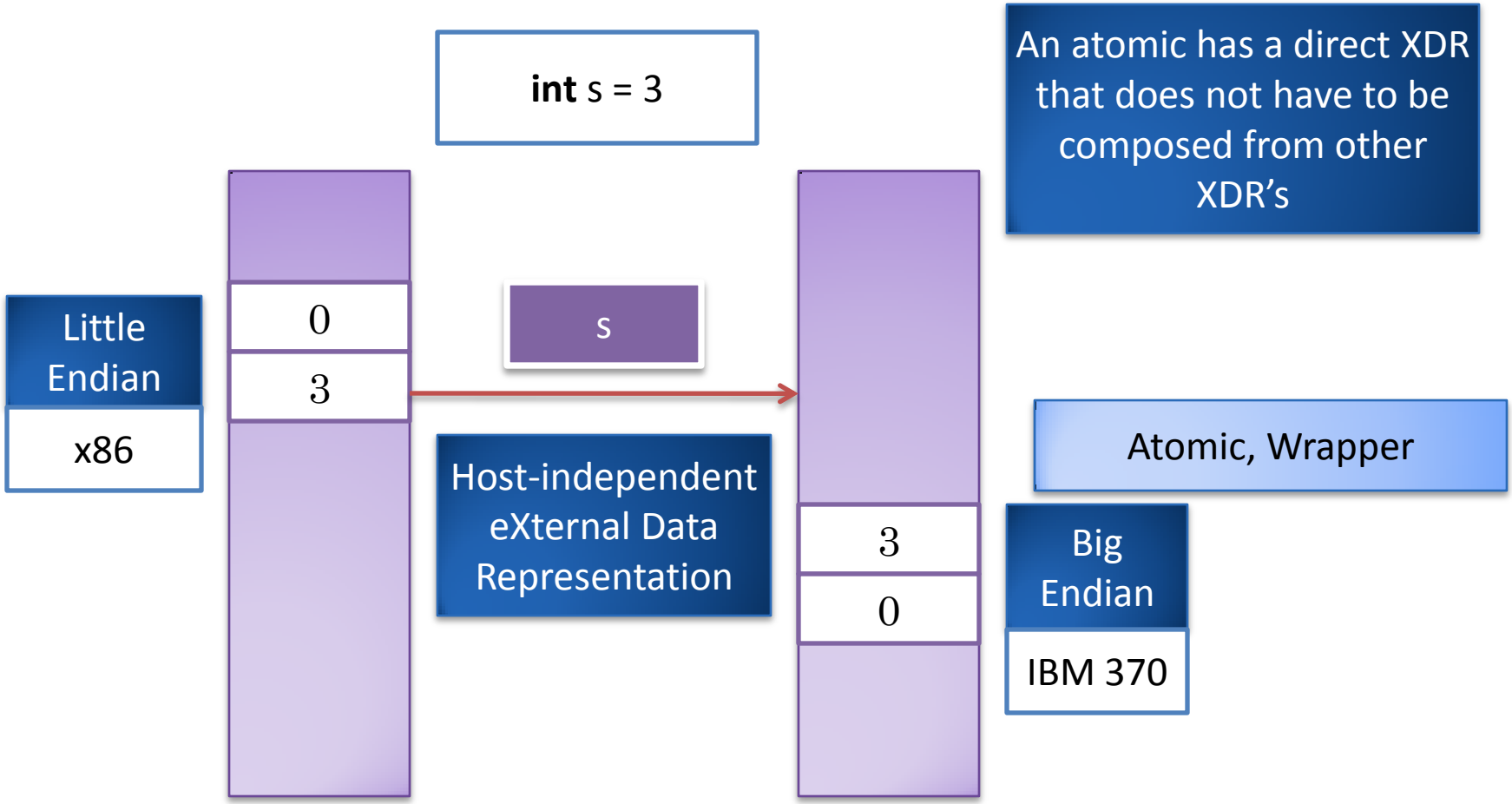
```
send add (5, 3)
```

Nature of value representation?

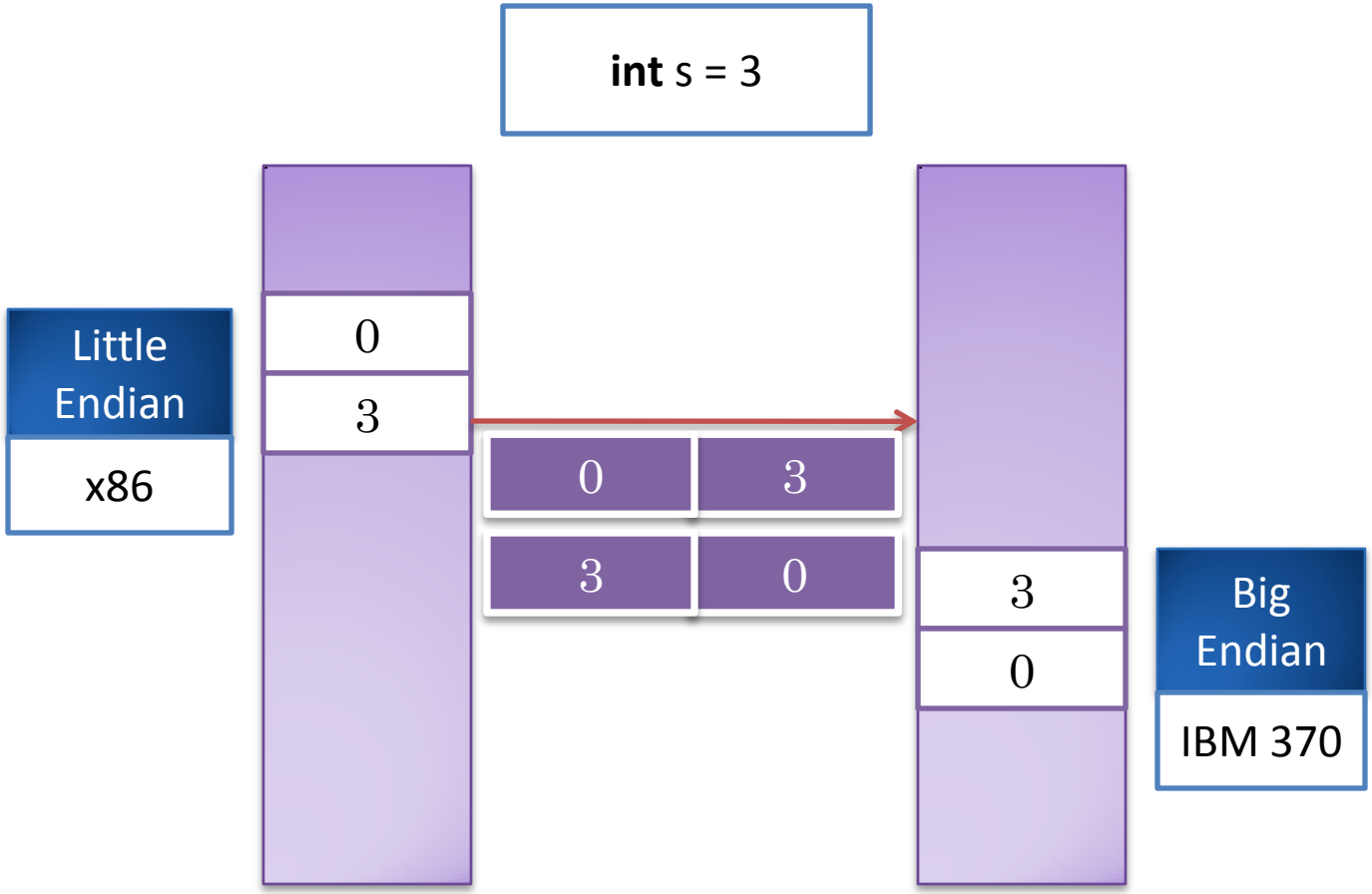
Atomic?


Handle Header (Node, ByteSequence)
Visit(Node, ByteSequence)
∀component, C
Visit(C, ByteSequence)
Handle Component Header (C, ByteSequence)
Atomic (C)?
Handle Atomic (C, ByteSequence)
Handle Atomic?

Atomic Values

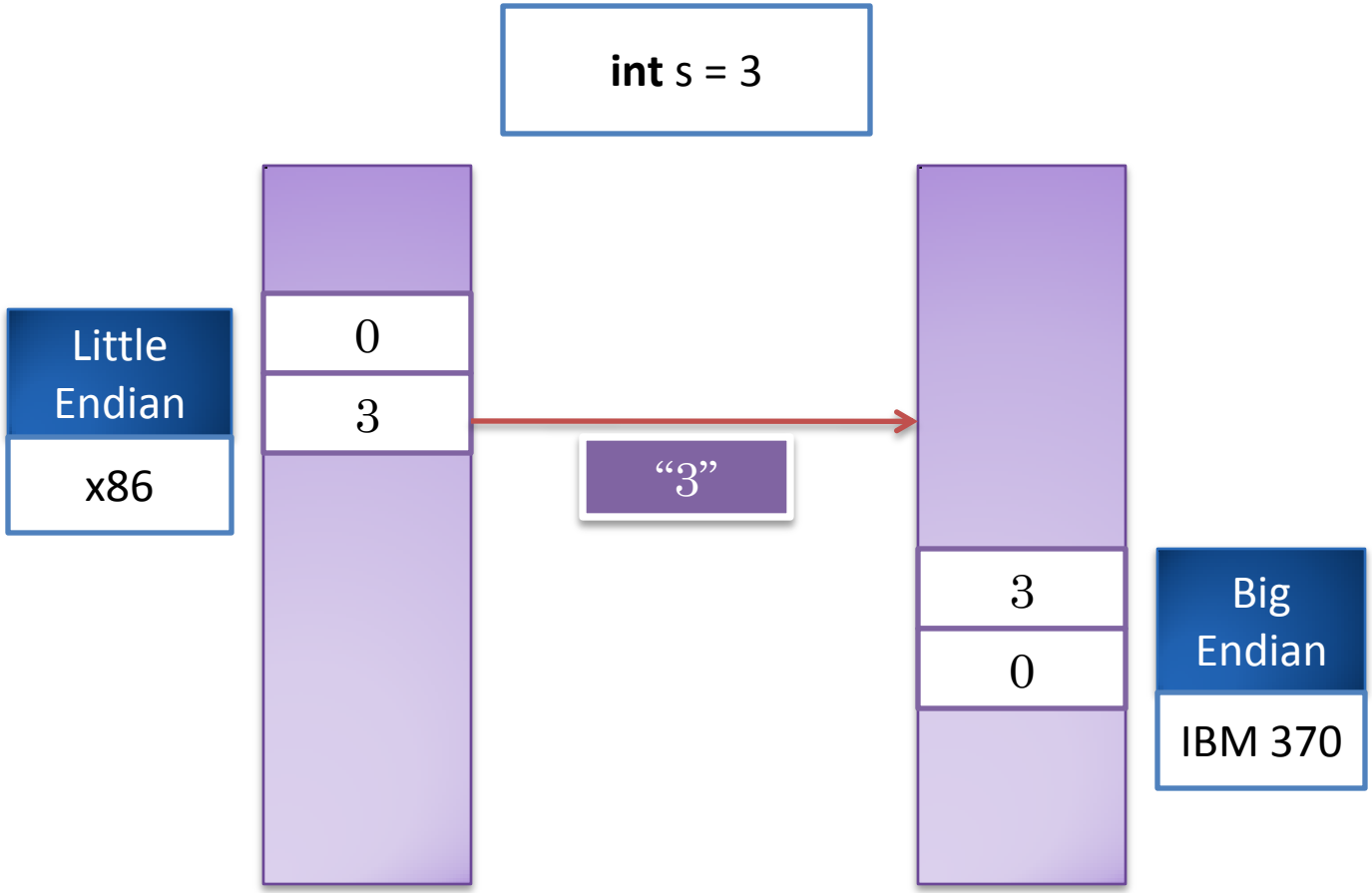


Atomic Binary



Network/External value representation: Some agreed upon binary representation of value in a sequence of bytes

Textual Primary



Textual: A standard string representation of value

Textual vs. Binary Atomic Values

Inefficient
Human readable: debugging and logging
Consistent with Composite Object Serialization for Heterogeneous Platforms
Can be sent over SMTP and HTTP (XML Serialization, SOAP, Web Services)

SMTP for IPC?

Slow – many Handlees involved on path

No connection set up

Sometimes speed not necessary

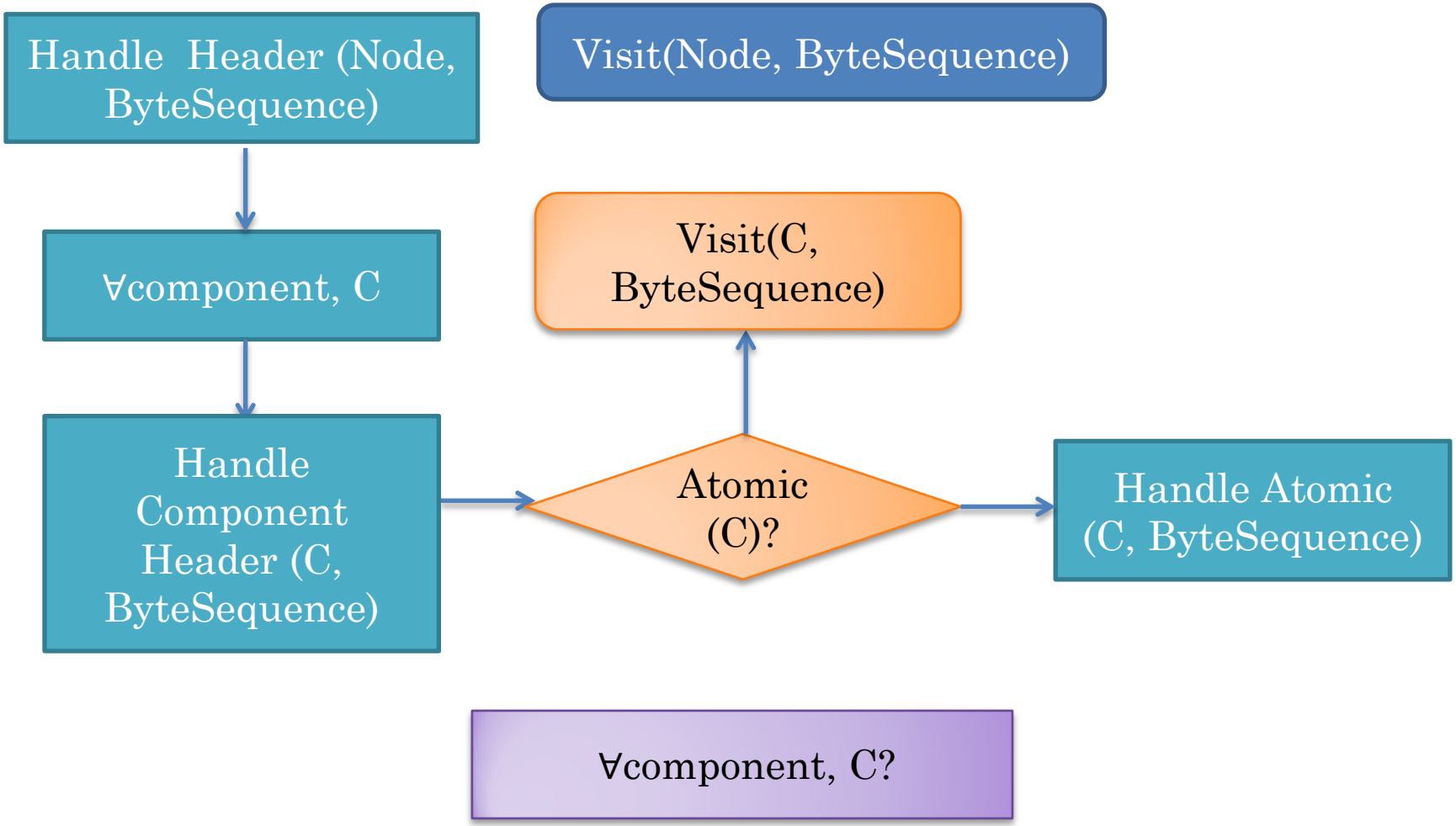
Chess program

Human to Handle communication

Invoke remote service with appropriately formatted email

Atomic Serialization

Components?


Handle Header (Node, ByteSequence)
Visit(Node, ByteSequence)
∀component, C
Visit(C, ByteSequence)
Handle Component Header (C, ByteSequence)
Atomic (C)?
Handle Atomic (C, ByteSequence)
∀component, C?

What are Components

∀component, C

Physical Structure : Memory representation

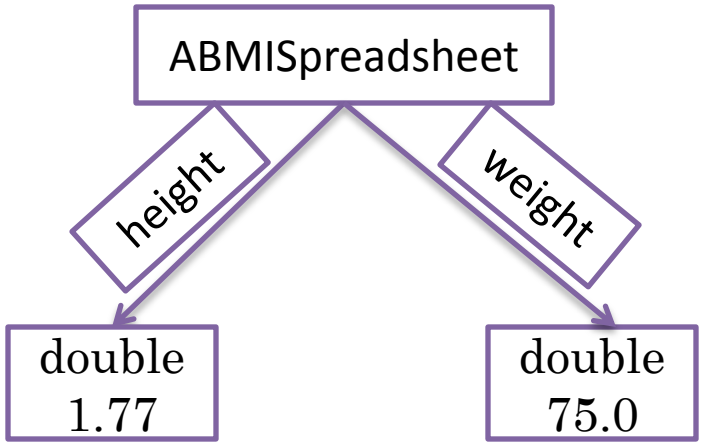
Logical Structure: Interface + Pattern

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);}
}
```

Physical Structure

BMISpreadsheet b = **new** ABMISpreadsheet()



Physical structure defined by instance variables stored in memory

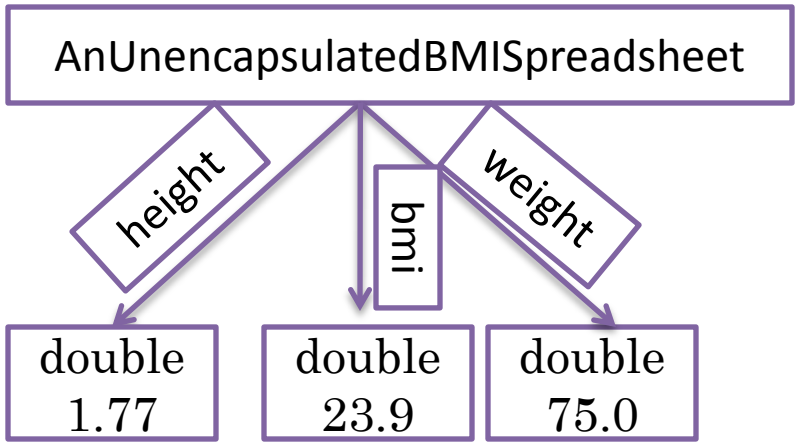
Internal nodes labeled by class name

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

Leaf nodes labeled type and value

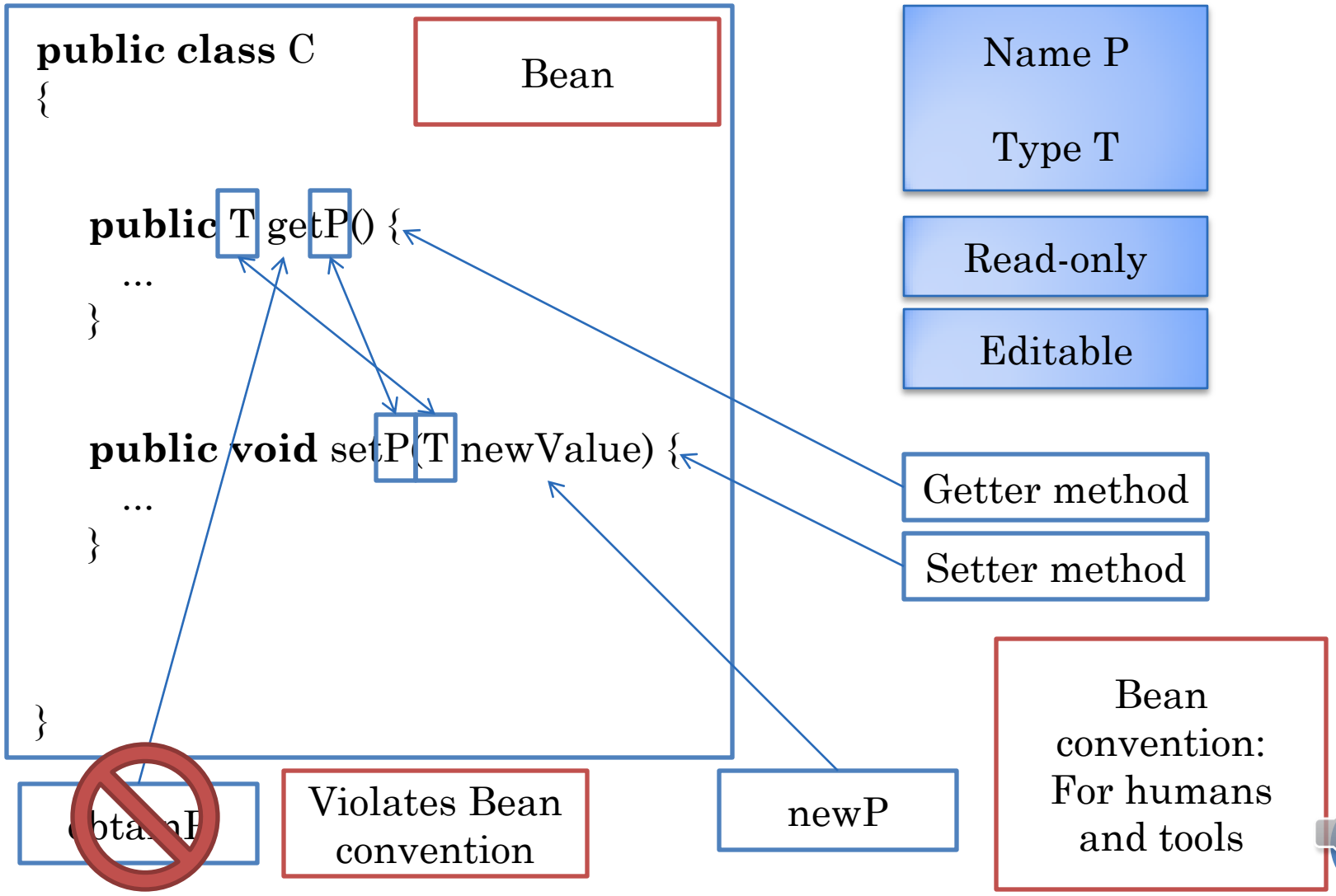
Logical Structure

BMISpreadsheet b = **new** AnUnencapsulatedBMISpreadsheet()



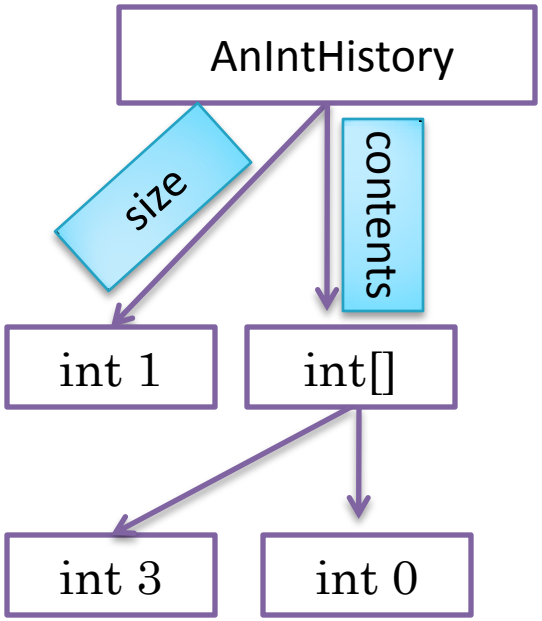
Bean Pattern

Typed, Named Unit of Exported Object State



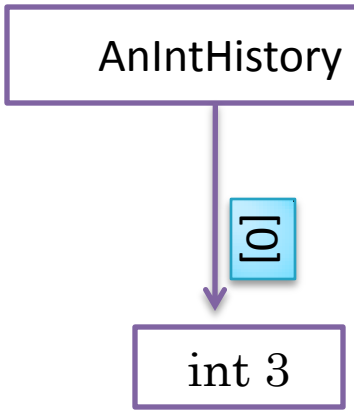
Physical Structure

```
IntHistory ints = new AnIntHistory();
ints.add(3)
```



Logical Structure

```
IntHistory ints = new AnIntHistory();
ints.add(3)
```



Can define Bean-like patterns for Lists (and Tables)

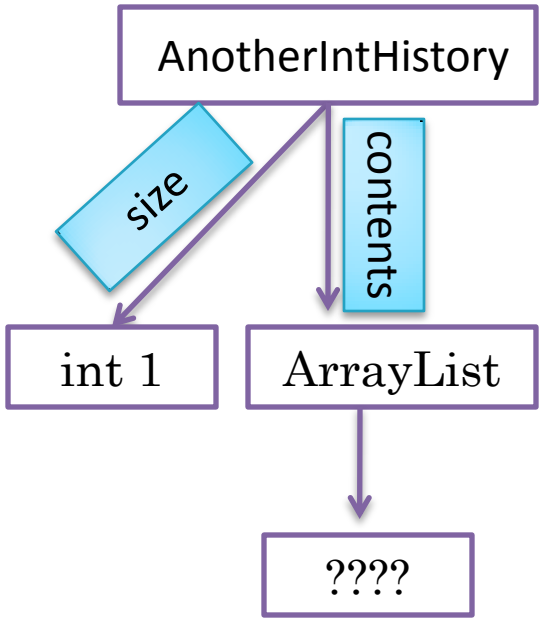
Vector and ArrayList follow these patterns but do not notify

AnotherIntHistory

```
public class AnotherIntHistory implements IntHistory {
  List<Integer> contents = new ArrayList();
  public int size() {
    return contents.size();
  }
  public int get (int index) {
    return contents.get(index);
  }
  public void add(int element) {
    contents.add(element);
  }
}
```

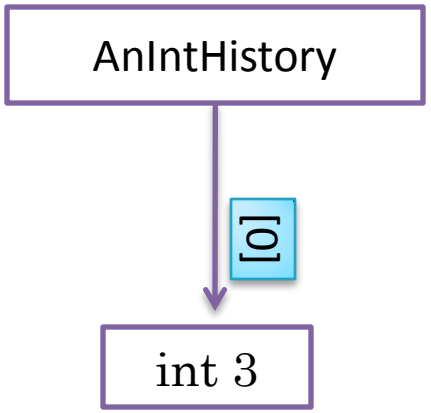
Physical Structure

```
IntHistory ints = new AnotherIntHistory();
ints.add(3)
```

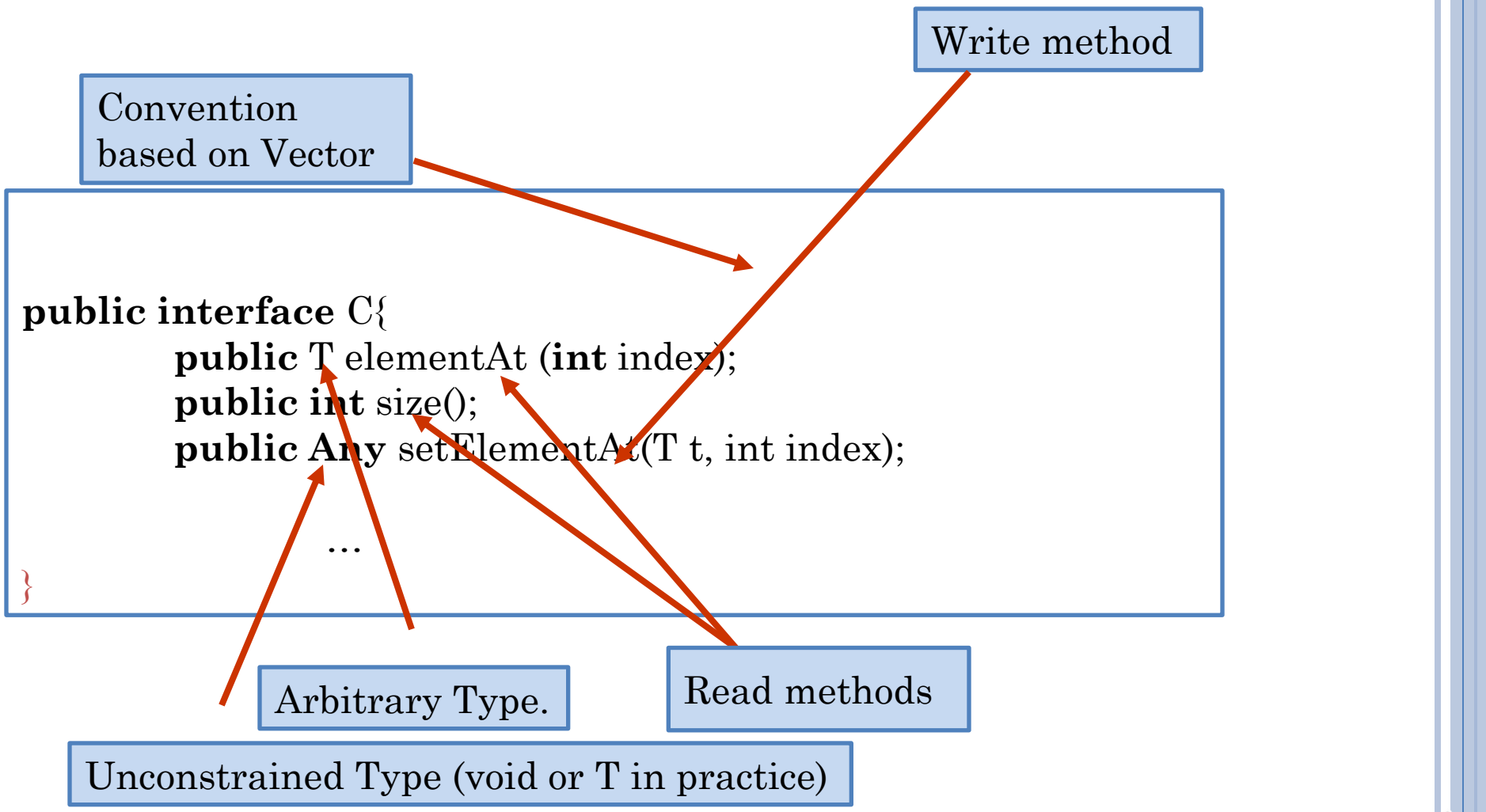


Logical Structure

```
IntHistory ints = new AnotherIntHistory();
ints.add(3)
```



Conventions for Variable-Sized Collection



How to Derive Components?

∀component, C

Physical Structure : Memory representation

Logical Structure: Interface + Pattern

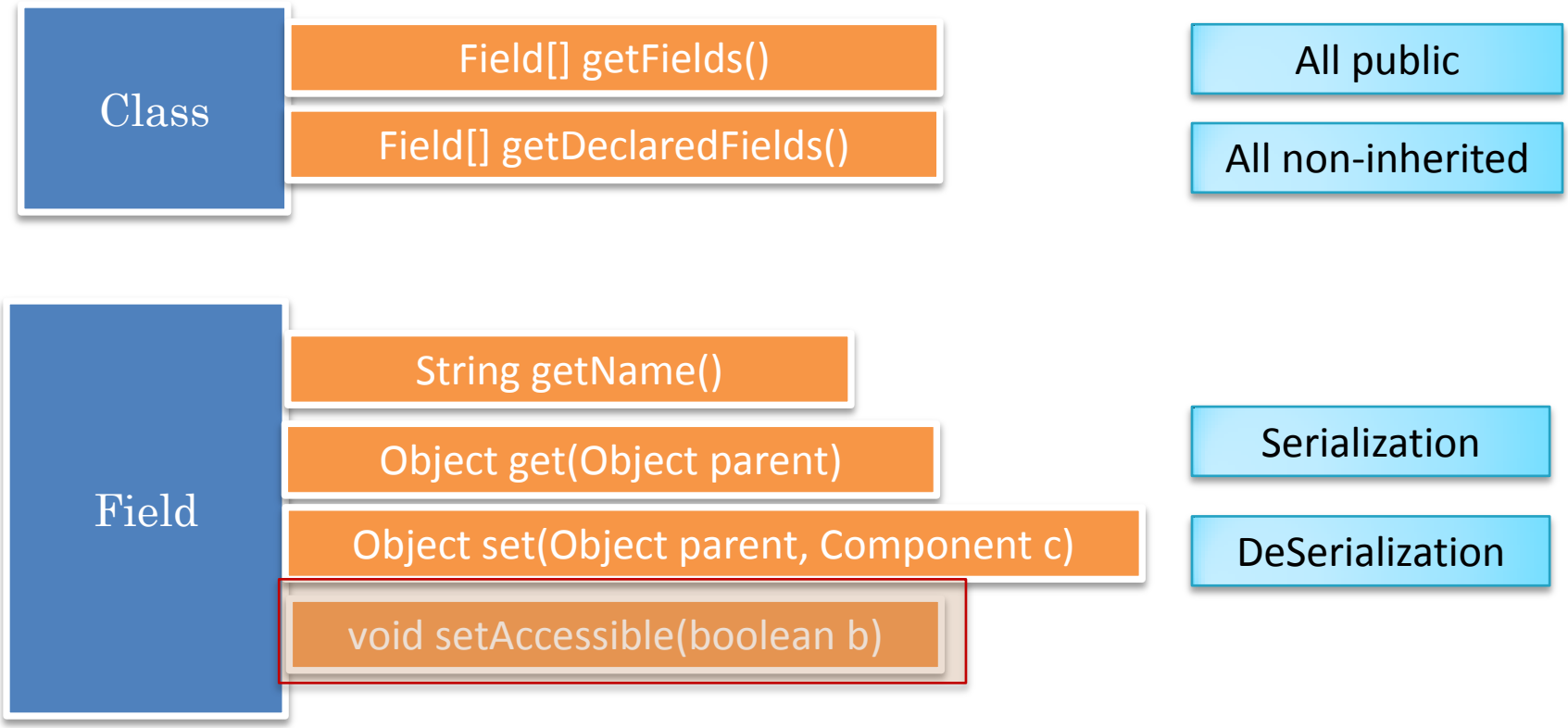
How to Determine Physical Components

Array elements can be directly accessed

Need to determine instance variables of programmer defined objects

Can we do this in a replaceable library?

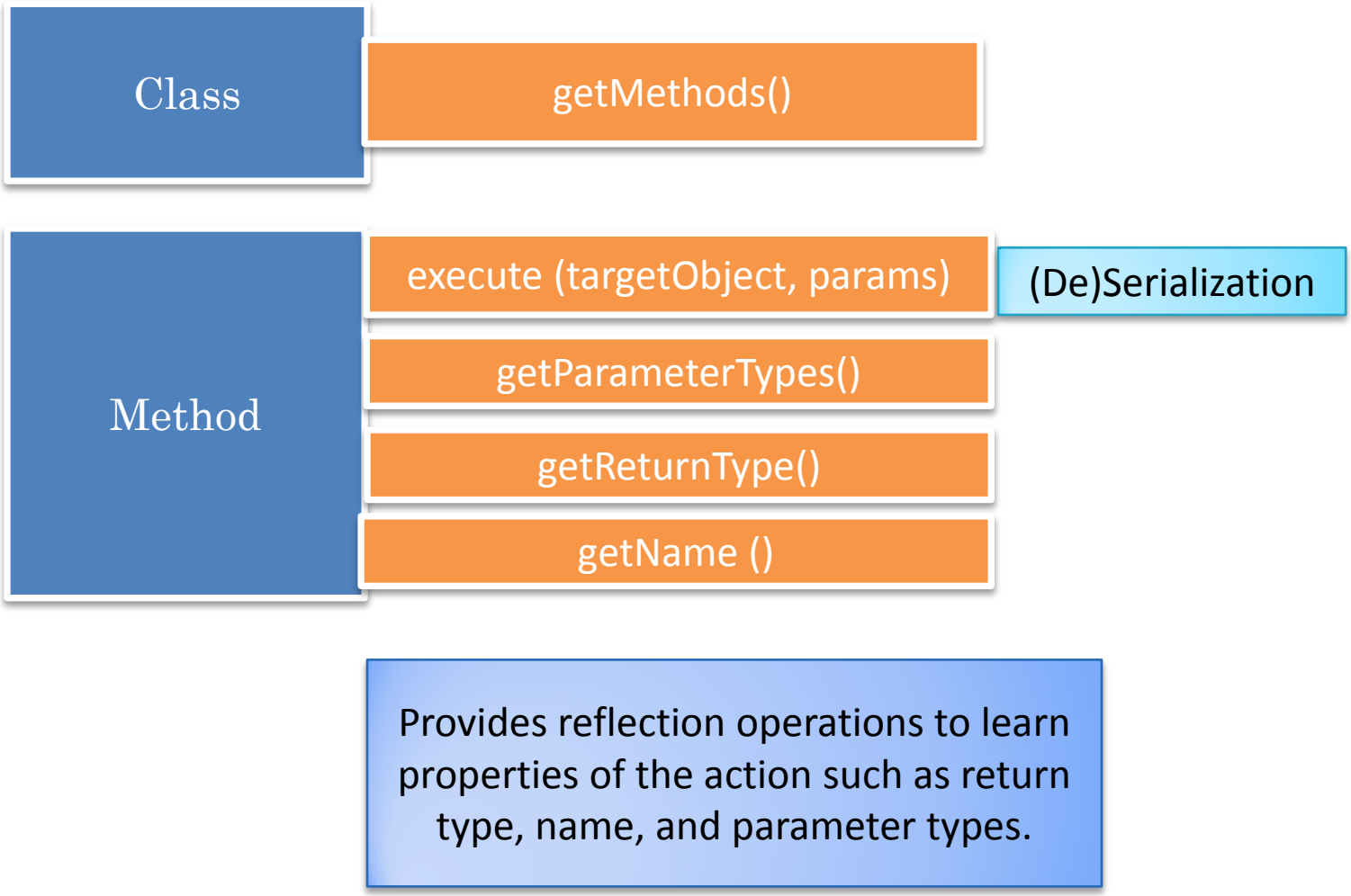
Field Reflection



Can break encapsulation!

Logical components?

Method Reflection



Provides reflection operations to learn properties of the action such as return type, name, and parameter types.

Reflection Example

```
public interface BMISpreadsheet {
  double getWeight();
  void setWeight(double newWeight);
  double getHeight();
  void setHeight(double newHeight);
  double getBMI();
}
```

Calling printProperties

```
public static void main(String[] args) {
        BMISpreadsheet bmi = new ABMISpreadsheet();
        printProperties(bmi);
}
```

printProperties() accepts an argument of arbitrary type

Class Reflection

```
public static void printProperties(Object object) {
    System.out.println("Properties of:" + object);
    Class objectClass = object.getClass();
    Method[] methods = objectClass.getMethods();
    Object[] nullArgs = {};
    for (int index = 0; index < methods.length; index++) {
        Method method = methods[index];
        if (isGetter(method)) {
            Object retVal = methodInvoke(object, method,
nullArgs);
            System.out.println(propertyName(method) + ":" +
retVal);
        }
    }
        System.out.println();
}
```

Class is a runtime variable vs. compile time as in generics.

Invoking methods on a class to create learn its properties.

Method Reflection

```
public static String GETTER_PREFIX = "get";
public static boolean isGetter (Method method) {
       return method.getParameterTypes().length == 0 &&
              method.getReturnType() != Void.TYPE &&
              method.getName().startsWith(GETTER_PREFIX);
}
```

```
public static String propertyName(Method getter) {
  return getter.getName().substring(GETTER_PREFIX.length());
}
```

Method Reflection: Invoking methods on a method.

Invoking Target Method

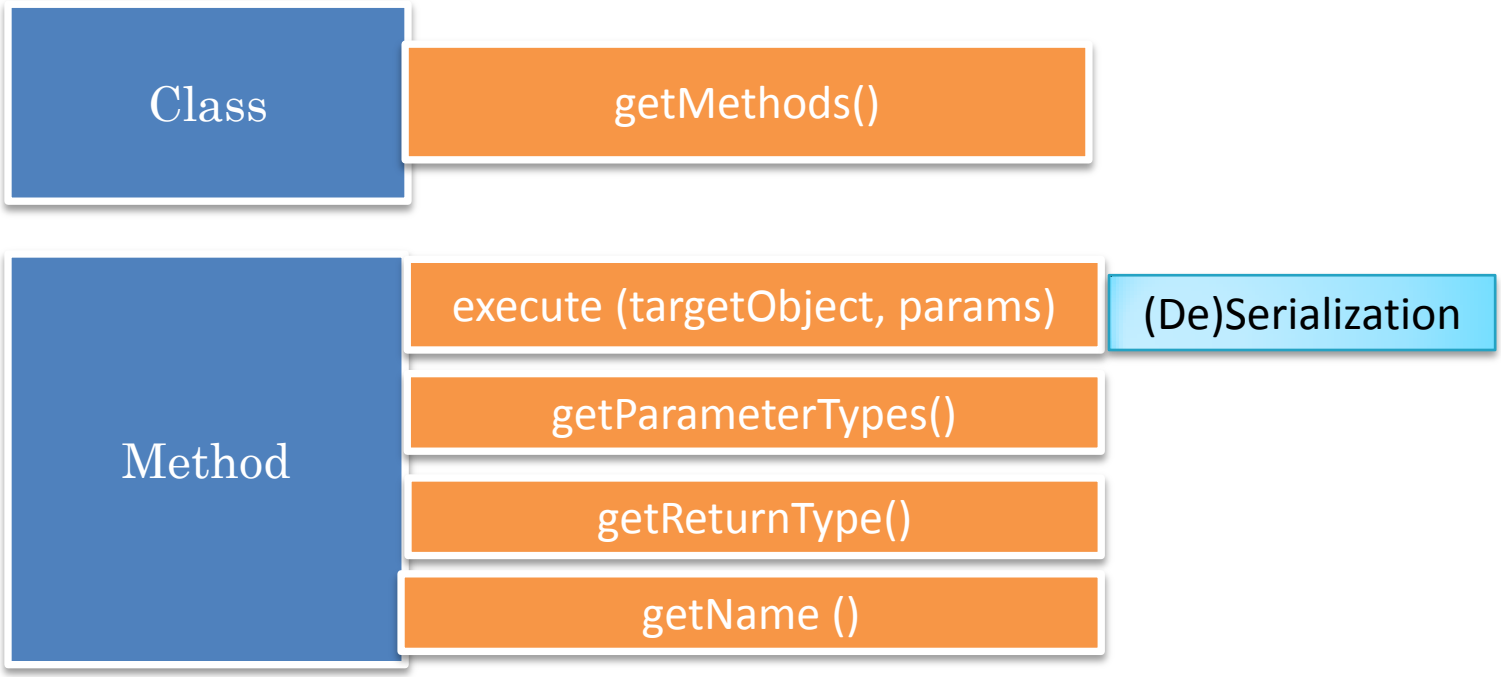
```
public static Object  methodInvoke(Object object,
                    Method method, Object[] args ) {
        try {
                return  method.invoke(object, args);
        } catch (IllegalAccessException e) {
                e.printStackTrace();
                return null;
        } catch (InvocationTargetException e ) {
                e.printStackTrace();
                return null;
        }
}
```

IllegalAccessException: Method not visible to caller.

InvocationTargetException: Exception thrown when parameters/target object do not match method or method throws exception such as ClassCastException.

Method that takes method as a parameter is 2nd-order method.

Method Reflection



Provides reflection operations to learn properties of the action such as return type, name, and parameter types.

Higher level abstraction?

Introspection

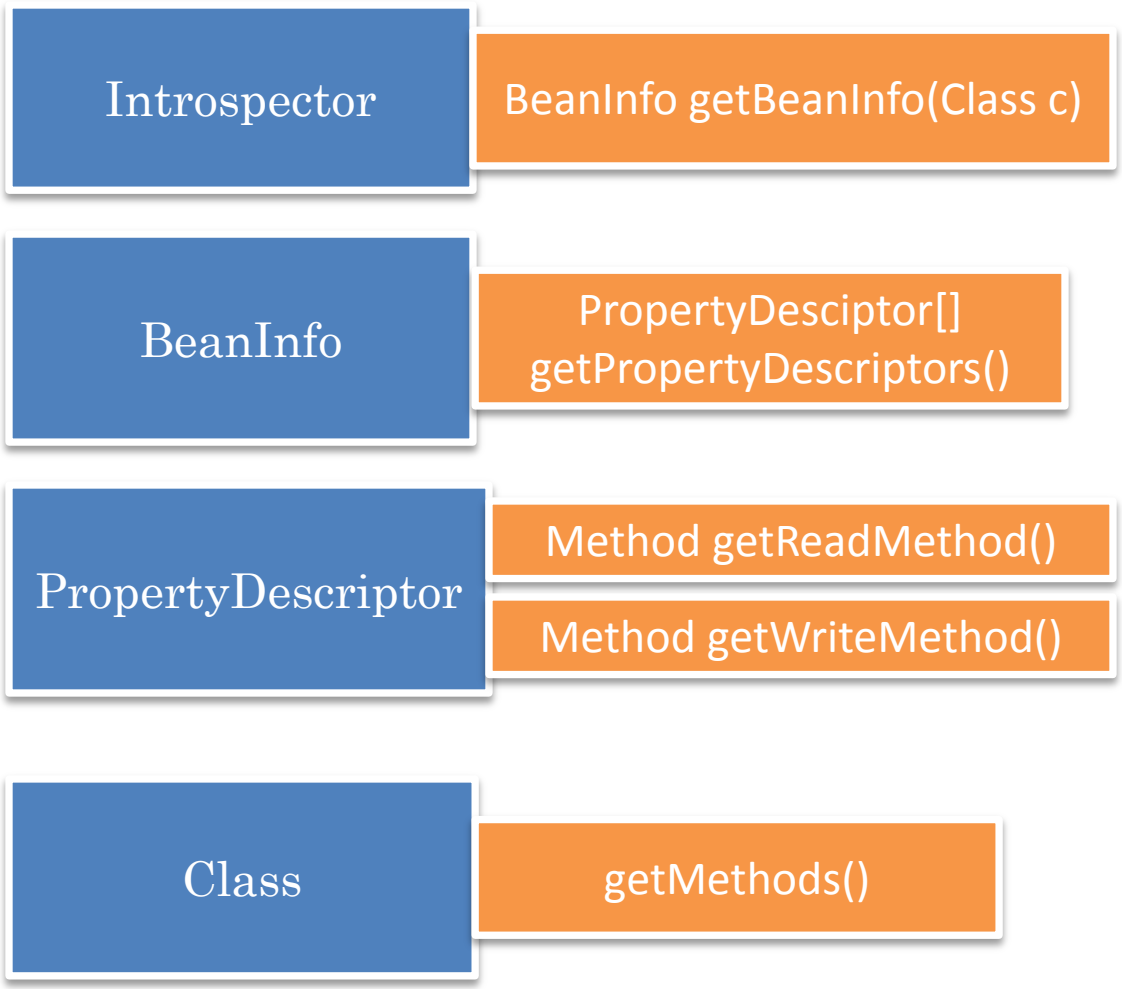
Identifies directly the components defined by some pattern

Provides a way to invoke read and write operations on the components

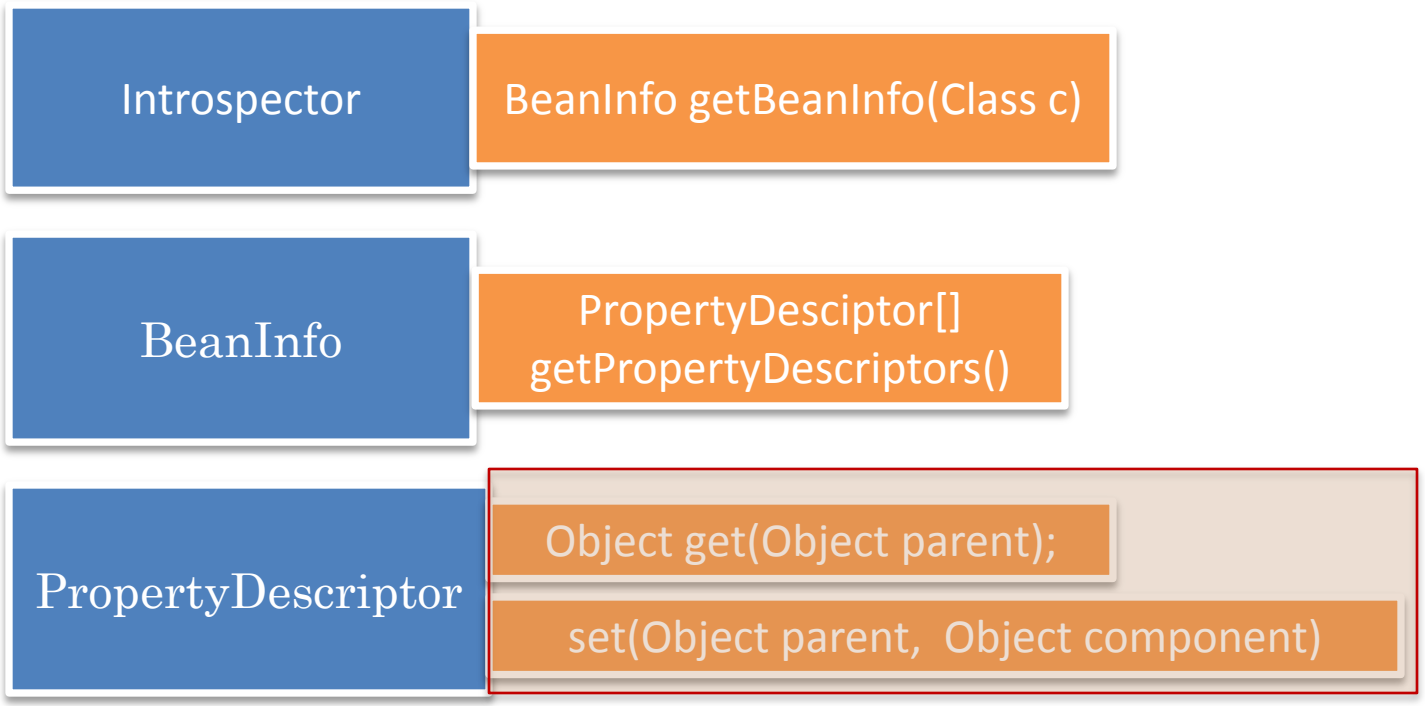
Pattern-Dependent

Relatively High-Level

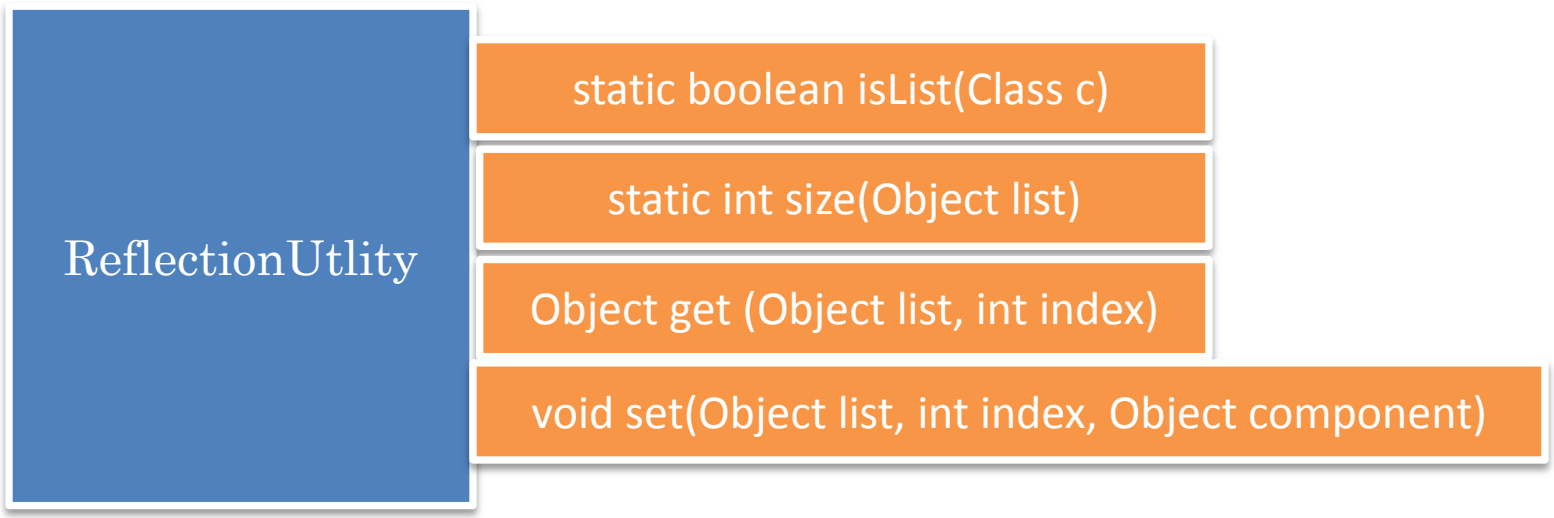
Bean Introspection



Better Bean Introspection Interface



List Introspection Provided by OEAll



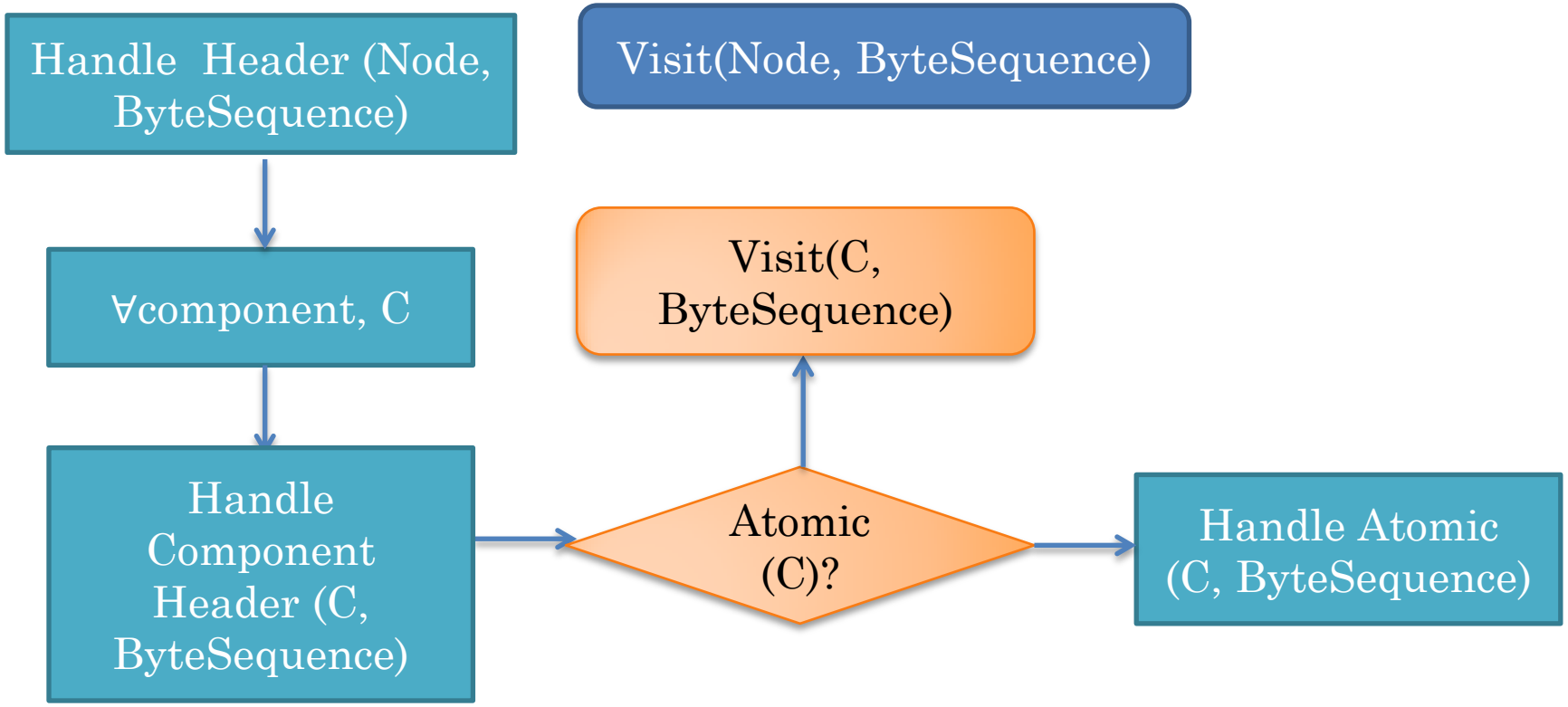
Why no isBean()?

Every object is a bean with 0 or more properties

Language Support for Library Support

Language support for libraries can be used to manipulate logical/physical structure
Libraries needed as new patterns emerge
Libraries can change language support for serialization of physical/logical structures

Physical vs. Logical Components (review)



Comparison?

Physical Structure : Memory representation

Logical Structure: Interface + Pattern

Physical vs. Logical (Automation)

Any object can be physically serialized automatically

Not all objects follow Bean Pattern

List pattern non standard

Not all communicated objects follow List and Bean pattern

In practice, Bean, List, and Hashmap patterns should suffice

Physical vs. Logical (Efficiency)

In general physical state larger than exported state

Accommodates growth an in BMI List

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

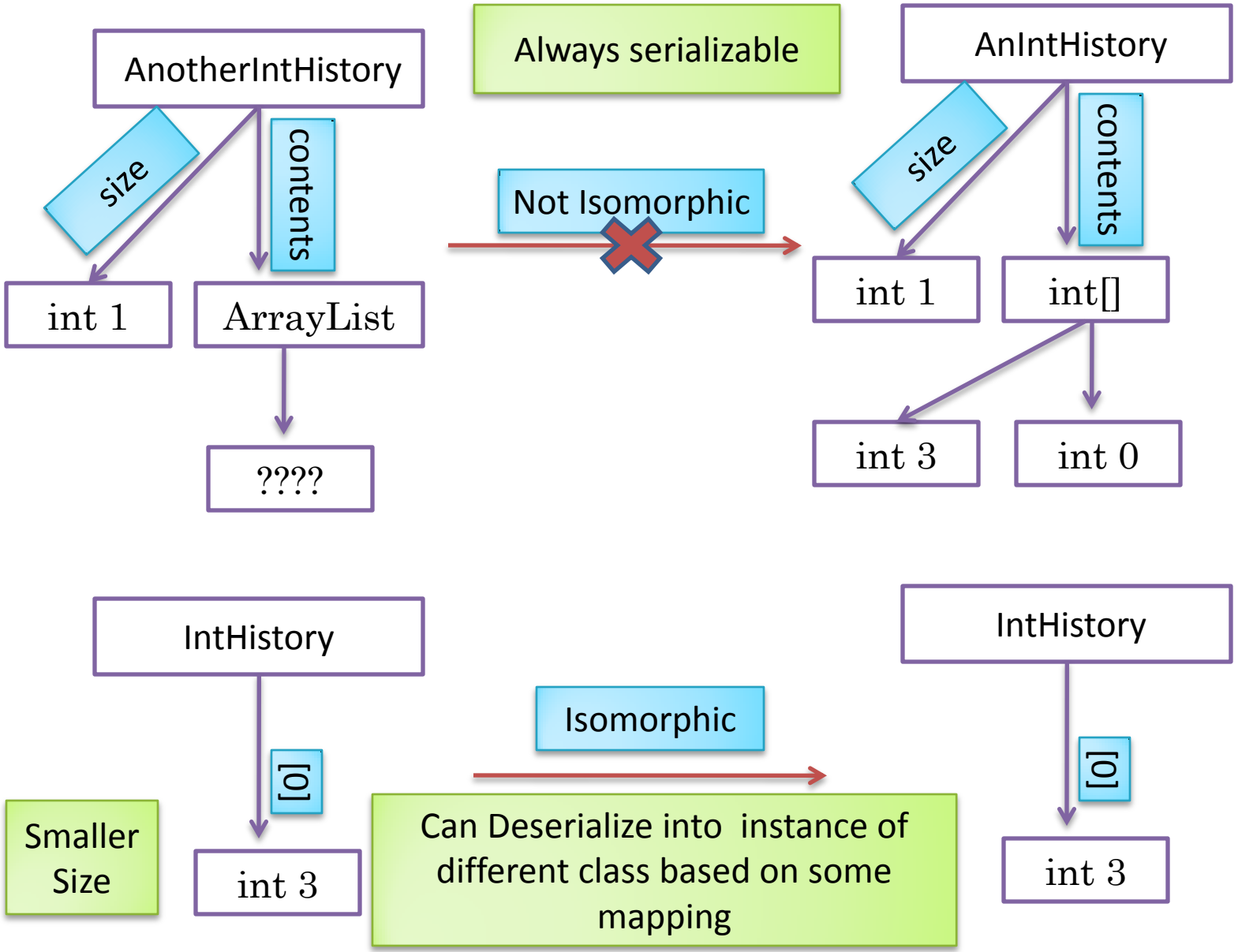
Often physical serialization overridden to serialize physical structures

Physical vs. Logical (Heterogeneity)

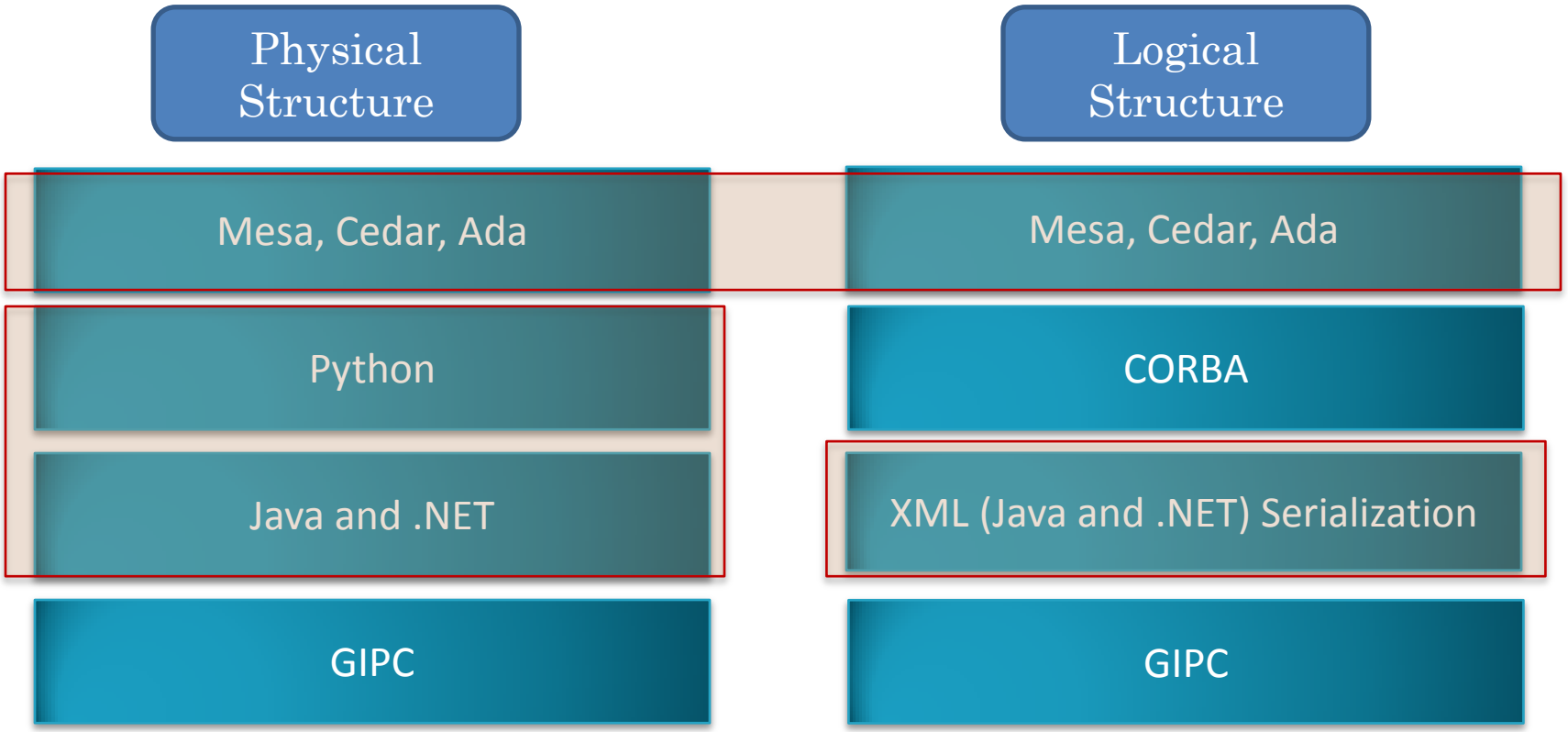
Different physical structures can imply same logical structure
Can translate into object of different class
Different logical structures can imply same physical structure but received object not semantically equivalent
Semantically equivalence→same logical structure

registerDeserializingClass(SerializedClass, DeserializedClass)
registerDeserializingClass(Vector.class, ArrayList.class)

Communicating. Physical vs. Logical Structure



Physical vs. Logical Infrastructure Supported

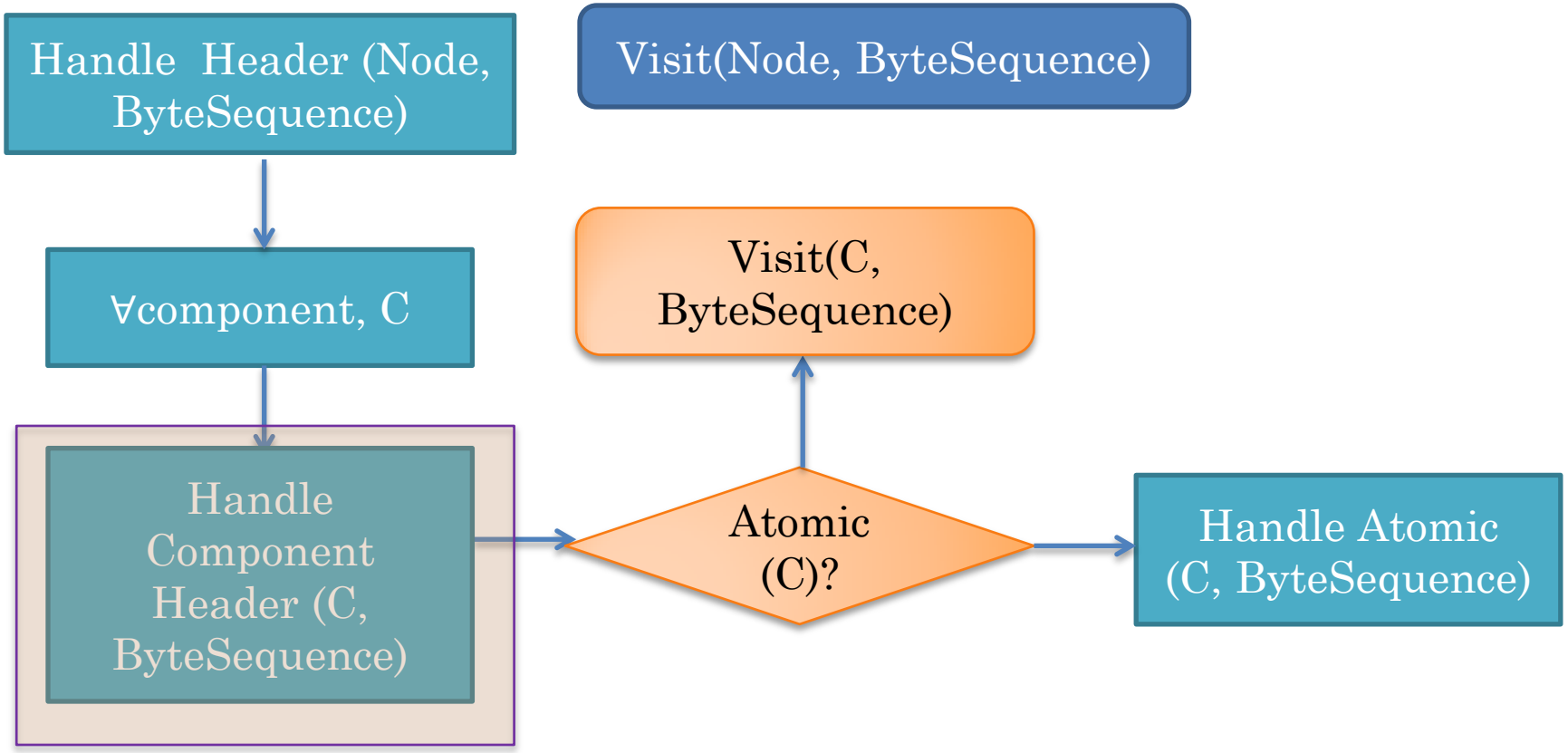


In (strongly typed) conventional language logical = physical structure

Performance problems addressed by extensibility

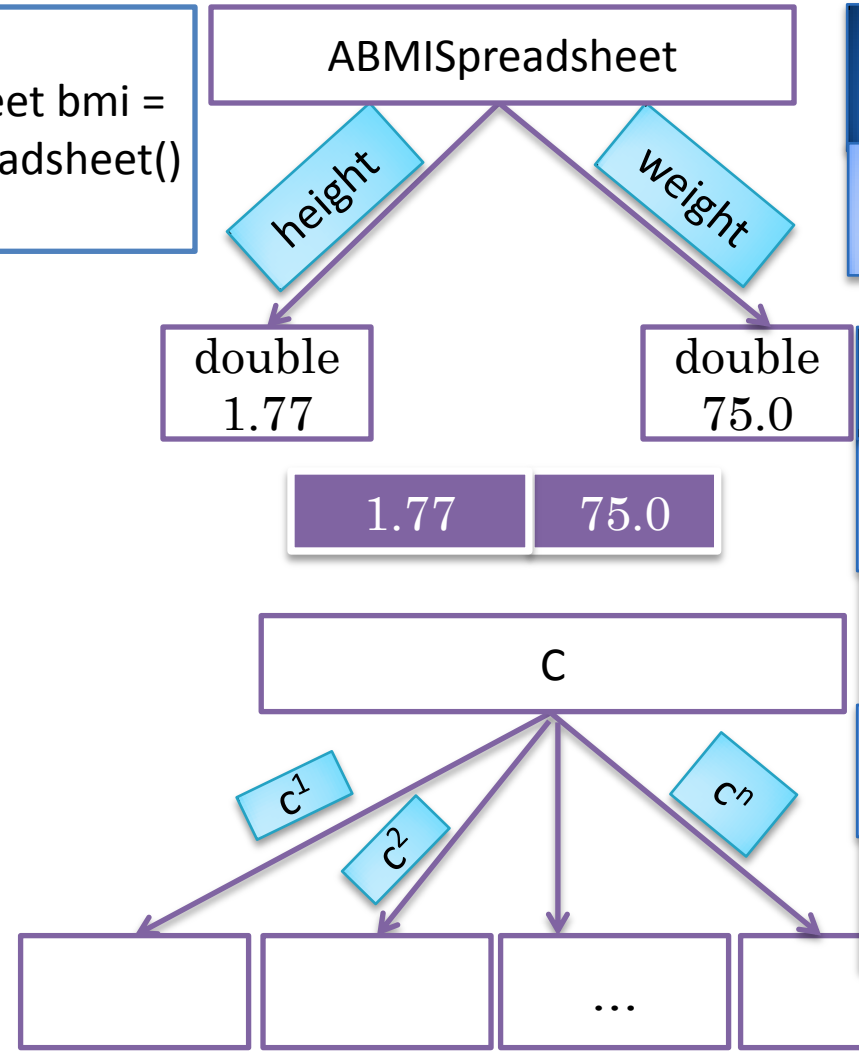
Heterogeneity problems addressed by extendible XML Bean Serialization

Algorithm


Handle Header (Node, ByteSequence)
Visit(Node, ByteSequence)
∀component, C
Visit(C, ByteSequence)
Handle Component Header (C, ByteSequence)
Atomic (C)?
Handle Atomic (C, ByteSequence)

No Component Name

BMISpreadsheet bmi =
new ABMISpreadsheet()



Works for unnamed components

Both ordered and unordered collections (Set, List)

Works when order of named fields is fixed

Same language and compiler

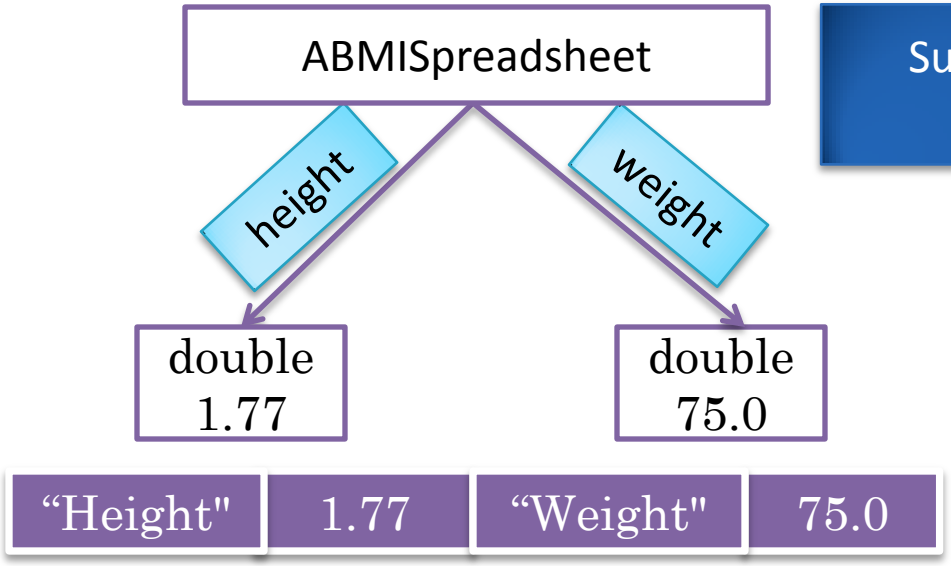
or Alphabetical sorting

Full deserialization

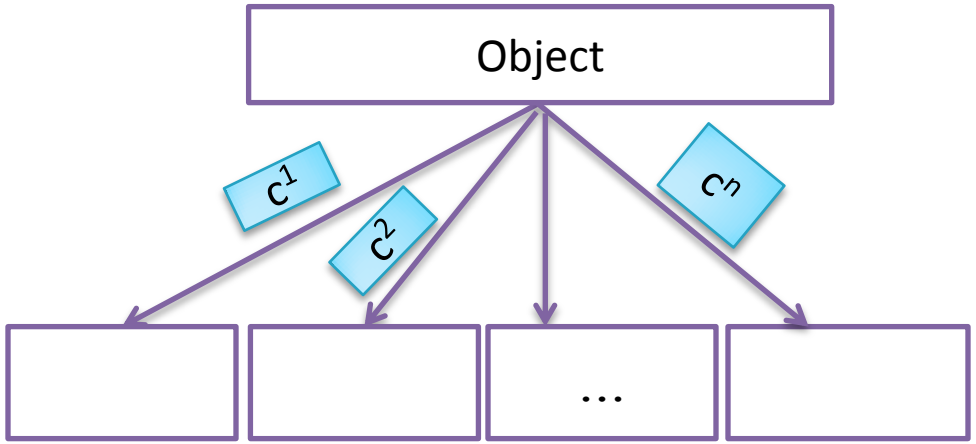
Known destination type

C^1 Value Serialization	C^2 Value Serialization	…	C^n Value Serialization

Sending Component Names

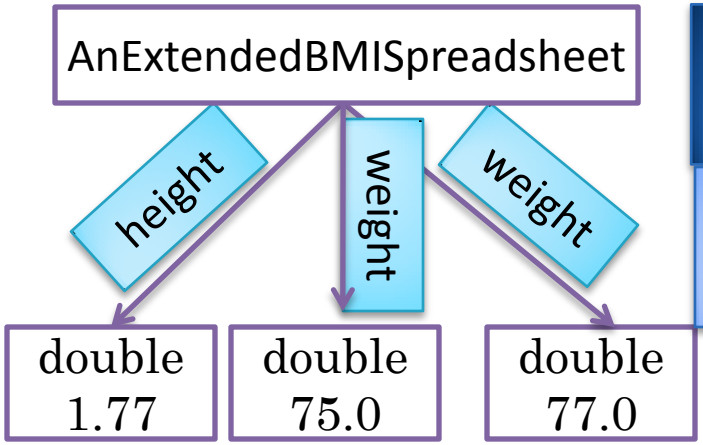


Supports heterogeneous platforms



C1 Name &Value Serialization	C2 Name &Value Serialization	…	Cn Name &Value Serialization

Duplicated Names



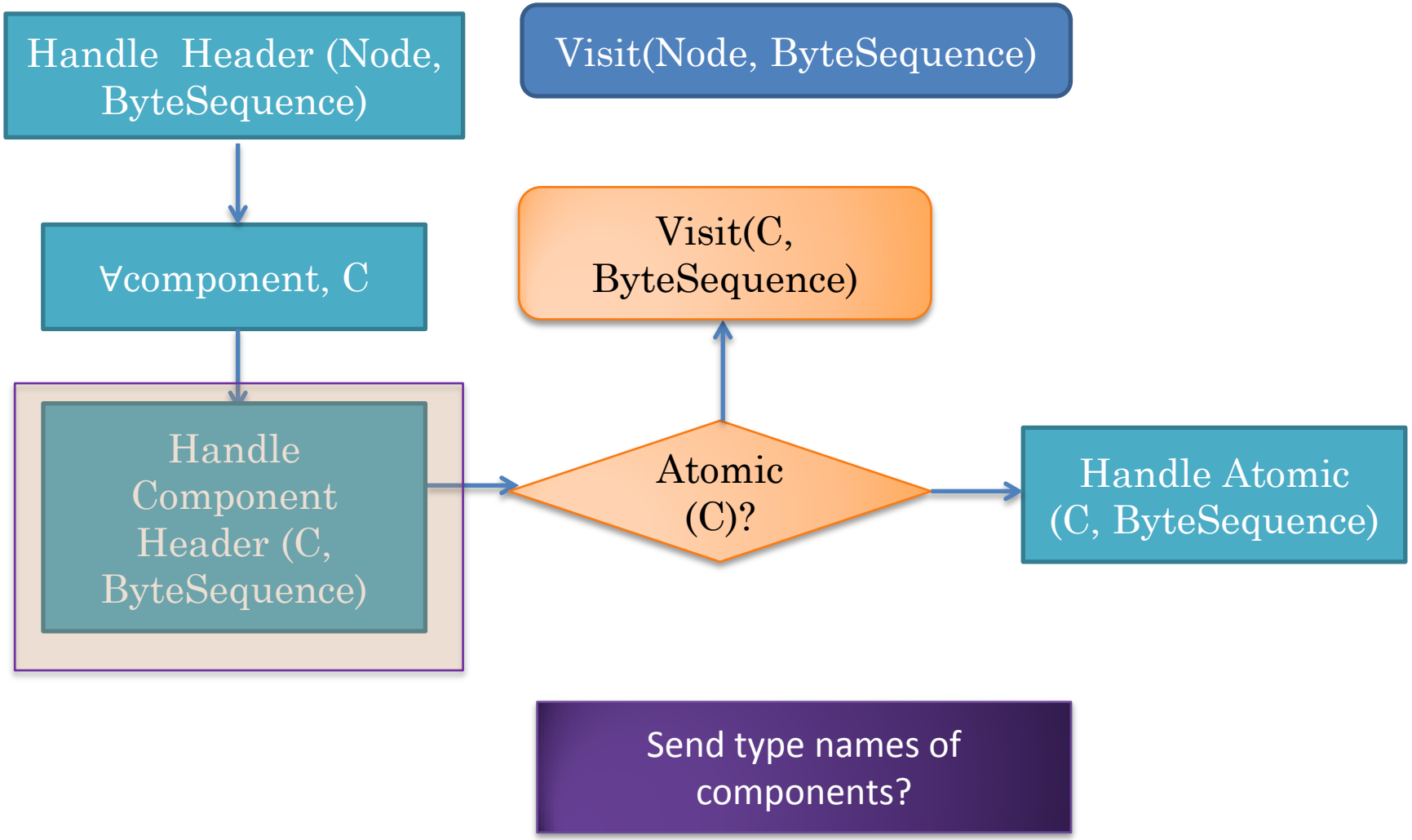
Can have multiple components with same name

public variable and property, variable in different classes

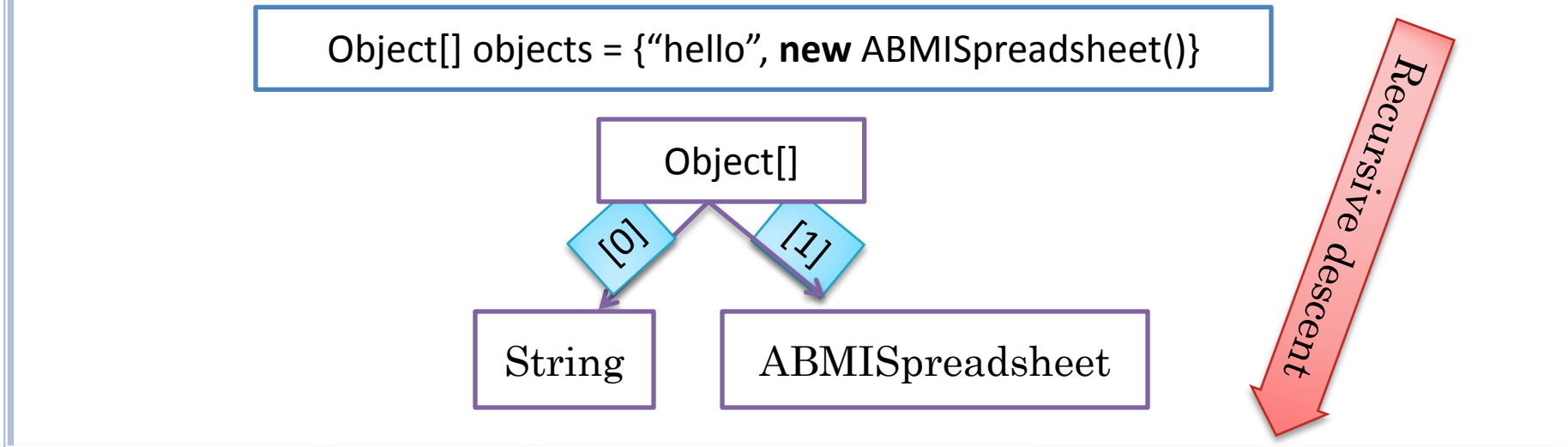
ABMISpreadsheet	"Height"	1.77	"Weight"	75.0
AnExtendedBMISpreadsheet	"Weight"	75.0		

Class T^1 Serialization	Class T^1 Declared Components	Class T^2 Serialization	Class T^2 Declared Components

Algorithm


Handle Header (Node, ByteSequence)
Visit(Node, ByteSequence)
∀component, C
Visit(C, ByteSequence)
Handle Component Header (C, ByteSequence)
Atomic (C)?
Handle Atomic (C, ByteSequence)
Send type names of components?

Composing (Polymorphic) Object Values



Polymorphic Object Component

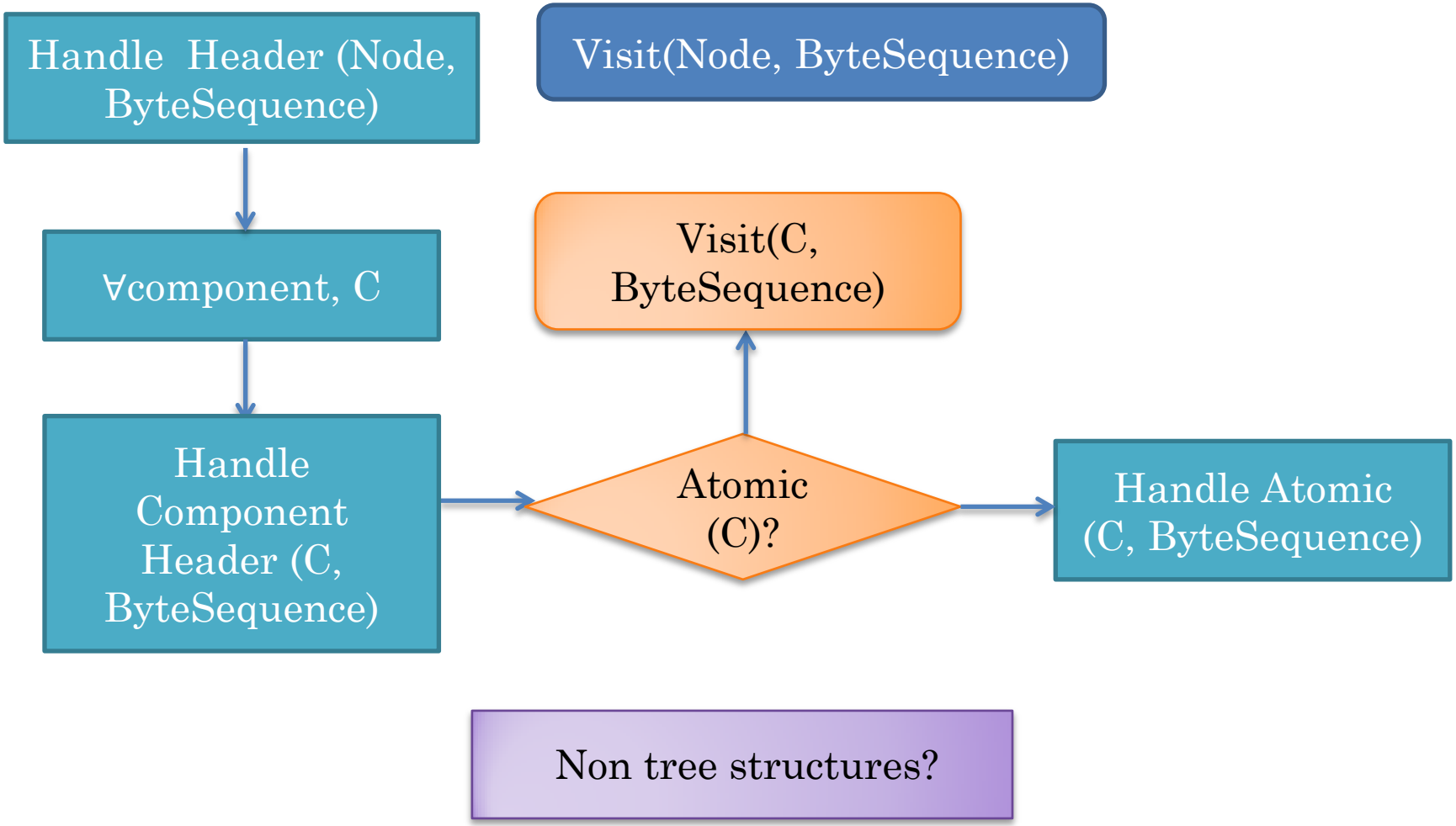
Other Component Serialization

Class Serialization

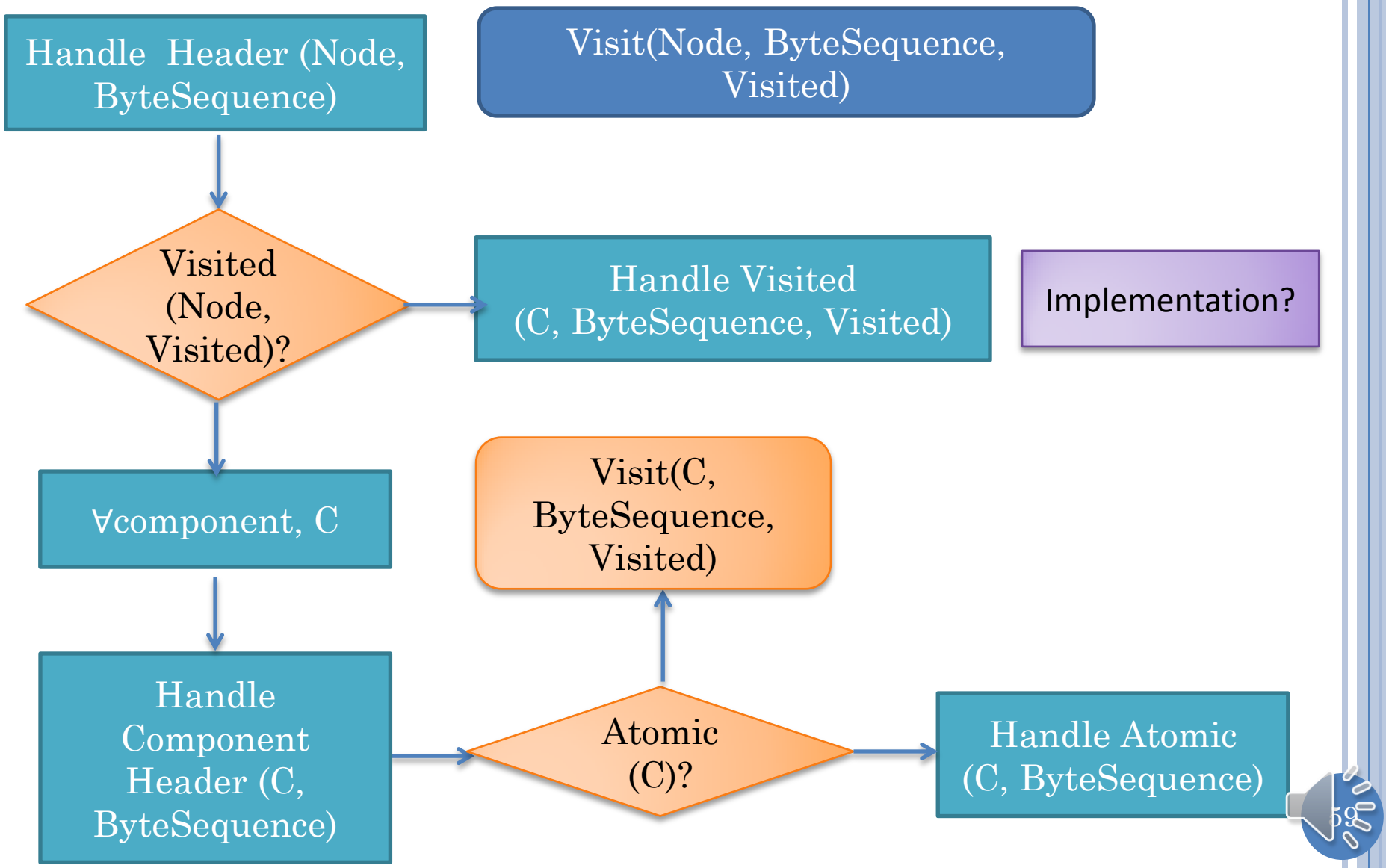
(Name &) Value Serialization

(Name &) Value Serialization

Basic Tree Algorithm

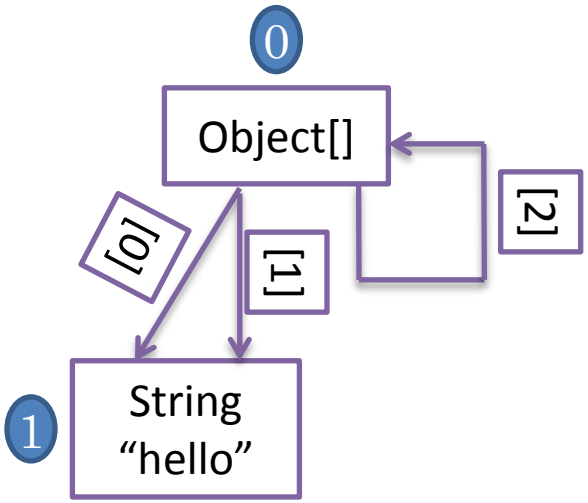


Visited Node and Types



Visited Nodes

```
Object[] objects = new Object[2];
objects[0] = “hello”;
objects[1] = objects[0];
objects[2] = objects;
```

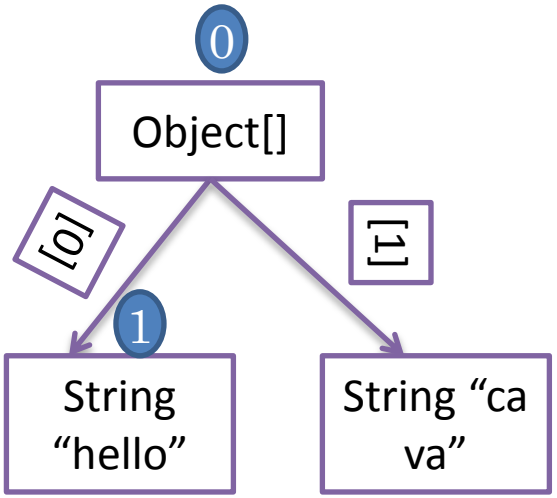


- Algorithm keeps track of visited nodes
- Visit number: number of first visits before a node first visited in recursive algorithm.
- Assume corresponding nodes visited in the order in serialization and deserialization
- Serialization serializes revisited node value by putting visit number of the node
- Deserialization converts visit number to already constructed node
- Node numbers are internal pointers within serialized representation

Repeated Types

```
Object[] objects = new Object[2];
objects[0] = “hello”;
objects[1] = “ ca va”;
```

Supported in Java



What if all elements happen to be String?

Message Pointers

Value (and type) pointer

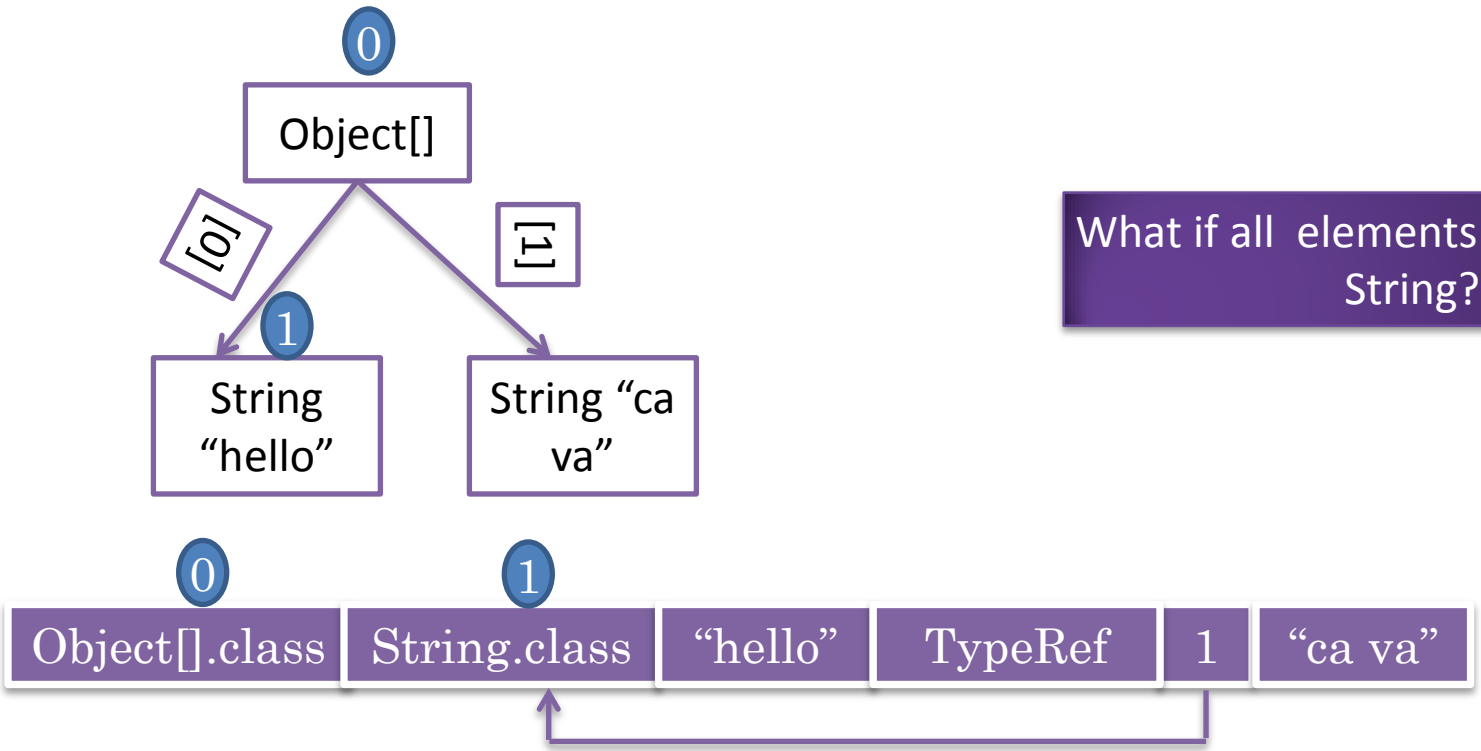
Point to previously visited value and type

Type pointer

Point to previously visited value type

Repeated Types

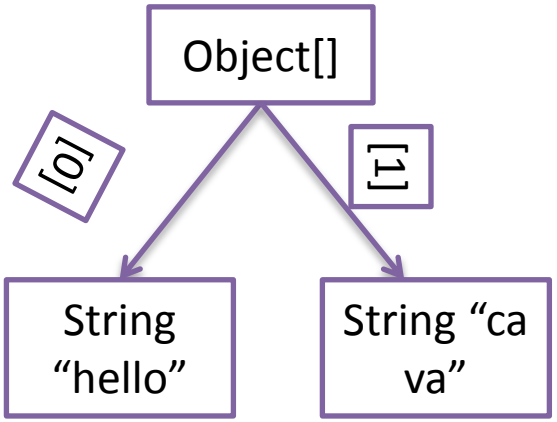
```
Object[] objects = new Object[2];
objects[0] = “hello”;
objects[1] = “ ca va”;
```



What if all elements happen to be String?

Repeated Types With Programmer Tagging

```
@StringElement
Object[] objects = new Object[2];
objects[0] = “hello”;
objects[1] = “ ca va”;
```



Element class name sent once

Supported in .NET XML Serialization

XML attribute used for manual XML composition

C# attributes used for dynamic XML generation

Object[].class	String.class	“hello”	“ca va”

Message Pointers

Value (and type) pointer

Point to previously visited value and type

Type pointer

Point to previously visited value type

Independent Serializations

No Inter Message Compression

String.class	"hello"

String.class	"ca va"

Inter Message with Local Addressing and Headers

Msg 0	String.class	"hello"

Msg 1	Type Ref	0.0	"ca va"

Inter message with Global (type) Addressing

String.class	"hello"

Type Ref	0	"ca va"

Inter message compression reduces generality and correctness of serialization scheme

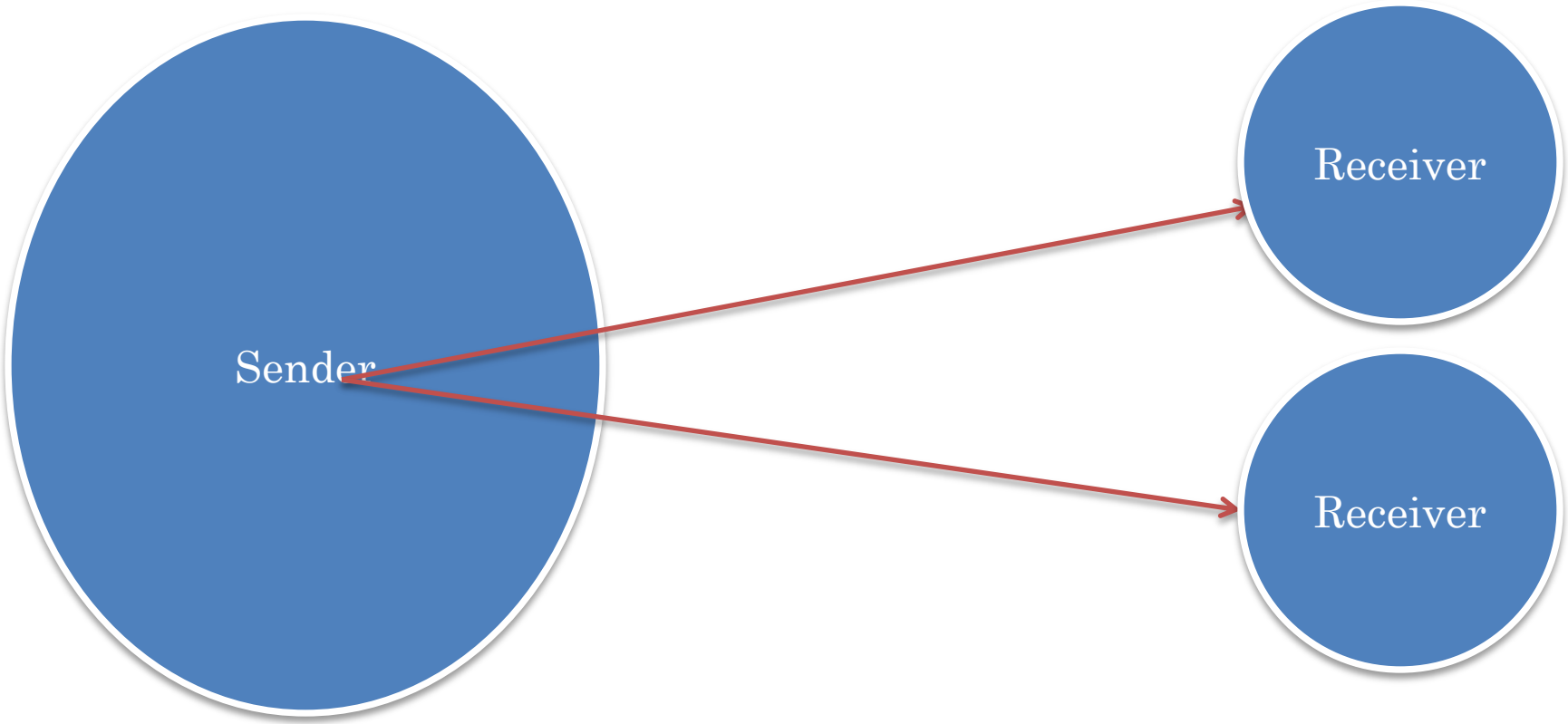
Inter-Message Compression

```
ObjectOutputStream socketOut = new ObjectOutputStream(socket.getOutputStream());
Object[] objects = new Object[1];
objects[0] = “hello”;
socketOut.writeObject(objects);
objects[0] = “ca va”;
socketOut.writeObjects(object);
```

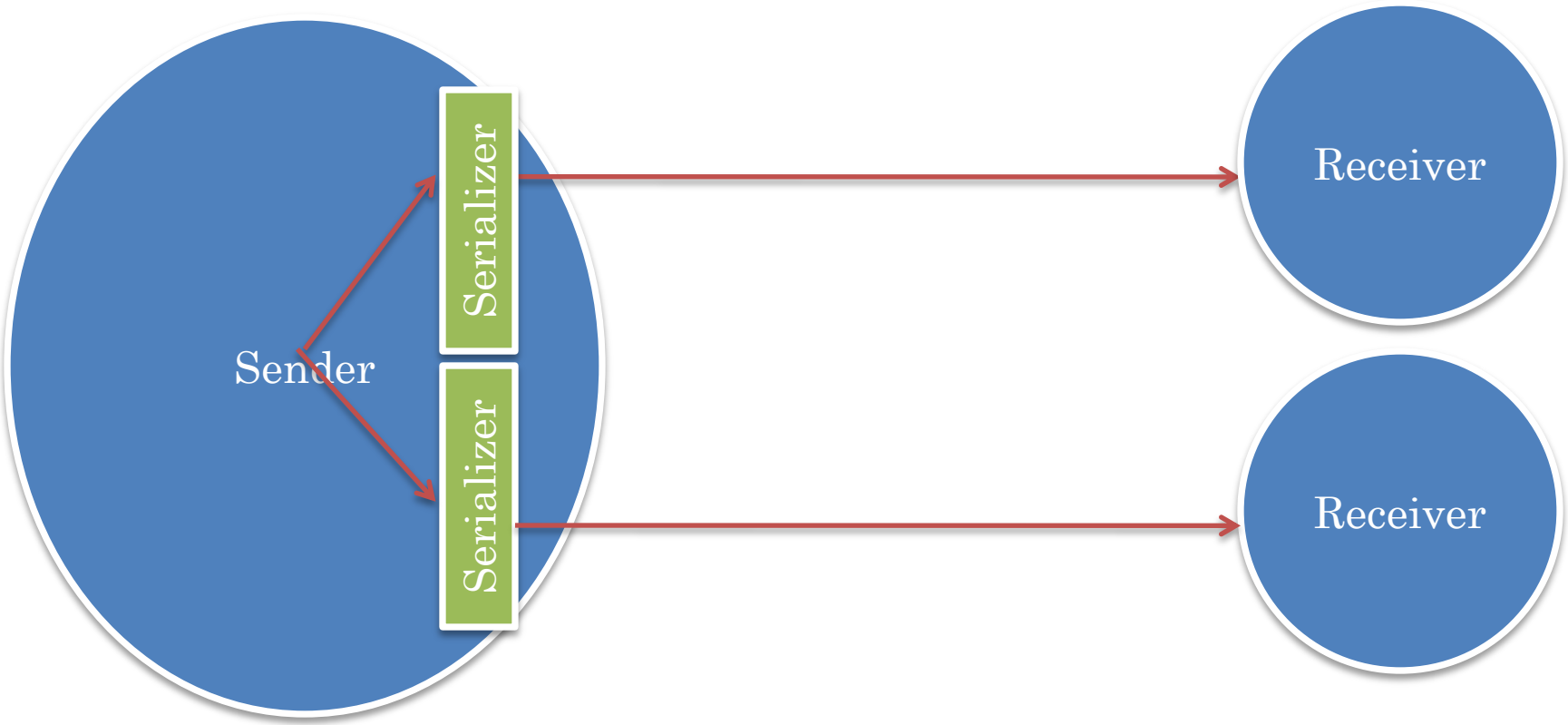
objects[0]	Reference	0

Stream could be re created or reset to release cache

Multicast and Serialization



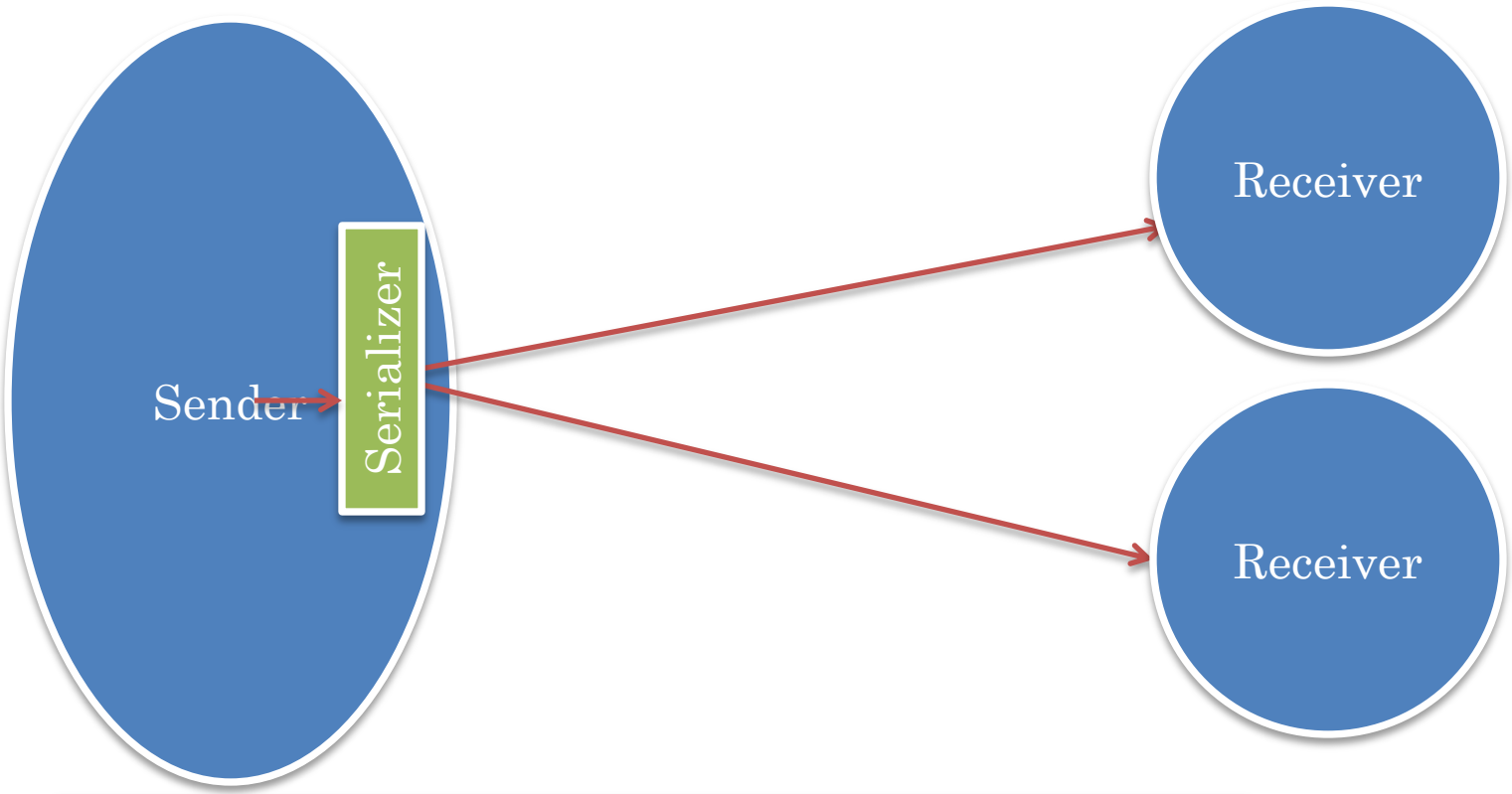
Separate Serializer



Could serialize separately for each destination

Serialization is expensive

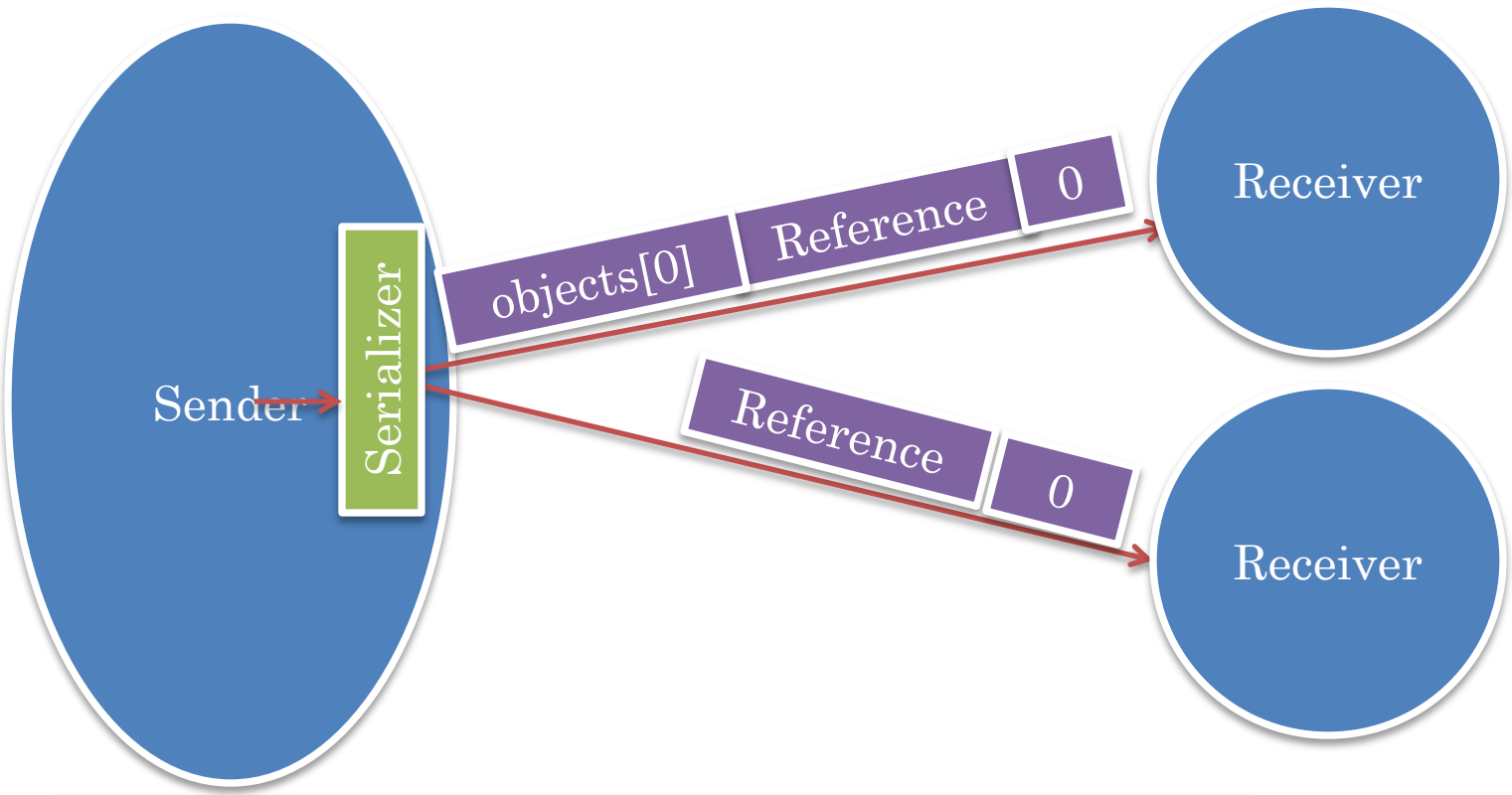
Common Serializer



Assuming single serializer object for all receivers

Interferes with compression when multicast is not broadcast

Multicast vs. Compression



Assuming single serializer object for all receivers

Stream is corrupted when assumption breaks

Relayer

Serialization Communication Implemented by Who

Infrastructure (Language, Library)

Programmer (of Communicated Objects)

Both (Programmer code called by Infrastructure)

Reasons for overriding?

Extensibility Reasons

- Some parts of objects cannot be serialized
- Some parts of objects do not need to be serialized and need to be derived
- May want a more efficient serialization scheme (sp. for physical structures)
- As new patterns emerge, new (de) serializers needed
- May need to determine which equivalent type to deserialize an object to

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);
  }
}
```

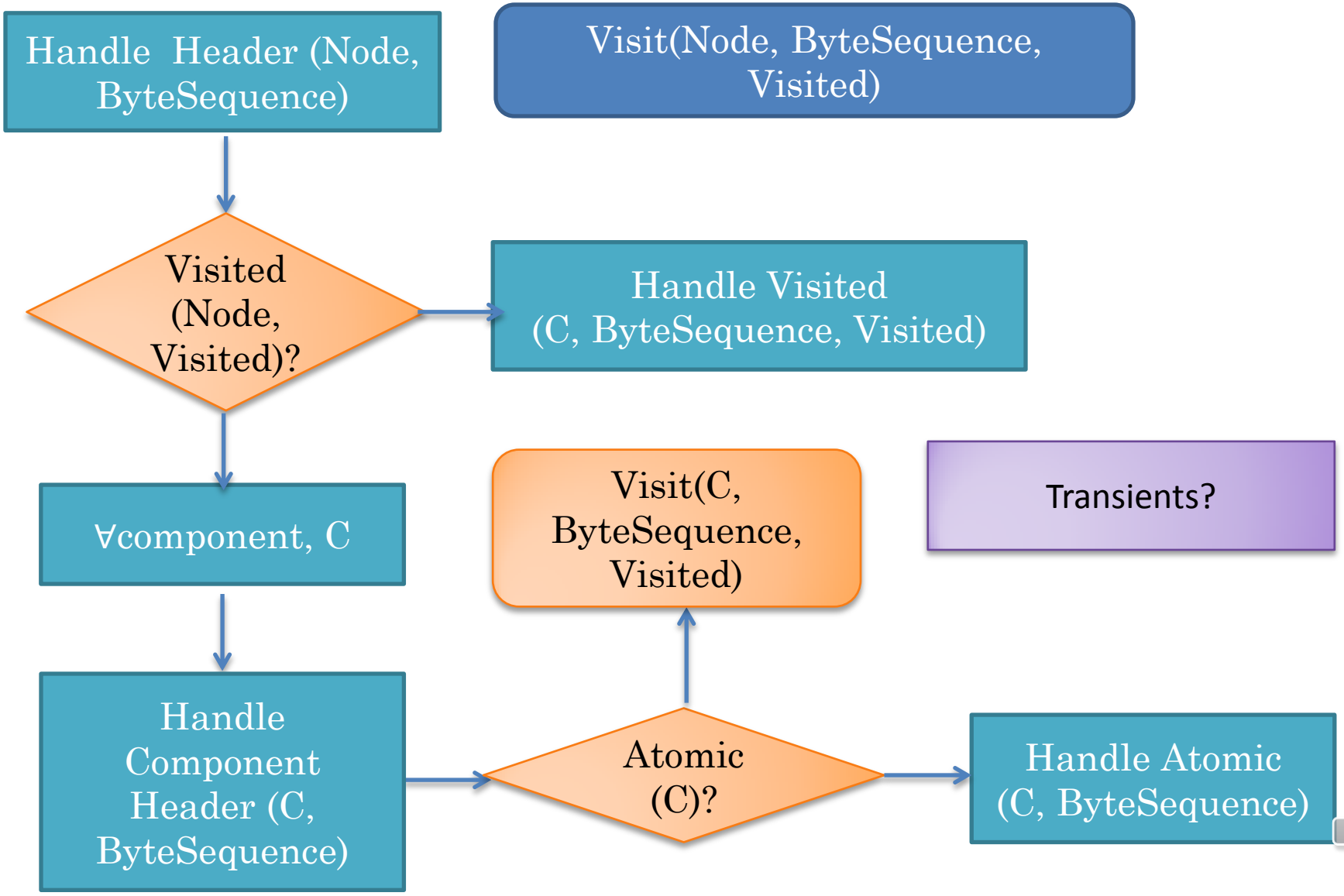
Logical?

GIPC: Transient Annotation for Logical

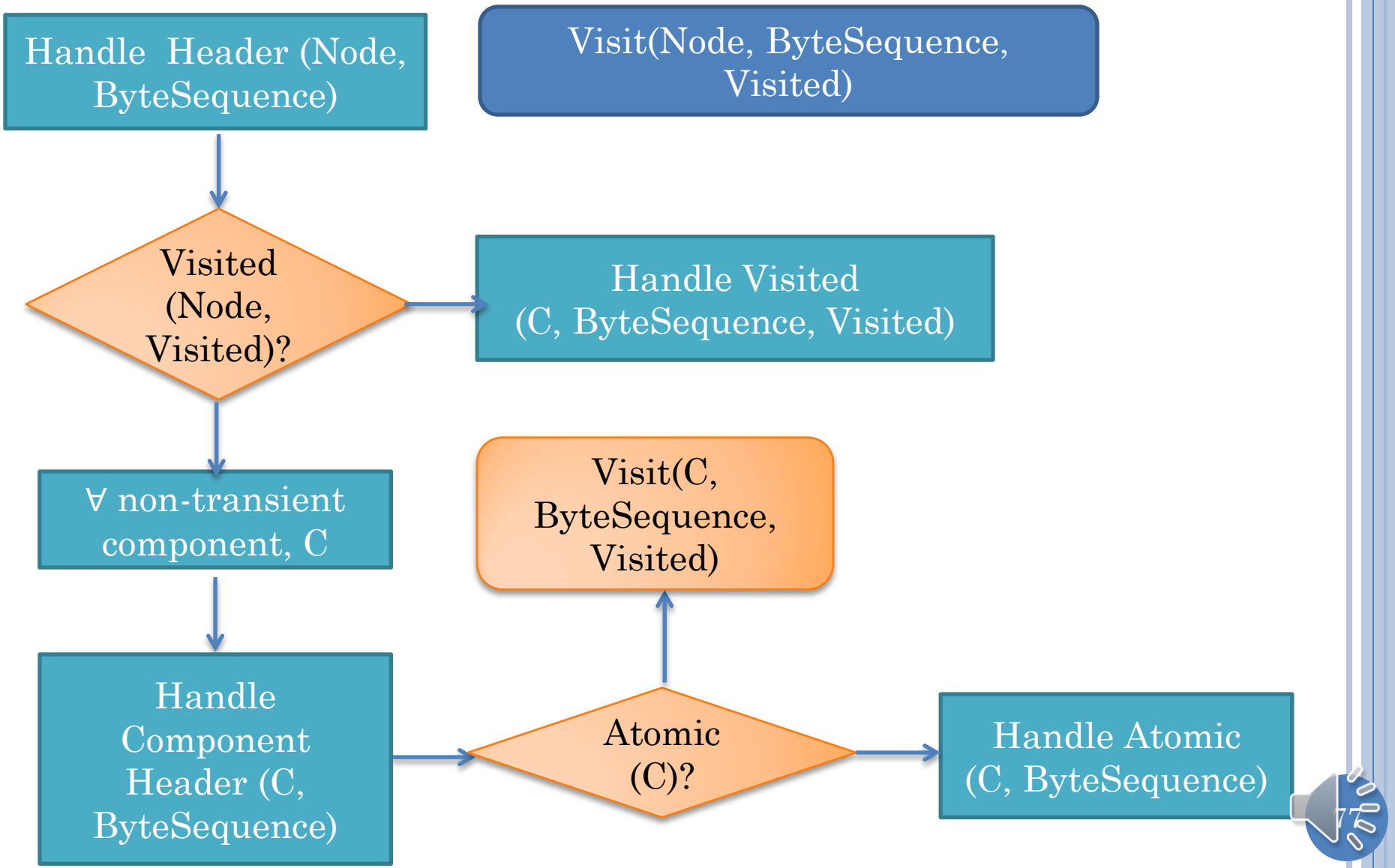
```
public class AnotherBMISpreadsheet implements BMISpreadsheet{
  double height = 1.77;
  double weight = 75;
  double bmi = calculateBMI();
  public double getHeight() {return height;}
  public void setHeight(double newHeight) {
    height = newHeight;
    bmi = calculateBMI();
  }
  public double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

Logical component returned by annotated method is transient

Without Transients



Without Transients



Extensibility Reasons

Some parts of objects cannot be serialized

Some parts of objects do not need to be serialized and need to be derived

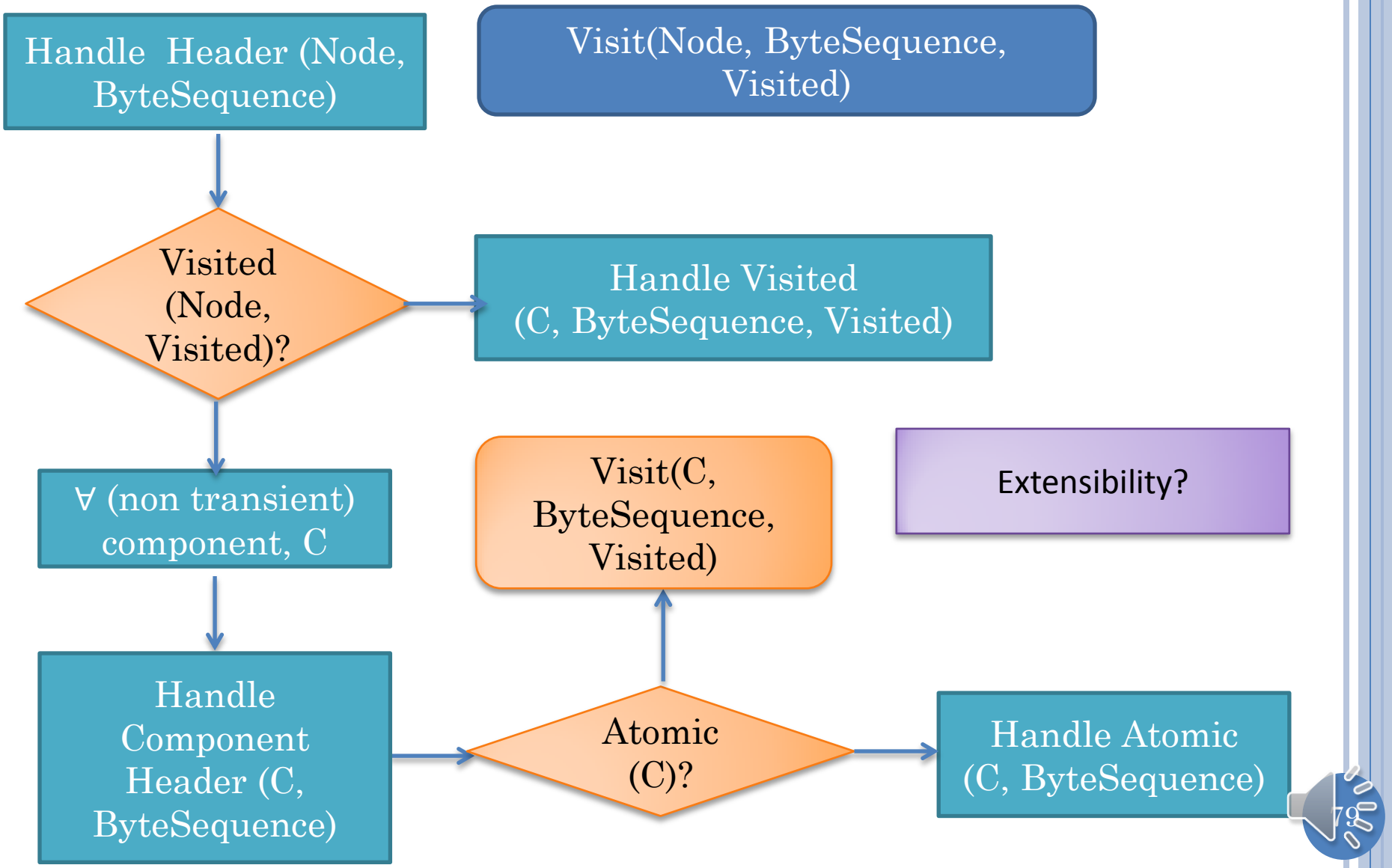
May want a more efficient serialization scheme (sp. for physical structures)

As new patterns emerge, new (de) serializers needed

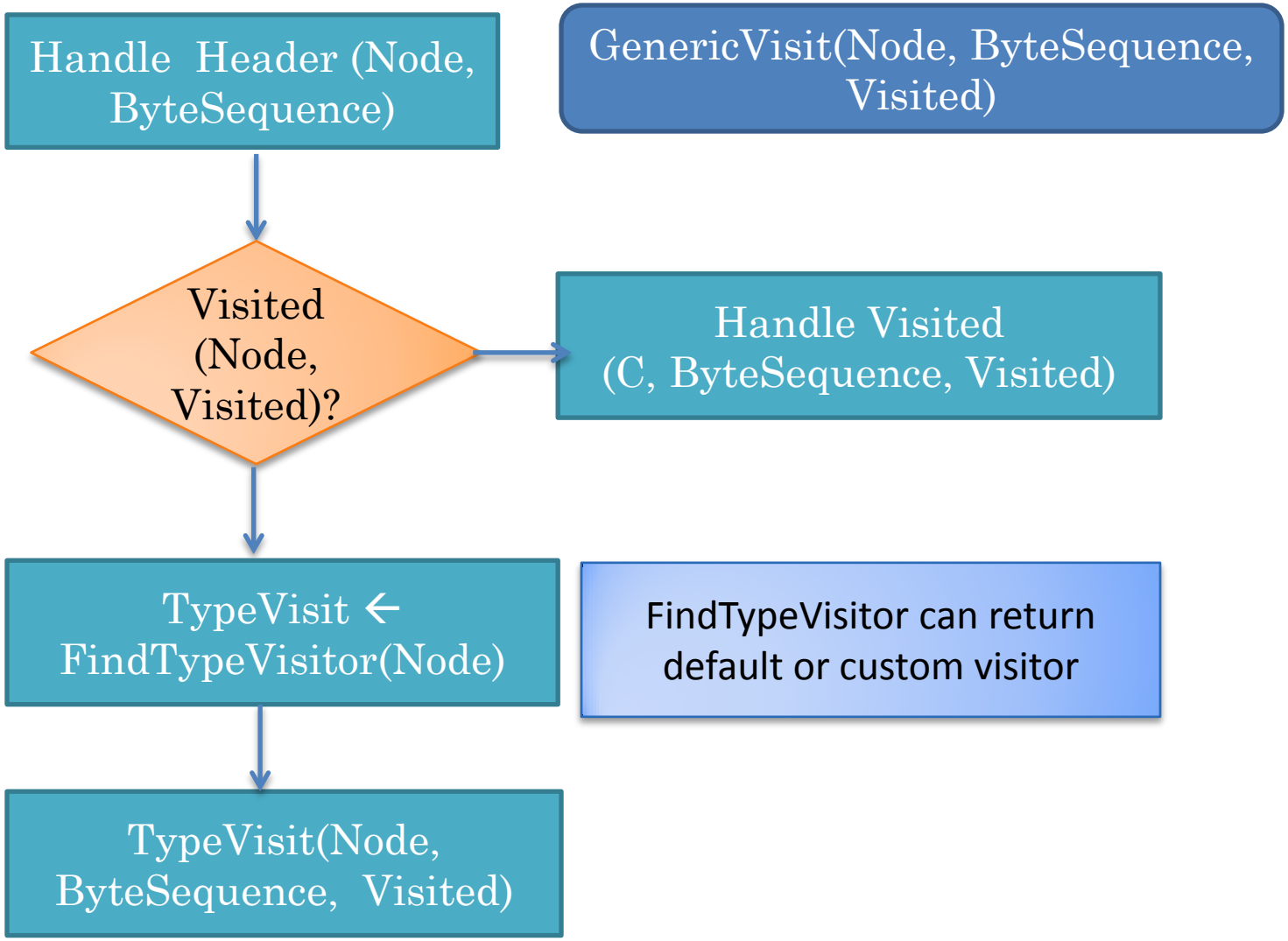
May need to determine which equivalent type to deserialize an object to

Custom (de)serializers?

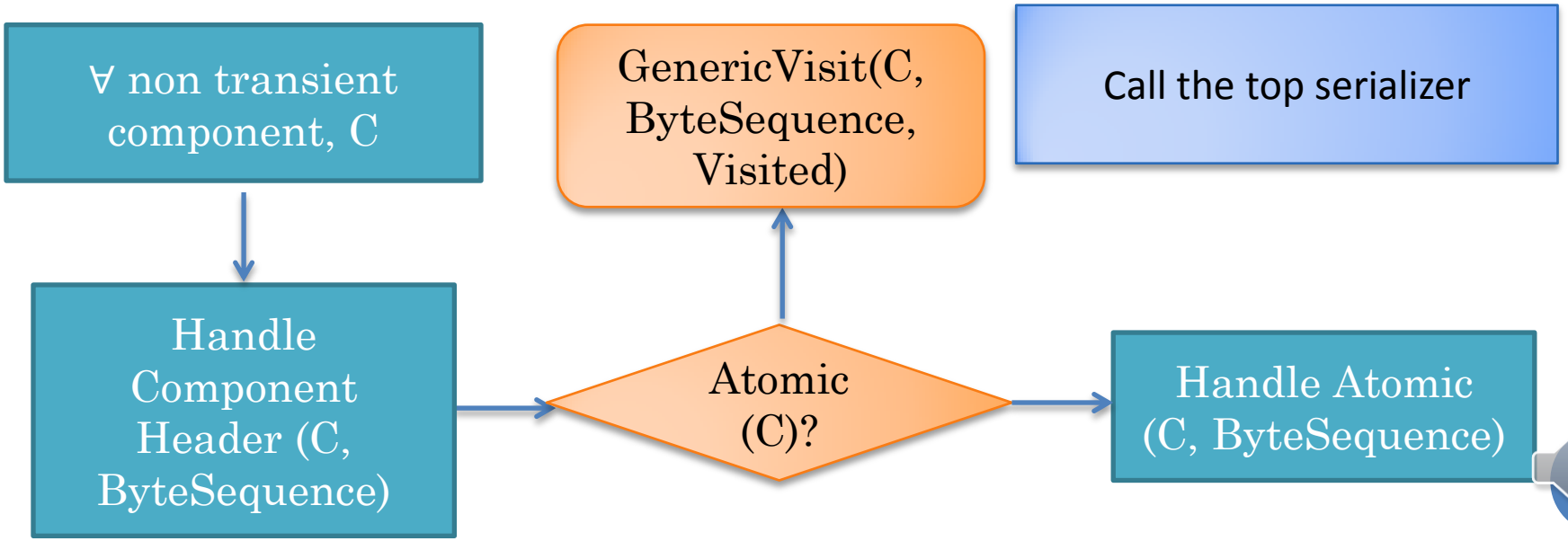
Basic Algorithm



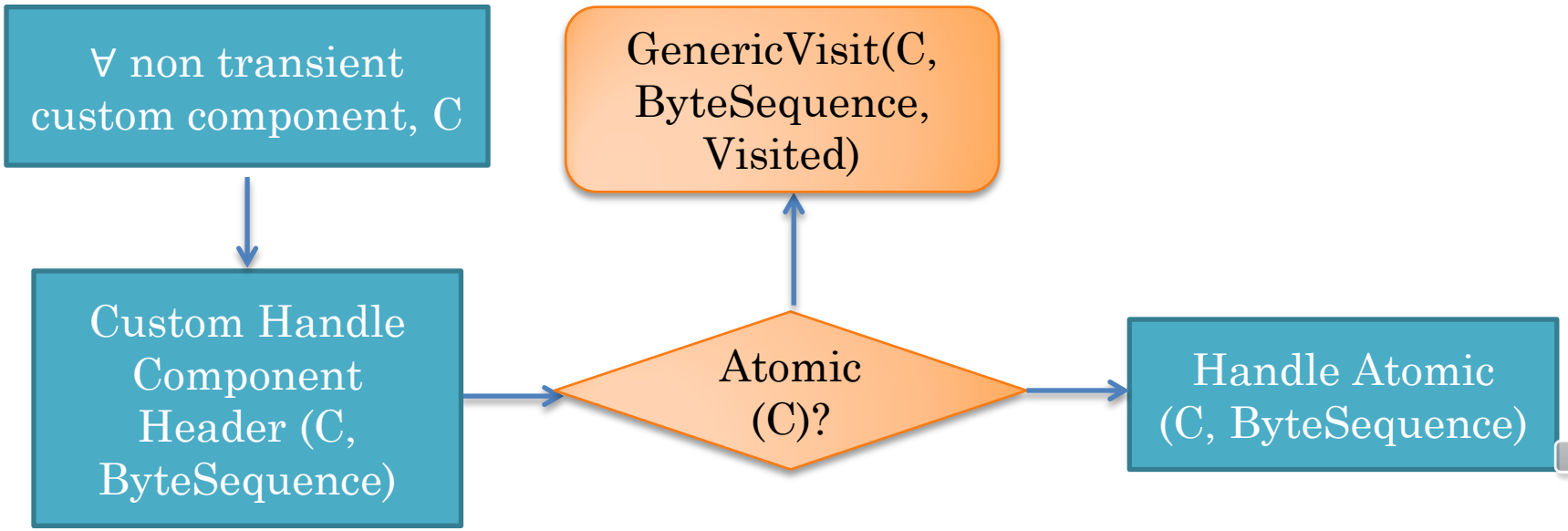
Generic vs. Type Visitor



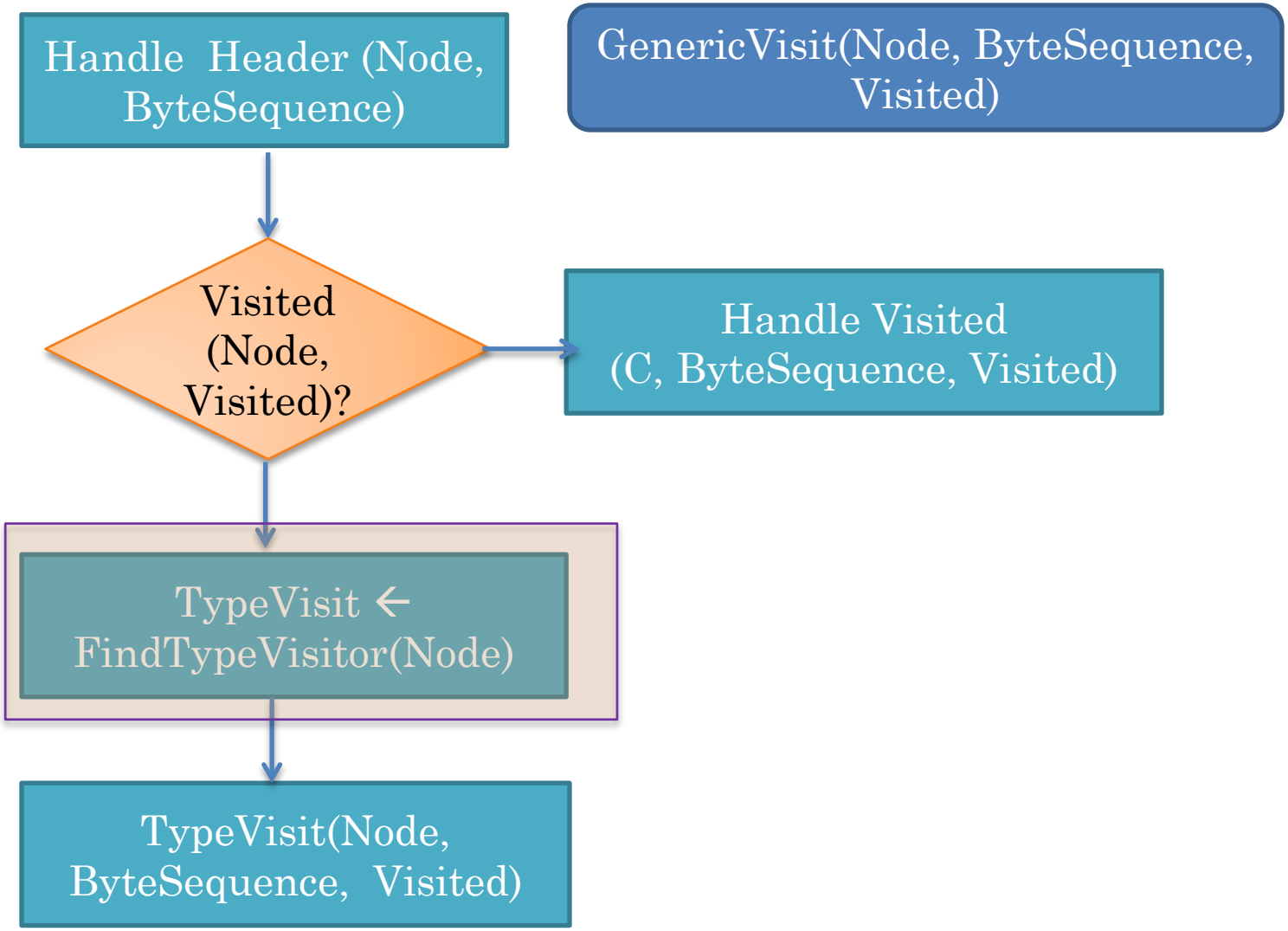
Default Type Visitor



Typical Custom Type Visitor



Generic vs. Type Visitor

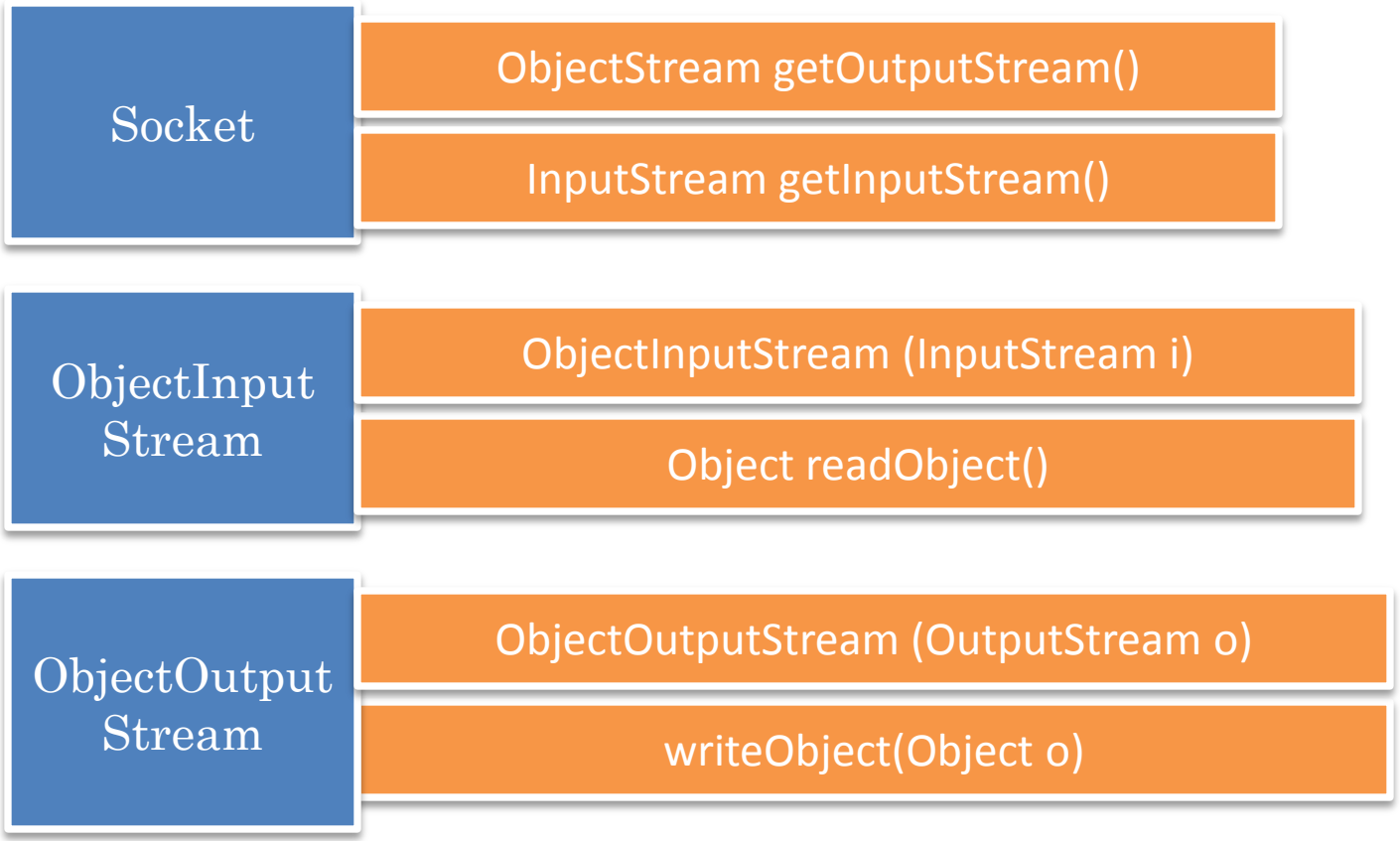


Specifying Custom (De)Serializers in Java

TypeVisit ← FindTypeVisitor(Node)

(De)Serializer Class Implements Custom Code

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();
```

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

Programmer-Defined ReadObject

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

Externalizable Interface based Functions

Externalizable

- readExternal (ObjectIn)
- writeExternal(ObjectOut)

These methods must exist if interface implemented

ObjectIn

- Object readObject(Atomic)(Object (Atomic))

ObjectOut

- writeObject(Atomic)(Object(Atomic))

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

Serializable vs. 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

Alternatives?

Extensibility Reasons

Some parts of objects cannot be serialized

Some parts of objects do not need to be serialized and need to be derived

May want a more efficient serialization scheme (sp. for physical structures)

As new patterns emerge, new (de) serializers needed

May need to determine which equivalent type to deserialize an object to

Alternatives?

Specifying Custom (De)Serialization

TypeVisit ← FindTypeVisitor(Node)

(De)Serialized Class Implements Custom Code for Overriding Serialization and Initialization

Specifying Custom Serializers

TypeVisit ← FindTypeVisitor(Node)

(De)Serialized Class Implements Custom Code for Overriding Serialization and Initialization

Special pattern for initializing serialized method

Call Special Methods: GIPC 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 d
  public v
     weight = newWeight;
     bmi =
  }
  public d
  double calculateBMI() {
     return weight/(height*height);
  }
  public void initSerializedObject() {
    bmi = getBMI();
  }
}
```

Like constructor, method with special signature called when object deserialized

Similar mechanism in other systems integrated with extendibility support such as testing

Extensibility Reasons

Some parts of objects cannot be serialized

Some parts of objects do not need to be serialized and need to be derived

May want a more efficient serialization scheme (sp. for physical structures)

As new patterns emerge, new (de) serializers needed

May need to determine which equivalent type to deserialize an object to

Custom Serialization

TypeVisit ← FindTypeVisitor(Node)

(De)Serialized Class Implements Custom Code for Overriding Serialization and Initialization

Special pattern for initializing serialized method

External Custom Serializer

Registry

Custom External Serializer API

registerSerializer(Type, Serializer)
registerSerializer(Integer.**class**, integerSerializer);
registerSerializer(HashSet.**class**, collectionSerializer);

registerSerializer(Pattern, Serializer)
registerSerializer(BEAN_PATTERN, listSerializer);
registerSerializer(LIST_PATTERN, collectionSerializer);

Comparison?

TypeVisit ← FindTypeVisitor(Node)

(De)Serialized Class Implements Custom Code for Overriding Serialization and Initialization

Special pattern for initializing serialized method

External Custom Serializer

Registry

Coupling of Transients Initialization and Extensibility

Extensibility and initialization of transients is coupled

Often they are done together

Only one set of abstraction learnt

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 Handling and Extended Serialization

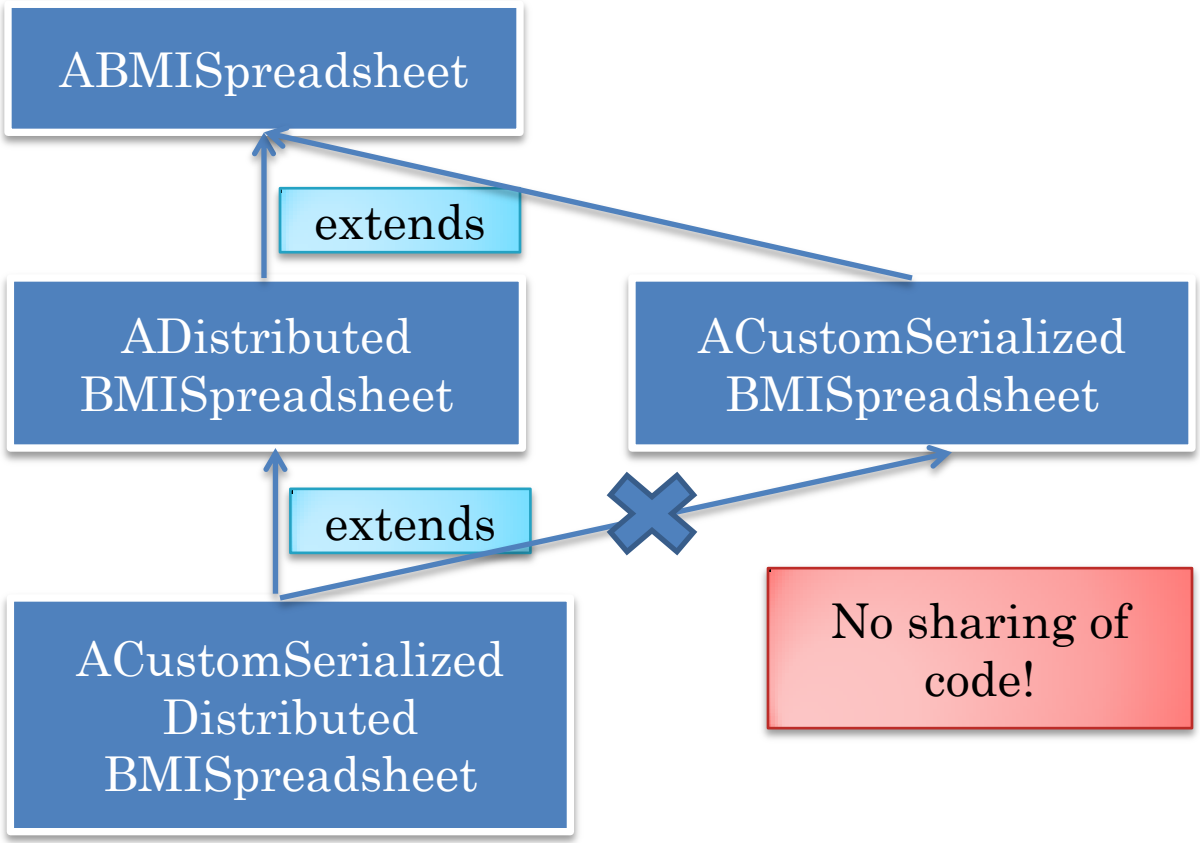
Serialization extension must be done by the class being serialized

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

Internal Serializer

Cannot add serializer for class without source code

Must inherit and change references and not reuse code

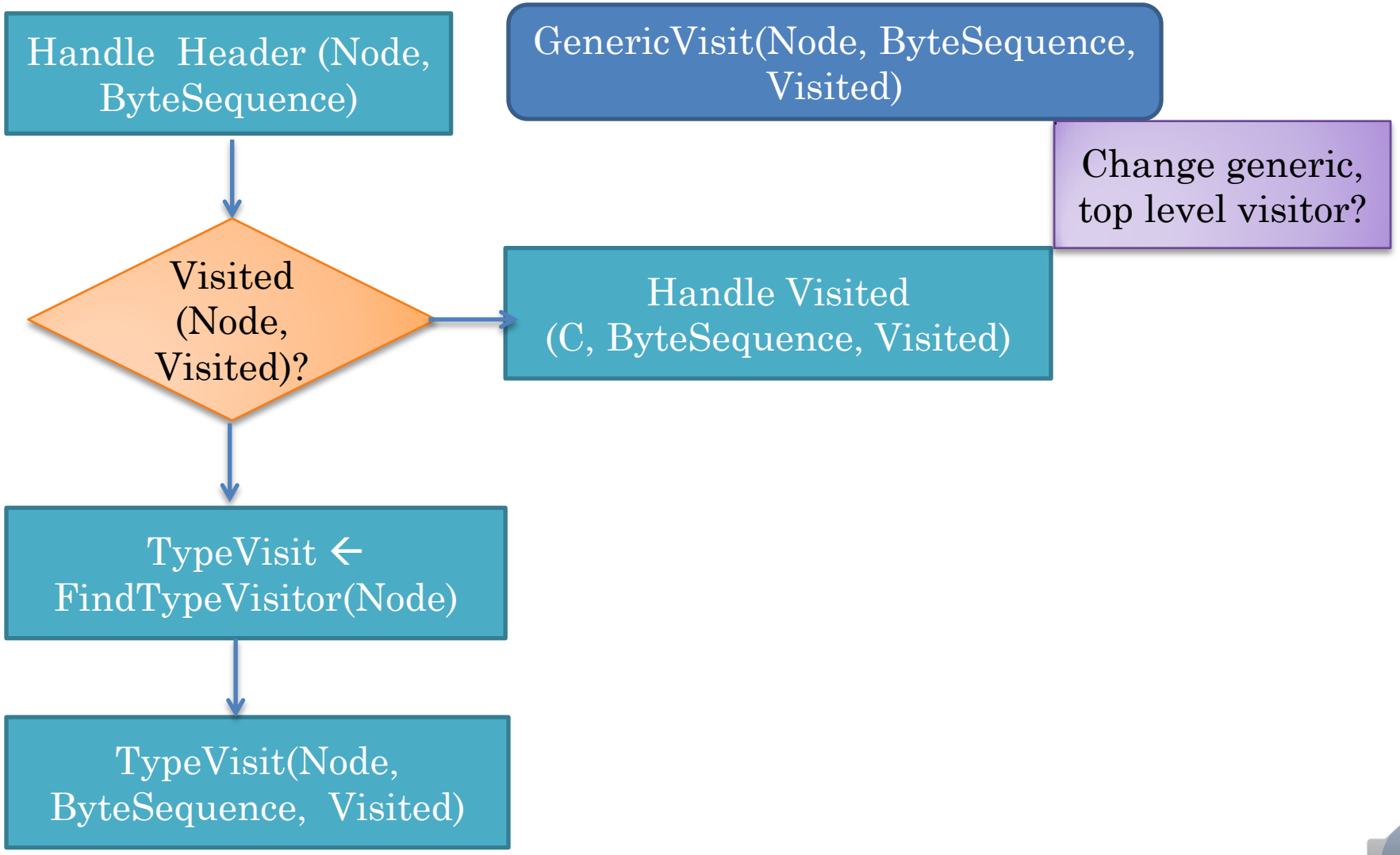


Coupling of Handling and Extended Serialization

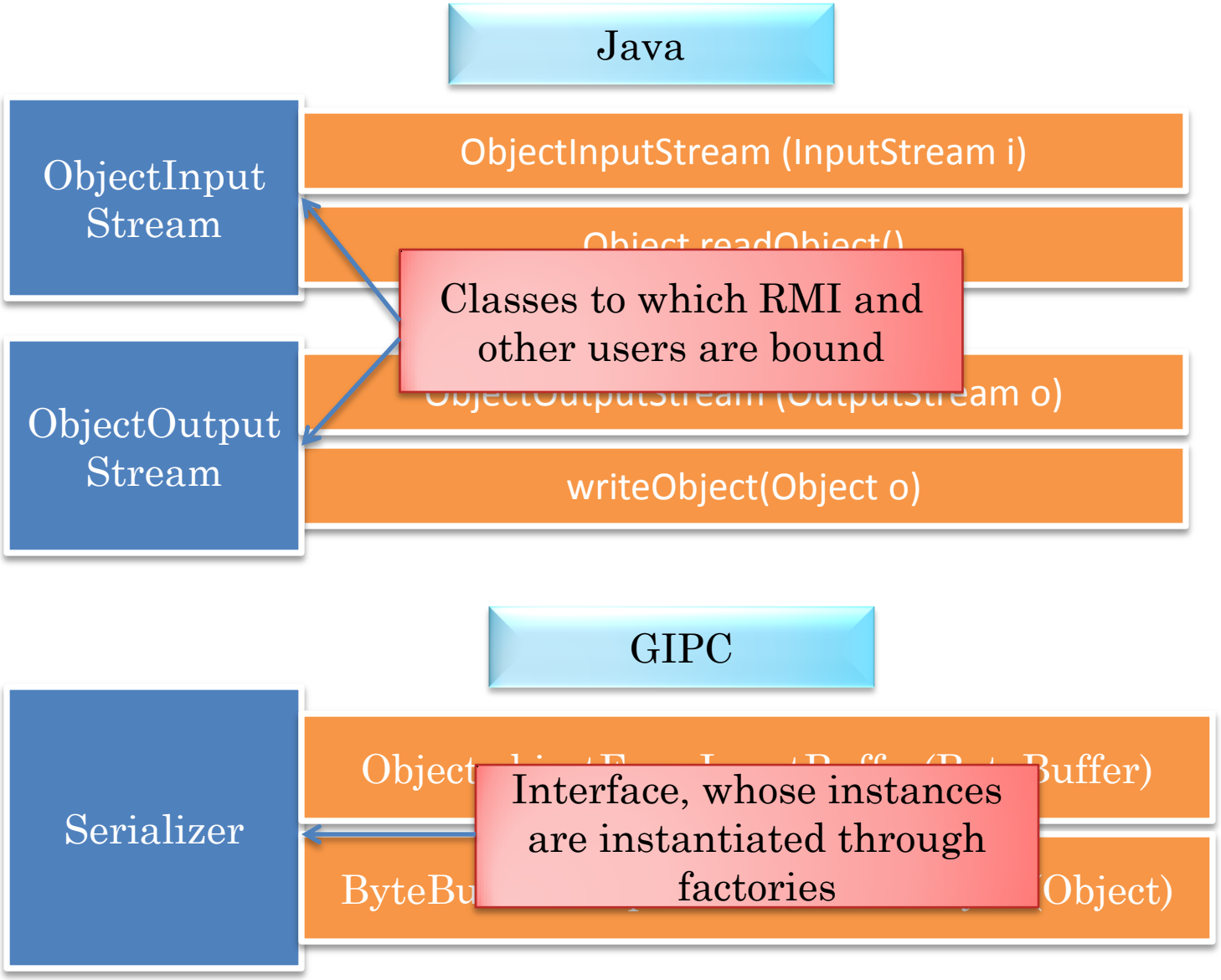
Cannot have multiple serializers for different contexts

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

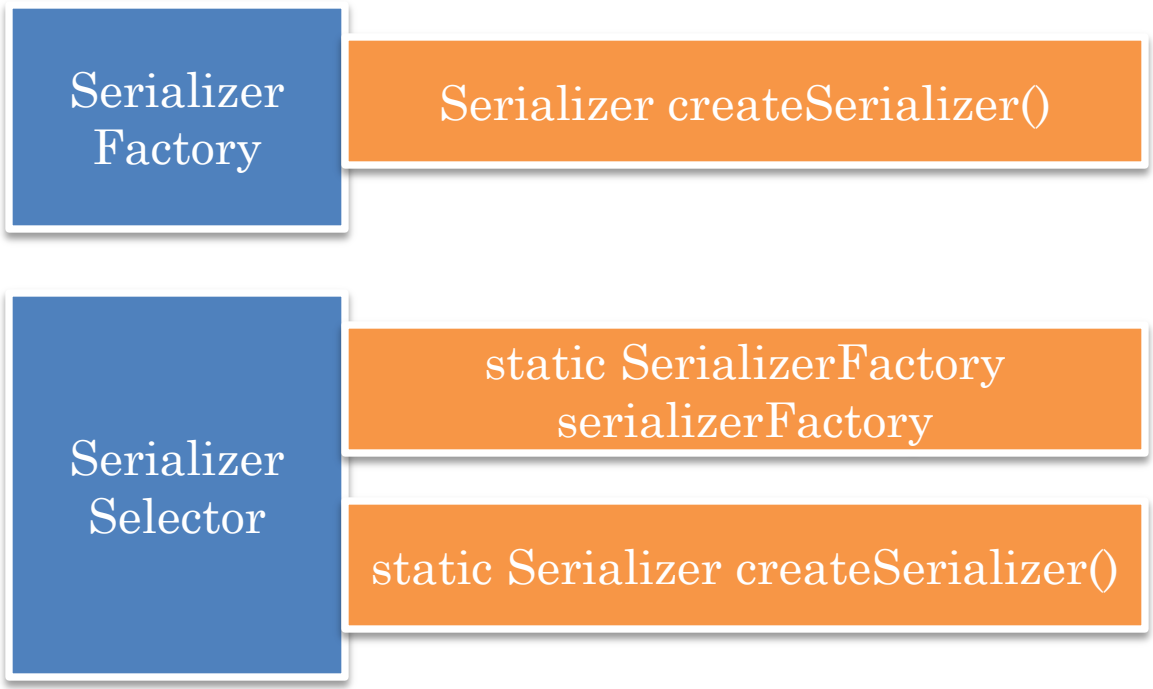
Generic vs. Type Visitor



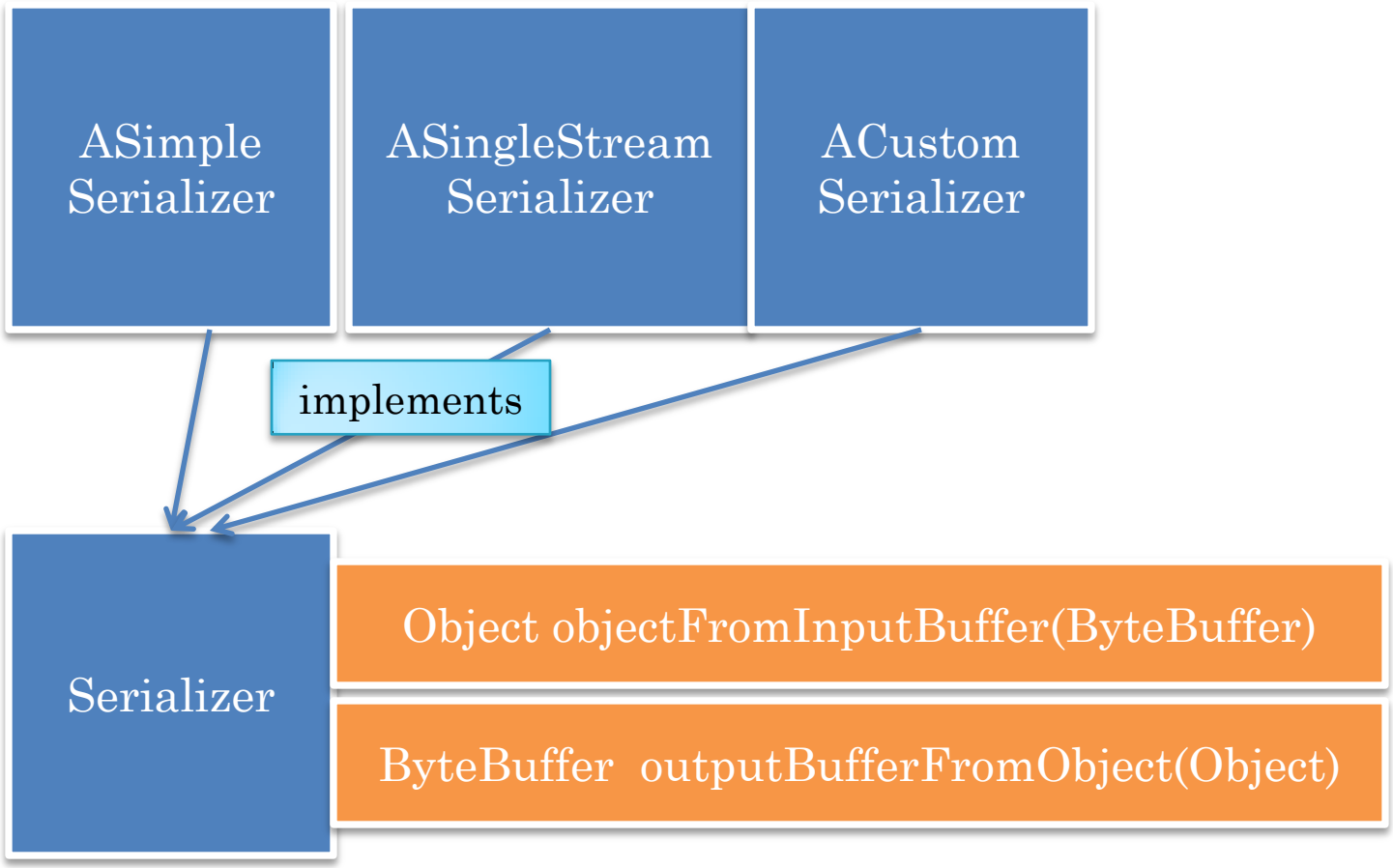
Class vs. Interface/Factory



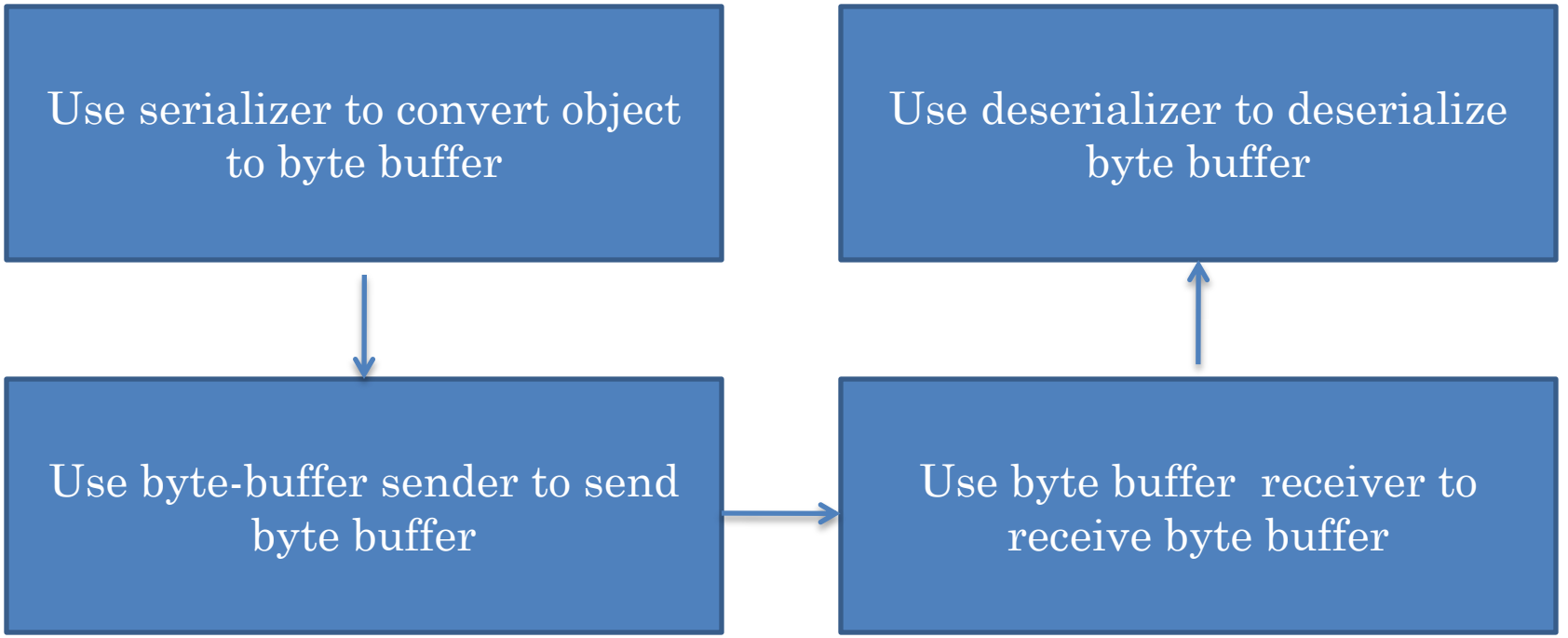
GIPC Serializer Factory and Selector



GIPC Serializer



Send/Receive Object Handle



Serializer may reuse buffer for serialization

Send is not synchronous

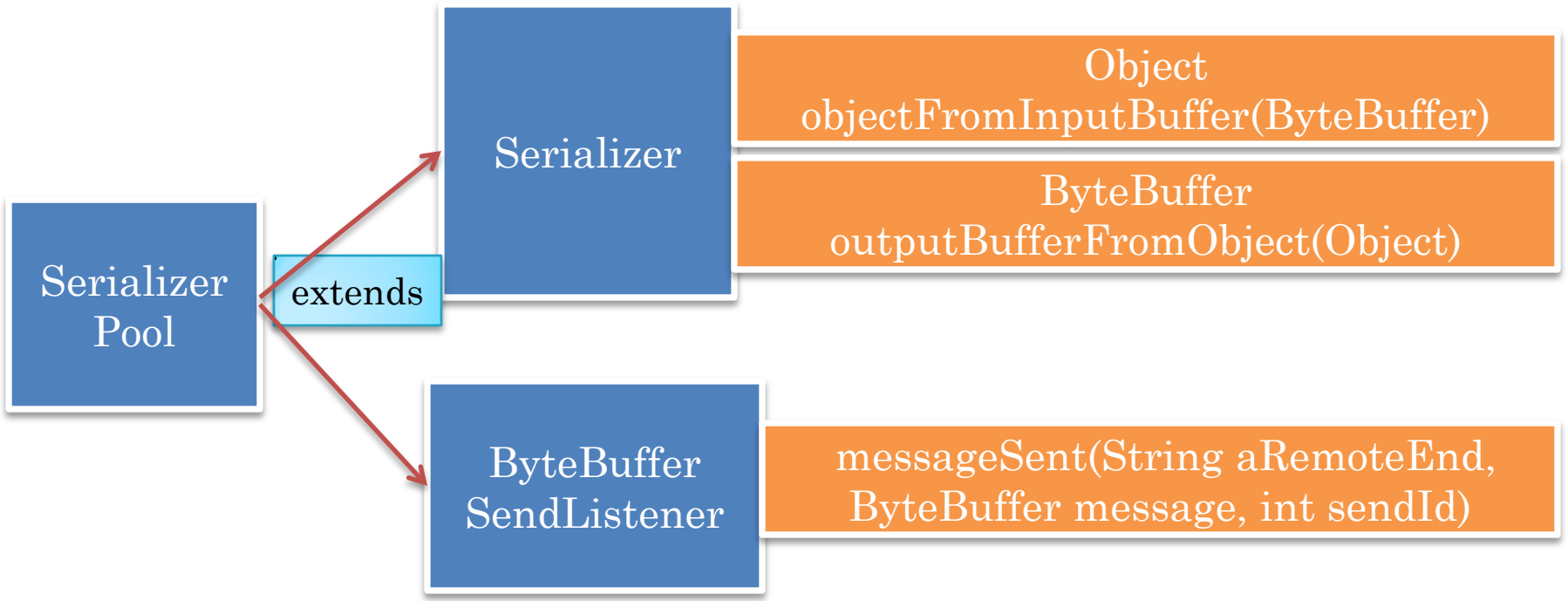
Output Buffer Management

Create separate serializer for each send

Does not allow sharing – defeats single buffer serialization

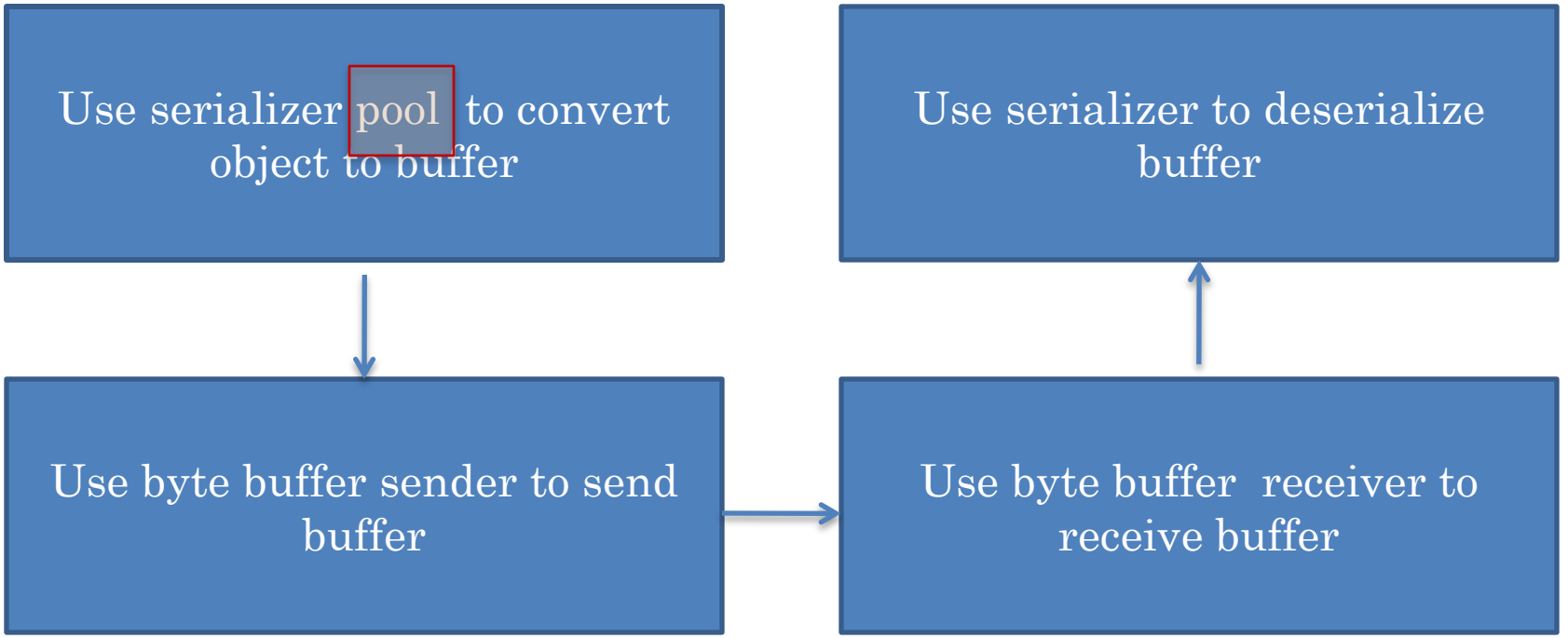
Some serializers create separate buffer for each send, wasted instantiation

Serializer Pool



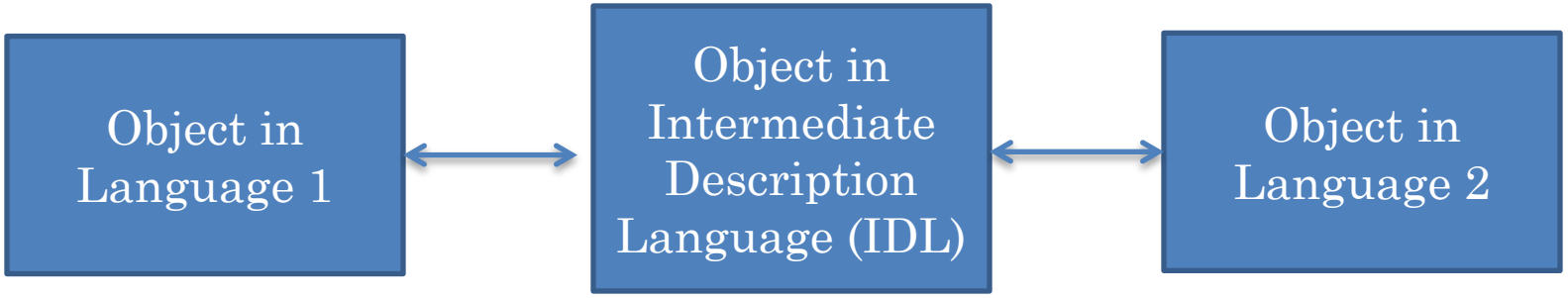
- Serializer pool used instead of serializer
- Next free serializer from pool used to serialize
- Serializer returned to pool when message sent

Serializer Pool Instead of Serializer



Deserializer buffer management?
Byte buffer can be reused by reading thread after object copies made
If deserialization fails, bytebuffer returned, which must be copied by receiver

Supporting Multiple Languages



If languages are OO does IDL for Serialization have to be O-O?

Describing structures, OO would require decoding the structure of the IDL object

Schema and Instance

```
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);}
}
```

```
new ABMISpreadsheet()
```

Schema/ Type Definition

```
typedef struct {
   double height;
   double weight;
} ABMISpreadsheet;
```

Schema/Type Instance

AMBMISpreadsheet myBMI = {1.77, 75}

XML Schema and Instance

```
<ABMISpreadsheet xmlns="urn:xmlns:25hoursaday-com: ABMISpreadsheet " >
<?xml version="1.0" encoding="UTF-8" ?>
 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
            targetNamespace="urn:xmlns:25hoursaday-com:ABMISpreadsheet"
            xmlns:bk="urn:xmlns:25hoursaday-com:ABMISpreadsheet“
   <xs:complexType name=“ABMISpreadsheet">
      <xs:sequence>
         <xs:element name=“height" type="xs:number" /> <xs:element />
         <xs:element name=“weight" type="xs:number" /> <xs:element />
      </xs:sequence>
   </xs:complexType>
 </xs:schema>
```

Common schema for all Beans?

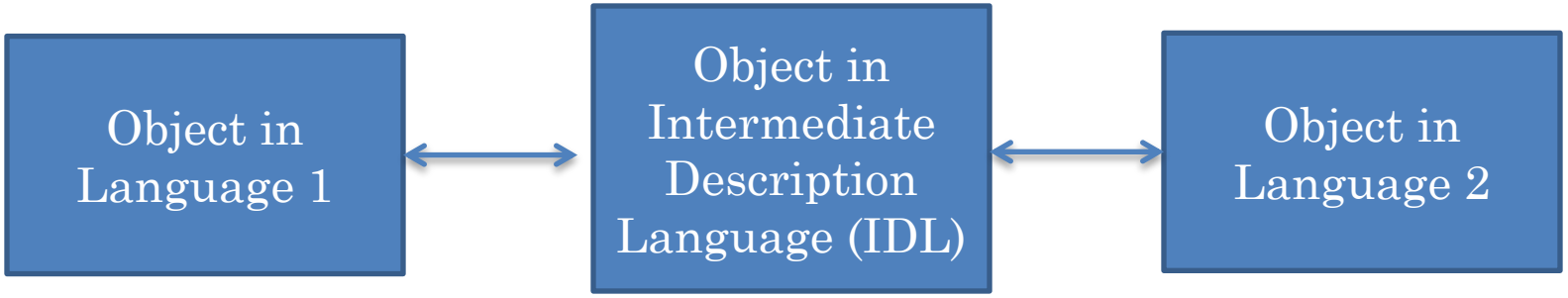
```
<ABMISpreadsheet xmlns=
      "urn:xmlns:25hoursaday-com:ABMISpreadsheet" >
 < ABMISpreadsheet >
       <height>1.77</height>
       <weight>75.0</weight>
 </ABMISpreadsheet>
```

Representation based on Pattern-Specific, Class Independent Schema

```
<Object xmlns=
      "urn:xmlns:25hoursaday-com:Object" >
 < Object >
      <Property><Key> height</Key> <number>1.77</number></Property>
       <Property><Key>weight</Key> <number>75.0</number></Property>
</Object>
```

More verbose, structural rather than type equivalence

Automatic vs. Manual Translation



- Automatic translation for Beans
  - Schema generated automatically
  - Converter generated automatically
- Constrains patterns programmers use
- No need to create and update schema
- No need to translate between language and schema representation

JSON: Java Script Based IDL

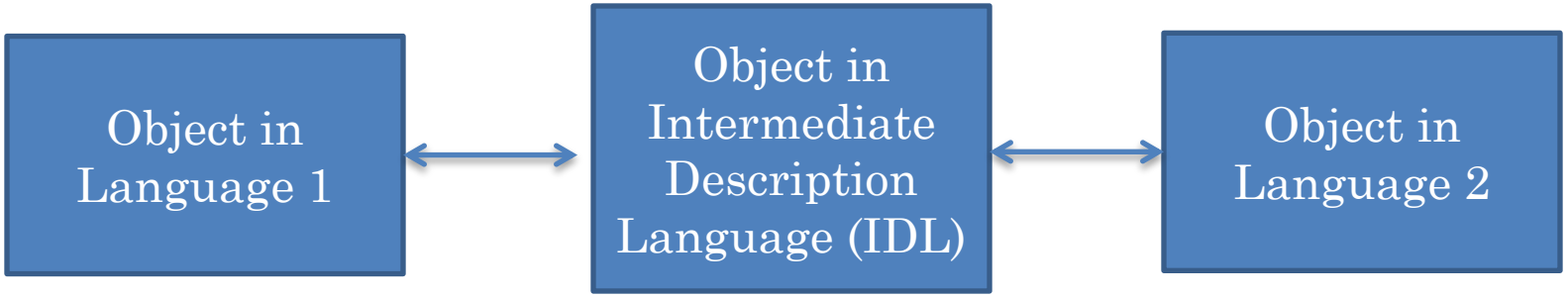
Schema/ Type Definition

```
{ "name":“ABMISpreadsheet", "properties":
    { “height": { "type":“number", "required":true },
   { “weight": { "type":“number", "required":true }
}
```

Schema/Type Instance

```
{ “height": 1.77, “weight" : 75.0 }
```

Automatic vs. Manual Translation (review)



C, XML, JSON, ….

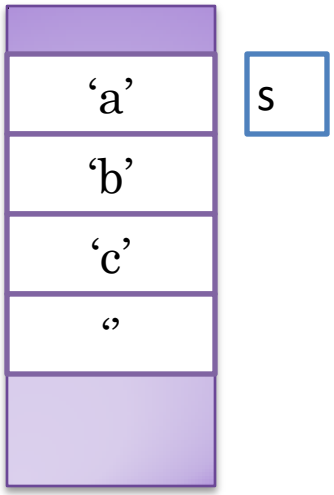
Need to create and update schema

Need to translate between language and schema representation

Automatic translation for language independent patterns such as Bean

Serialization and Strong Typing

```
char *s;
```

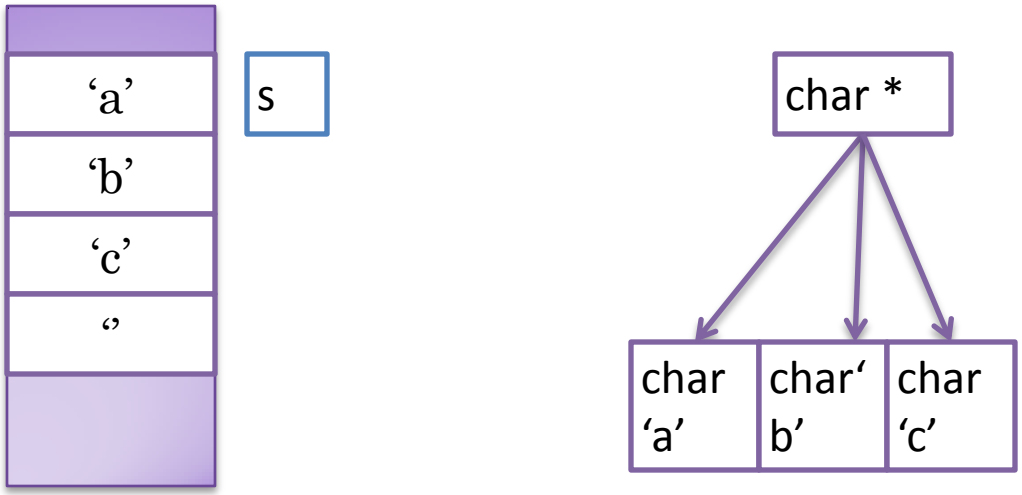


Physical structure communicated and reconstructed?

Array Interpretation

char *s;

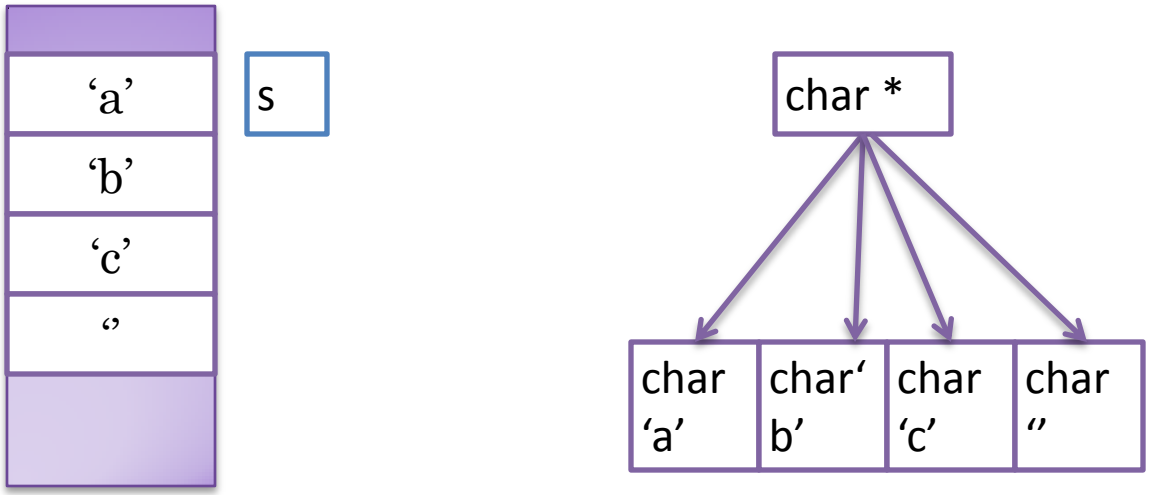
s = {‘a’, ‘b’, ‘c’}



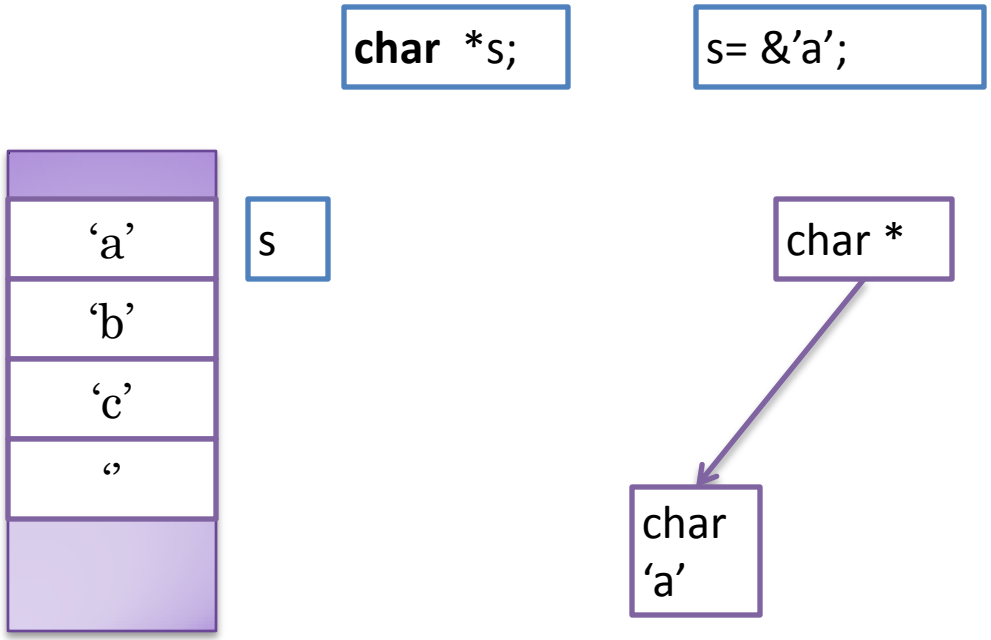
String Interpretation

char *s;

s = “abc”;



Char Pointer Interpretation

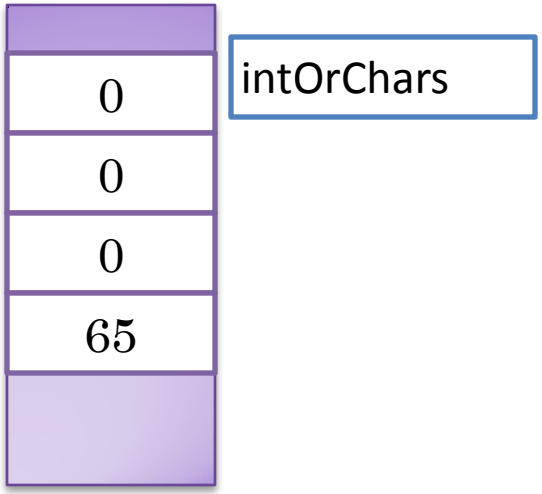


UNION

```
union AnIntOrChars{
        int s;
        char c[4];
}  intOrChars
```

Like subclasses, union describes alternatives

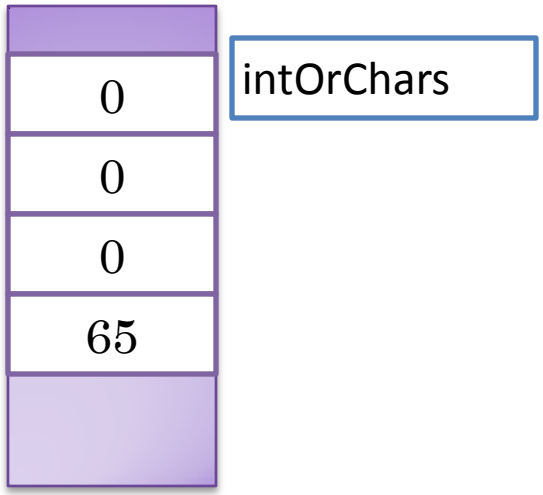
Each alternative is a way of interpreting memory

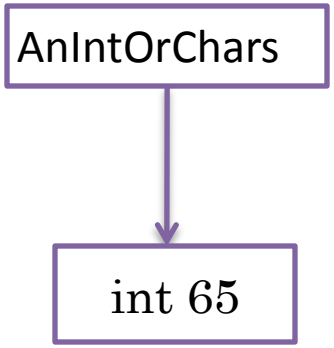


Int Interpretation

```
union AnIntOrChars{
        int s;
        char c[4];
}  intOrChars
```

intOrChars.s = 65

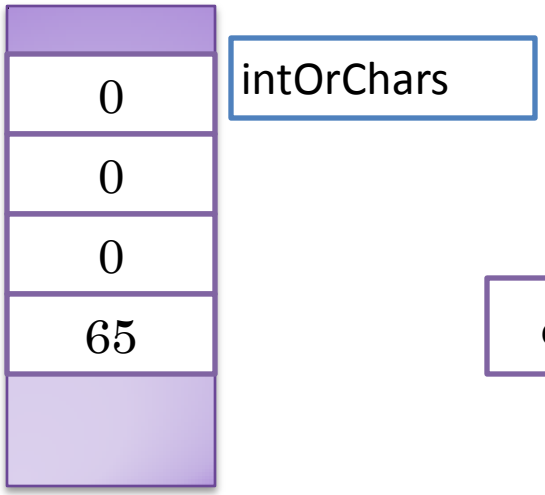


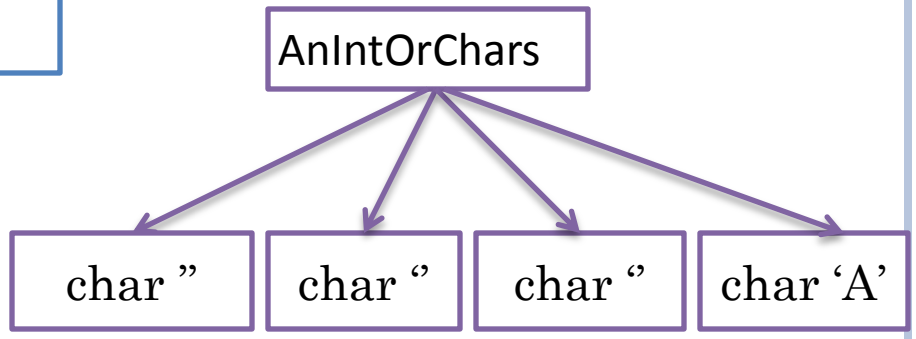


Char Array Interpretation

```
union AnIntOrChars{
        int s;
        char c[4];
}  intOrChars
```

intOrChars.s [0]= ‘’





Strong Typing and Serialization

In weakly typed languages type of a variable is ambiguous

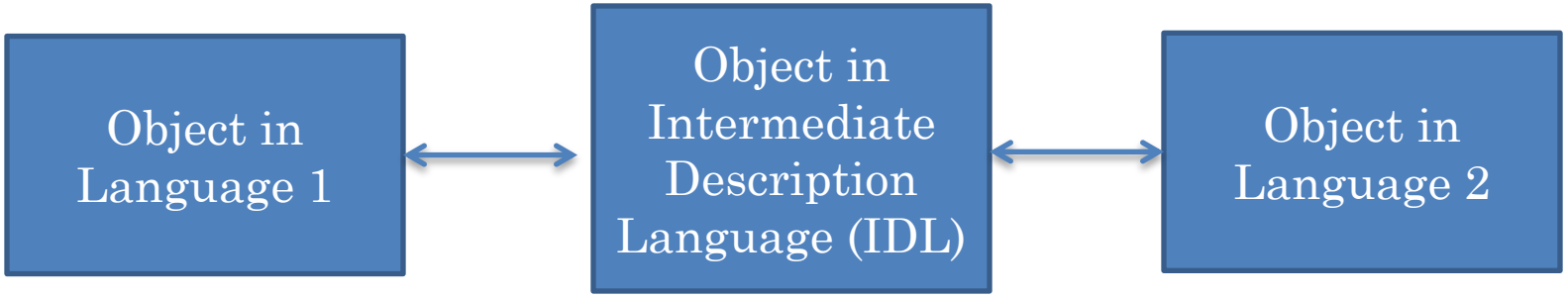
Programmers disambiguate based on their application-specific knowledge

Serialization depends on type of variable: same memory can be serialized differently based on its type

Serialization requires unambiguous typing

Give up on C?

Standard Solution: IDL



Need to create and update schema

Need to translate between language and schema representation

SUITE DIRECTIVE-BASED SERIALIZATION

```
char  *s;
```

Interpret as string (default)

```
/*oc
DiscriminatedUnion  AnIntOrCharsStruct
*/
struct AnIntOrCharsStruct{
      enum (isShort, isChars} tag;
      union {
        int s;
        char c[4];
    }  intOrChars;
}   intOrCharsStruct;
```

Use enum field storing one of N choices preceding union with N alternatives as discriminant, with choice I implying alternative I.

Special comments used as directives for serialization

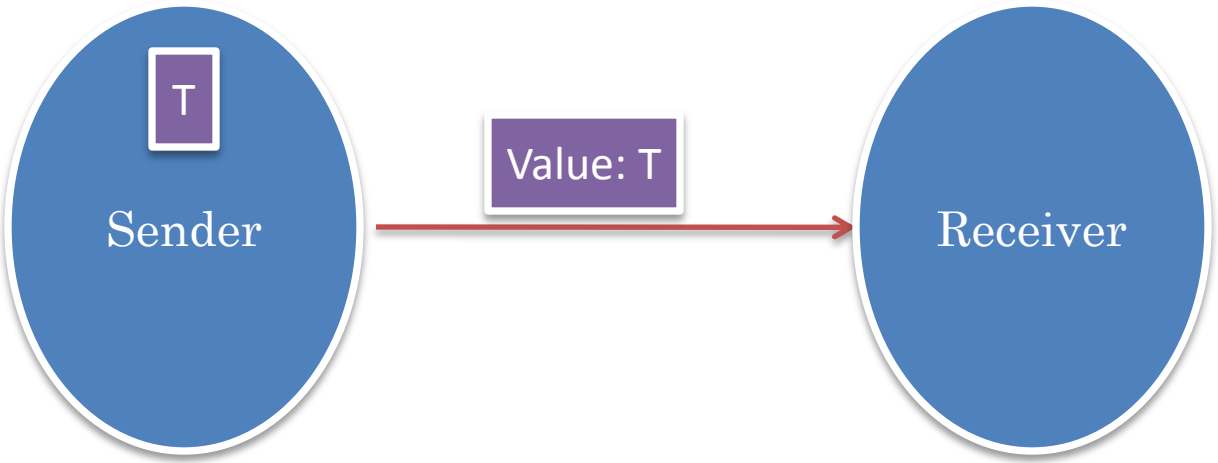
Serialization directive is a general idea supported in later systems through other language-supported constructs

Strong Typing and Serialization

If the type of an object is unambiguous, is it serializable?

Conceptual and implementation issues

Unknown Type



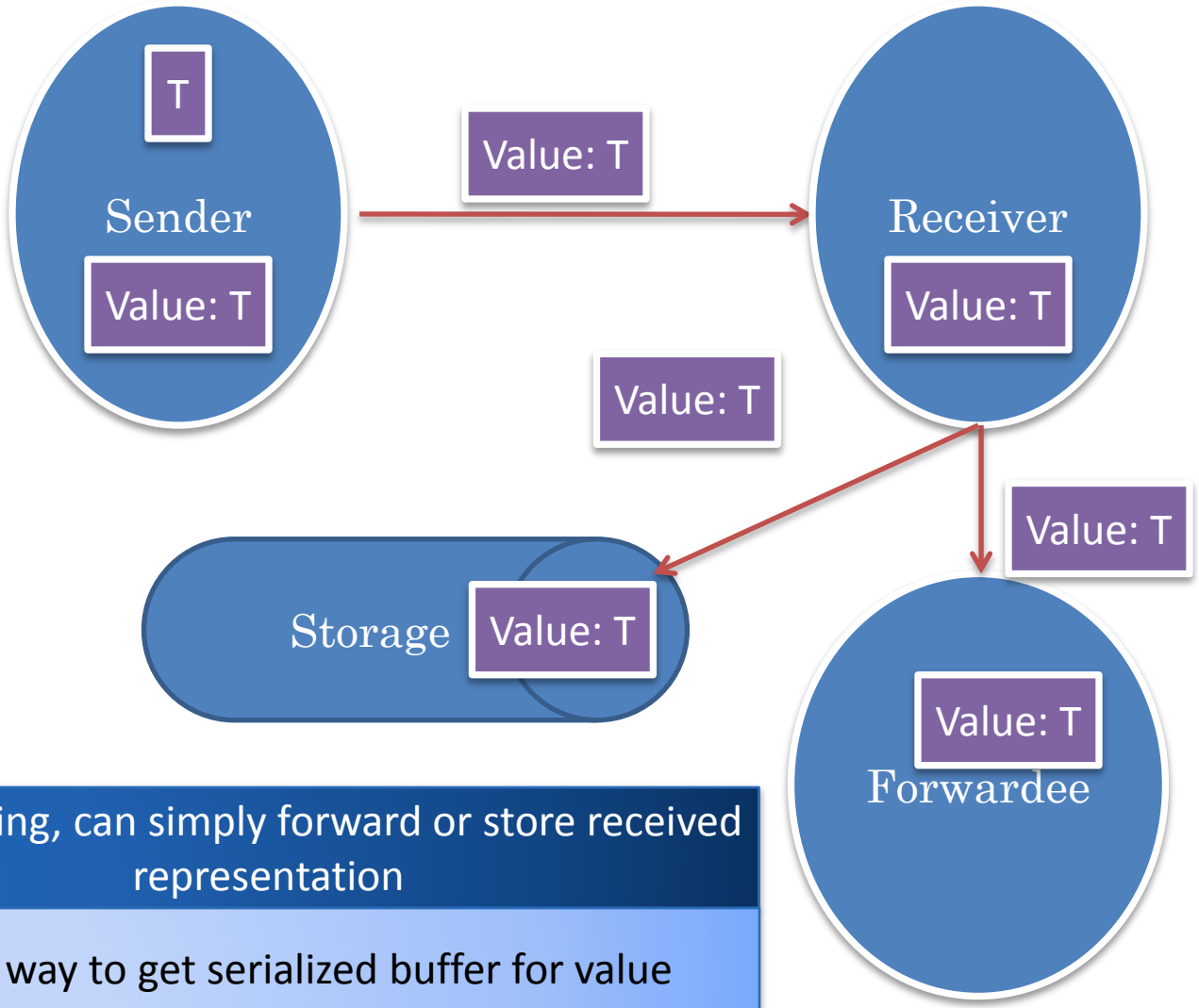
Receiver does not know about the type of sent values

Application may be type-independent (Logger, Monitor, Registry, Realeyer)

Type may have evolved

How to support receipt of value of unknown type

No Handling of Received Unknown Value



If no Handling, can simply forward or store received representation

Need a way to get serialized buffer for value

Not easy in Java

What Handling to do With Unknown Type

Can invoke some method of known super type

Applet, Thread start method

Relies on dynamic dispatch in virtual methods (Java)

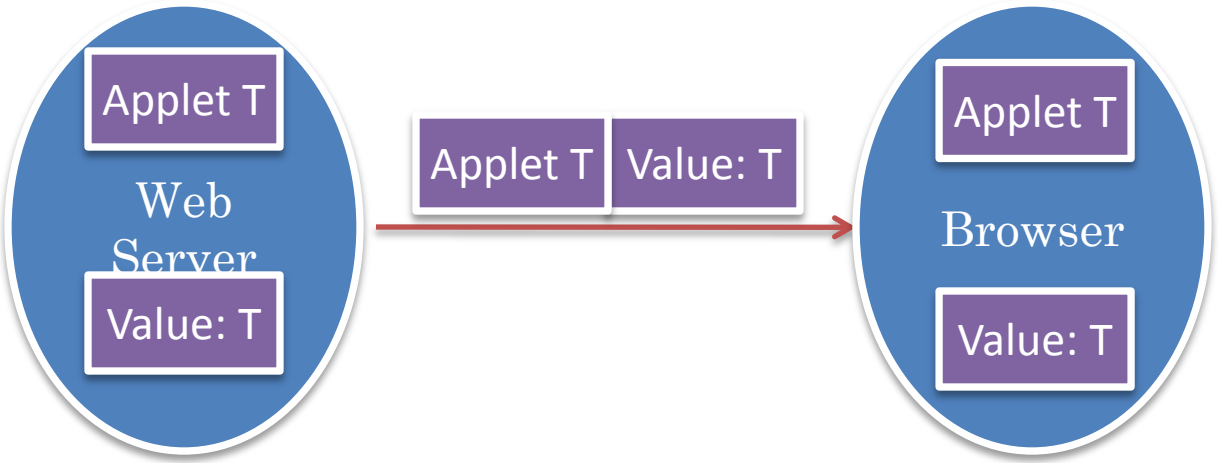
Can invoke some method of known signature using reflection

Invoke main method, getter, setter

Handle method invocation request from requester that knows type and possibly sent the object

A client loads ABMISpreadsheet and same and other clients make remote calls in it

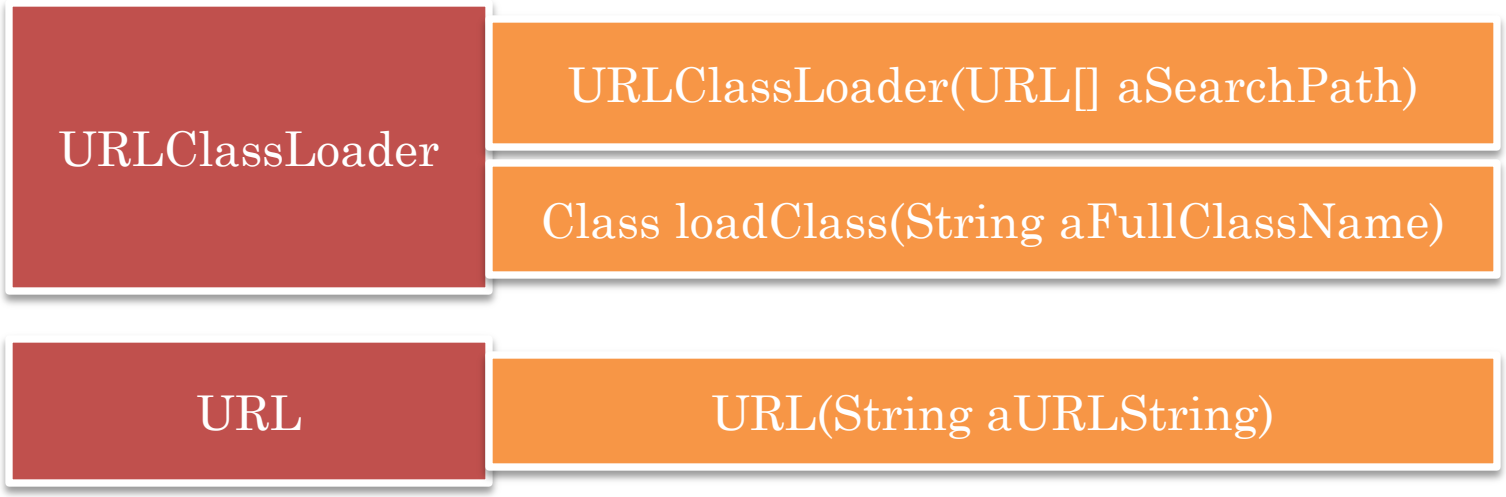
Message Handling: Same Language



Applets are loaded dynamically

There is a way in Java to load classes from the network

Dynamic Loading

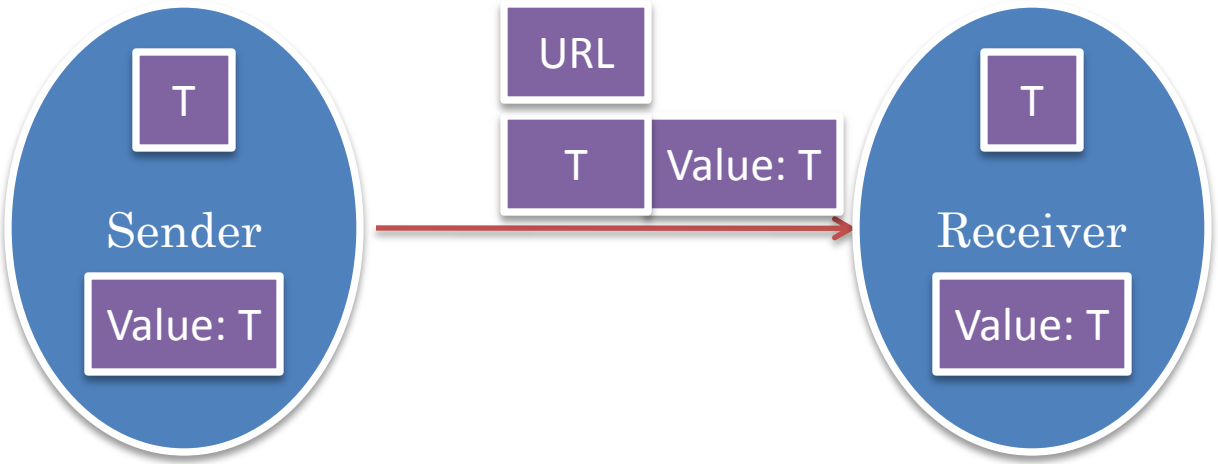


Loading Remote Class

```
public static Class loadClass(
    String aURLOfClassesFolder,String aFullClassName) {
  try {
    URL[] urlSearchPath = { new URL(aURLOfClassesFolder) };
    URLClassLoader loader =
          new URLClassLoader(urlSearchPath);
    return loader.loadClass(aFullClassName);
  } catch (MalformedURLException e) {
      e.printStackTrace();
      return null;
   } catch (ClassNotFoundException e) {
      e.printStackTrace();
      return null;
   }
}
```

loadClass(“ www.cs.unc.edu/~dewan/comp734/gipc/classes ” , “bmi.ABMISpreadsheet”)

Message Handling: Same Language



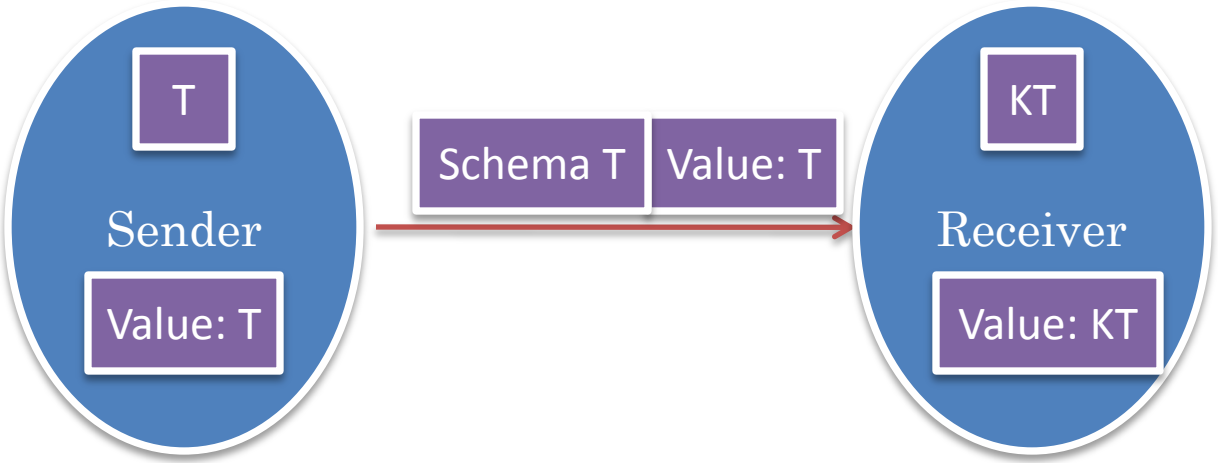
In Java URL for location to load class can be sent to receiver

URL returned by RMIClassLoader.getClassAnnotation (class)

Must return non null in Web Servers

It would be nice if classes were communicatedwithout requiring URLs

Message Handling: Different Language



Could convert to special data structure of known type to which graph is translated using sent schema

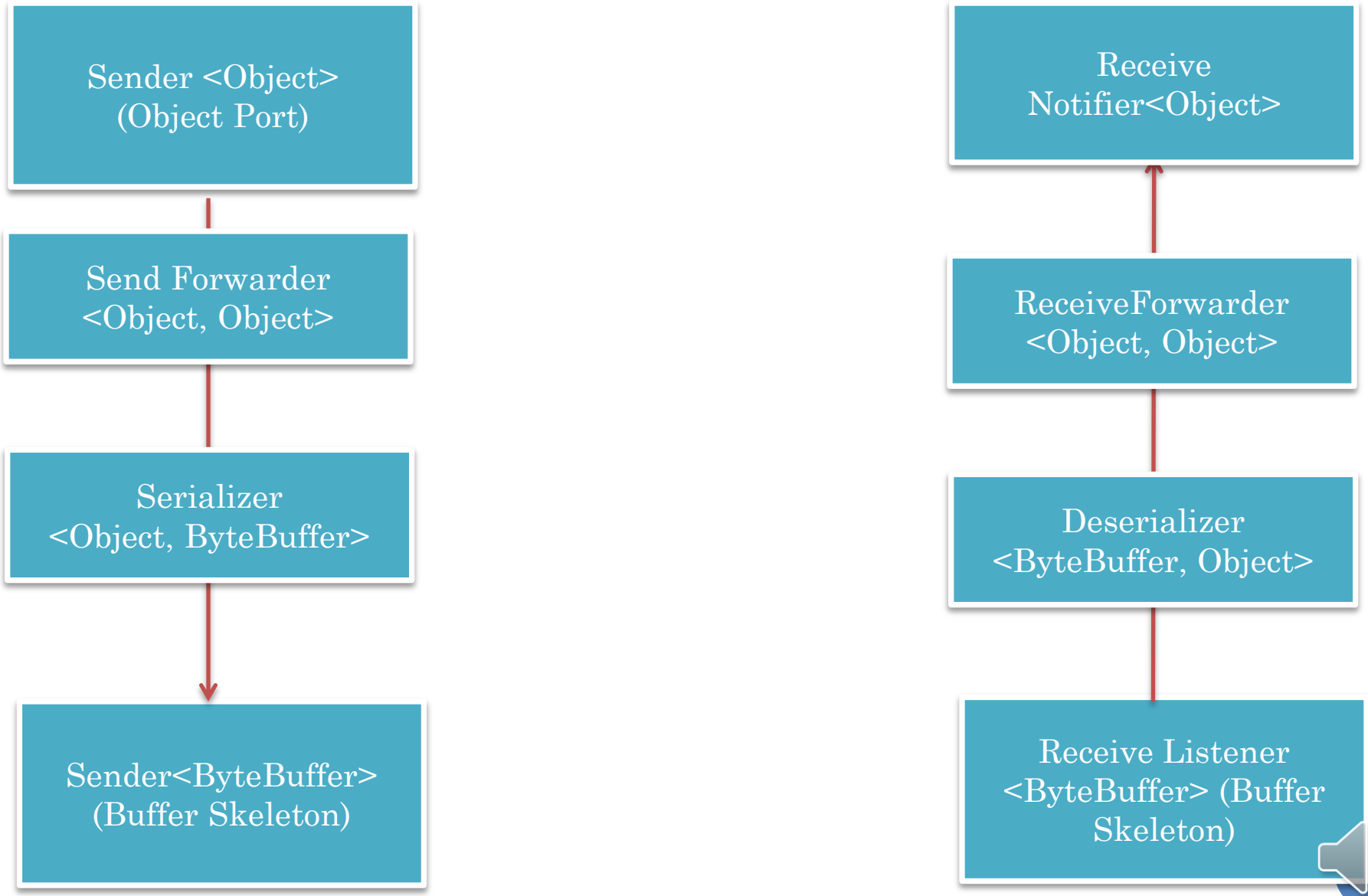
e.g. XML Node, Generic Dynamic Bean with property names as arguments to setters and getters

e.g. New ABMISpreadsheet with setters/getters

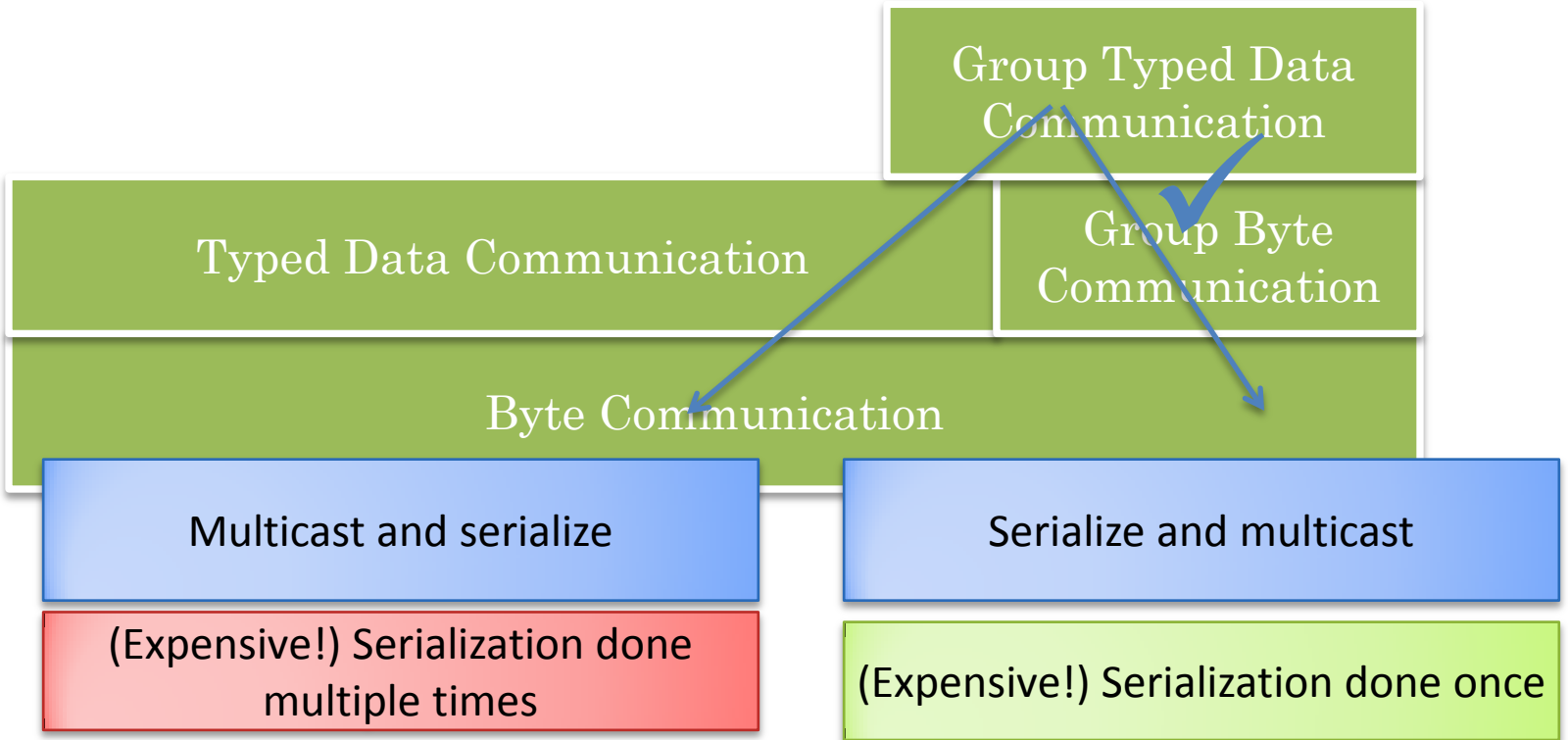
Would lose semantics such as getBMI()

Cannot do app-specific computations unless we have an O-O IDL

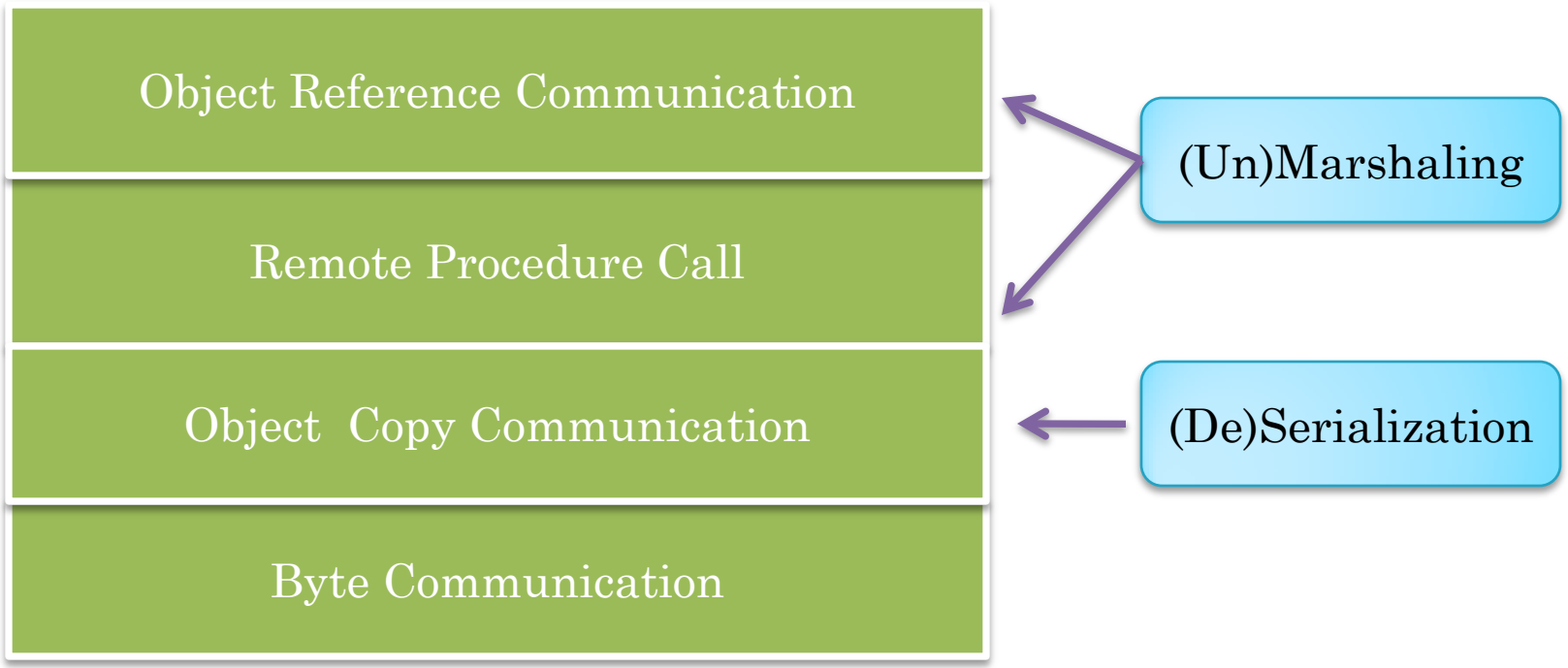
Serializer and Communication



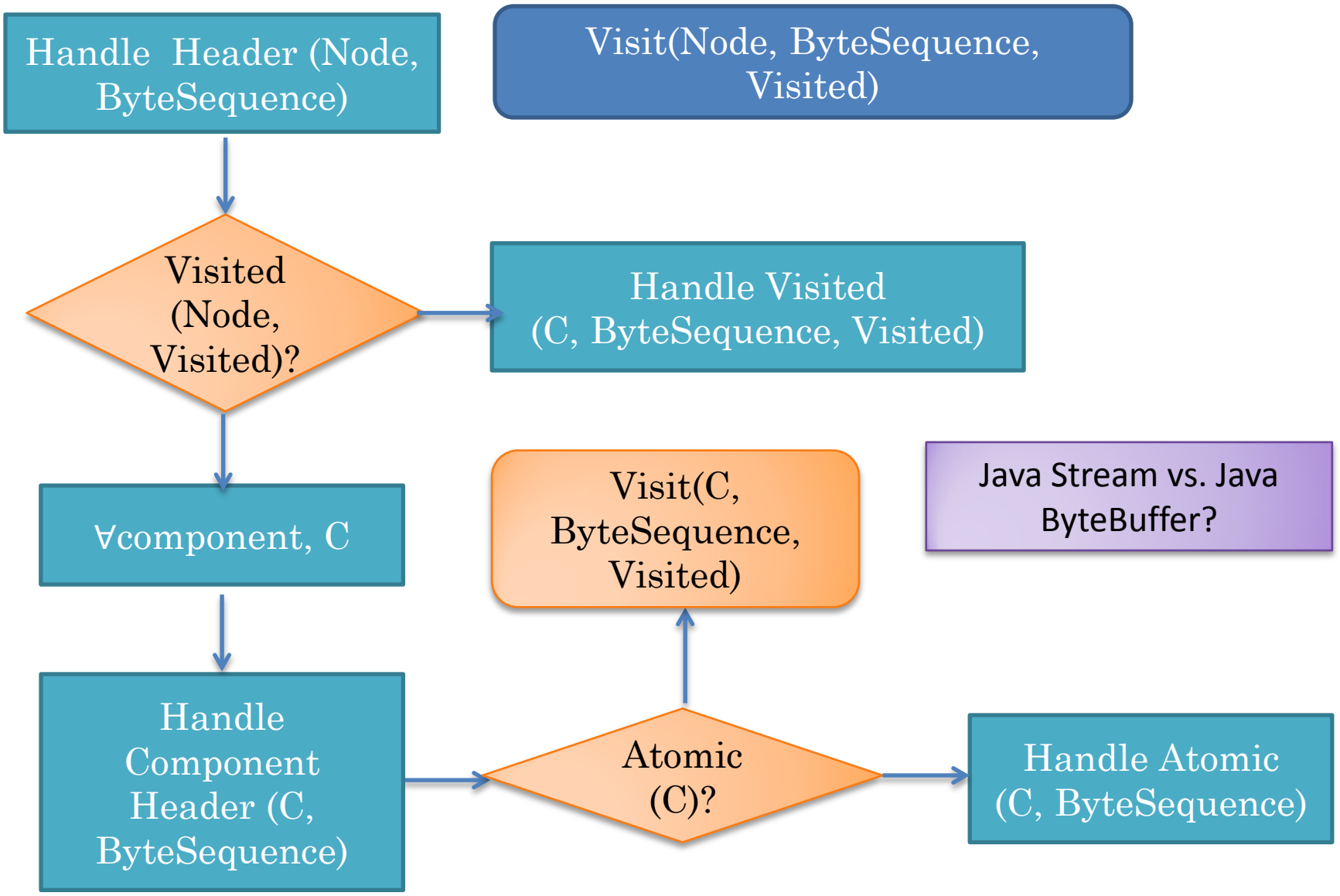
Serialize and Multicast Alternatives



Object Communication vs. RPC, Serialization vs. Marshaling of Values



Byte Buffer?


Handle Header (Node, ByteSequence)
Visit(Node, ByteSequence, Visited)
Visited (Node, Visited)?
Handle Visited (C, ByteSequence, Visited)
∀component, C
Visit(C, ByteSequence, Visited)
Java Stream vs. Java ByteBuffer?
Handle Component Header (C, ByteSequence)
Atomic (C)?
Handle Atomic (C, ByteSequence)

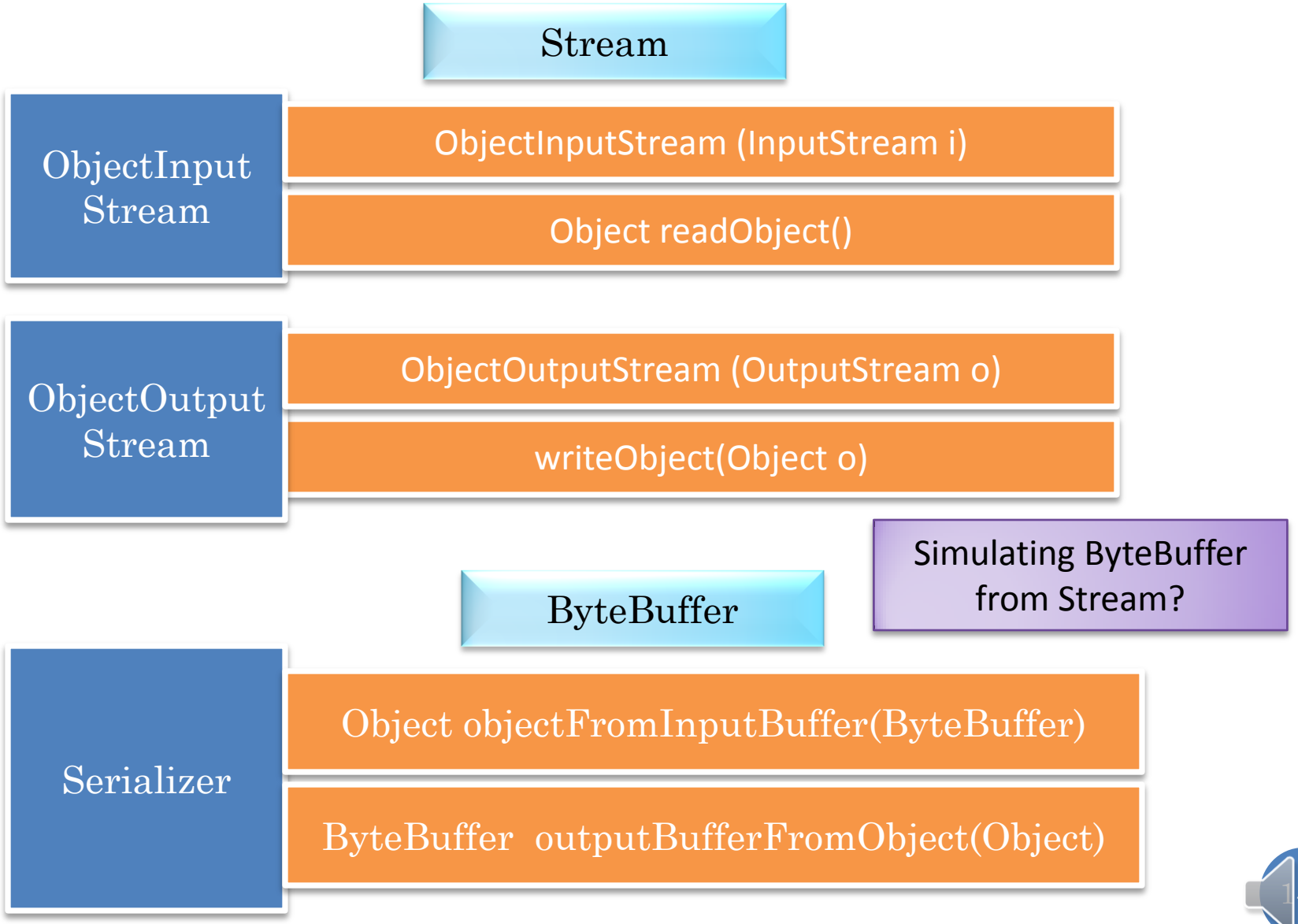
Stream vs. ByteBuffer

In stream based serialization, conceptually one stream for all communication between two parties

In byte buffer based serialization, conceptually each byte buffer is independent of the previous ones sent from that destination

Simulating ByteBuffer from Stream?

Stream vs. ByteBuffer API Examples



From Stream to Byte Buffer

Custom Stream to extract filled bytes

New streams created and flushed on each serialization

```
public ByteBuffer getOutputBufferFromObjects(Object[] objects) {
  try {
    SeekableByteArrayOutputStream byteArrayOutputStream =
       new SeekableByteArrayOutputStream();
    ObjectOutputStream objectOutputStream =
       new ObjectOutputStream(byteArrayOutputStream);
    for (int i = 0; i < objects.length; i++) {
      objectOutputStream.writeObject(objects[i]);
    }
    objectOutputStream.flush();
    ByteBuffer retVal = ByteBuffer.wrap(byteArrayOutputStream
                        .getBuffer(), 0, byteArrayOutputStream.size());
    return retVal;
  } catch (Exception e) {
    e.printStackTrace();
    return null;
  }
}
```

Extracting bytes

DeSerializing

```
public List<Object> objectsFromInp                    uffer inputBuffer) {
  try {
    ObjectInputStream objectInputSt      m = new ObjectInputStream(
      new ByteArrayInputStream(inputBuffer.array(),
           inputBuffer.position(), inputBuffer.remaining()));
    List<Object> retVal = new ArrayList();
    while (true) {
      try {
        Object readObject = objectInputStream.readObject();
        retVal.add(readObject);
      } catch (EOFException eof) {break;}
    }
    return retVal;
  }catch (Exception e) {
    e.printStackTrace();return null;
  }
}
```

New streams created and flushed on each serialization

Cost of Serializing

- Each serialization creates a new byte array and streams around it
- Could reuse the same stream for multiple serializations
- Java Serializer adds 4 synchronization bytes at start and Deserializer removes them
- Plus Java does inter message compression
- Scheme reduces correctness and generality (same object being sent multiple times and multicast)
- ObjectOutputStream reset does not seem to work