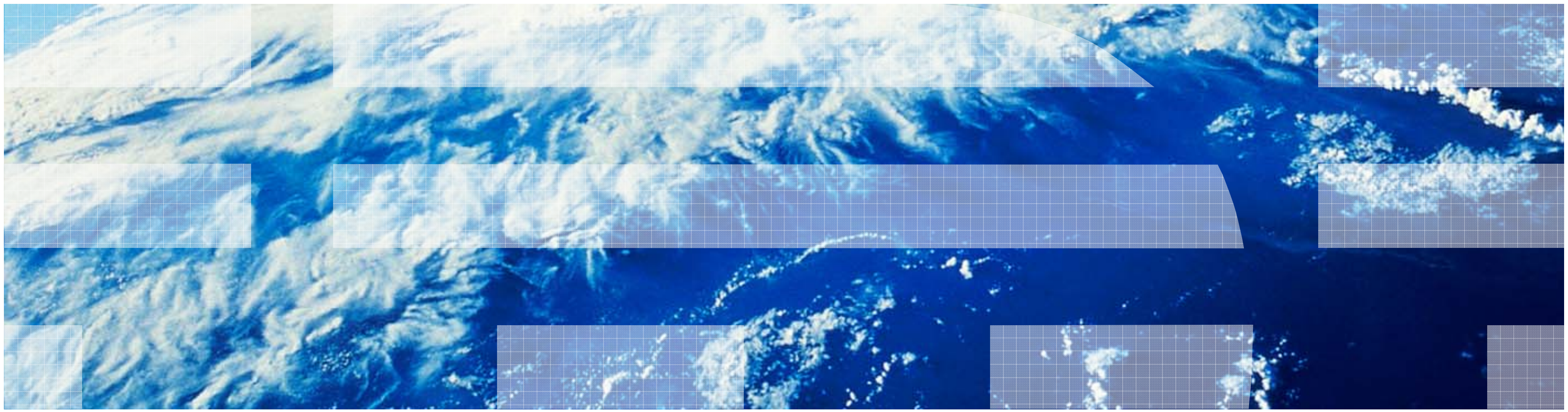
Compiling X10 to Java

Mikio Takeuchi, Yuki Makino[†], Kiyokuni Kawachiya, Hiroshi Horii,
Toyotaro Suzumura, Toshio Suganuma, and Tamiya Onodera



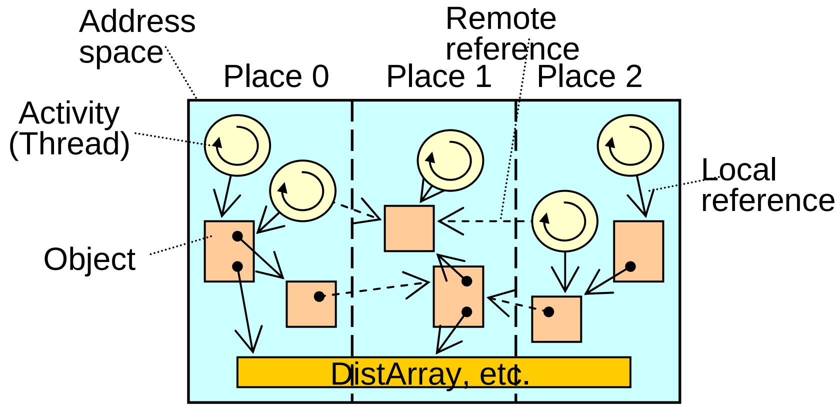
IBM Research – Tokyo [†]IBM Yamato Software Development Laboratory

Outline

- Value Proposition of Managed X10
- Challenges in Compiling X10 to Java
  - Types
  - Generics
  - Arrays
- Experimental Results
  - Sequential Benchmarks
  - Parallel Benchmarks
  - Distributed Benchmarks
- Conclusions
- Future Work

“Managed X10” (X10 on Java VMs)

- **Managed X10** is an implementation of X10 on Java
  - Is a tool for Java programmers to easily scale-out their existing applications with built-in distributed execution feature and better inter-operability with Java.

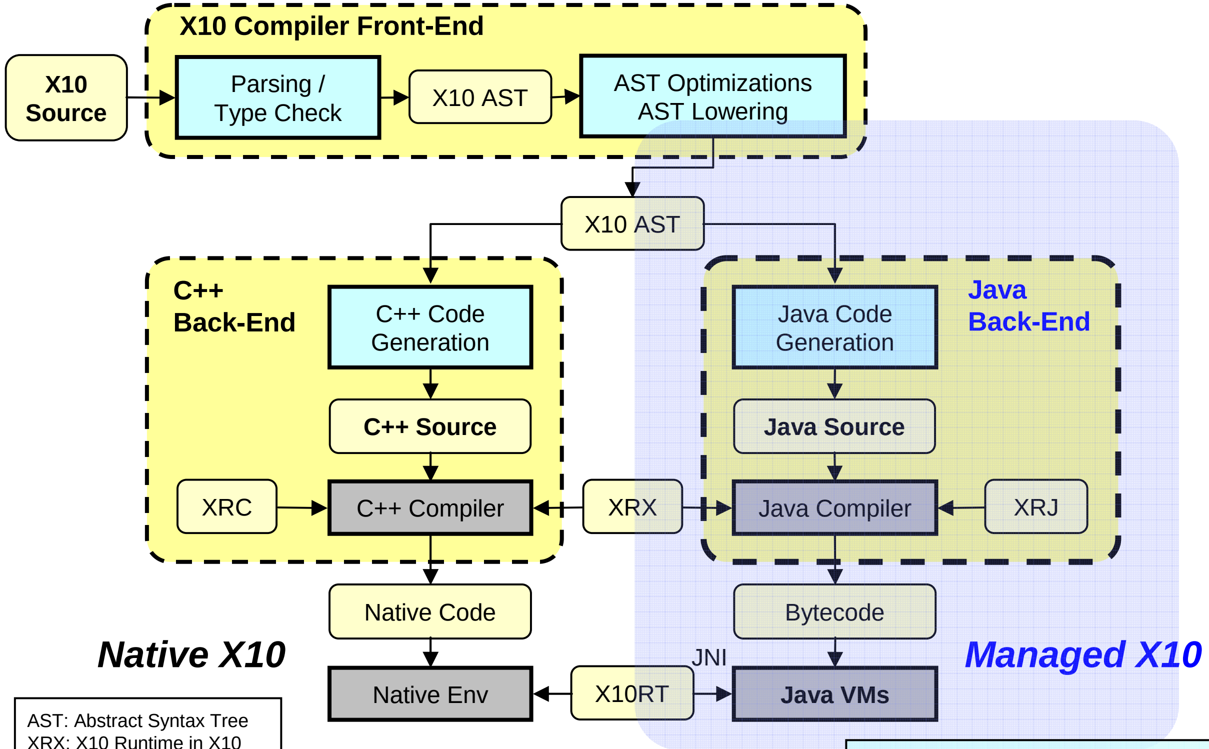


APGAS programming model

```
class MyHello {
  public static def main(Array[String]) {
    finish for (pl in Place.places()) {
      async at (pl) Console.OUT.println(
         "Hello World from place " + here.id);
}}}
```

```
$ x10c MyHello.x10          # Compile
$ x10 MyHello               # Run
Hello World from place 3
Hello World from place 0
Hello World from place 1
Hello World from place 2
```

Hello World in X10



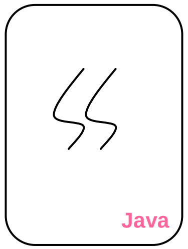
AST: Abstract Syntax Tree
XRX: X10 Runtime in X10
XRJ: X10 Runtime in Java
XRC: X10 Runtime in C++
X10RT: X10 Comm. Runtime

X10 compilation flow

X10 program is compiled into Java source, and executed on Java VMs

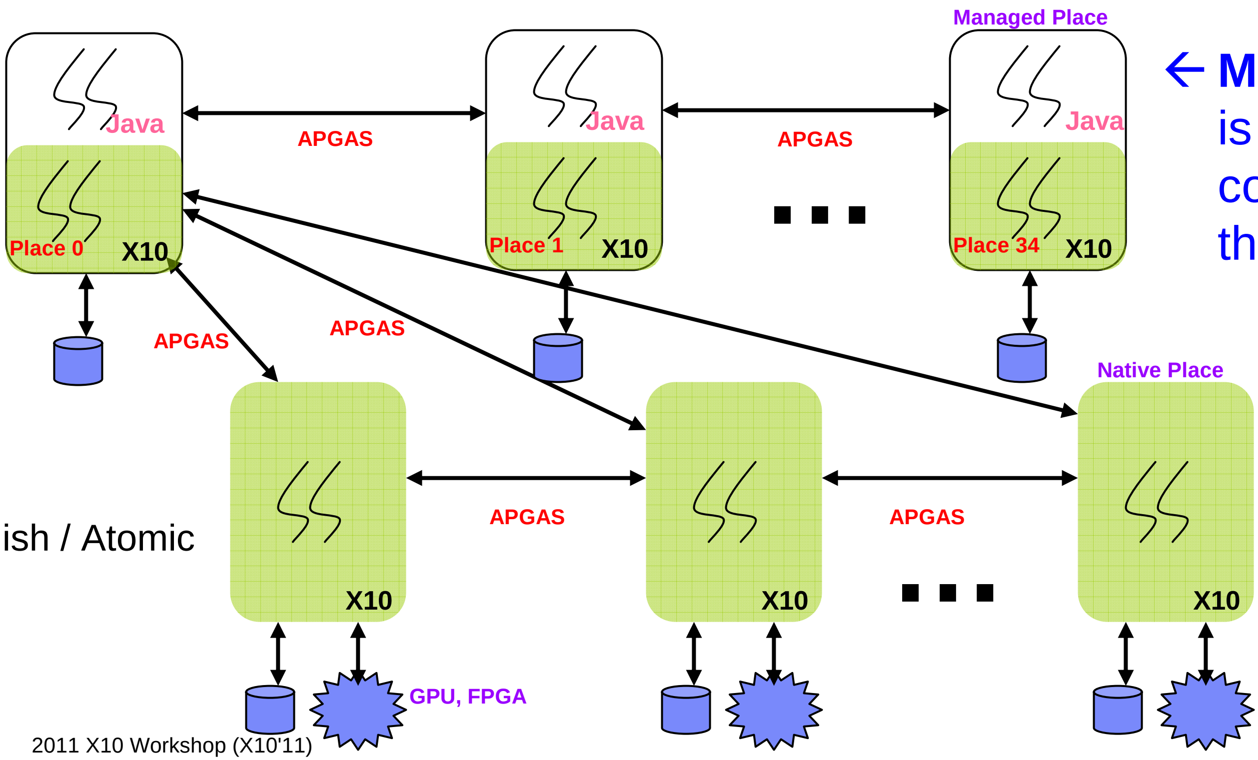
X10 as a co-language for Java – for acceleration and scale-out

- **In Java ...**



- How do you deal with petabytes of data?
- How do you take advantage of GPUs and FPGAs?

- **In X10** – Performance and Productivity at Scale



← **Managed X10** is one of key components in this architecture

X10 features

- Places
- Asyncs / Finish / Atomic
- Clocks
- ...

A Sample X10 Program

Java-like syntax, but uses var, val, def, ...

Rich array class

X10

```
class Sample[T] implements (String)=>String {
  var data:T;
  def this(d:T) { data = d; } // constructor
  public operator this(str:String) = str + data;

  static struct MyPair[T,U](fst:T,snd:U) {
    public def toString() = "(" + fst + "," + snd +")";
  }

  public static def main(args:Array[String](1)) {
    /* Class example */
    val o = new Sample[Double](1.2);
    Console.OUT.println(o.data);          // -> 1.2
    Console.OUT.println(o("Data is "));   // -> Data is 1.2
    var a:Any = o;
    Console.OUT.println(o.typeName());// -> Sample[x10.lang.Double]
    /* Struct example */
    val p = MyPair[Int,Double](1,2.3);
    Console.OUT.println(p);               // -> (1,2.3)
    val x = 4;
    /* Function example */
    val q = MyPair[(Int)=>Int, Int]((i:Int)=>i*x, 5);
    Console.OUT.println(q.fst(q.snd));   // -> 20
```

Operators can also be defined

X10

```
    /* Array example */
    val pt = [2,4] as Point;          // Point{rank==2}
    val R1 = (1..2)*(3..5);           // Region{rank==2}
    val arr = new Array[Int](R1); // Array[Int]{rank==2}
    arr(2,4) = 8;
    Console.OUT.println(arr(pt));  // -> 8
    /* Parallel processing */
    var m:Int = 0; val i = 1;         // mutable/immutable data
    finish async { m = i * 2; }
    Console.OUT.println(m);           // -> 2
    /* Distributed processing */
    at (here.next()) o.data = 3.4;  // copy of o is modified
    Console.OUT.println(o.data); // -> 1.2
    /* GlobalRef example */
    val g = GlobalRef(o);
    at (here.next()) { at (g.home) g().data = 5.6; }
    Console.OUT.println(o.data); // -> 5.6
  }
}
```

Parallel/distributed processing

Global data access

Strong type system – type parameters are not erased

Primitive types (Int, Double, ...) are defined as structs

New 1st class data types, structs and functions

Challenges in Compiling X10 to Java

- **Types** – mapping of X10 data types to Java
  – How to translate X10 types (incl. structs and functions) to Java?
  – How to utilize Java primitive types?

- **Generics** – gaps between X10 generics and Java generics
  – How to implement X10's richer generics semantics?

- **Arrays** – optimizations of array access
  – How to make X10 Array performance comparable to Java?

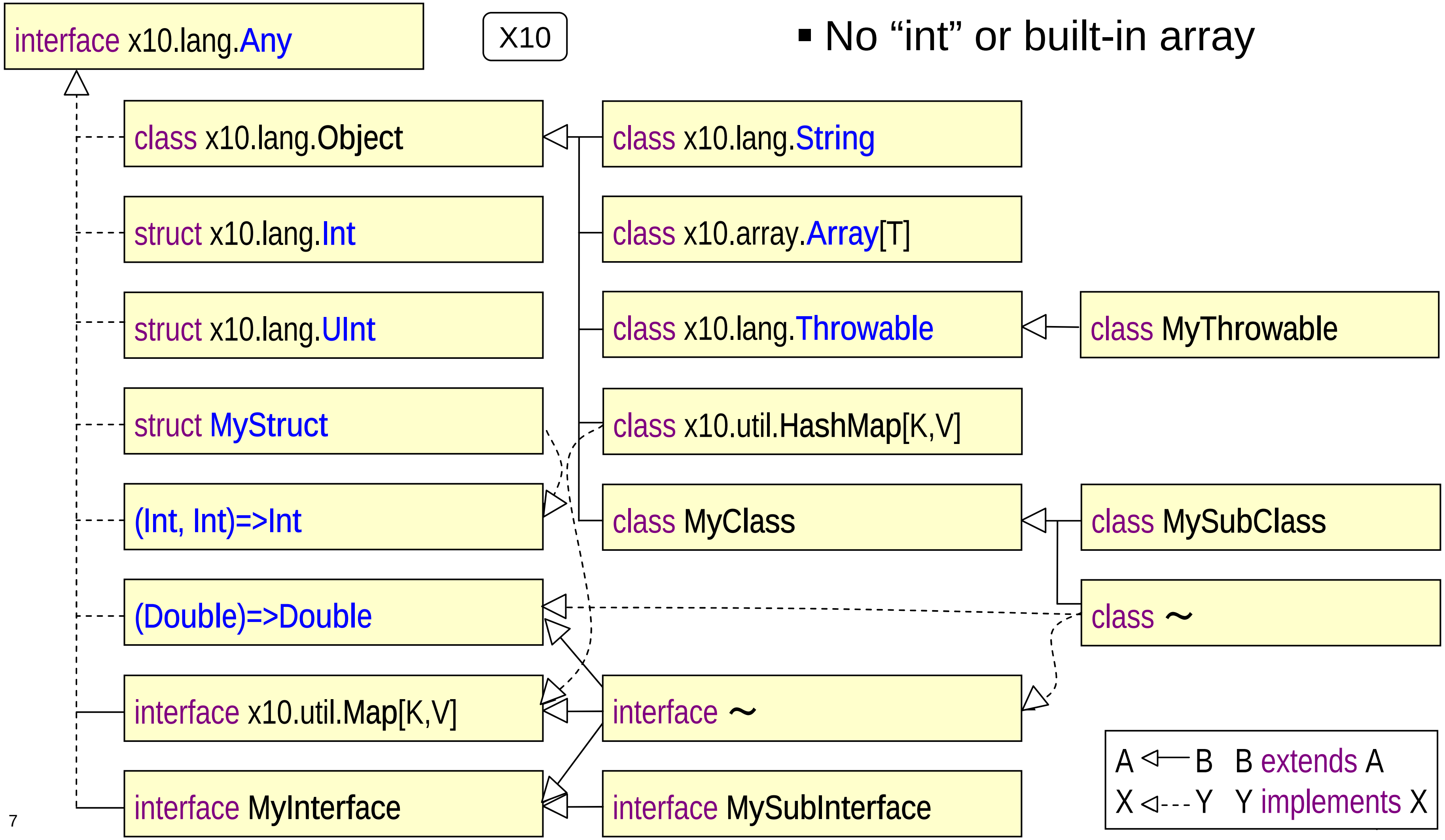
- **Places** – distributed execution
  – How to utilize multiple computer nodes to run an X10 application?

- and others ...

X10 Types

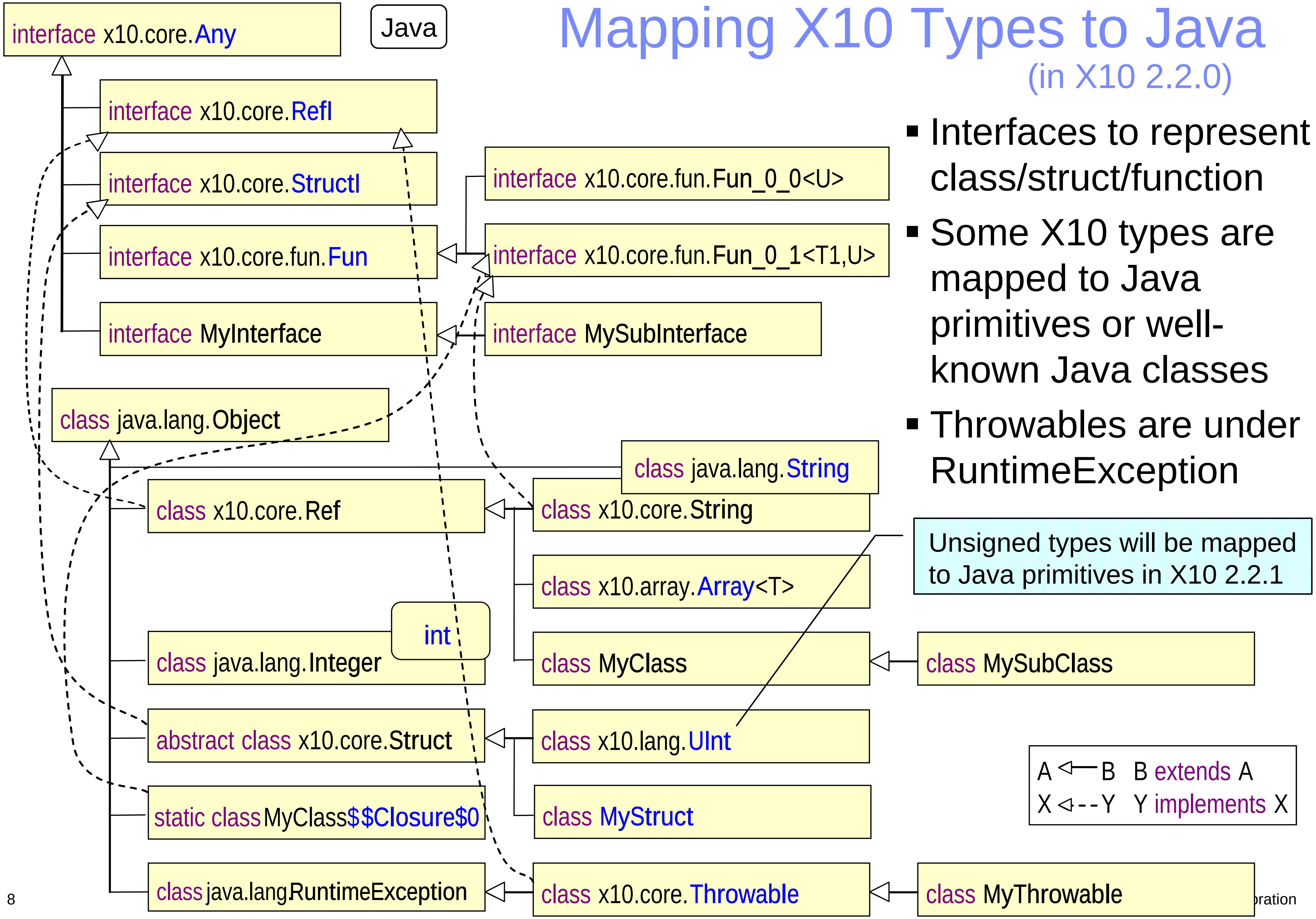
- X10 has richer types than Java, and they need to be mapped to Java

- Interface "Any" as a top-level
- Structs and functions
- Int (and UInt, ...) are structs
- No "int" or built-in array



Mapping X10 Types to Java
(in X10 2.2.0)

- Interfaces to represent class/struct/function
- Some X10 types are mapped to Java primitives or well-known Java classes
- Throwables are under RuntimeException



Example of Compilation

X10

```
class C(p:Int) {
  val q:Int; var r:Int;
  def u() { }
  private def v() { }
  def w() { return super.hashCode(); }
  def this(p:Int) {
    property(p);
    q = 1;
  }
  static val s = Place.MAX_PLACES;
  static def t() { }
}
```

x10c

Java

```
import x10.lang.Place;
import x10.lang.Runtime;
import x10.runtime.impl.java.InitDispatcher;
public class C extends x10.core.Ref {
  public int p;                              // property
  public int q, r;                           // instance fields
  final public int p() { return this.p; }    // property method
  public void u() { }                        // instance methods
  private void v() { }
  // bridge method for private instance method
  public static void v$P(final C C) { C.v(); }
  public int w() { return super.hashCode(); }
  // bridge method for super.hashCode()
  final public int C$hashCode$S() { return super.hashCode(); }
  public C(final int p) {                    // constructor
    super();
    this.p = p;
    this.__fieldInitializers173();
    this.q = 1;
  }
  // instance field initializer
  final private void __fieldInitializers173() { this.r = 0; }
  // bridge method for field initializer
  final public static void __fieldInitializers173$P(final C C) {
    C.__fieldInitializers173();
  }
  public static int s = 0;                   // static field
  public static void t() { }                 // static method
  // static initialization
  public static int fieldId$s;               // static field id
  // static field initializer
  public static int getInitialized$s() {
    if (Runtime.hereInt() == 0) {
      C.s = Place.getInitialized$MAX_PLACES();
      InitDispatcher.broadcastStaticField(C.s, C.fieldId$s);
    }
    return C.s;
  }
  // static field deserializer
  public static void getDeserialized$s(byte[] buf) {
    C.s = (Integer) InitDispatcher.deserializeField(buf);
  }
  static {                                   // Java static initializer
    C.fieldId$s = InitDispatcher.addInitializer("C", "s");
  }
}
```

(some lines are omitted)

- Class/field names are basically same
- Constructor is separated into Java constructor and field initializer
- For private instance method, additional bridge method is generated
  - to support inlining by front-end
- Static-field initializer and deserializer are generated
  - to support multi-place initialization

Static Initialization

- In X10, static fields are immutable and have the same value in all places.
- General approach: evaluate static fields in place 0 and broadcast them to all other places.
  - Difficulty of the use of final keyword impacts performance.
- Optimization: evaluate as many static fields as possible in each place if compiler can determine it is safe.

X10

```
public class C {
  static val t = System.currentTimeMillis();
  static val m = "Welcome to " + "X10 world!";
}
```

x10c

Java

```
import x10.lang.Place;
import x10.lang.Runtime;
import x10.runtime.impl.java.InitDispatcher;
public class C extends x10.core.Ref {
 // method call will be evaluated in place 0
 public static long t;
 public static int fieldId$t;
 public static void getDeserialized$t(byte[] buf) {
   C.t = (Long) InitDispatcher.deserializeField(buf);
 }
 public static long getInitialized$t() {
   if (Runtime.hereInt$O() == 0) {
     C.t = x10.lang.System.currentTimeMillis$O();
     InitDispatcher.broadcastStaticField(C.t, C.fieldId$t);
   }
   // wait until initialized (omitted)
   return C.t;
 }
 static {
   C.fieldId$t = InitDispatcher.addInitializer("C", "t");
 }
 // constant will be evaluated in each place
 public static final String m = "Welcome to X10 world!";
 public static String getInitialized$s() { return C.m; }
}
```

(some lines are omitted)

- X10 also requires all static fields must be initialized successfully before the main method is executed.
- Current approach: preload all X10 classes.
  - Large footprint
  - Massive class loading in a short time has impact to some JIT compilers (e.g. IBM J9)

Generics

- X10 generics must be implemented by **type reification**
  - Type parameters are kept for each instance, like in C++ templates
    - e.g. "new Sample[Int](1).typeName()" returns "Sample[x10.lang.Int]"
- However, Java generics are implemented by **type erasure**
  - Type parameters are checked and erased by javac

➔ Quiz: How this X10 code can be translated to Java?

X10

```
interface I[T] {
  def m(T):T;
}
class B[T] {
  def m(a:T):T {return a;}
}
class C[T1,T2] extends B[Int] implements I[String], I[Int] {
  def this(T1){}
  def this(T2){}
  def m(a:T1){return a;}
  def m(a:T2){return a;}
  public def m(a:String){return a;}
  public def m(a:Int){return a;}
}
```

How to implement both I[String] and I[Int]?

How to overload m(T1) and m(T2)?

How to pass primitive (unboxed) data to get better performance, while allowing access through B[Int]

Generics Compilation

X10

```
interface I[T] {
  def m(T):T;
}
class B[T] {
  def m(a:T):T {return a;}
}
class C[T1,T2] extends B[Int] implements I[String], I[Int] {
  def this(T1){}
  def this(T2){}
  def m(a:T1){return a;}
  def m(a:T2){return a;}
  public def m(a:String){return a;}
  public def m(a:Int){return a;}
}
```

x10c

Java

```
import x10.rtt.RuntimeType;
import x10.rtt.Type;
import x10.rtt.Types;
interface I<T> {
  public static final RuntimeType<I> $RTT = ...
  Object m(T id$0, Type t1);
}
public class B<T> extends x10.core.Ref {
  public static final RuntimeType<B> $RTT = ...
  private Type T;
  public T m_0_$$B_T$G(T a) {return a;}
}
public class C<T1, T2> extends B<Integer> implements I {
  public static final RuntimeType<C> $RTT = ...
  private Type T1, T2;
  // dispatcher for abstract public I.m(id$0:T):T
  public Object m(Object a1, Type t1) {
   if (t1.equals(Types.STRING)) {
     return m((String) a1);
   } else if (t1.equals(Types.INT)) {
     return m((int)(Integer) a1);
   }
   return null;
  }
  // bridge for B.m(a:T):T
  public Integer m_0_$$B_T$G(Integer a1) {return m((int) a1);}
  // constructors need signature mangling
  public C(Type T1, Type T2,
              T1 id$1, Class $dummy0) {...}
  public C(Type T1, Type T2,
              T2 id$2, Class[] $dummy0) {...}
  // generic methods need signature mangling
  public T1 m_0_$$C_T1$G(T1 a) {return a;}
  public T2 m_0_$$C_T2$G(T2 a) {return a;}
  // instantiated generic methods
  public String m(String a) {return a;}
  public int m(int a) {return a;}
}
```

X10 generics are mapped to Java generics, but ...

Additional fields (T1,T2) are generated to hold type parameters

Static field "$RTT" holds X10-level class info

Dispatch method is generated for overloading

Method name is modified to include type parameters

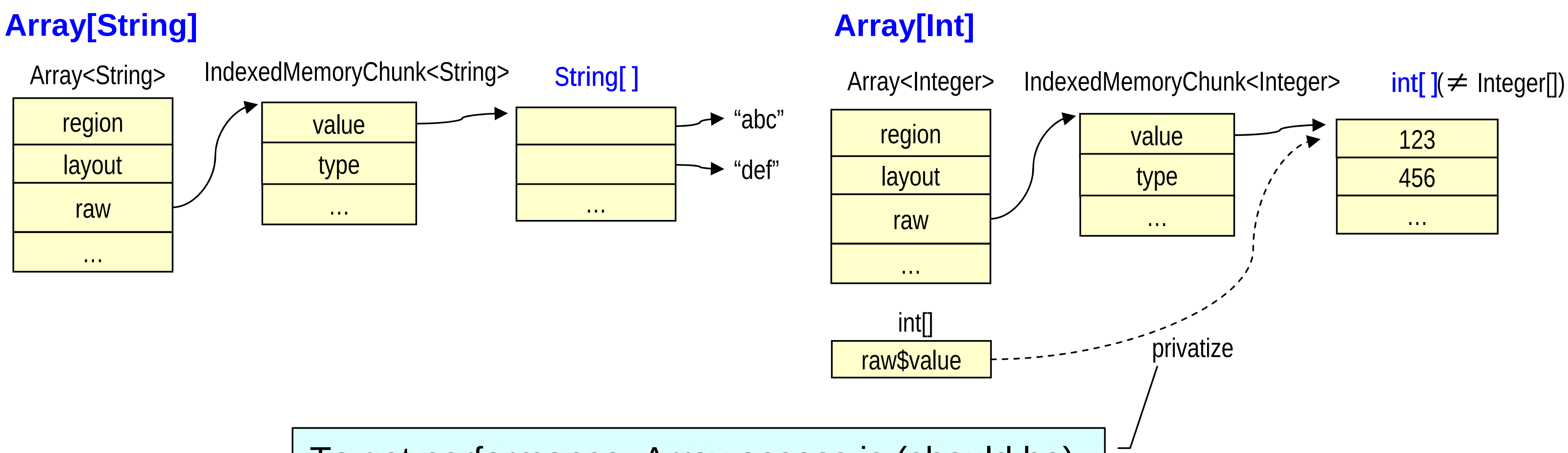
Primitives are used as much as possible, and bridge method is generated to access it through boxed types

Arrays

X10

```
val arr = new Array[Int](3..5);
for ([i] in 3..5) arr(i) = i
var s:Int = 0;
for (pt in arr) s += arr(pt)*arr(pt);
Console.OUT.println(s);  // -> 50
```

- X10 array is not a built-in data type, but is a class
  - It is generic, multi-dimensional, and sparse array
- An X10 array instance consists of three objects
  - **Array**[T] holds array attributes
  - **IndexedMemoryChunk**[T] represents a contiguous 1-dimensional array
  - Actual **Java array** (e.g. String[ ] or int[ ]) to hold array elements



To get performance, Array access is (should be) inlined and privatized in some situations

Array Access (e.g. Array[Int]{rail})

Array.x10

```
public final class Array[T]
  (region:Region{self!=null}, rank:Int, rail:Boolean)  {…}
  implements (Point(rank))=>T, Iterable[Point(rank)] {
  private val raw:IndexedMemoryChunk[T];
  private val layout:RectLayout;
  public operator this(i0:Int){rank==1}:T {
    if (rail) {
      return raw(i0);
    } else {
      if (CompilerFlags.checkBounds() && !region.contains(i0)) {
        raiseBoundsError(i0);
      }
      return raw(layout.offset(i0));
  } }
  public operator this(i0:Int)=(v:T){rank==1}:T{self==v} {
    if (rail) {
      raw(i0) = v;
    } else {
      if (CompilerFlags.checkBounds() && !region.contains(i0)) {
        raiseBoundsError(i0);
      }
      raw(layout.offset(i0)) = v;
    }
    return v;
  } }
}
```

Extra overhead in X10
- 3 field accesses
- 2 virtual calls
- 1 cast
- 1 boxing

IndexedMemoryChunk.x10

```
@NativeRep("java", "x10.core.IndexedMemoryChunk<#T$box>", ...)
public struct IndexedMemoryChunk[T] {
  @Native("java", "(#this).$apply$G(#index)")
  public native operator this(index:Int):T;
  @Native("java", "(#this).$set(#index, #value)")
  public native operator this(index:Int)=(value:T):void;
}
```

IndexedMemoryChunk.java

```
public final class IndexedMemoryChunk<T> extends x10.core.Struct {
  public int length;
  public Object value;
  public Type<T> type;
  public T $apply$G(int i) { return type.getArray(value, i); }
  public void $set(int i, T v) { type.setArray(value, i, v); }
}
```

virtual call
field/array access

int[ ] (≠Integer[])

0	1	…	i

IntType.java

```
public class IntType extends RuntimeType<Integer> {
  public Integer getArray(Object array, int i) {
    return ((int[]) array)[i];
  }
  public void setArray(Object array, int i, Integer v) {
    ((int[]) array)[i] = v;
  }
}
```

Optimized Array Access

X10

```
val arr: Array[Int]{rail} = ...
var sum:Int = 0;
for (var i:Int = 0; i < arr.size; ++i) {
    sum += arr(i);
}
```

x10c

Java

```
x10.array.Array<Integer> arr = ...
int sum = 0;
for (int i = 0; i < arr.size; i = i + 1) {
    sum = sum + (int) arr.$apply$G(i);
}
```

Extra overhead in X10
3 field accesses
2 virtual calls
1 cast
1 boxing

+ Operator inlining

Java

```
x10.array.Array<Integer> arr = ...
int sum = 0;
for (int i = 0; i < arr.size; i = i + 1) {
  sum = sum + ((int[]) arr.raw.value)[i];
}
```

Extra overhead in X10
2 field accesses
1 cast

+ Privatization of Java backing array

Java

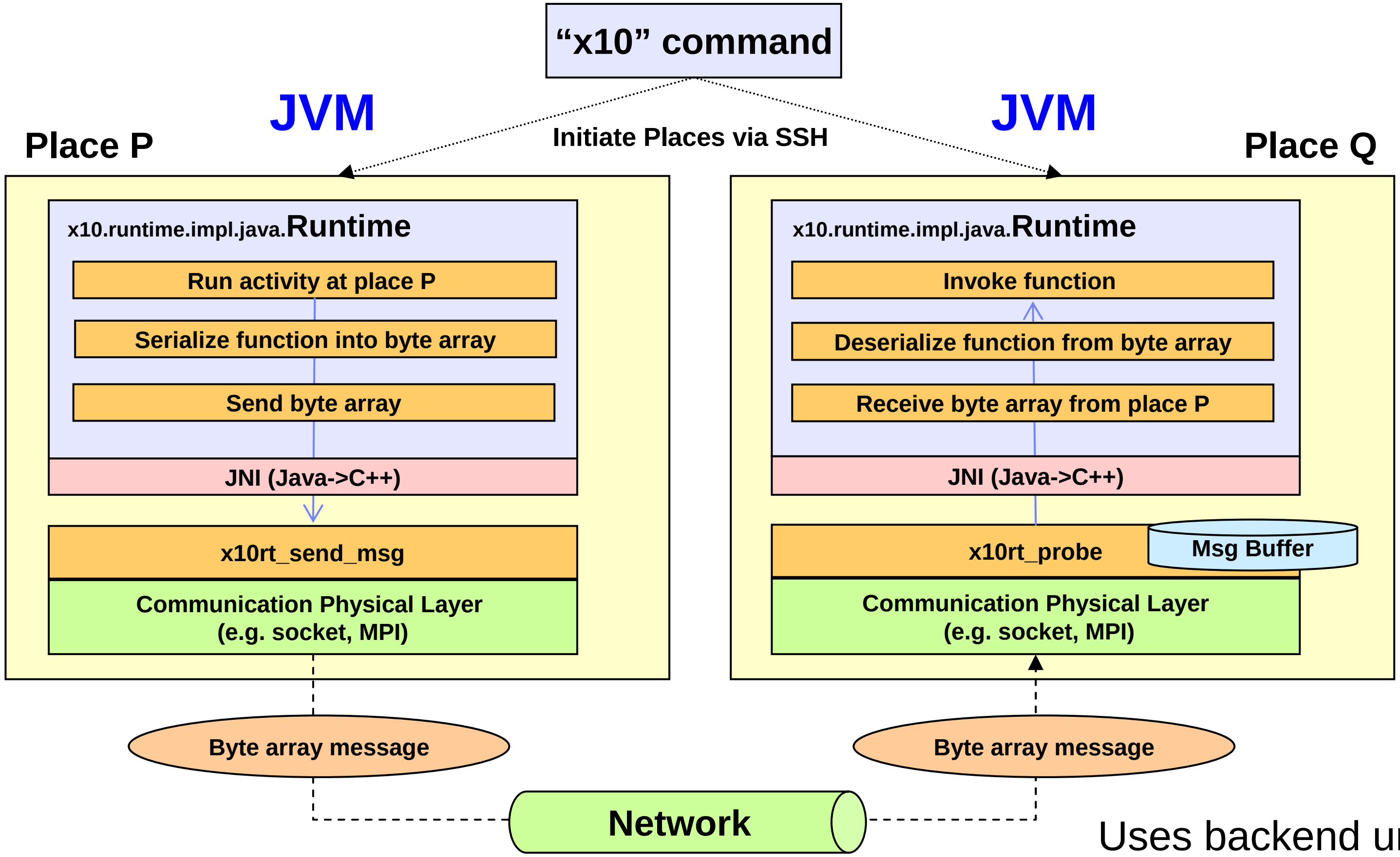
```
x10.array.Array<Integer> arr = ...
int sum = 0;
int[] arr$raw$value = (int[]) arr.raw.value;
for (int i = 0; i < arr.size; i = i + 1) {
  sum = sum + arr$raw$value[i];
}
```

Extra overhead in X10
None!

We have implemented operator inlining and privatization for IndexedMemoryChunk but not for Array.

Distributed Execution

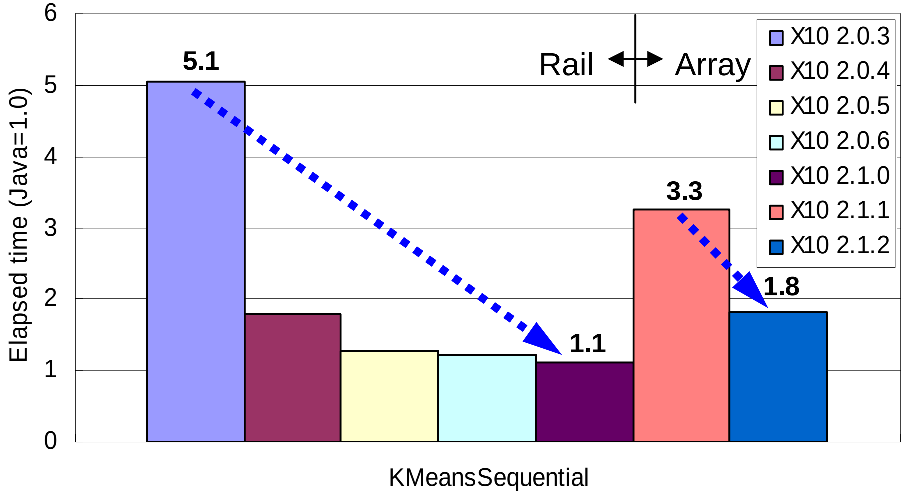
- Multi-JVM is supported in X10 2.1.2
  - Each X10 "Place" uses its own Java VM
  - Uses common X10RT (in C++) through JNI



Uses backend unique wire format in X10 2.2.0

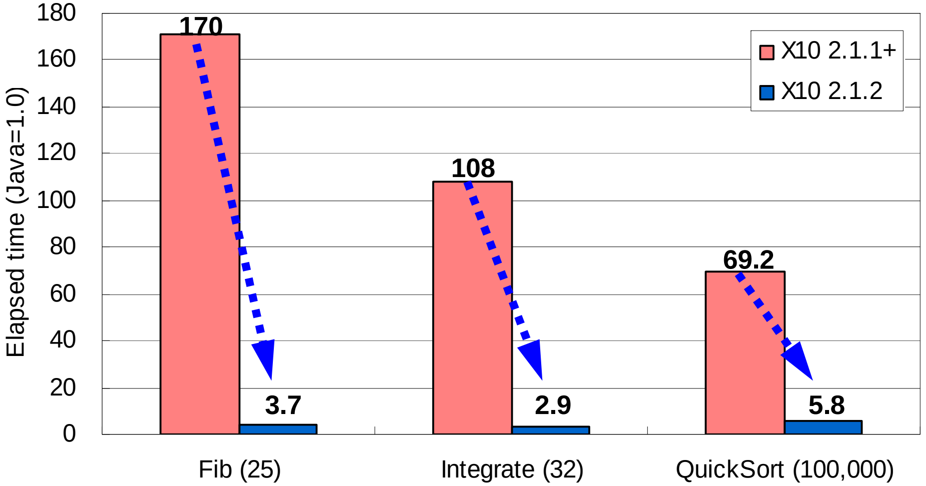
Performance Improvement

Intel Xeon X5670 (6-core, SMT-off, 2.93GHz) x 2, 16GB memory
64-bit Red Hat Enterprise Linux Server release 5.5 (kernel 2.6.18-194.el5)
IBM J9 VM (build 2.4, JRE 1.6.0 IBM J9 2.4 Linux amd64-64 jvmxa6460sr9-20110203_74623 (JIT enabled, AOT enabled)).

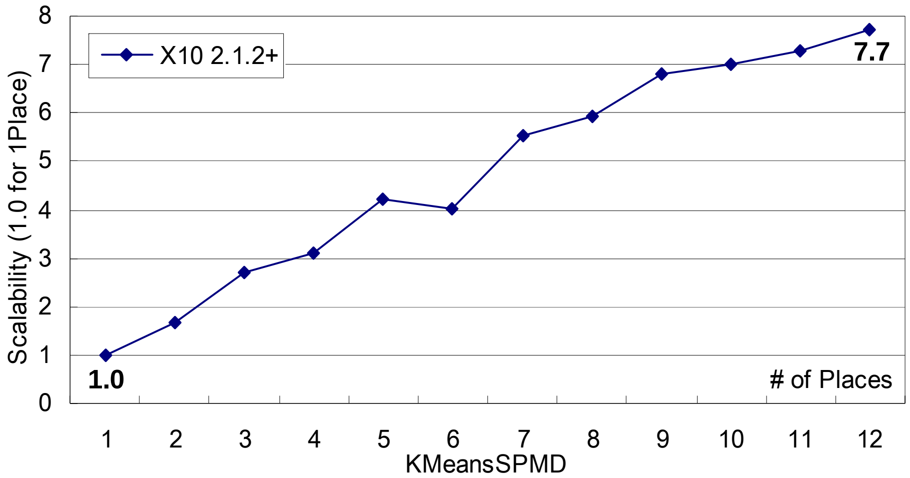


(a) Sequential benchmark results

- Performance on **single JVM** was much improved in 2.1.2
  - Both in sequential (a), and parallel (b) benchmarks
- **Multi-JVM** also shows good scalability (c)
  - Need further tuning



(b) Parallel benchmark results



(c) Distributed benchmark results

Conclusions

- Presented value proposition of Managed X10
- Explained challenges in compiling X10 to Java
  - For performance and functionality
- Discussed compilation techniques
- Demonstrated performance improvement history

Future Work

- Better Java Interoperability
- Heterogeneous Interoperability

- Sequential Performance (Array, Map, etc.)
- Parallel Performance (atomic)
- Distributed Performance and Scalability (Multi-JVM)
- Smaller Footprint

Thank You