Ryan Sciampacone – IBM Managed Runtime Architect
29th July 2013

Packed Objects

Important Disclaimers

THE INFORMATION CONTAINED IN THIS PRESENTATION IS PROVIDED FOR INFORMATIONAL PURPOSES ONLY.

WHILST EFFORTS WERE MADE TO VERIFY THE COMPLETENESS AND ACCURACY OF THE INFORMATION CONTAINED IN THIS PRESENTATION, IT IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED.

ALL PERFORMANCE DATA INCLUDED IN THIS PRESENTATION HAVE BEEN GATHERED IN A CONTROLLED ENVIRONMENT. YOUR OWN TEST RESULTS MAY VARY BASED ON HARDWARE, SOFTWARE OR INFRASTRUCTURE DIFFERENCES.

ALL DATA INCLUDED IN THIS PRESENTATION ARE MEANT TO BE USED ONLY AS A GUIDE.

IN ADDITION, THE INFORMATION CONTAINED IN THIS PRESENTATION IS BASED ON IBM’S CURRENT PRODUCT PLANS AND STRATEGY, WHICH ARE SUBJECT TO CHANGE BY IBM, WITHOUT NOTICE.

IBM AND ITS AFFILIATED COMPANIES SHALL NOT BE RESPONSIBLE FOR ANY DAMAGES ARISING OUT OF THE USE OF, OR OTHERWISE RELATED TO, THIS PRESENTATION OR ANY OTHER DOCUMENTATION.

NOTHING CONTAINED IN THIS PRESENTATION IS INTENDED TO, OR SHALL HAVE THE EFFECT OF:

- CREATING ANY WARRANT OR REPRESENTATION FROM IBM, ITS AFFILIATED COMPANIES OR ITS OR THEIR SUPPLIERS AND/OR LICENSORS

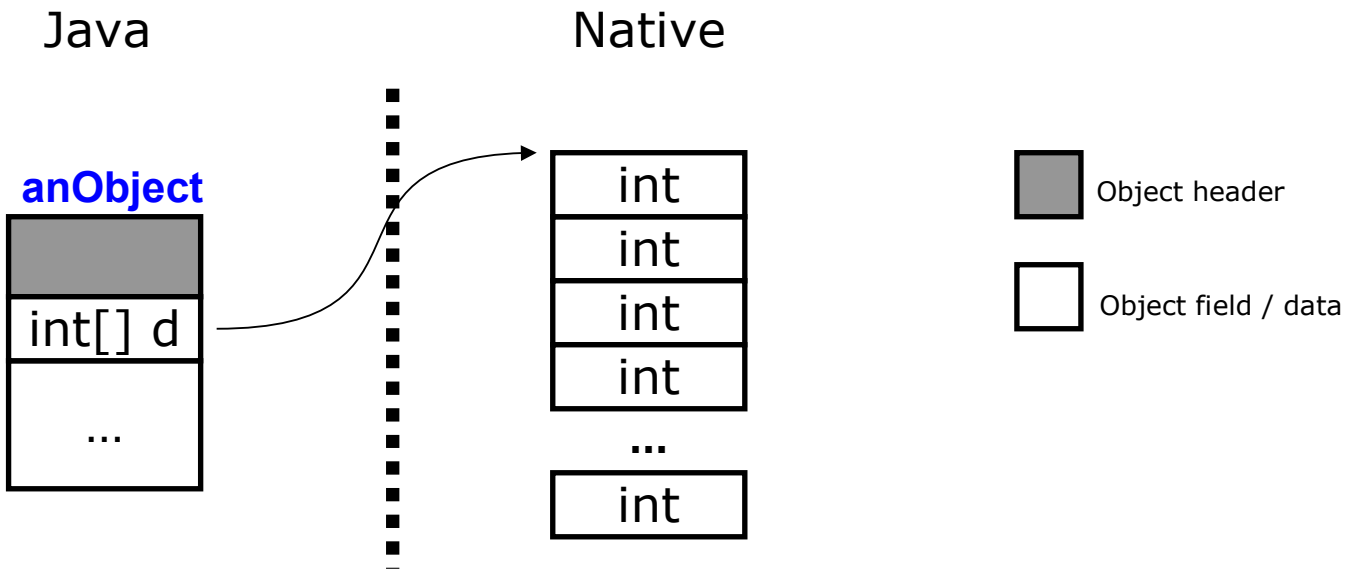
Problem

Problem? What problem?

- JNI just isn't a great way to marshal data
- Locality in Java can matter (e.g., JEP 142)
- Existing native and data placement stories aren't very good
- In many cases, legacy systems exist – the interop is just terrible

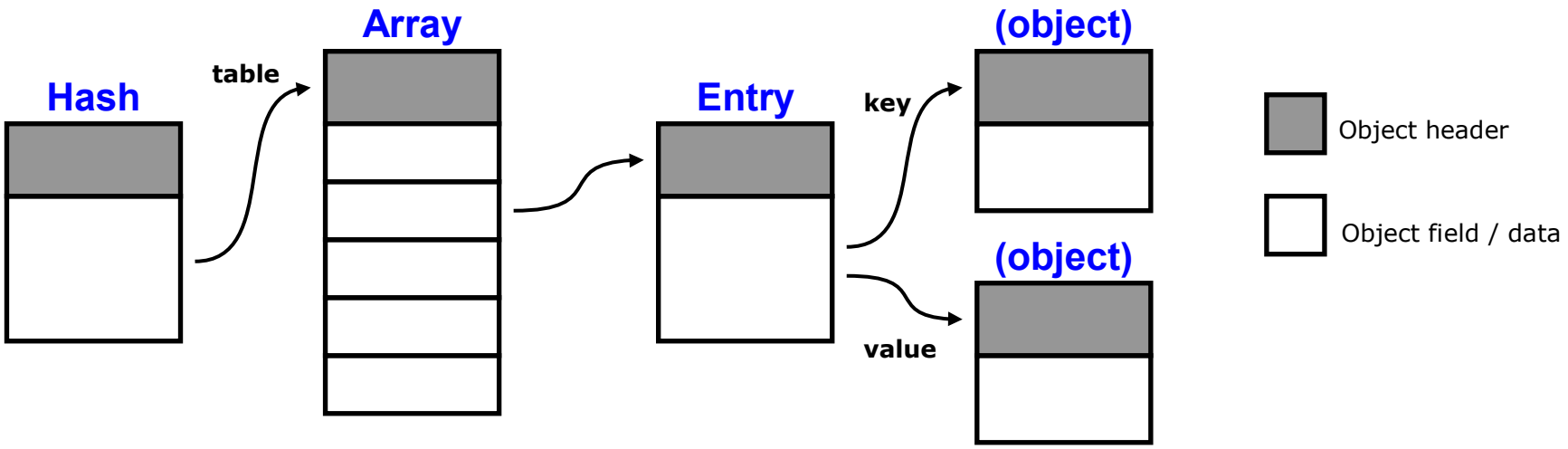
- So we want something that integrates well with the Java language and helps us…

Native Access

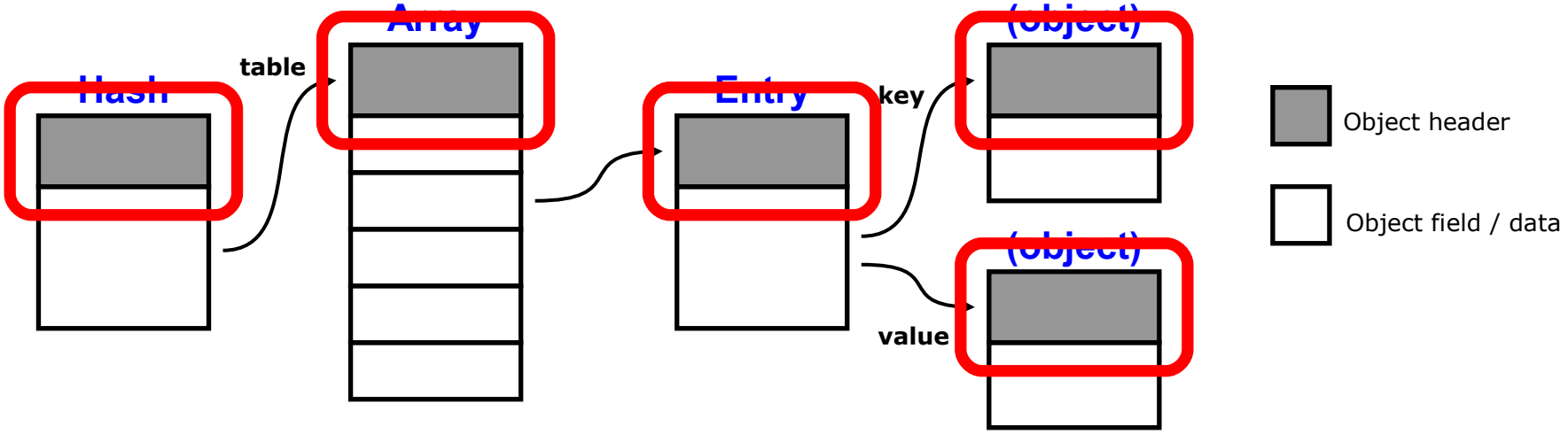


Fighting the Java/Native interface

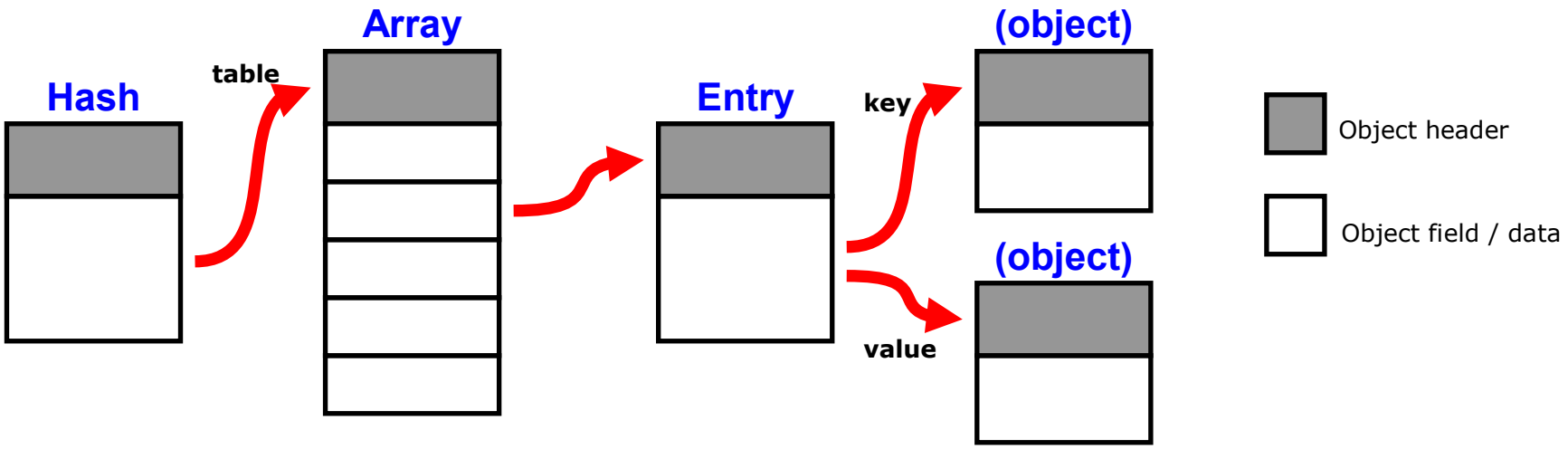
Everything is an Object



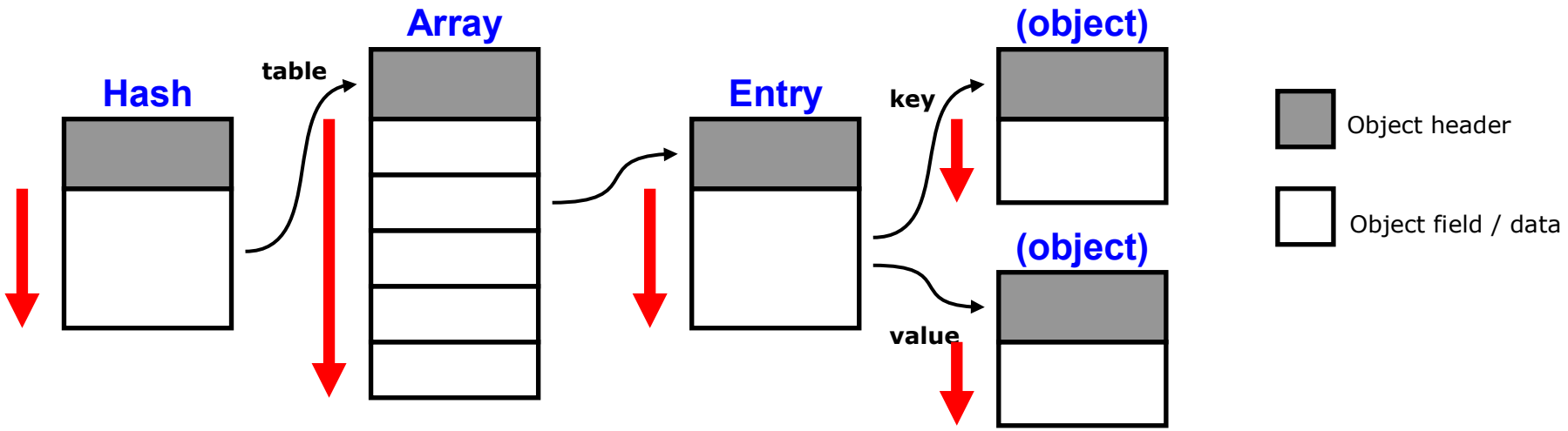
Everything is an Object



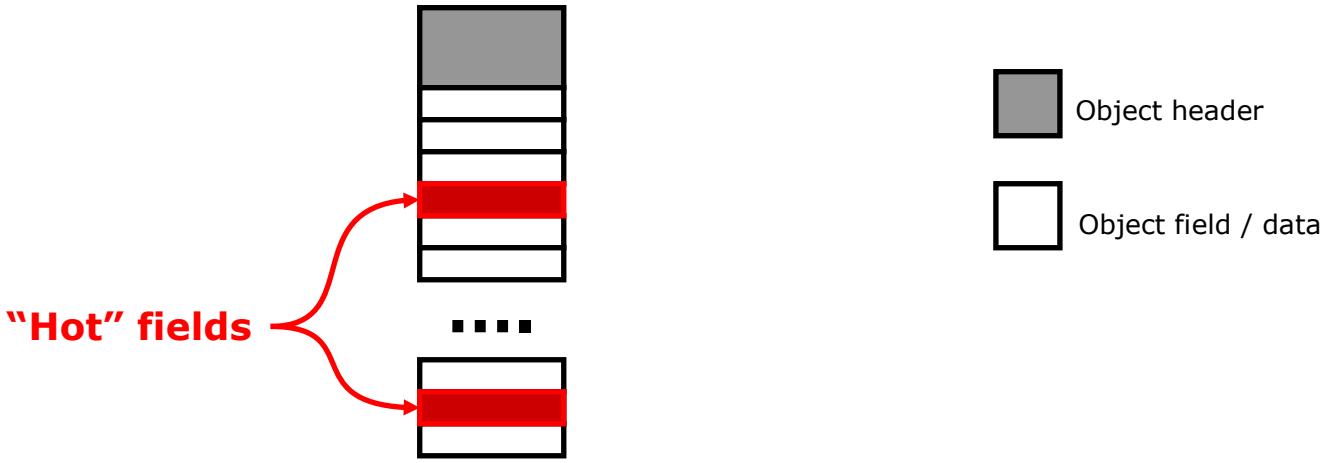
Everything is an Object



Everything is an Object



Object Internals



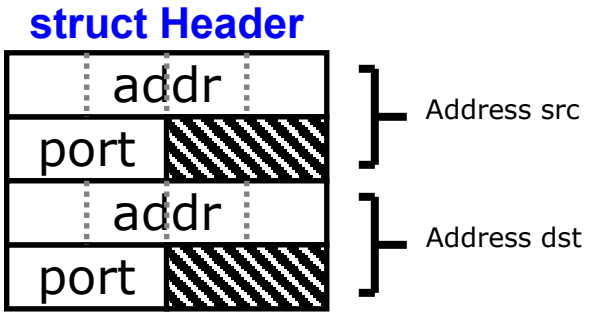
- Field ordering has performance implications
- JVM can potentially reorder your fields for you

Establishing Goals

- On heap / off heap seamless referencing of data
- Ability to do away with headers
- Ability to bring related objects close together

- This actually sounds a lot like C structure types

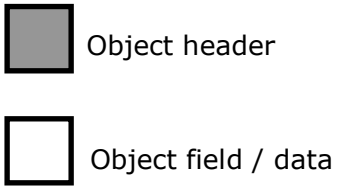
```
struct Address {
  char[4] addr;
  short port;
}
struct Header {
  struct Address src;
  struct Address dst;
}
```

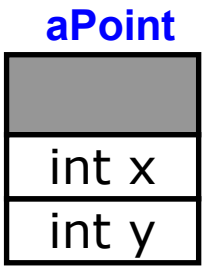


- Packed Objects!

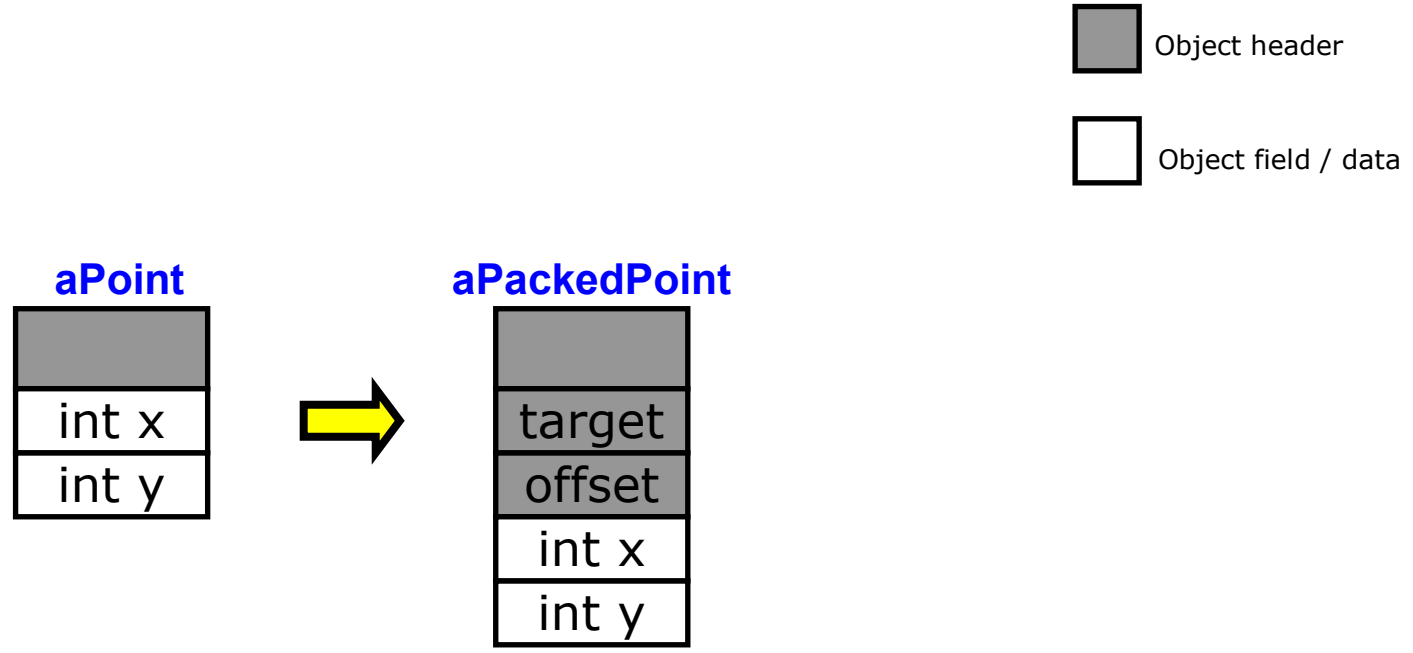
Basics

Packed Objects: Under the covers

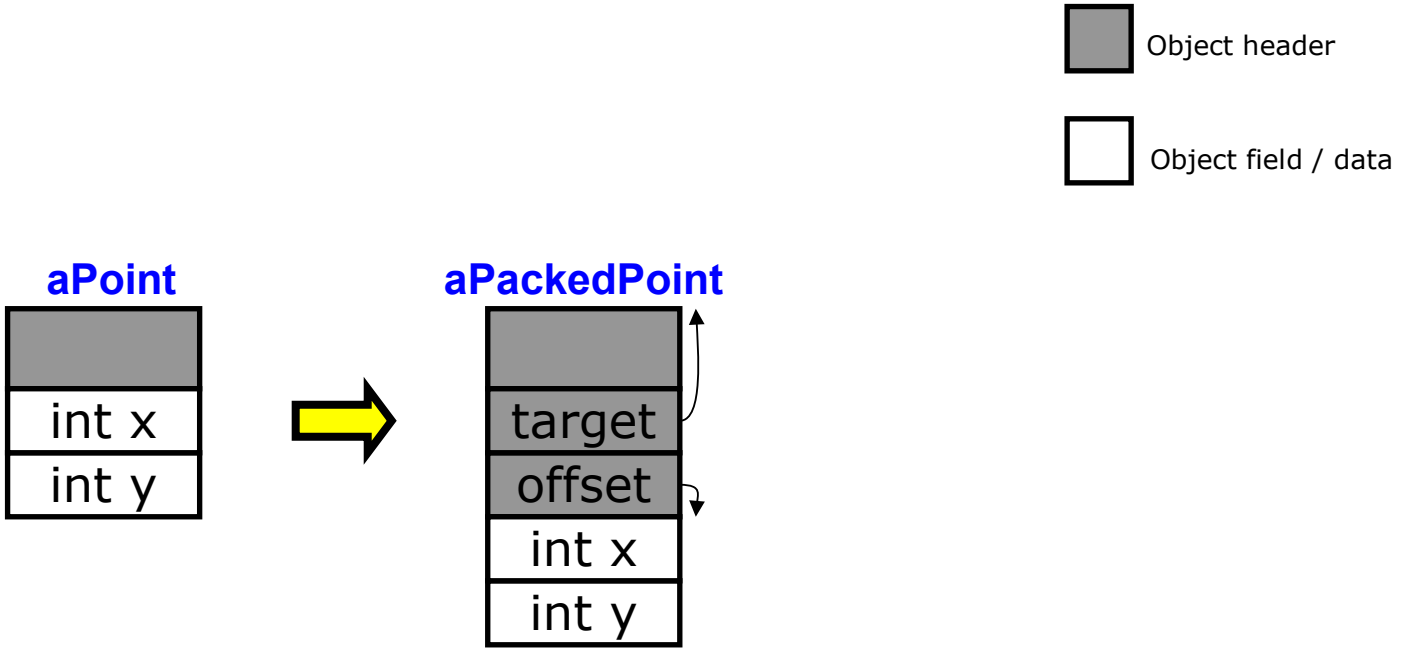




Packed Objects: Under the covers

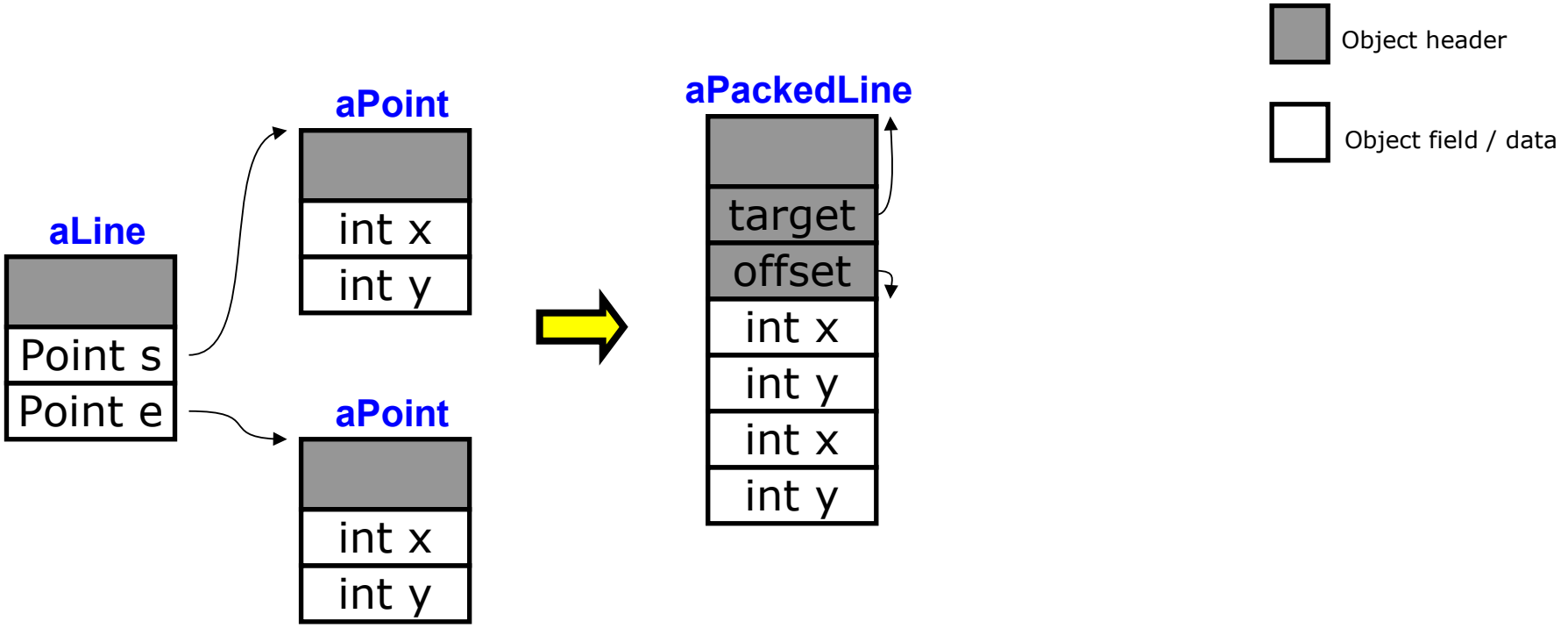


Packed Objects: Under the covers

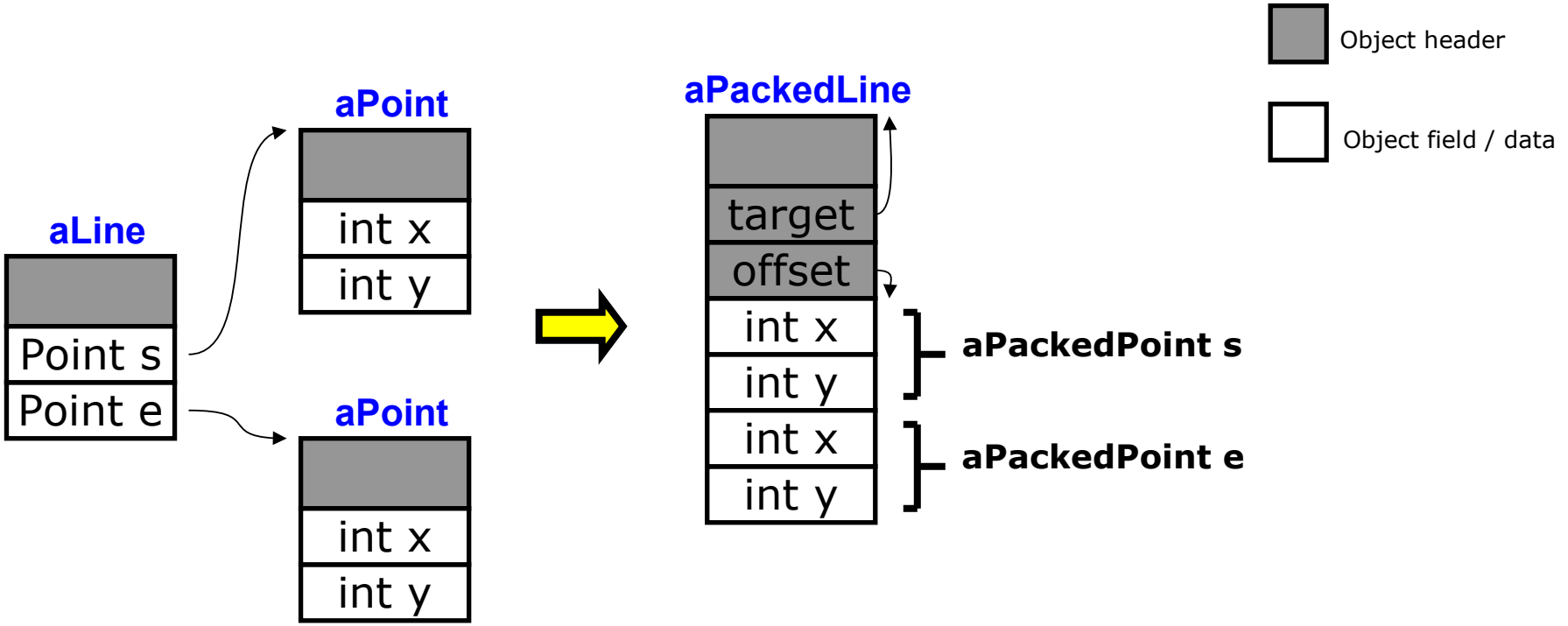


Packed Objects: In Practice

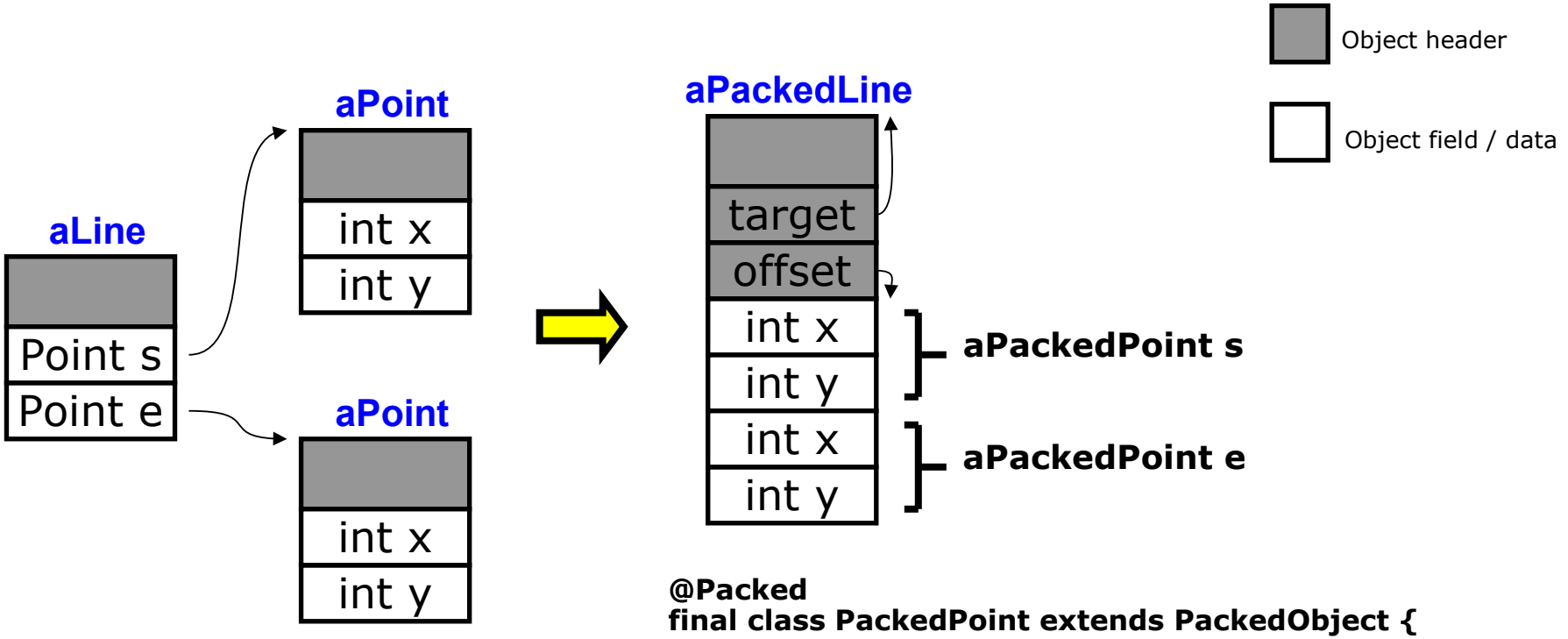
Packed Objects: In Practice



Packed Objects: In Practice

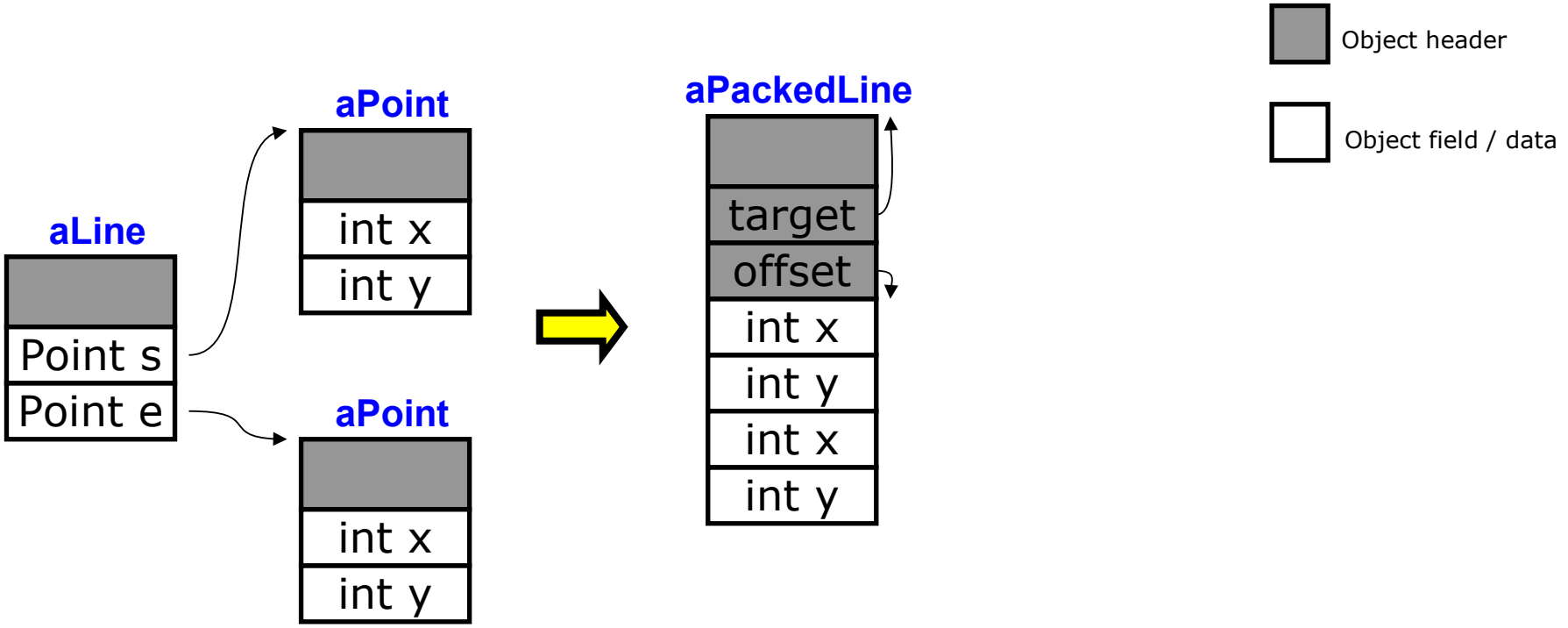


Packed Objects: In Practice

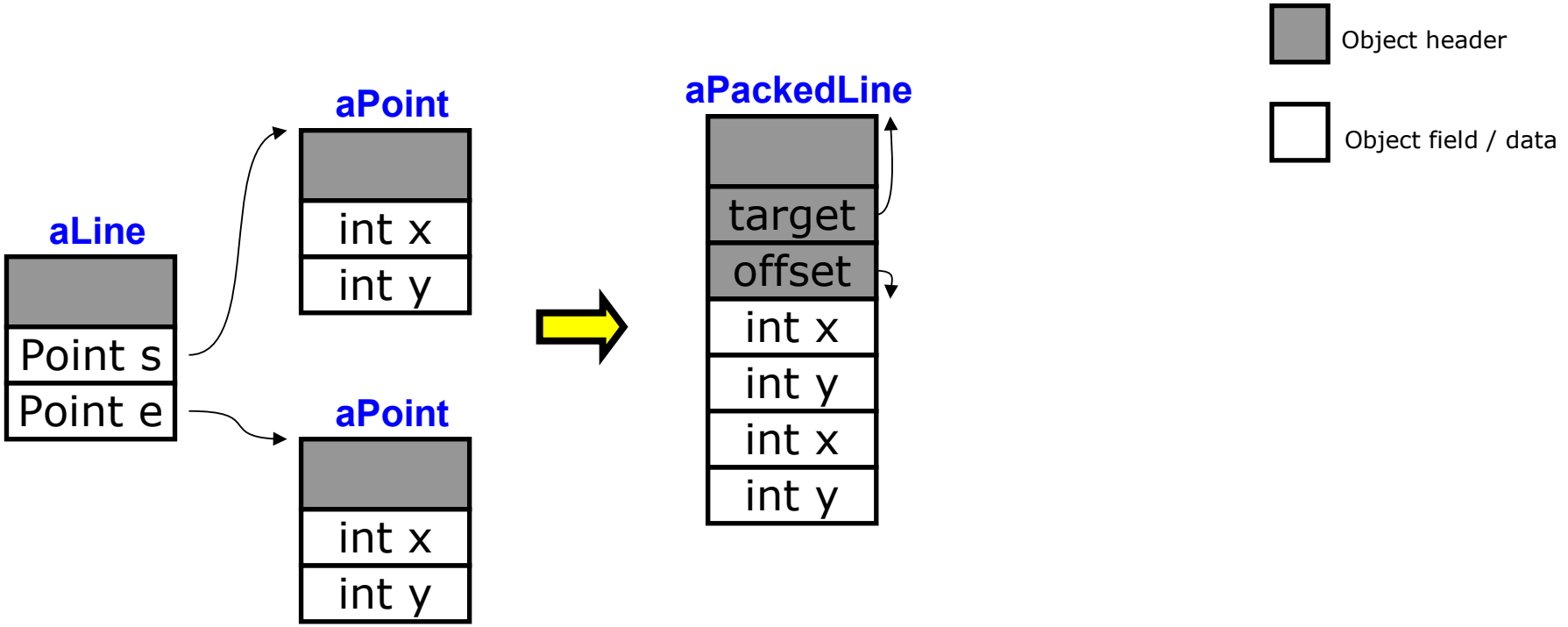



```
@Packed
final class PackedPoint extends PackedObject {
    int x;
    int y;
}

@Packed
final class PackedLine extends PackedObject {
    PackedPoint s;
    PackedPoint e;
}
```

Packed Objects: In Practice

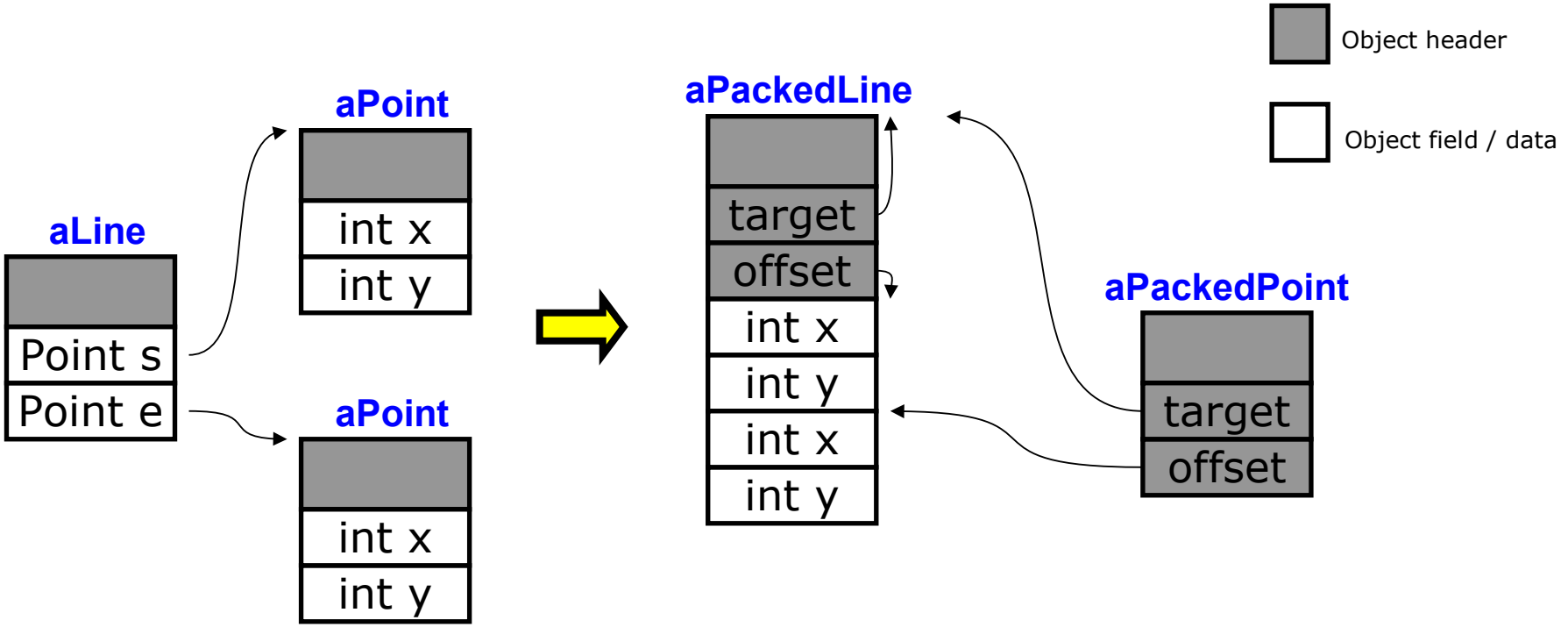


Packed Objects: In Practice



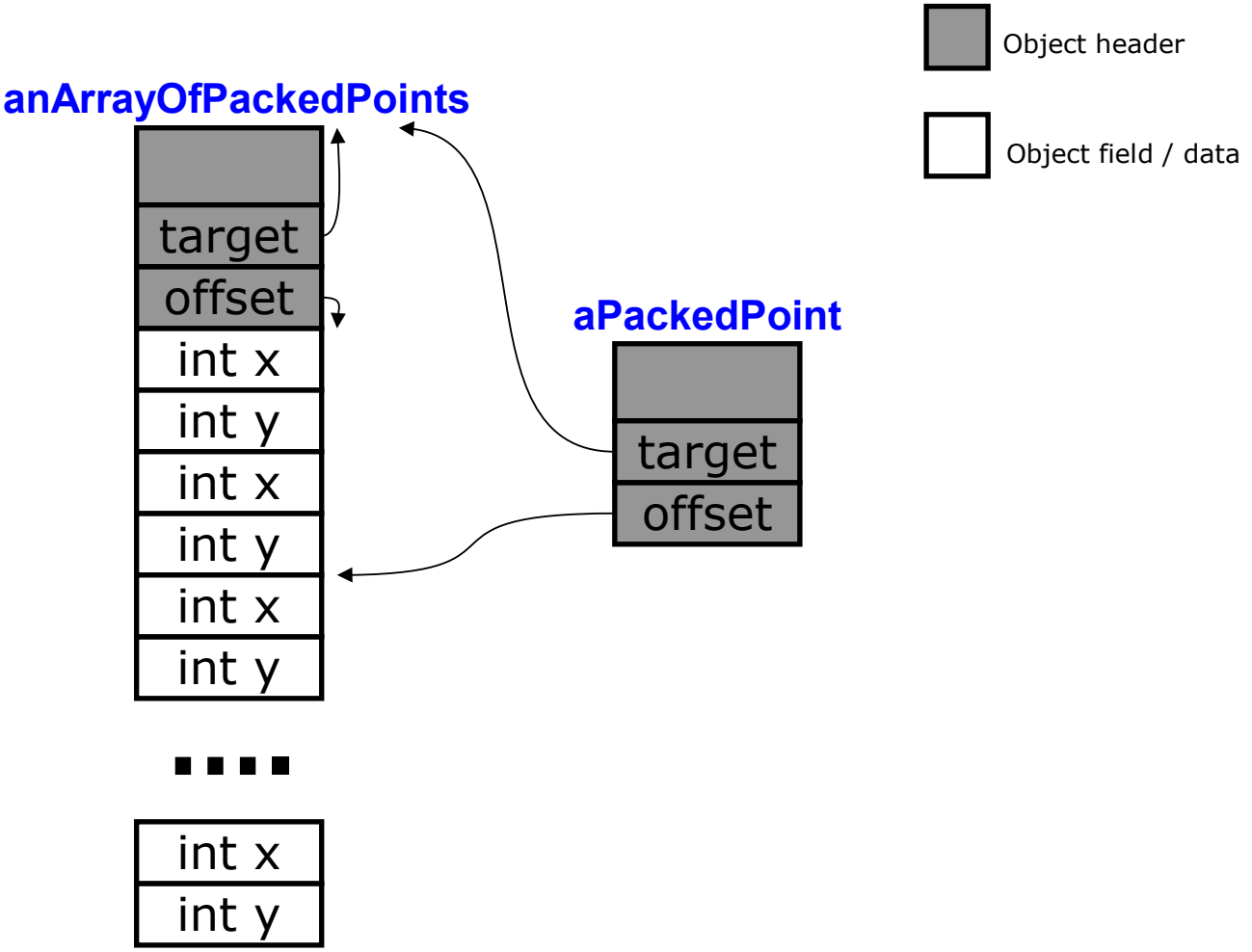
aPackedLine.e

Packed Objects: In Practice

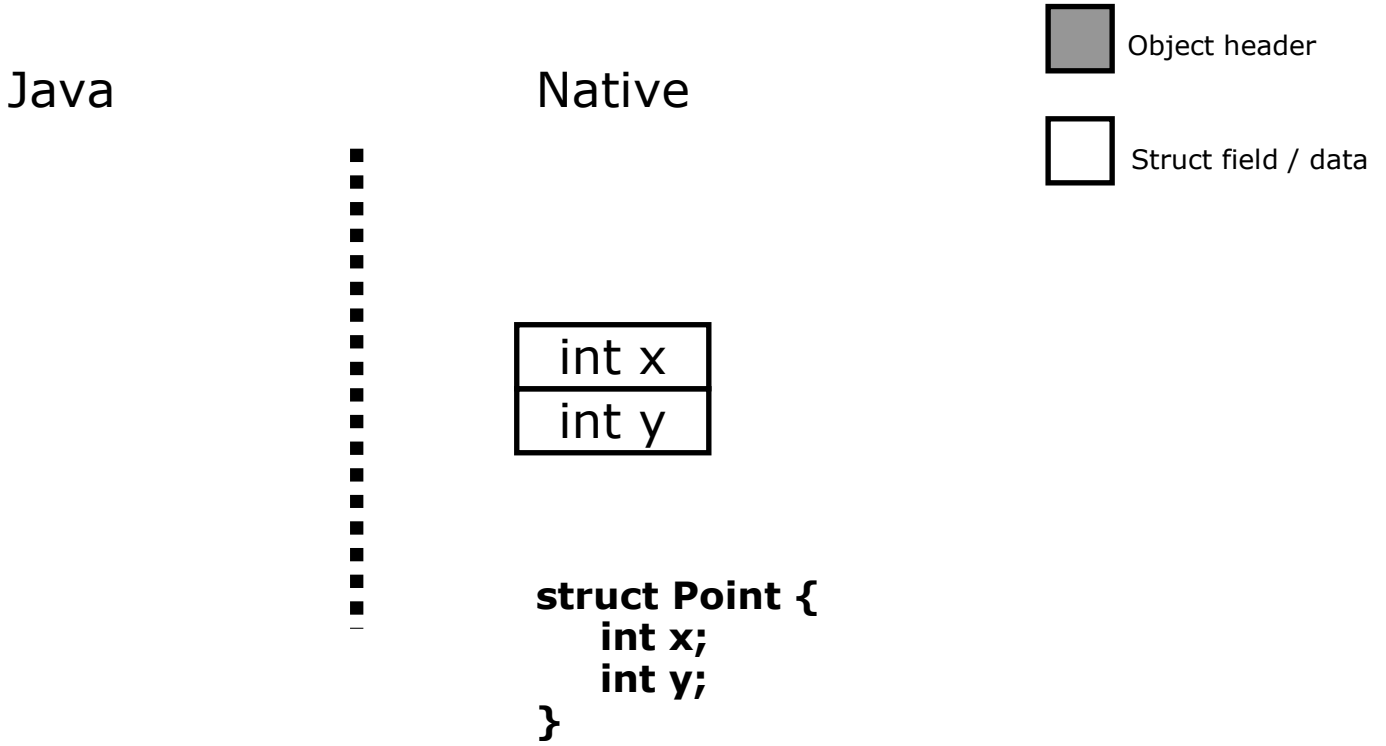


aPackedLine.e

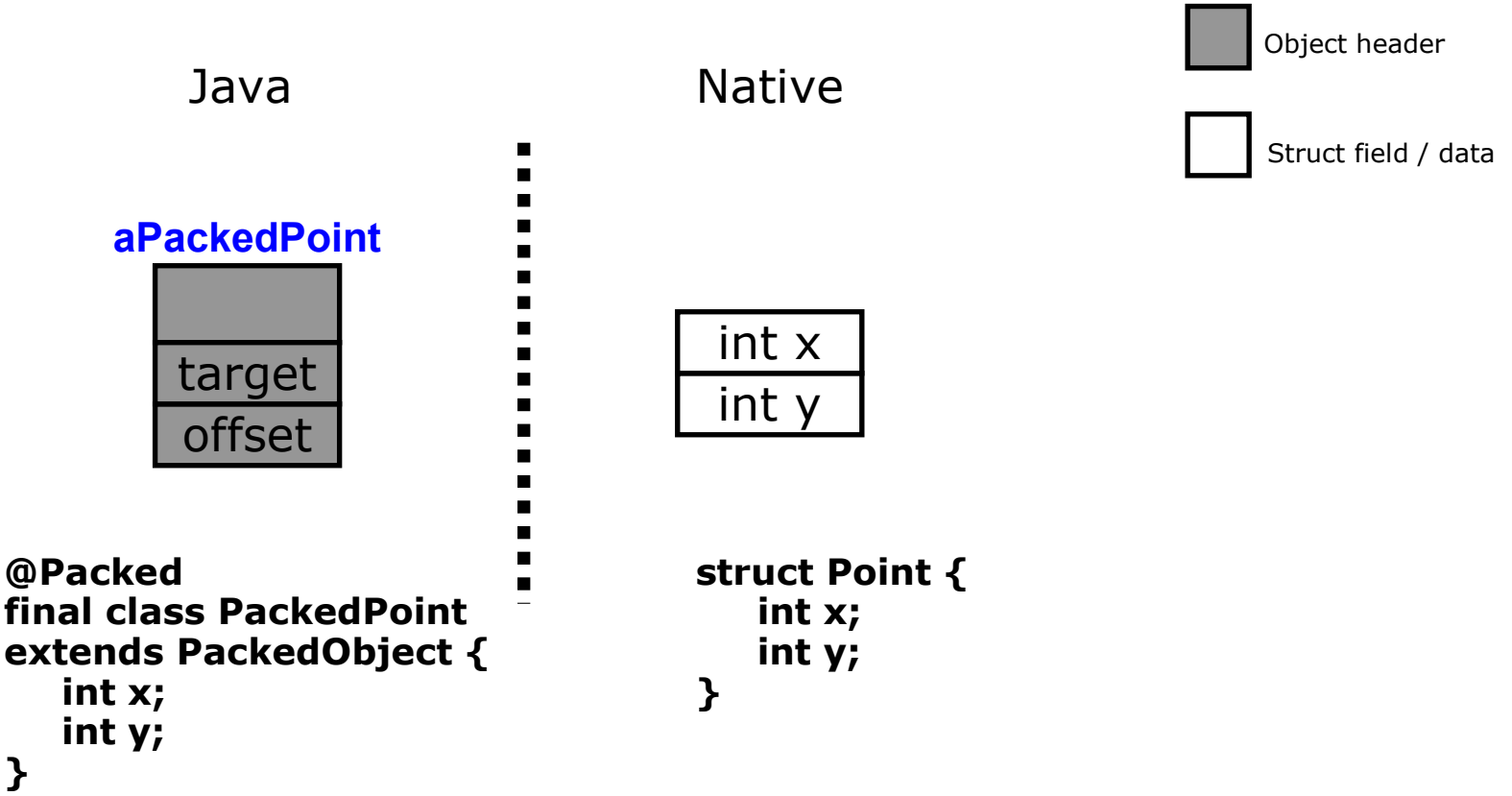
Packed Objects: In Practice with Arrays



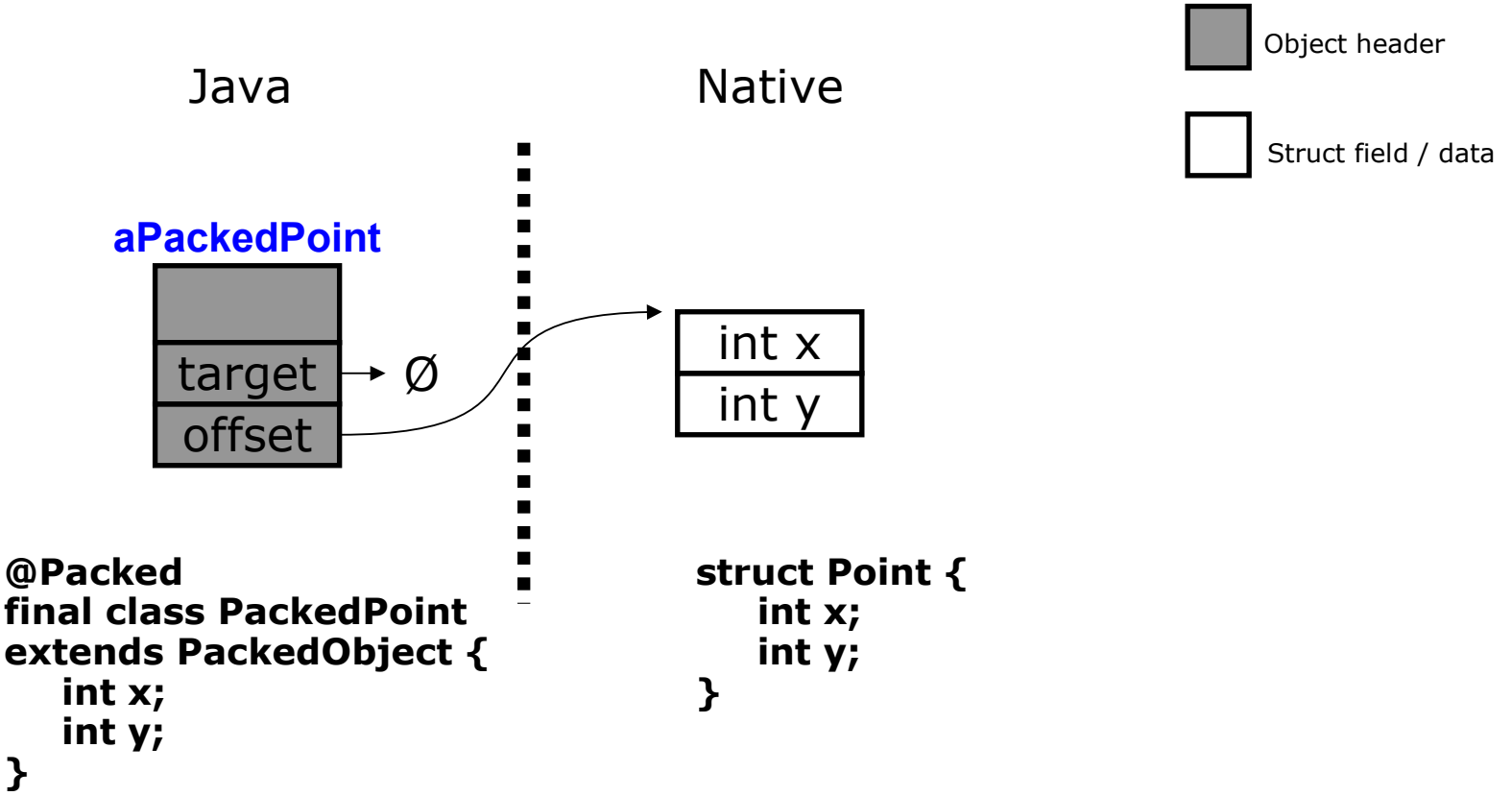
Packed Objects: In Practice with Native Access



Packed Objects: In Practice with Native Access



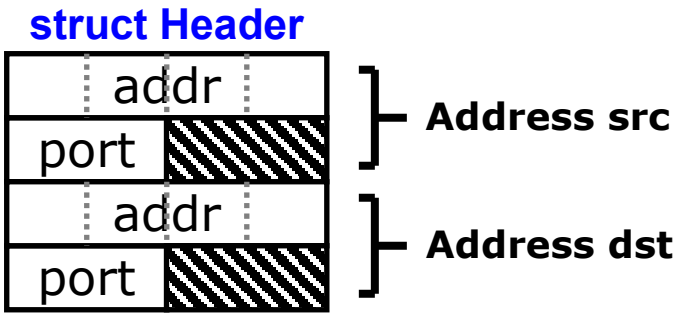
Packed Objects: In Practice with Native Access



Advantages

Lets Build Something in C!

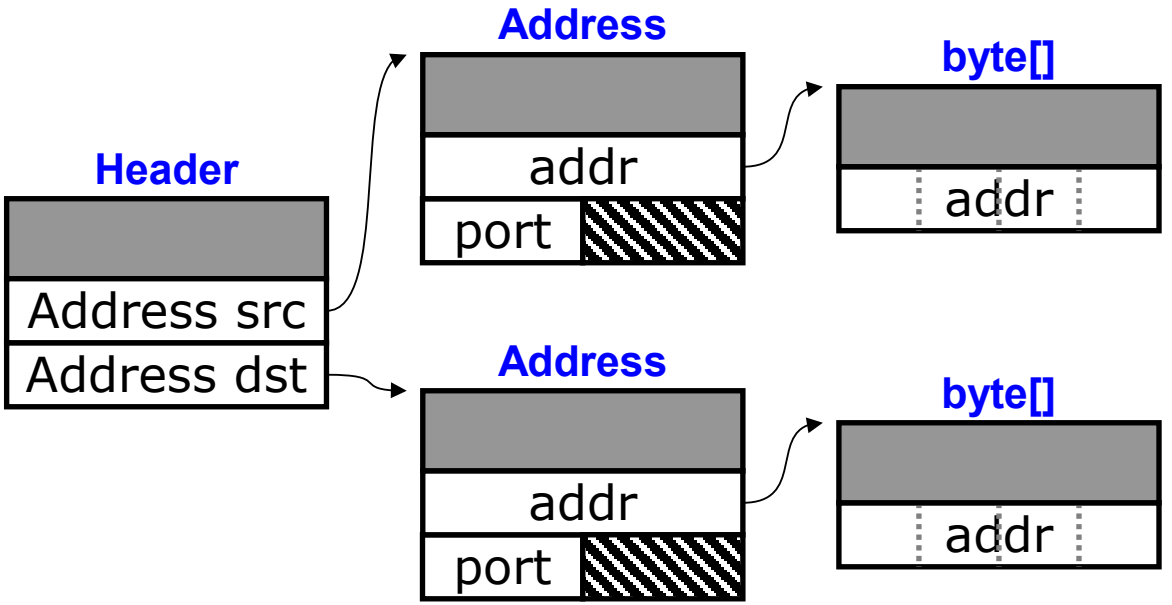
```
struct Address {
   char[4] addr;
   short port;
}
struct Header {
   struct Address src;
   struct Address dst;
}
```



- Nested substructures
- Compact
- Alignment

Let's Build the Same "Something" in Java!

```
class Address {
      byte[] addr;
      short port;
}

class Header {
      Address src;
      Address dst;
}
```



- Headers
- Locality
- Alignment

What does the Java code look like under the covers?

```
if(header.dst.addr[0] == (byte)192) {
    // ...
}
```

Bytecodes:

```
aload1
getfield Header.dest  LAddress;
getfield Address.addr [B
iconst0
baload
bipush 192
ificmpeq …
```

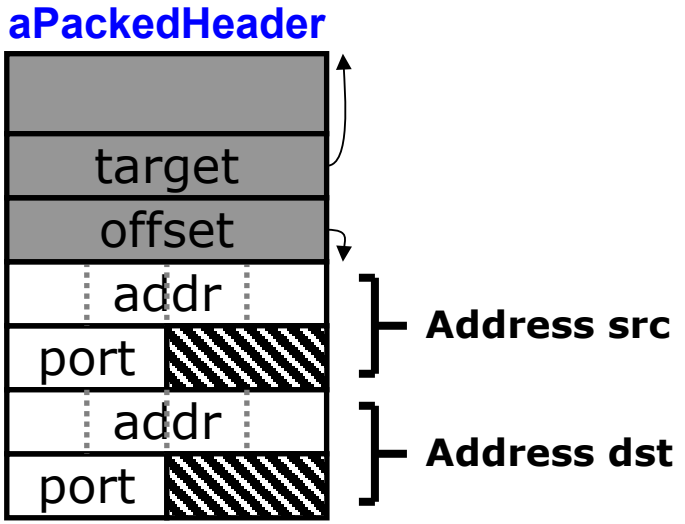
JIT (32 bit):

```
mov EBX, dword ptr -4[ECX]    // load temp1
mov EBX, dword ptr 8[EBX]     // load dest
mov EBX, dword ptr 4[EBX]     // load addr
movsx EDI, byte ptr 8[EBX]    // array[0]
cmp EDI, 192
```

- From a code point of view, this isn't terrible…

What if we did this with Packed Objects?

```
@Packed
final class Address extends PackedObject {
      PackedByte[[4]] addr;
      short port;
}

@Packed
final class PacketHeader extends PackedObject
{
      Address src;
      Address dest;
}
```



- The Java code is pretty clean… and a pretty good result!

Ok, what about the code under the covers?

```
if(header.dst.addr[[0]] == (byte)192) {
    // ...
}
```

Bytecodes:

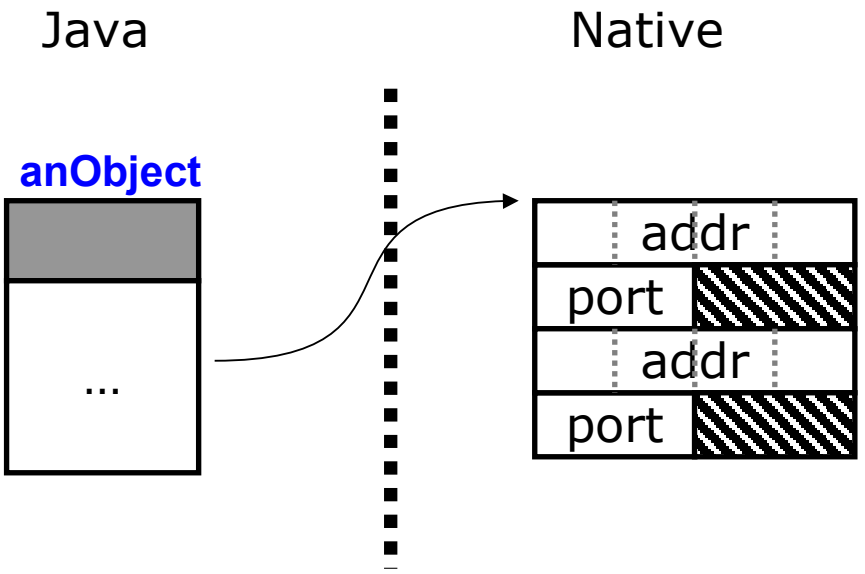
```
aload1
getfield PackedHeader.dest
    LAddress;
getfield Address.addr [B
iconst0
baload
bipush 192
ificmpeq …
```

JIT (32 bit):

```
mov EBX, dword ptr -4[ECX]    // load temp1
mov EAX, dword ptr 4[EBX]     // load target
mov EDX, dword ptr 8[EBX]     // load offset
lea EBX, dword ptr [EAX + EDX]
movsx EDI, byte ptr 0[EBX]    // array[0]
cmp EDI, 192
```

- Bytecodes don’t change… JIT code is pretty good too!

What about native access?



How do we implement this normally?

JNI implementation

```
public class PackedHeader {
    private long pointer;

    public byte[] getSourceAddress() { return getSourceAddressImpl(pointer); }
    public short getSourcePort() { return getSourcePortImpl(pointer); }
}
```

```
JNICALL jshort Java_pkg_PackedHeader_getSourcePort(JNIEnv* env, jobject recv, jlong pointer) {
   struct PacketHeader* header = (struct PacketHeader*)pointer;
   return (jshort)header->src.port;
}

JNICALL jbyteArray Java_pkg_PackedHeader_getSourceAddress(JNIEnv* env, jobject recv, jlong pointer) {
   struct PacketHeader* header = (struct PacketHeader*)pointer;
   jbyteArray result = (*env)->NewByteArray(env, 4);
   (*env)->SetByteArrayRegion(env, result, 0, 4, &(header->src.addr));
   return result;
}
```

- Usual “stash pointers in long types” tricks
- JNI costs tend to be high

JNI implementation

```
public class PackedHeader {
    private long pointer;

    public byte[] getSourceAddress() { return getSourceAddressImpl(pointer); }
    public short getSourcePort() { return getSourcePortImpl(pointer); }
}
```

```
JNICALL jshort Java_pkg_PackedHeader_getSourcePort(JNIEnv* env, jobject recv, jlong pointer) {
   struct PacketHeader* header = (struct PacketHeader*)pointer;
   return (jshort)header->src.port;
}

JNICALL jbyteArray Java_pkg_PackedHeader_getSourceAddress(JNIEnv* env, jobject recv, jlong pointer) {
   struct PacketHeader* header = (struct PacketHeader*)pointer;
   jbyteArray result = (*env)->NewByteArray(env, 4);
   (*env)->SetByteArrayRegion(env, result, 0, 4, &(header->src.addr));
   return result;
}
```

- Usual “stash pointers in long types” tricks
- JNI costs tend to be high

Unsafe implementation

```
class PackedHeader {
    private Unsafe unsafe;
    private long pointer;
    private static final int SRC_ADDR_OFFSET = 0;
    private static final int SRC_PORT_OFFSET = 4;
    private static final int DEST_ADDR_OFFSET = 8;
    private static final int DEST_PORT_OFFSET = 12;

    public short getSourcePort() { return unsafe.getShort(pointer + SRC_PORT_OFFSET); }
    public byte[] getSourceAddress() {
        byte[] result = new byte[4];
        unsafe.copyMemory(null, pointer + SRC_ADDR_OFFSET, result, 0, 4);
        return result;
    }
}
```

- You shouldn’t be here
- Keeping your indices straight is never fun

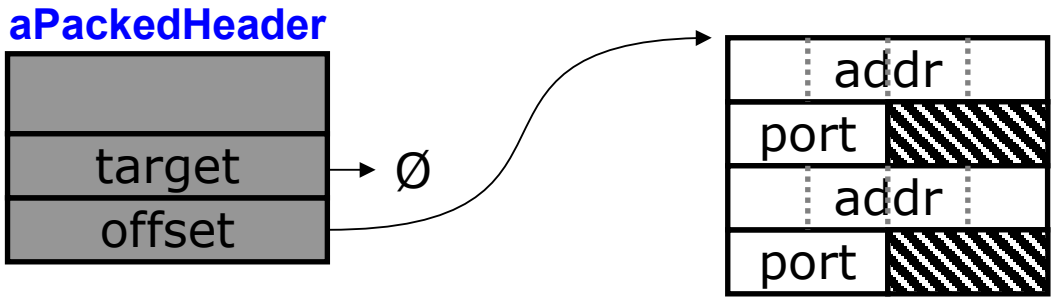
DirectByteBuffer implementation

```
class PackedHeader {
    private ByteBuffer buffer;
    private static final int SRC_ADDR_OFFSET = 0;
    private static final int SRC_PORT_OFFSET = 4;
    private static final int DEST_ADDR_OFFSET = 8;
    private static final int DEST_PORT_OFFSET = 12;

    public short getSourcePort() { return buffer.getShort(SRC_PORT_OFFSET); }
    public byte[] getSourceAddress() {
        byte[] result = new byte[4];
        buffer.get(result, SRC_ADDR_OFFSET, 4);
        return result;
    }
}
```

- No extra JNI to write (this is good)
- Still playing the indices game

PackedObject answer

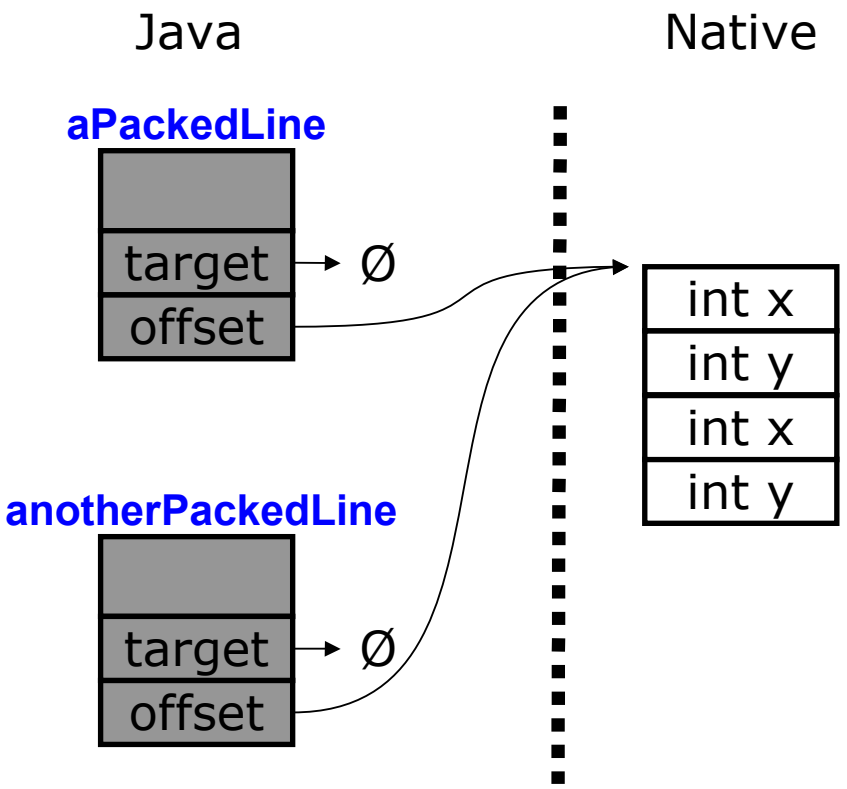
```
final class PacketHeader extends PackedObject {
   Address src;
   Address dest;

   public short getSourcePort() { return src.port; }
   public PackedByte[] getSourceAddress() { return src.addr; }
}
```



- Looks like natural Java code
- Foregoes JNI
- Same type capable of on-heap representation

Challenges

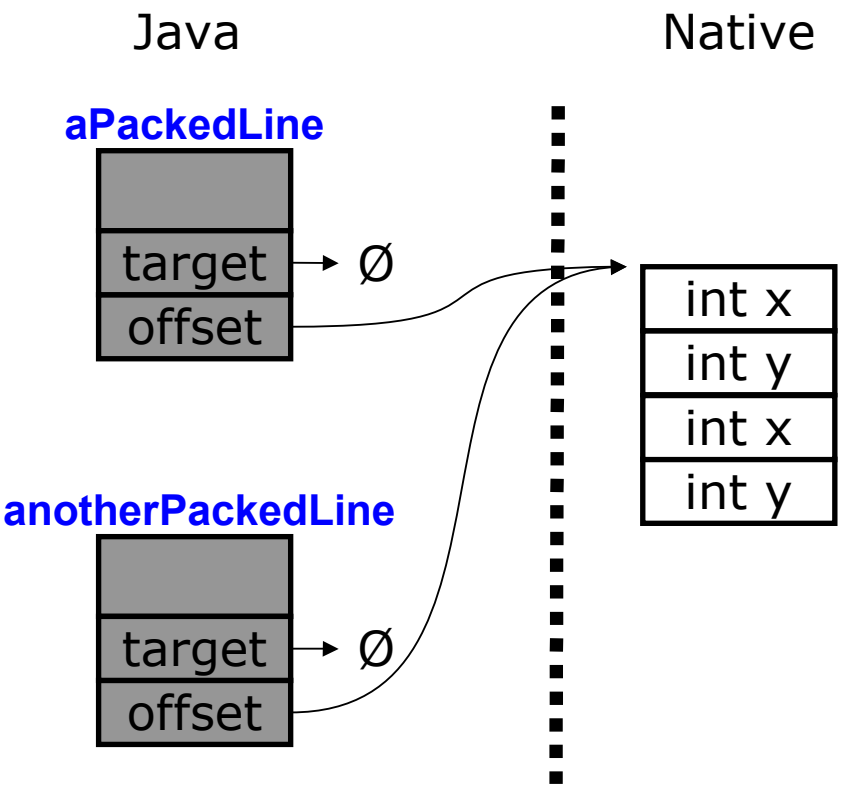
Identity Crisis



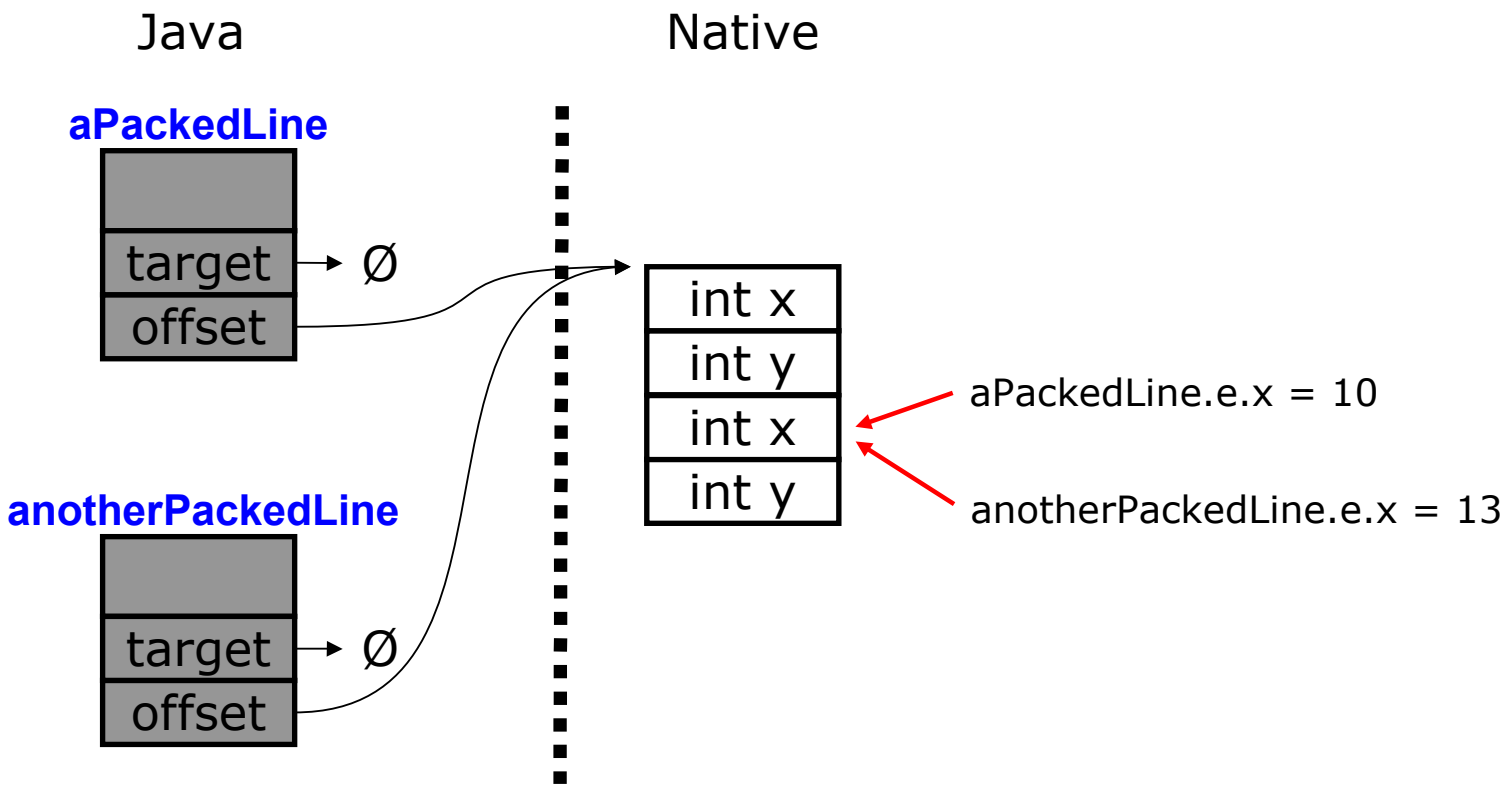
What does `aPackedLine == anotherPackedLine` mean?

→ The data is what really matters

Synchronization



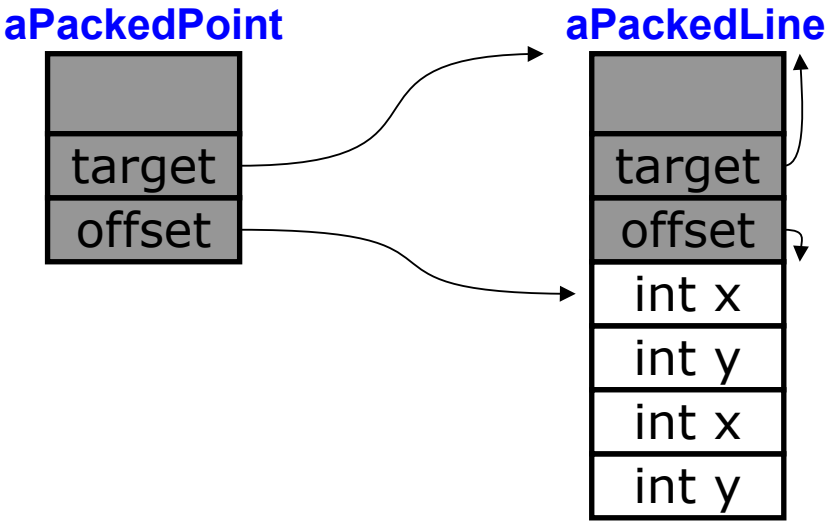
Synchronization



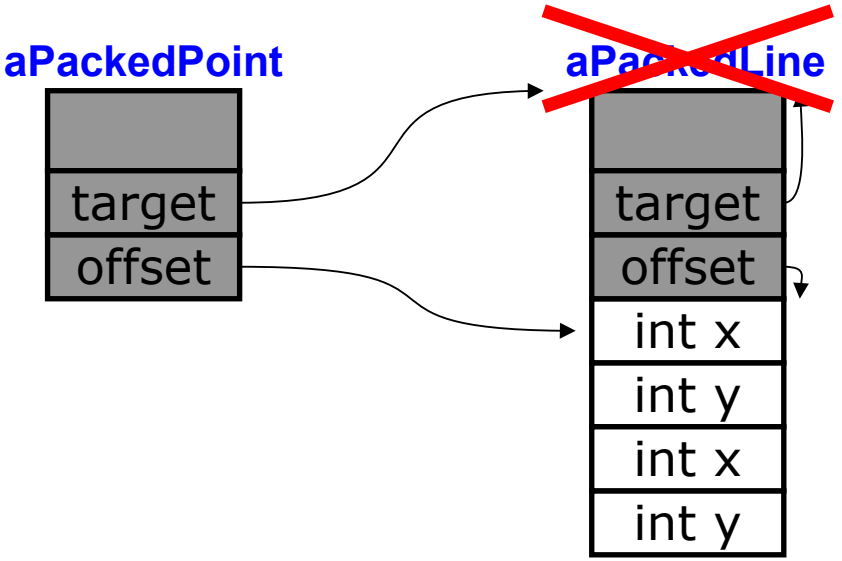

```
Thread T1:
synchronized (aPackedLine) {
        aPackedLine.e.x = 10;
}
```

```
Thread T2:
synchronized (anotherPackedLine) {
        anotherPackedLine.e.x = 13;
}
```

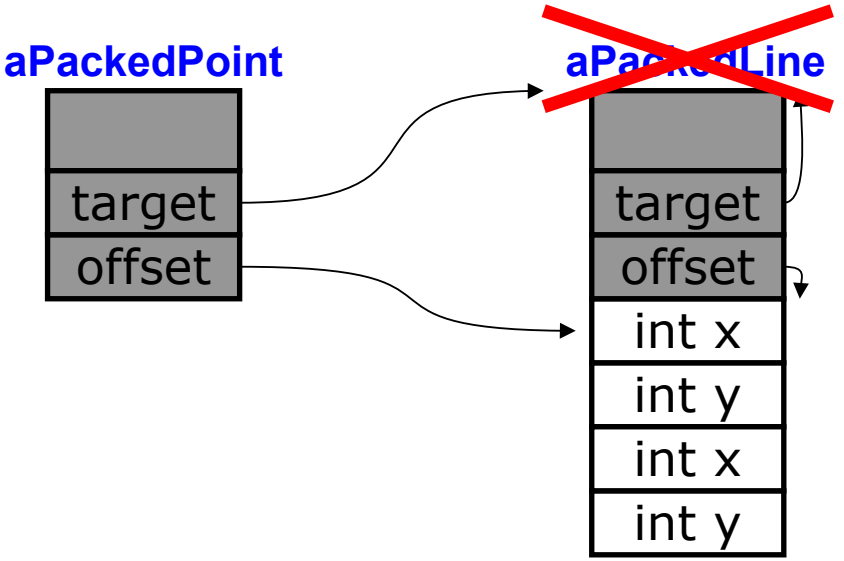
Finalization



Finalization

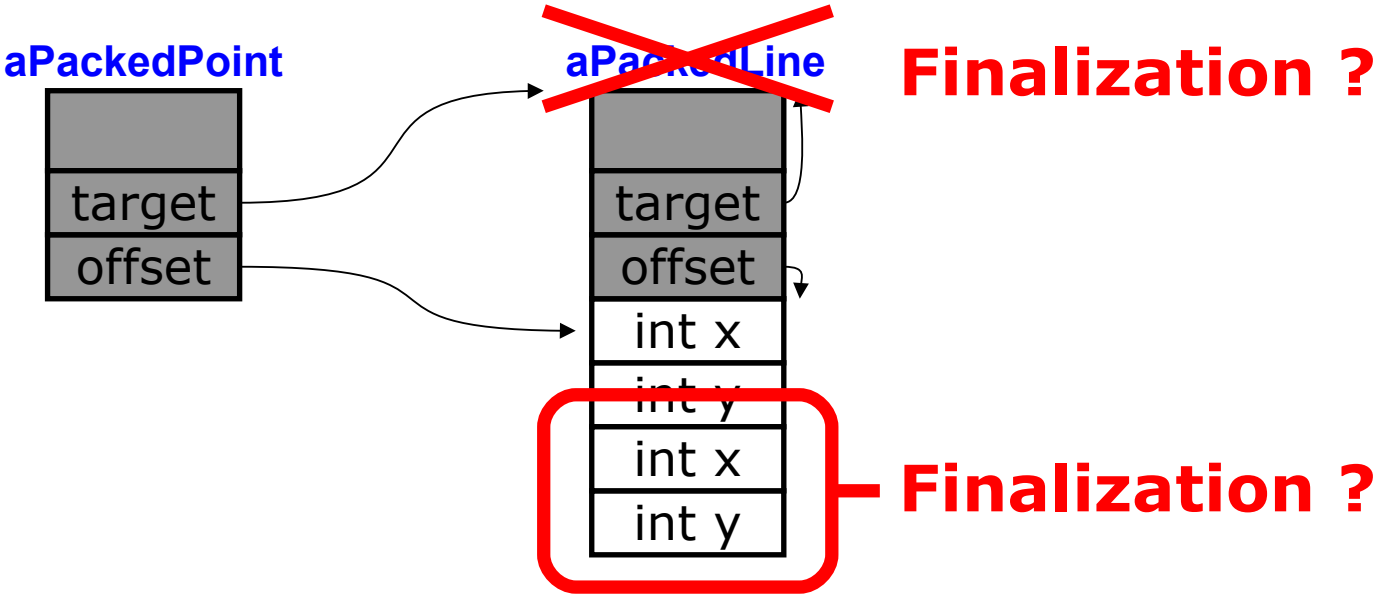


Finalization

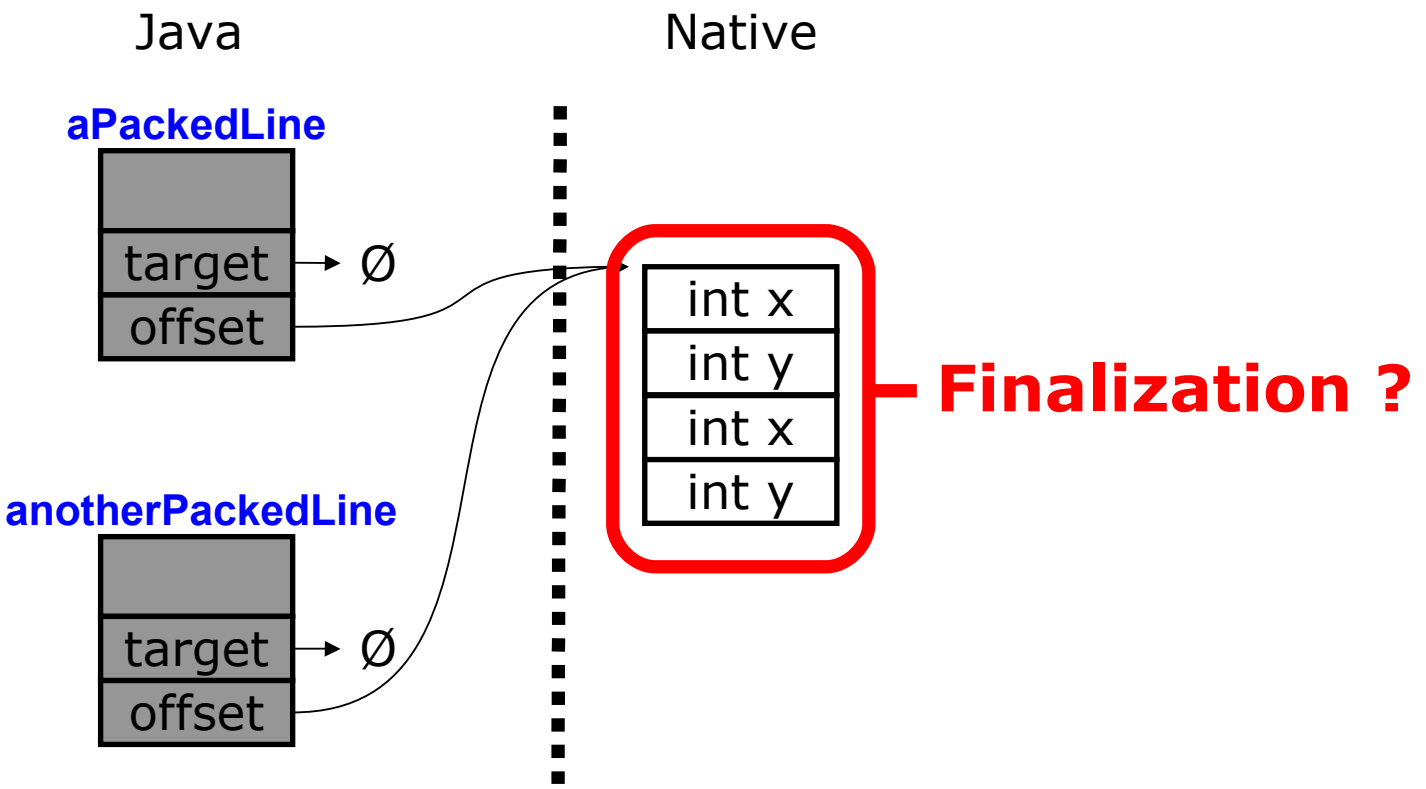


Finalization ?

Finalization

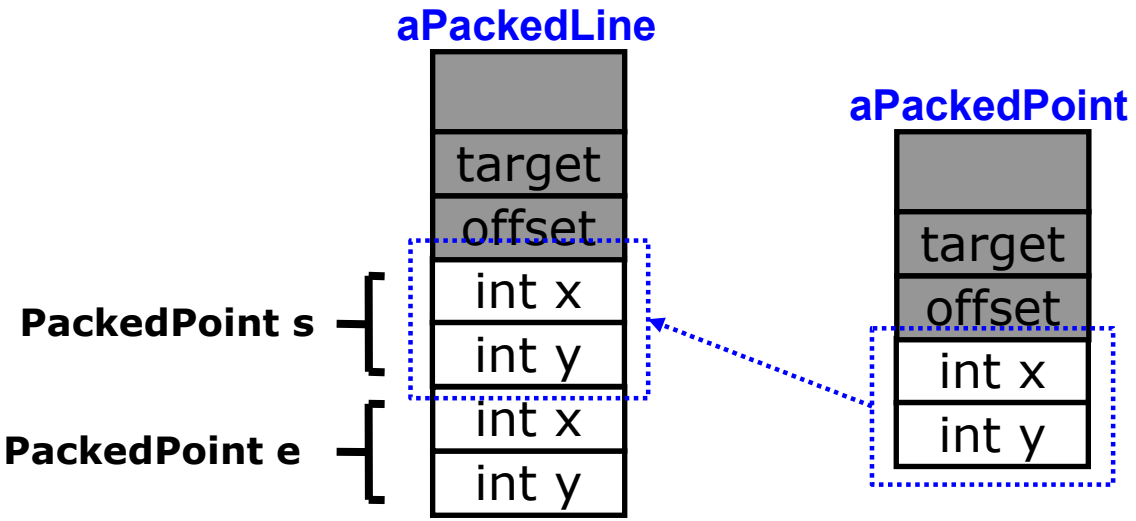


Finalization



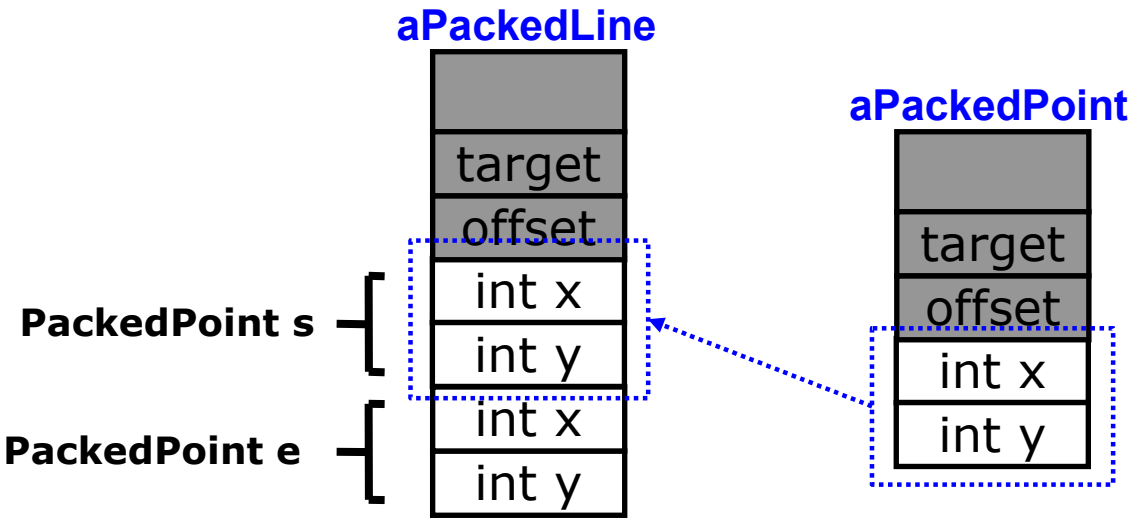
Nested Data Structures

```
aPackedLine.s = aPackedPoint;
```



Nested Data Structures

```
aPackedLine.s := aPackedPoint;
```



- Base types **do** share the same assignment operator
- Helps convey `aPackedLine == anotherPackedLine` as meaningless

Field Initialization

```
@Packed
final class PackedPoint extends PackedObject {
    int x;
    int y;

    PackedPoint(int x, int y) { … }
}

@Packed
final class PackedLine extends PackedObject {
    PackedPoint s;
    PackedPoint e;

    PackedLine(int sx, int sy, int ex, int ey) { … }
}
```

No no-argument constructor

Implicitly instantiates PackedPoint objects for s & e fields

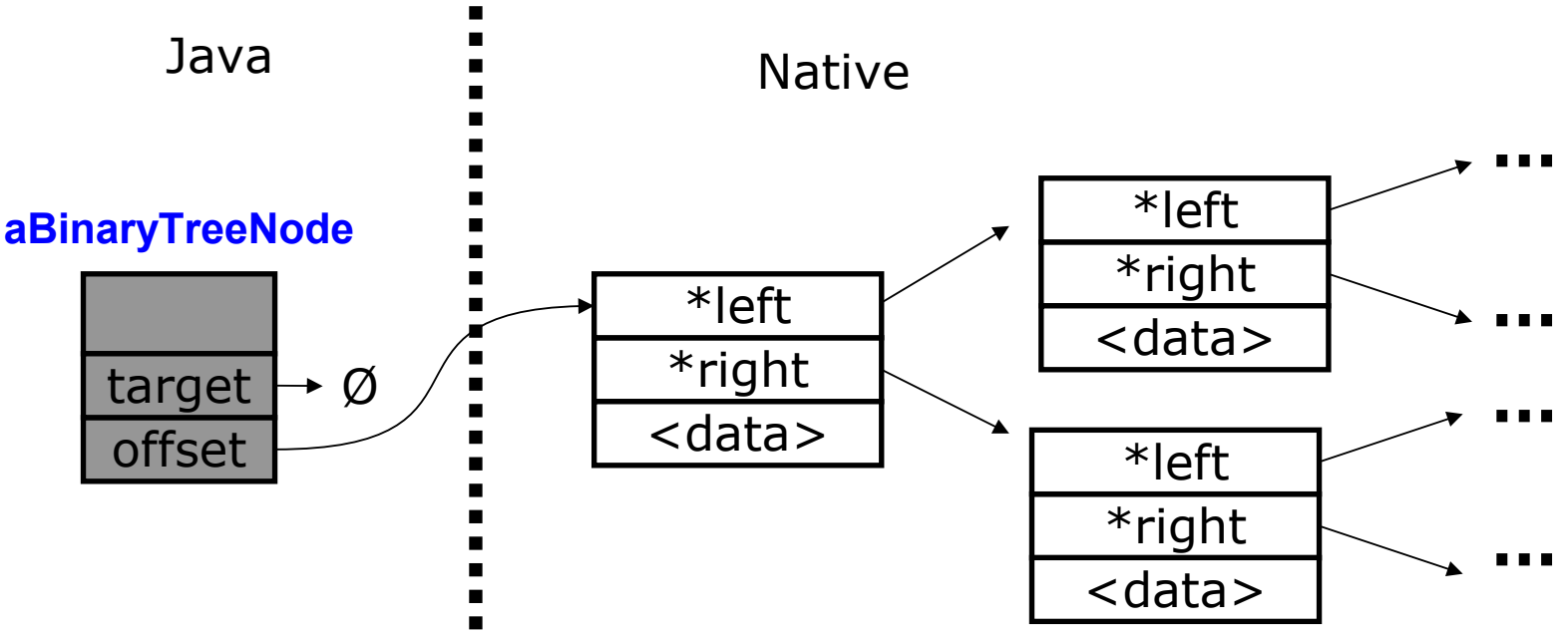
Field Initialization

```
@Packed
final class PackedPoint extends PackedObject {
    int x;
    int y;

    void init(int x, int y) {
        this.x = x;
        this.y = y;
    }

    PackedPoint(int x, int y) { … }
}

@Packed
final class PackedLine extends PackedObject {
    PackedPoint s;
    PackedPoint e;

    PackedLine(int sx, int sy, int ex, int ey) {
        s.init(sx, sy);
        e.init(ex, ey);
    }
}
```

Advanced

Modeling Native Data Pointers

```
struct BinaryTreeNode {
    struct BinaryTreeNode* left;
    struct BinaryTreeNode* right;
    // data
}
```

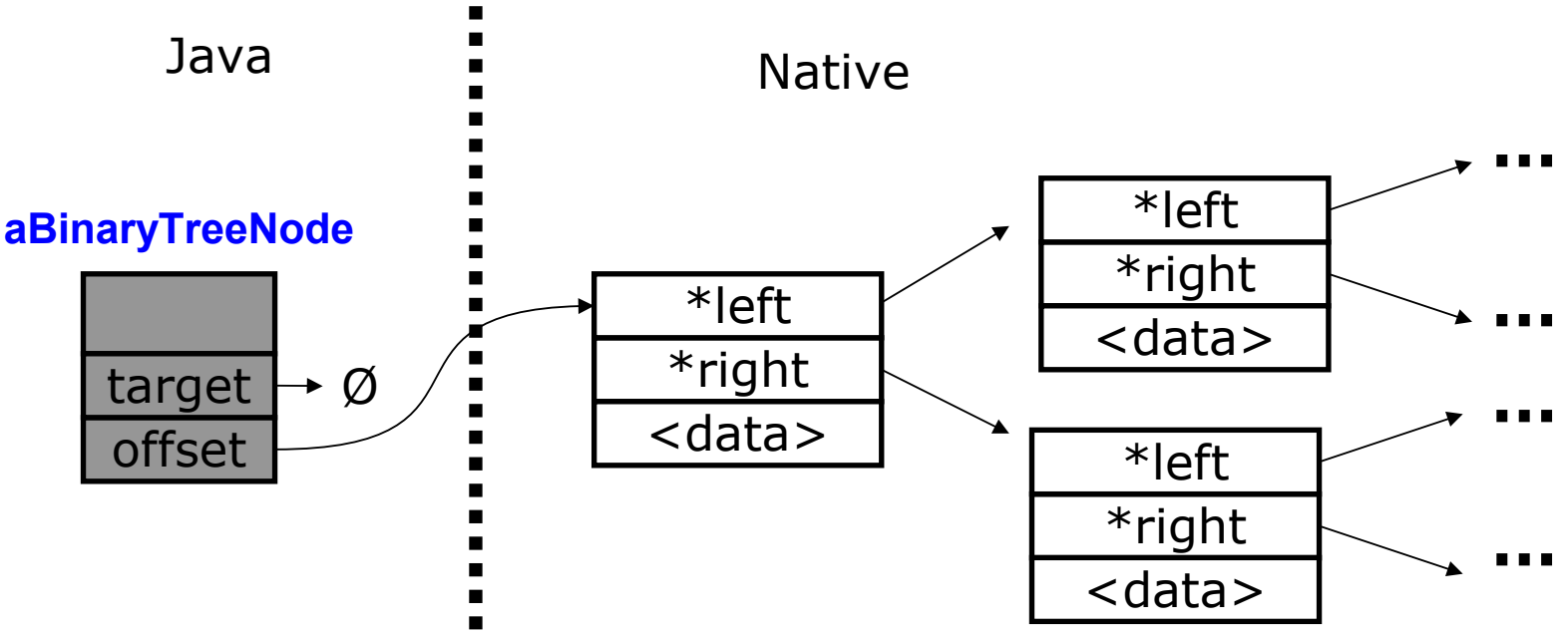


Modeling Native Data Pointers

?

aBinaryTreeNode.right

```
struct BinaryTreeNode {
    struct BinaryTreeNode* left;
    struct BinaryTreeNode* right;
    // data
}
```

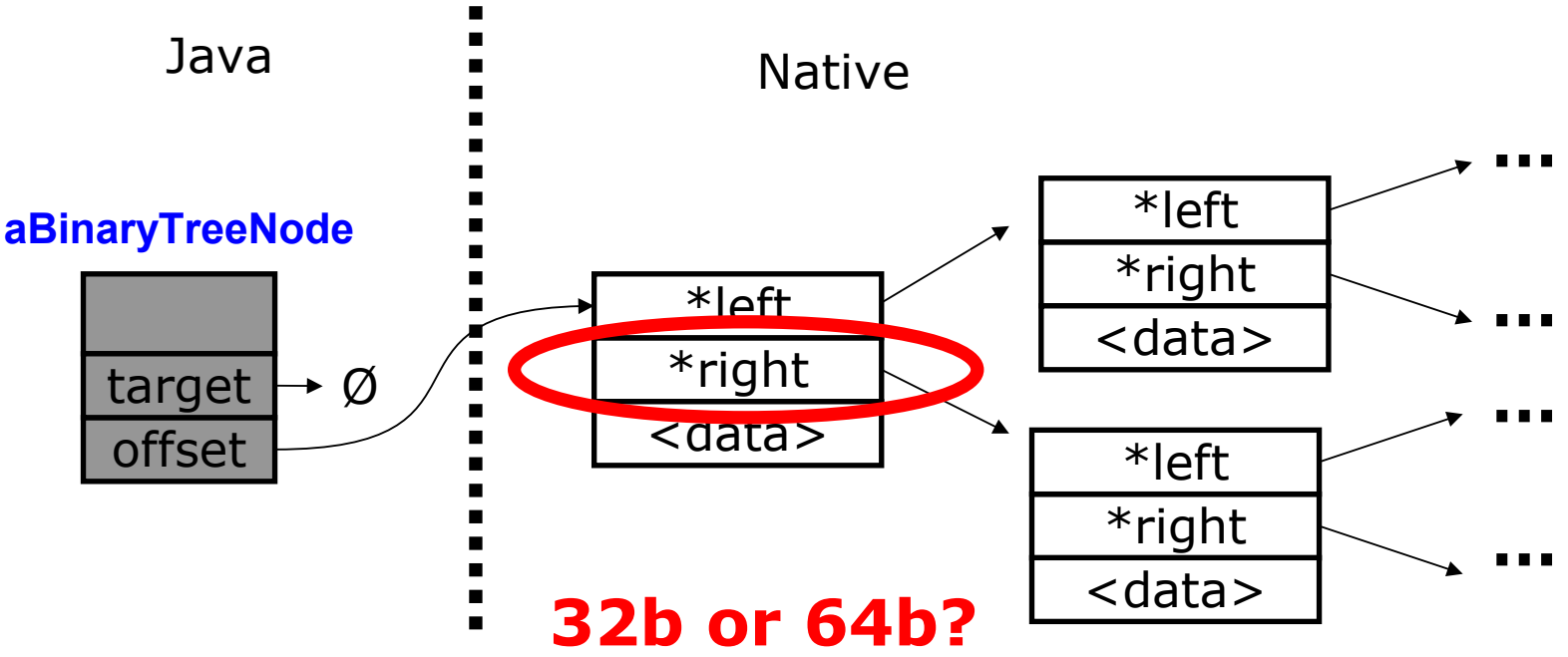


Modeling Native Data Pointers

?

aBinaryTreeNode.right

```
struct BinaryTreeNode {
    struct BinaryTreeNode* left;
    struct BinaryTreeNode* right;
    // data
}
```

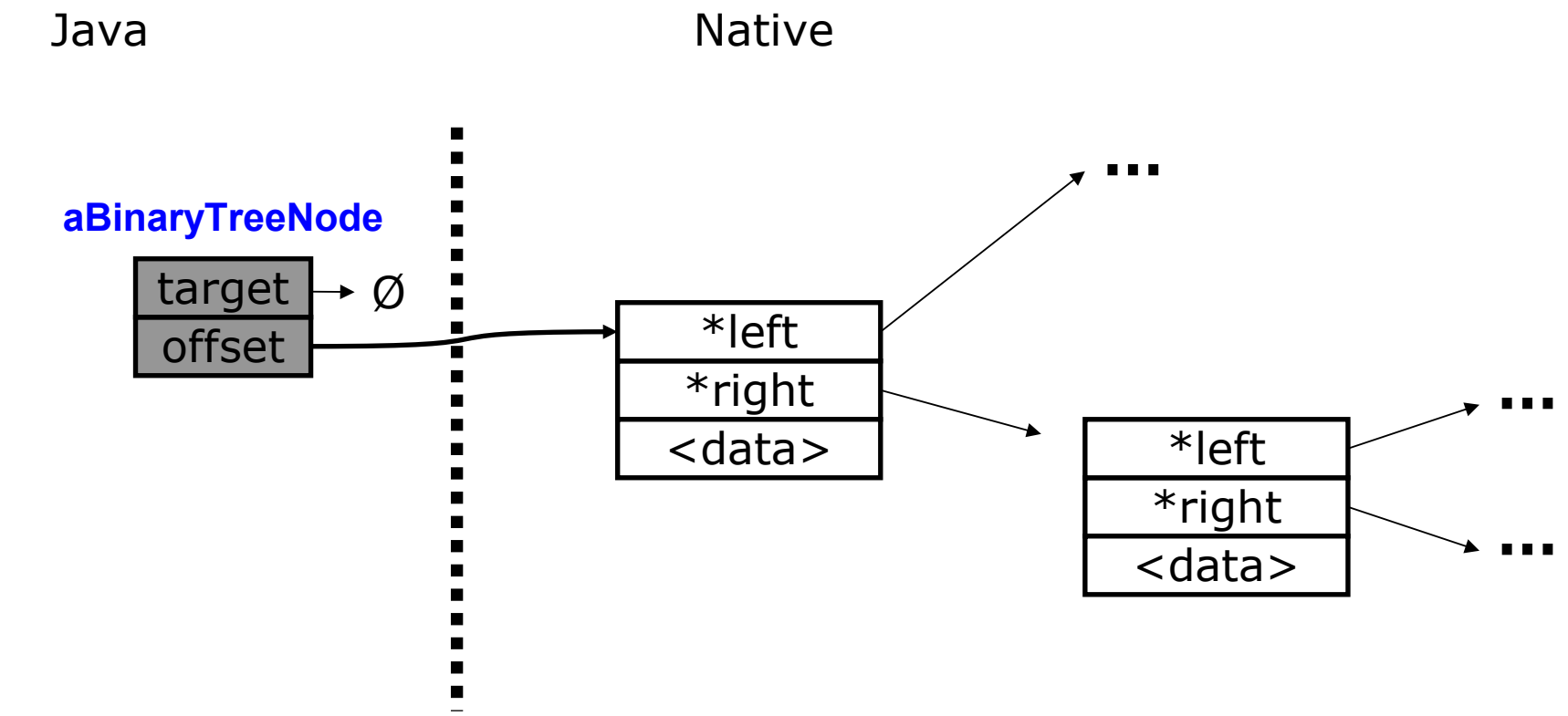


Modeling Native Data Pointers

```
@Packed
class BinaryTreeNode extends PackedObject {
    @NativePointer BinaryTreeNode left;
    @NativePointer BinaryTreeNode right;
    // data
}
```

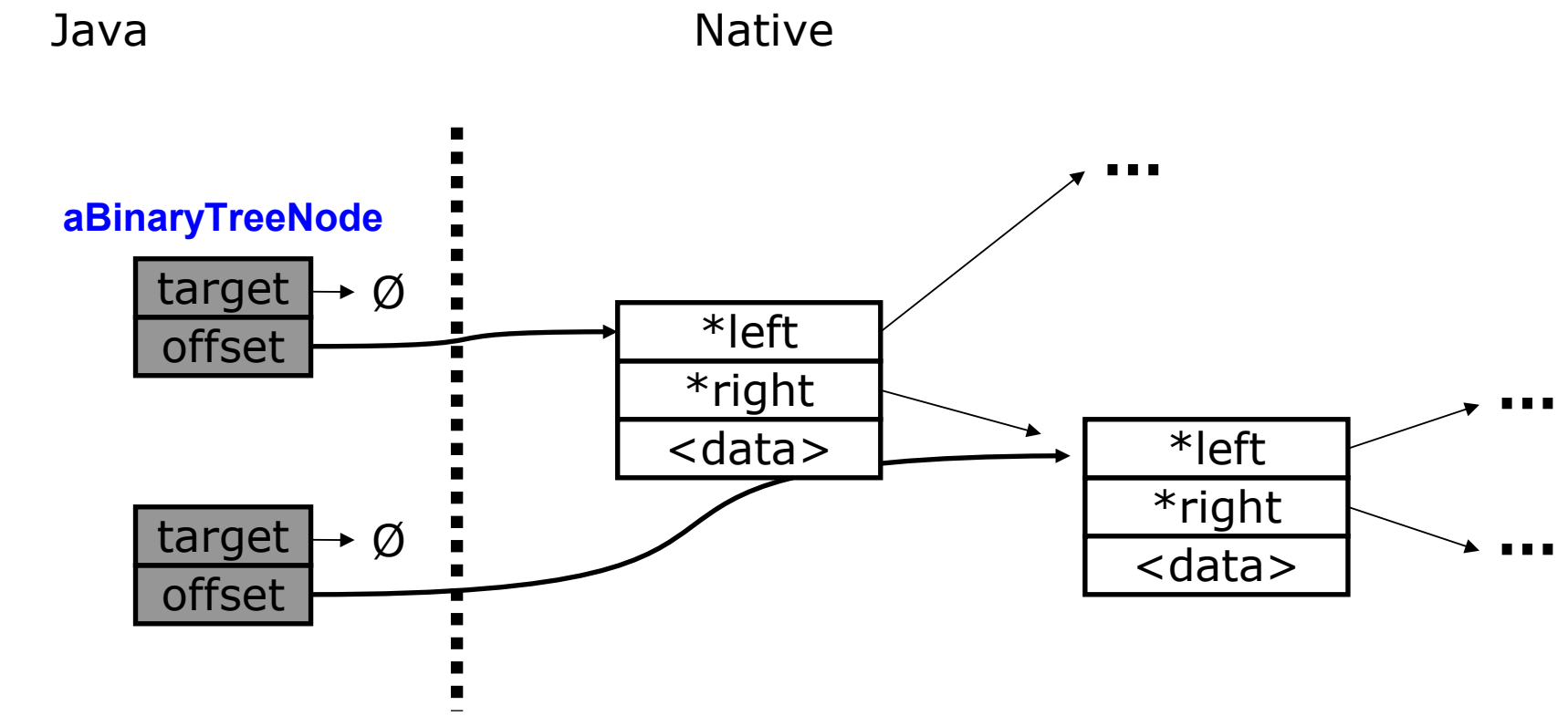
- Annotation to mark a field as a native pointer (rather than a Java one)
- Enhance `getfield` / `putfield` to recognize
- Restrict for security reasons
- Unmanaged pointers (no GC involvement)

Modeling Native Data Pointers




```
aBinaryTreeNode.right
```

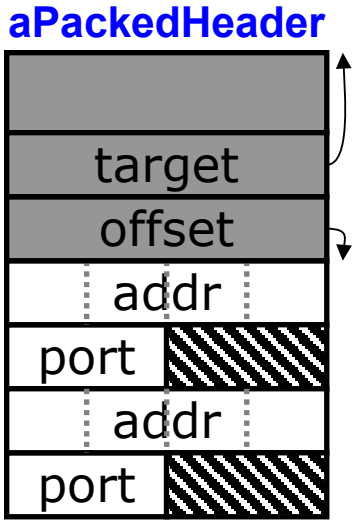
Modeling Native Data Pointers



`aBinaryTreeNode.right`

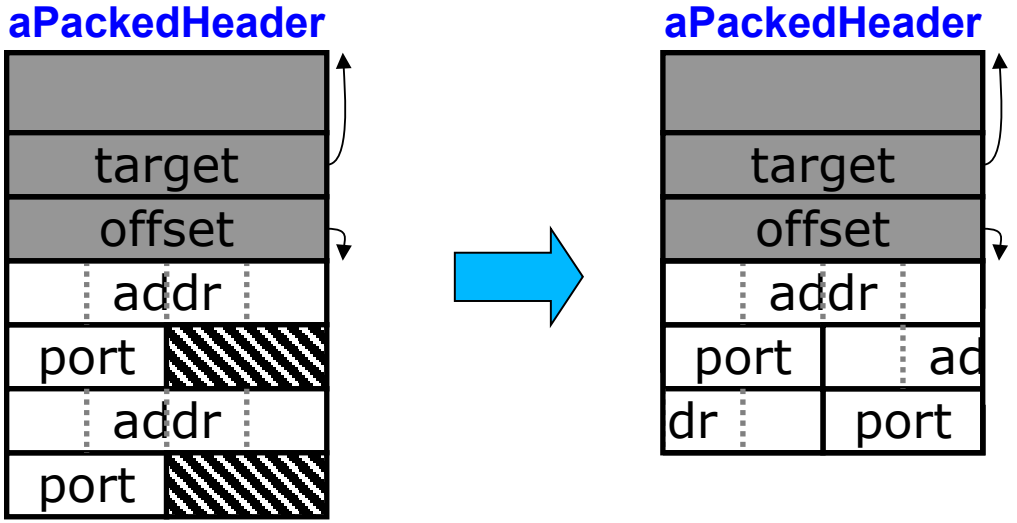
Alignment

```
@Packed
final class Address
extends PackedObject {
      PackedByte[[4]] addr;
      short port;
}

@Packed
final class PacketHeader
extends PackedObject {
      Address src;
      Address dest;
}
```



Alignment

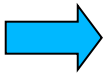
```
@Packed
final class Address
extends PackedObject {
      PackedByte[[4]] addr;
      short port;
}

@Packed
final class PacketHeader
extends PackedObject {
      Address src;
      Address dest;
}
```



- Which is the correct default behavior?
- How do you get the alternate if that's what you want?

Alignment

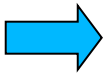
```
class A {
    int i;
    short s;
    short padding; // align
    long l;
}
```



```
class A {
    int i;
    short s;
    @Align long l;
}
```

Alignment

```
class A {
    int i;
    short s;
    short padding; // align
    long l;
}
```



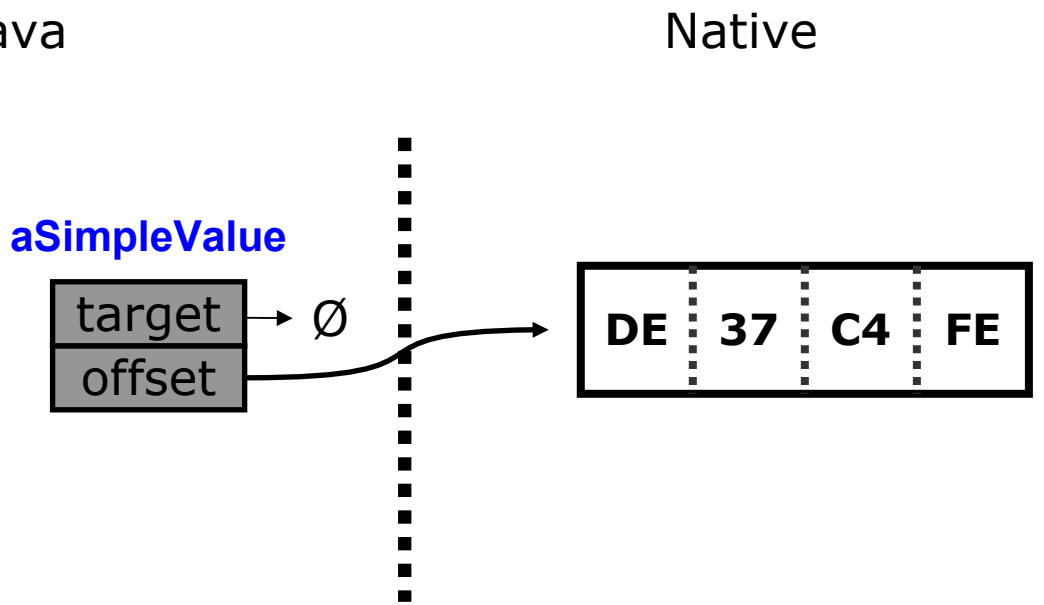
```
class A {
    int i;
    short s;
    @Align long l;
}
```

- Padding isn't quite right in the context of nested structures…

```
@Packed
final class Address extends PackedObject {
    PackedByte[[4]] addr;
    short port;
}

@Packed
final class PacketHeader extends PackedObject {
    @Align Address src;
    @Align Address dest;
}
```

Endian

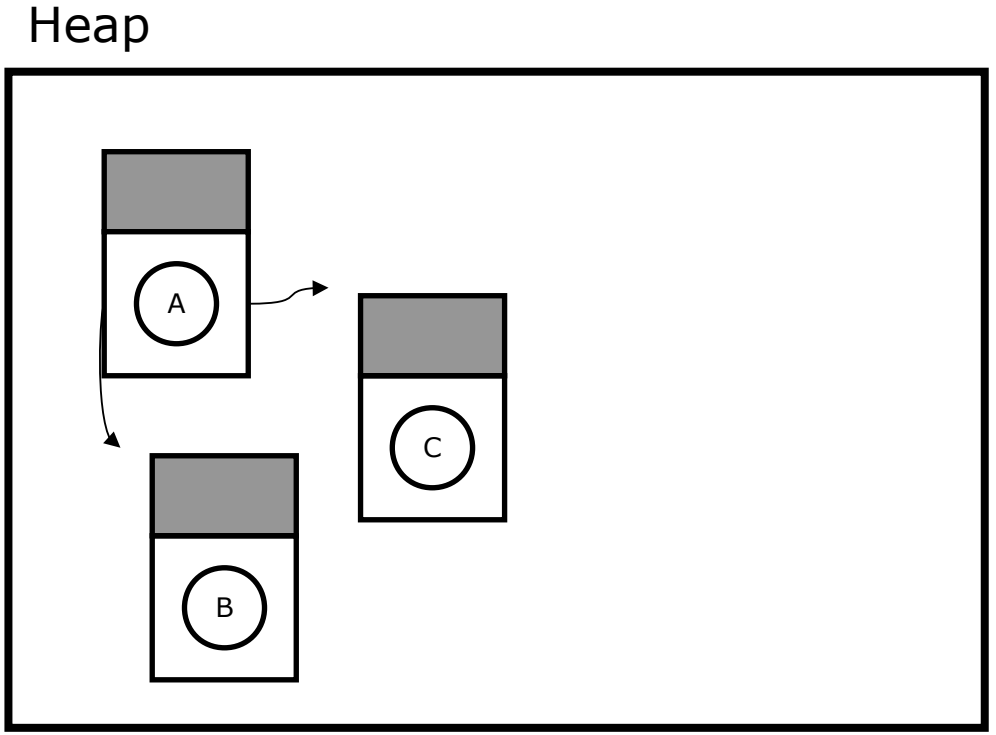
```
@Packed
final class SimpleValue extends PackedObject {
      int value;
}
```



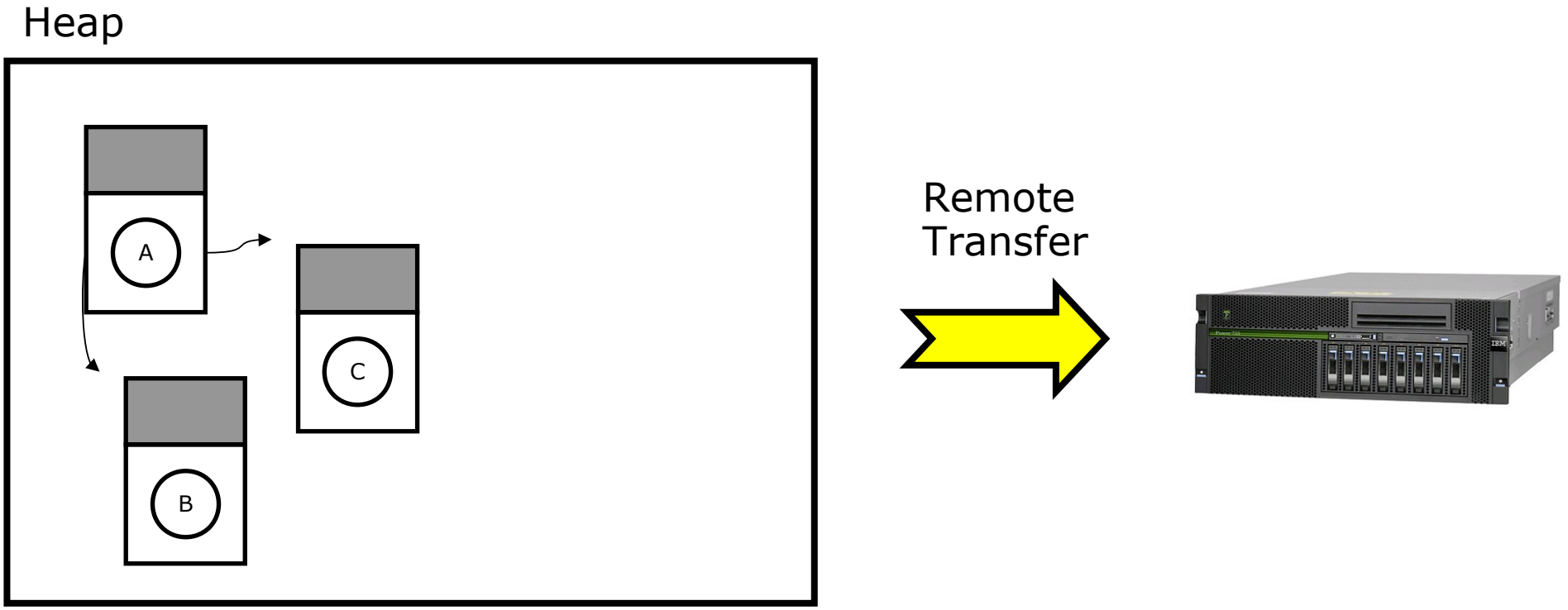
- Provide a field annotation `@BigEndian` (and `@LittleEndian`)

Possibilities

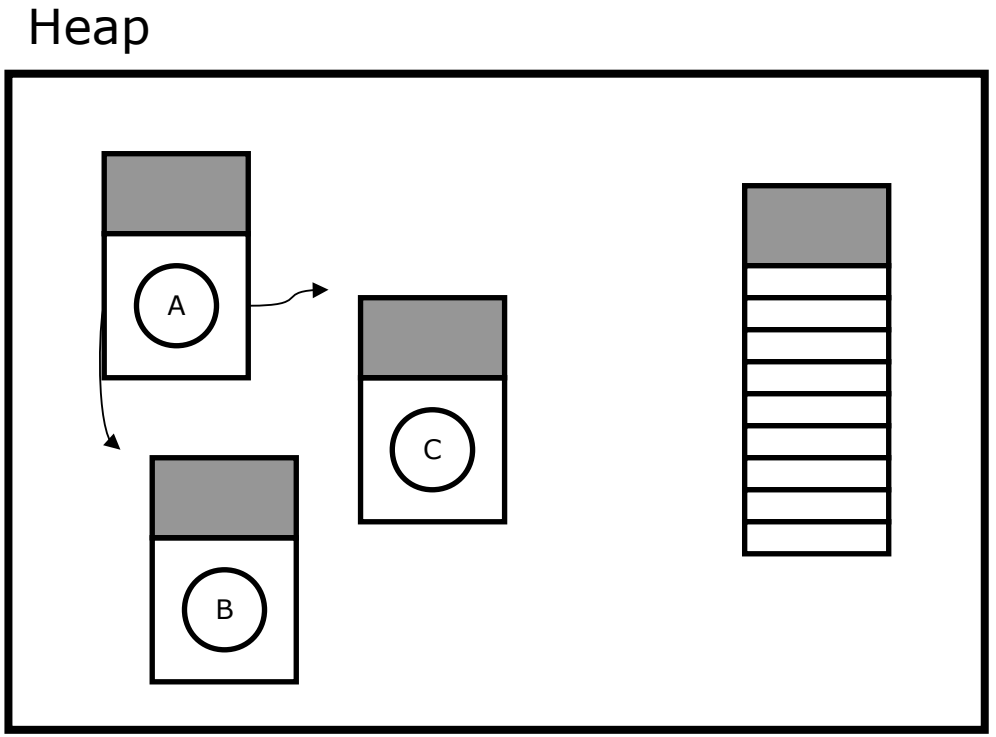
Let's look at transferring data



Let's look at transferring data

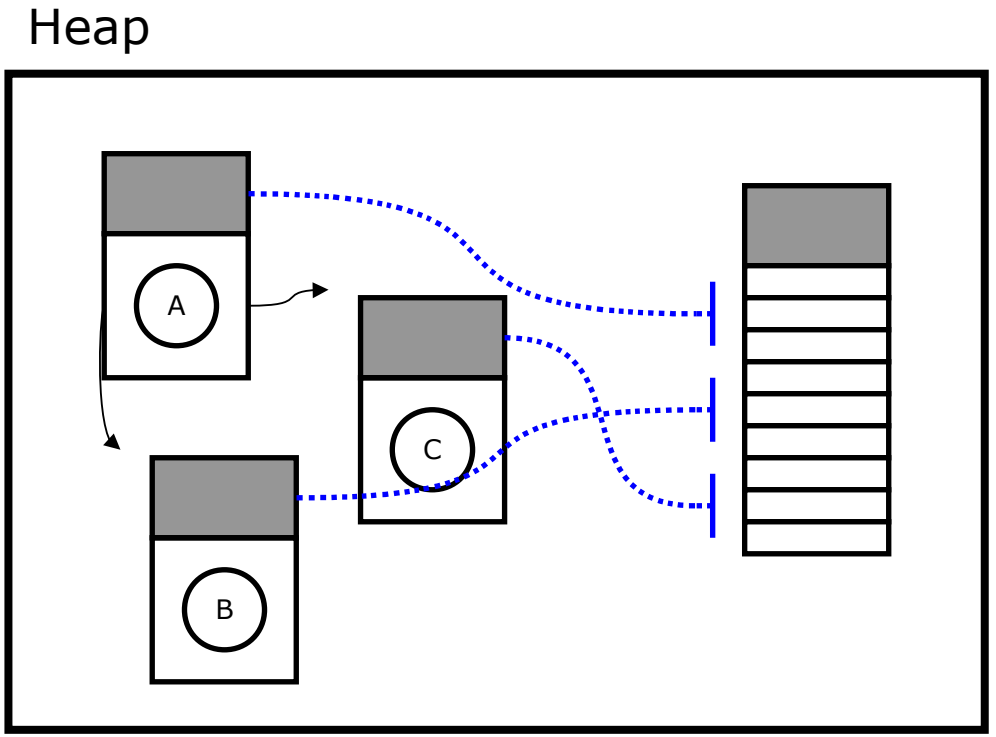


Let's look at transferring data



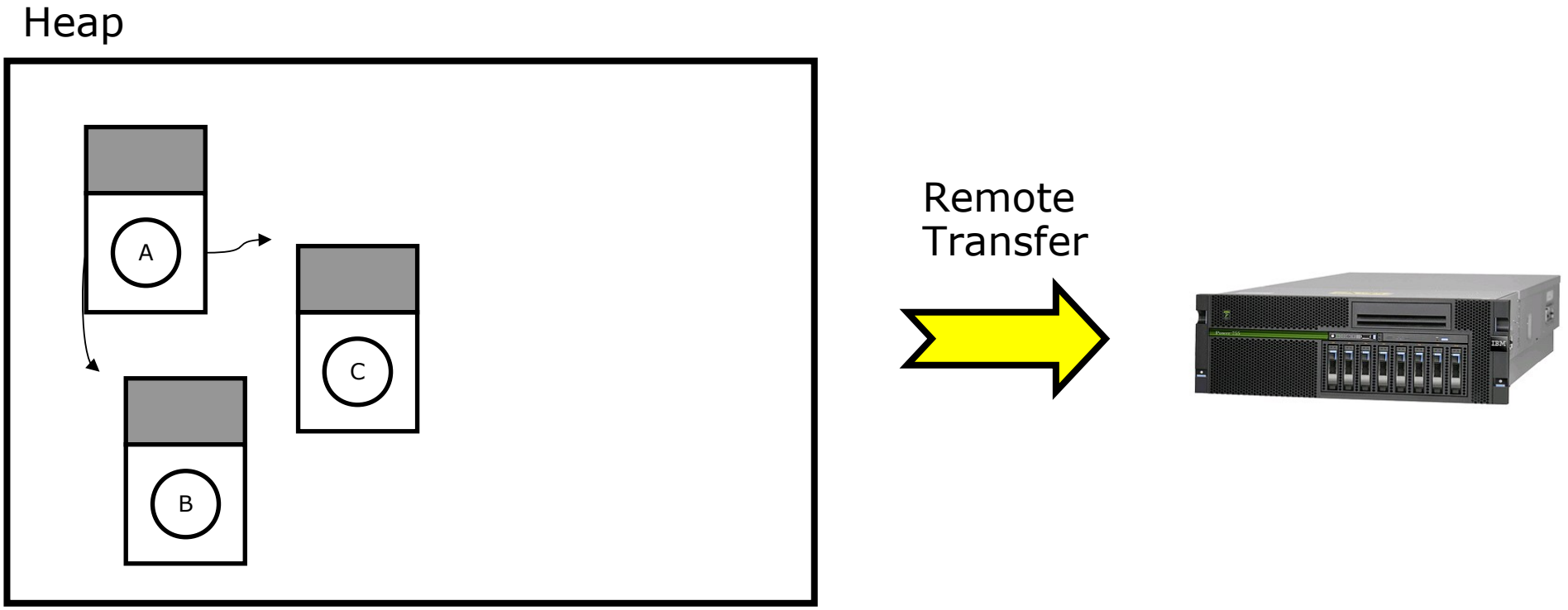
Remote Transfer

Let's look at transferring data

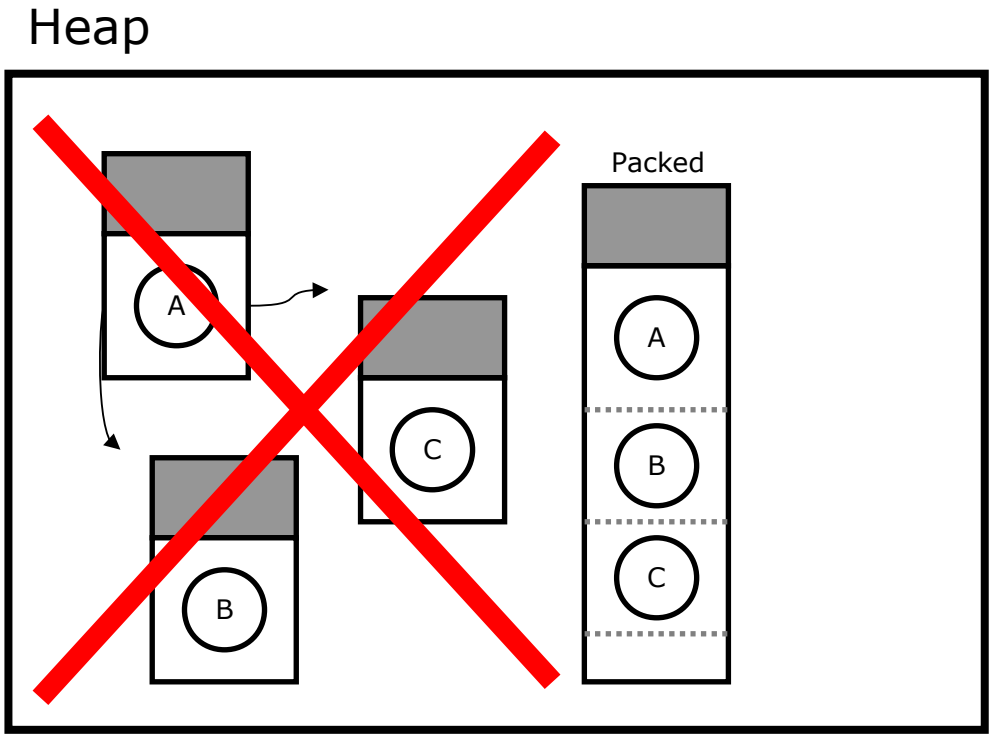


Remote Transfer

PackedObjects could help…

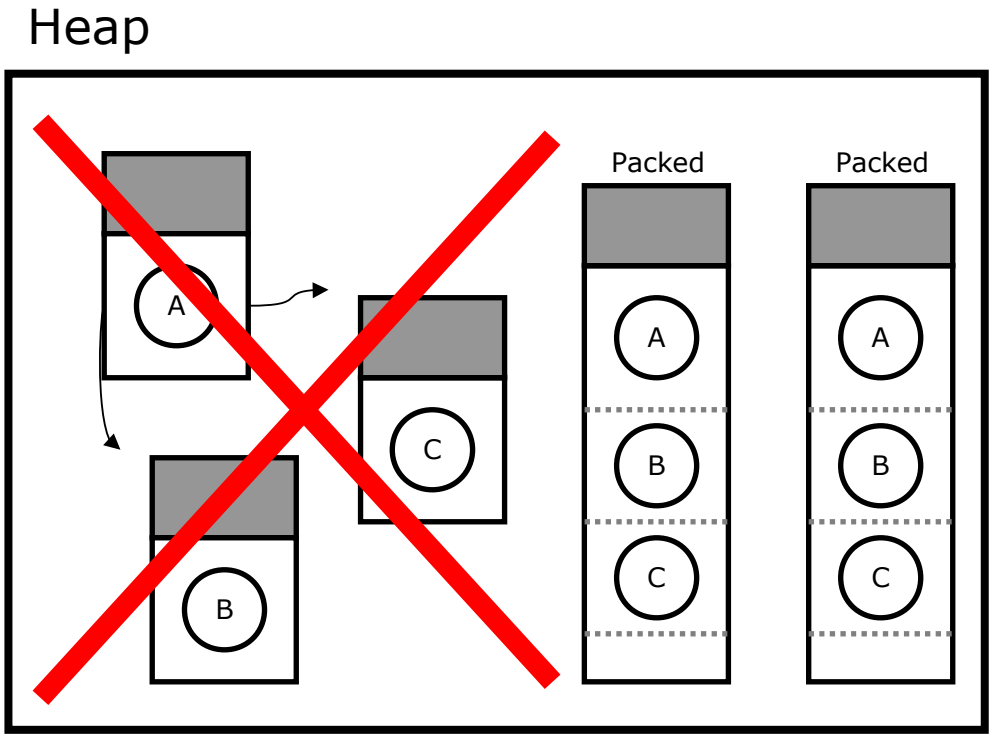


PackedObjects could help…



Remote Transfer

PackedObjects could help…

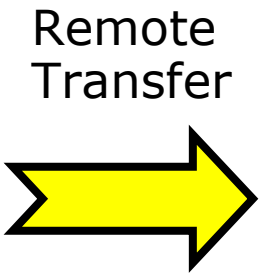


Remote Transfer

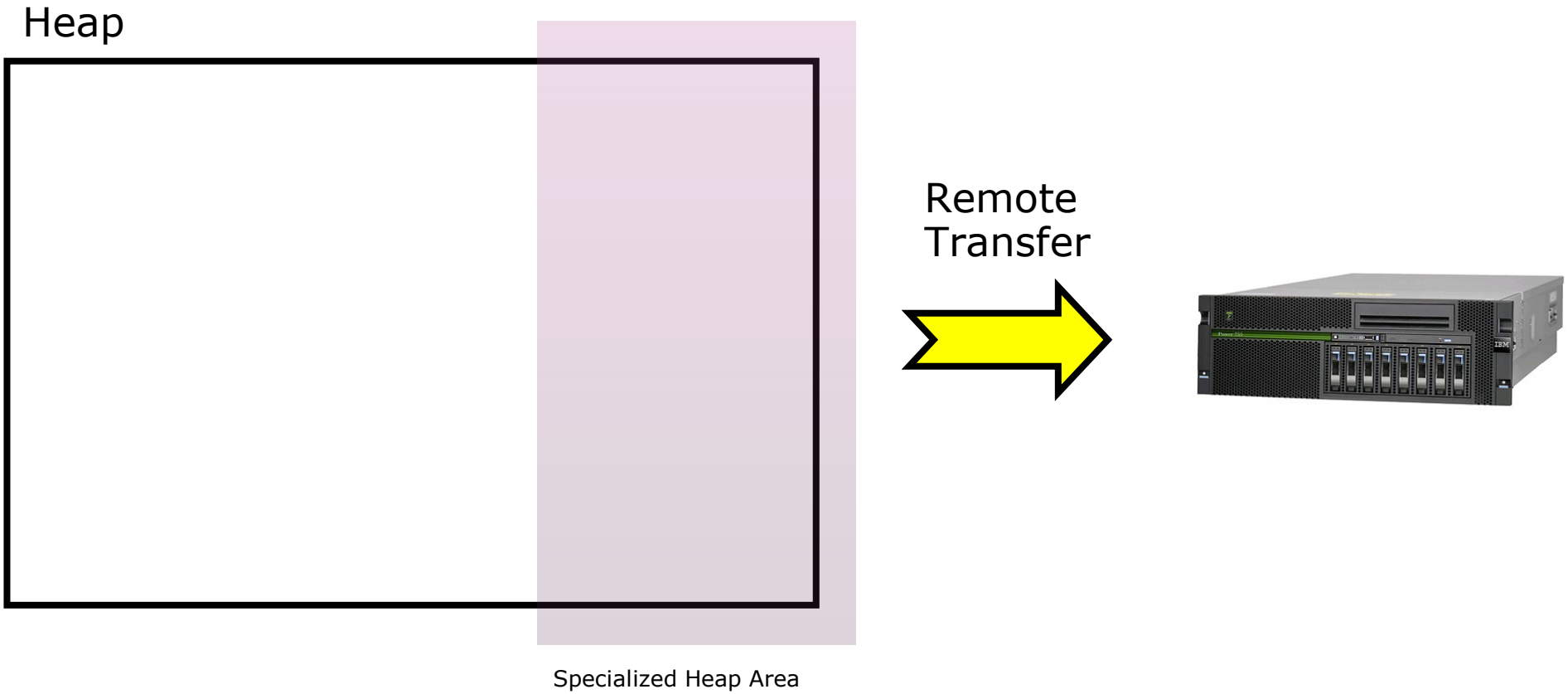
Making the data transfer easier…

Heap

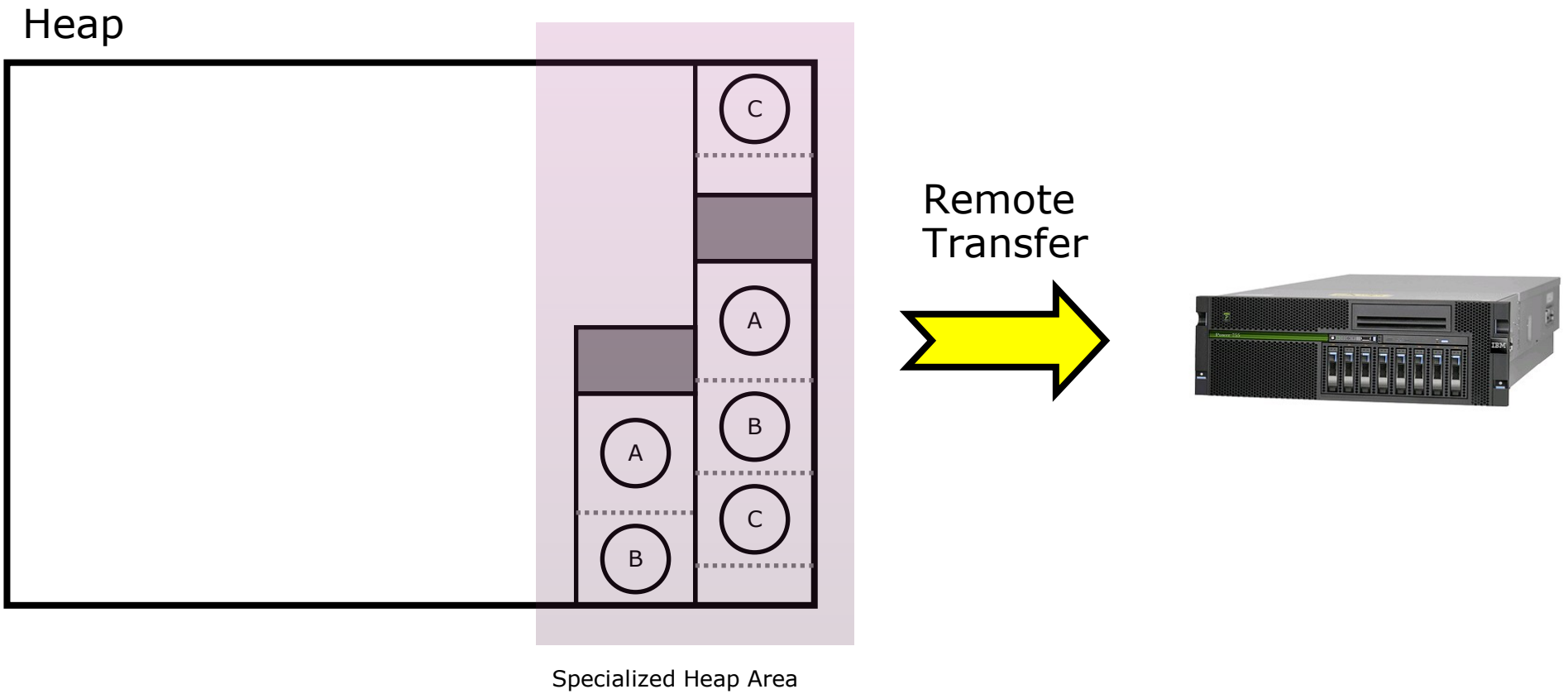
Remote
Transfer



Making the data transfer easier…

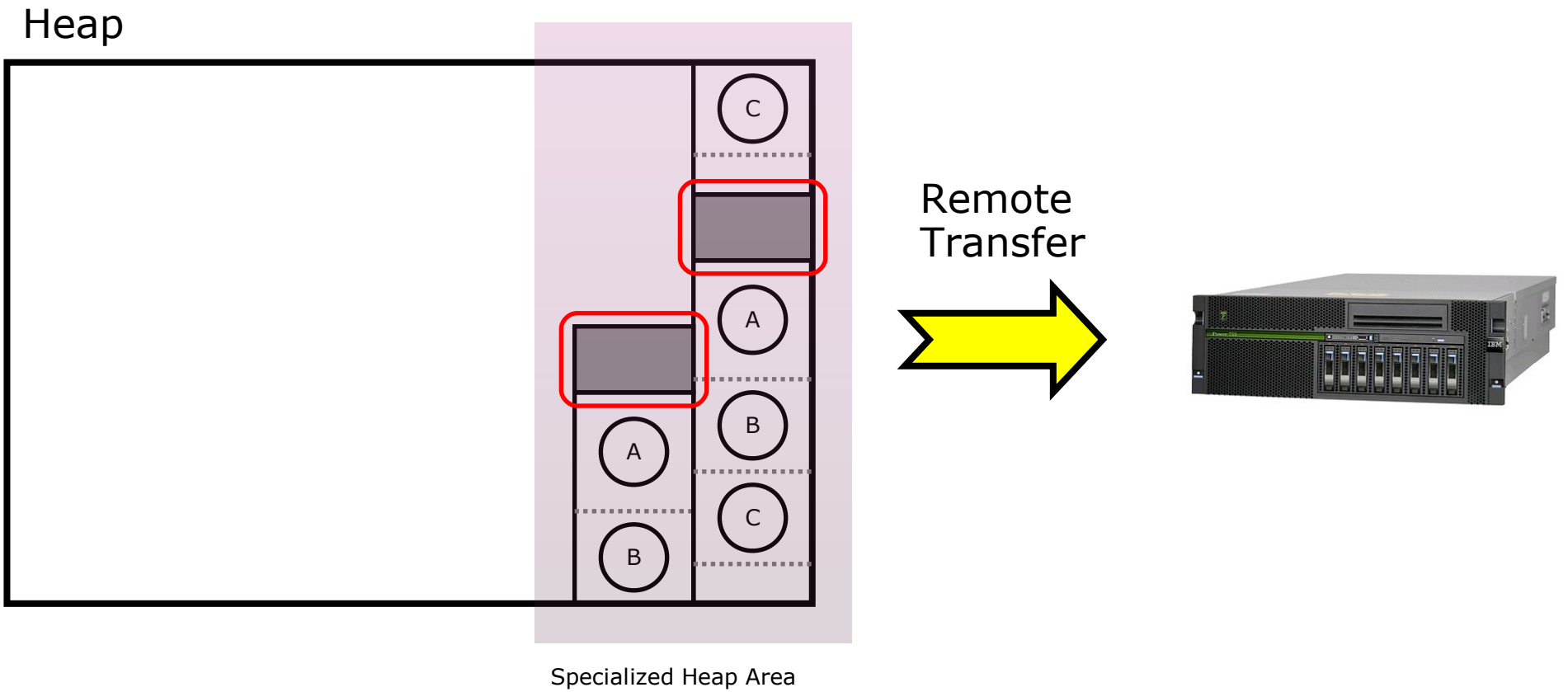


Making the data transfer easier…



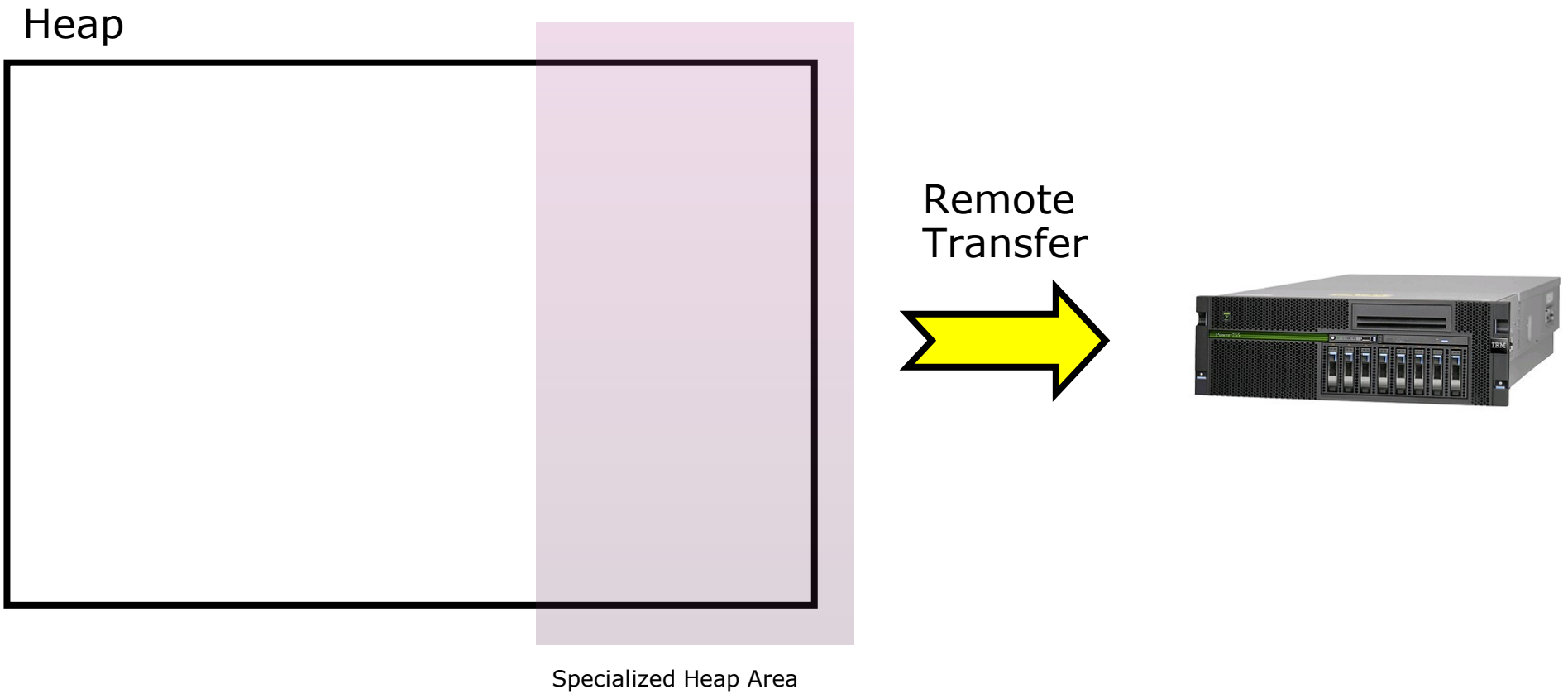
Specialized Heap Area

Making the data transfer easier…

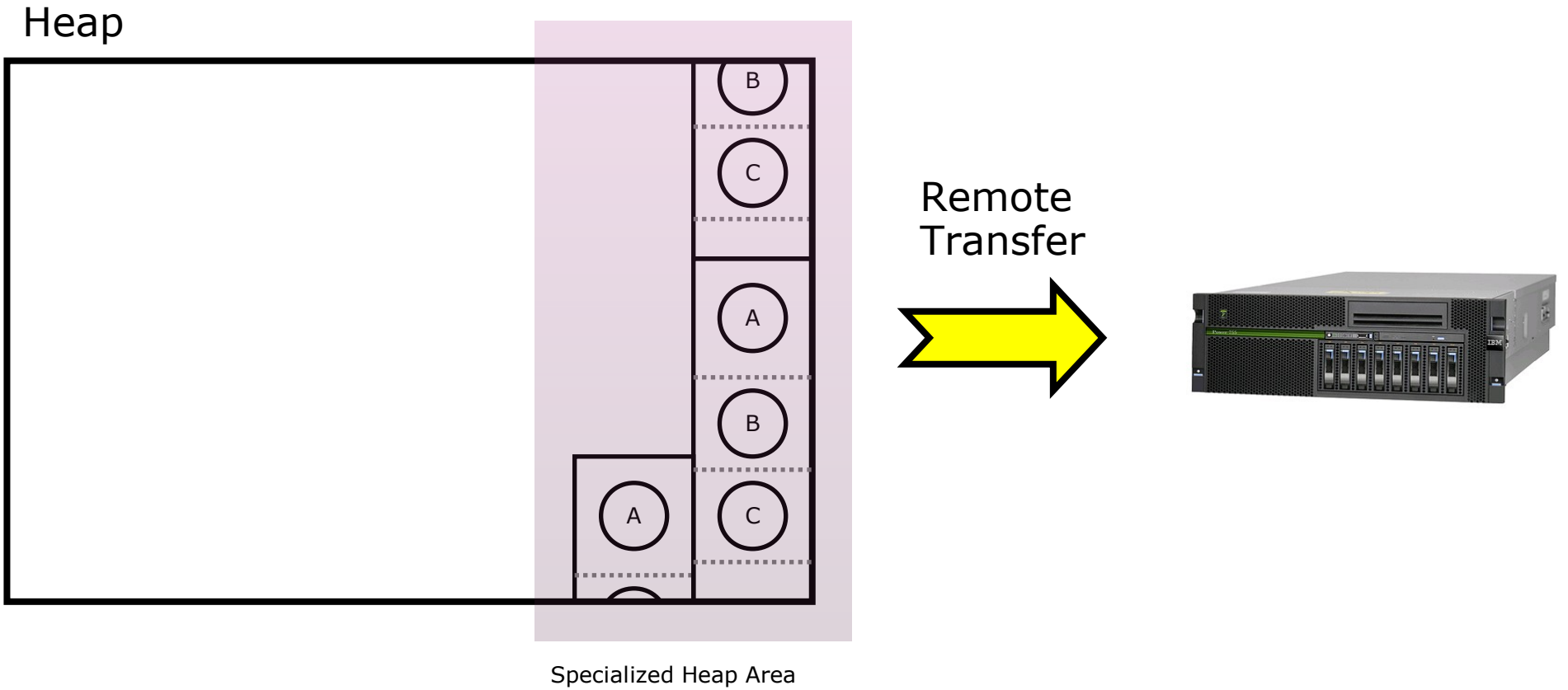


Specialized Heap Area

Making the data transfer seamless

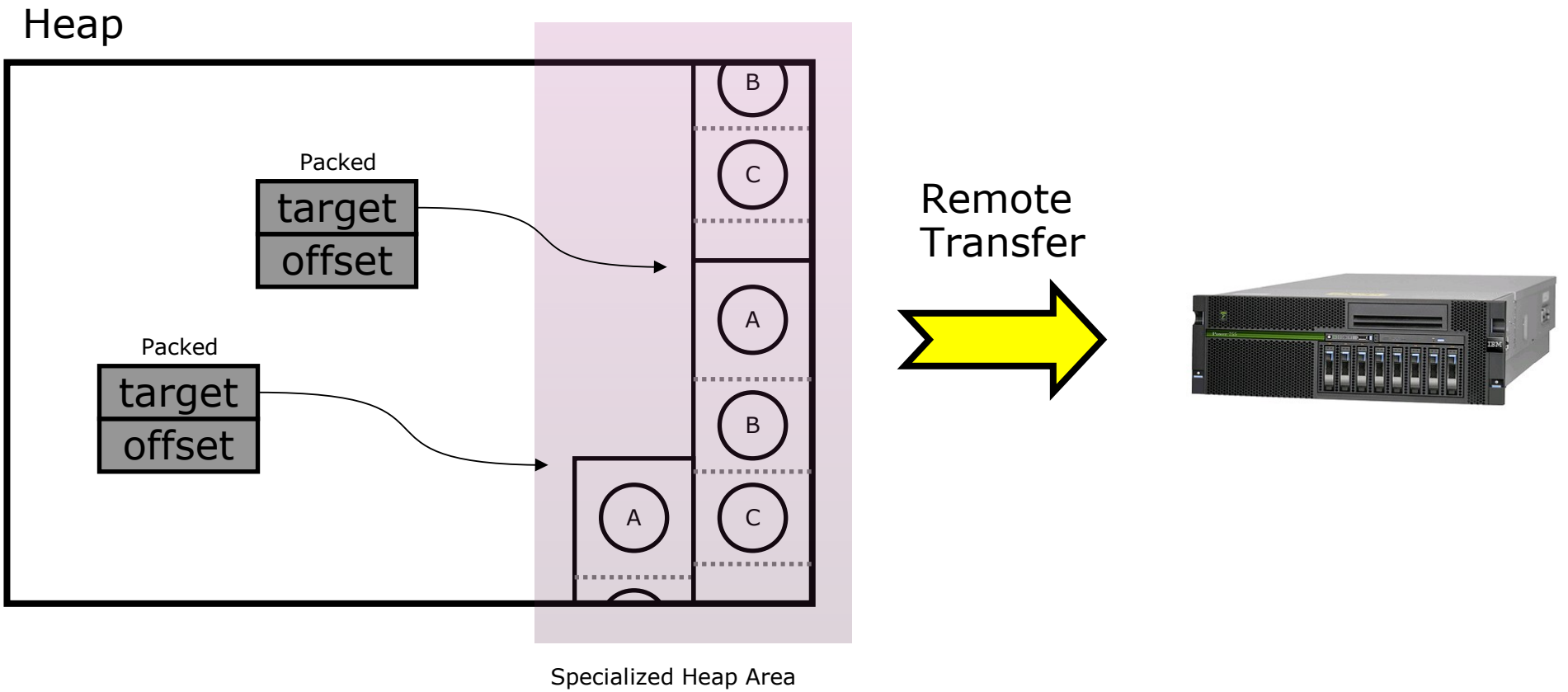


Making the data transfer seamless



Specialized Heap Area

Making the data transfer seamless



Questions?

References

- **Get Products and Technologies:**
  - IBM Java Runtimes and SDKs:
    - https://www.ibm.com/developerworks/java/jdk/
  - IBM Monitoring and Diagnostic Tools for Java:
    - https://www.ibm.com/developerworks/java/jdk/tools/
- **Learn:**
  - IBM Java InfoCenter:
    - http://publib.boulder.ibm.com/infocenter/java7sdk/v7r0/index.jsp
- **Discuss:**
  - IBM Java Runtimes and SDKs Forum:
    - http://www.ibm.com/developerworks/forums/forum.jspa?forumID=367&start=0

Copyright and Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., and registered in many jurisdictions worldwide.

Other product and service names might be trademarks of IBM or other companies.

A current list of IBM trademarks is available on the Web – see the IBM “Copyright and trademark information” page at URL: www.ibm.com/legal/copytrade.shtml