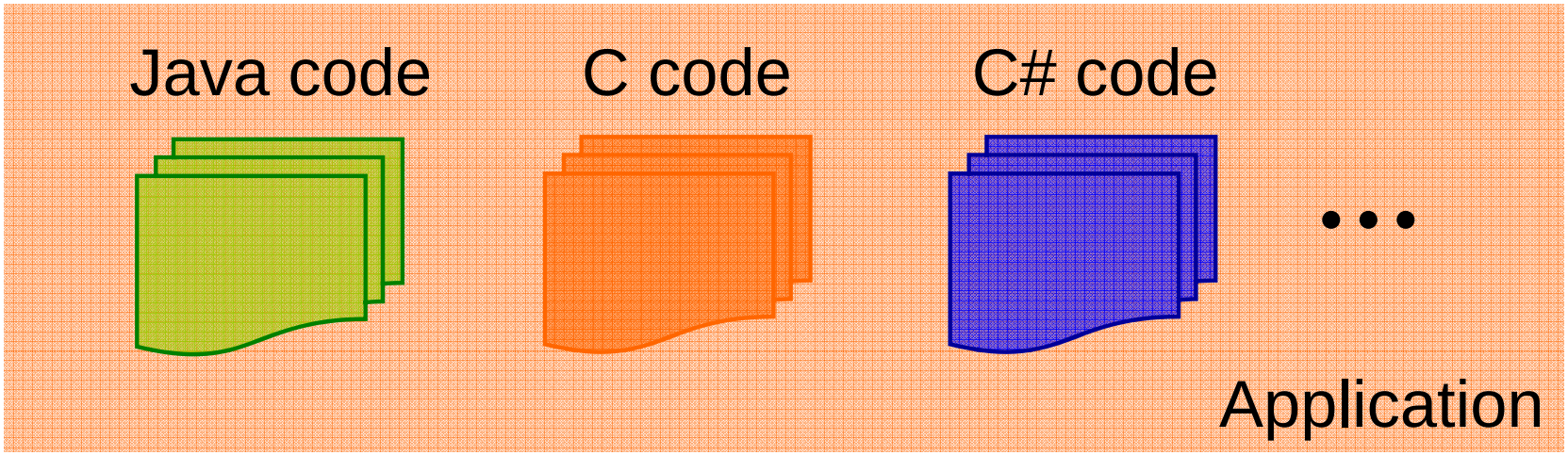
ILEA: Inter-Language Analysis across Java and C

Gang Tan, Boston College
Greg Morrisett, Harvard University

Multi-Lingual Programming

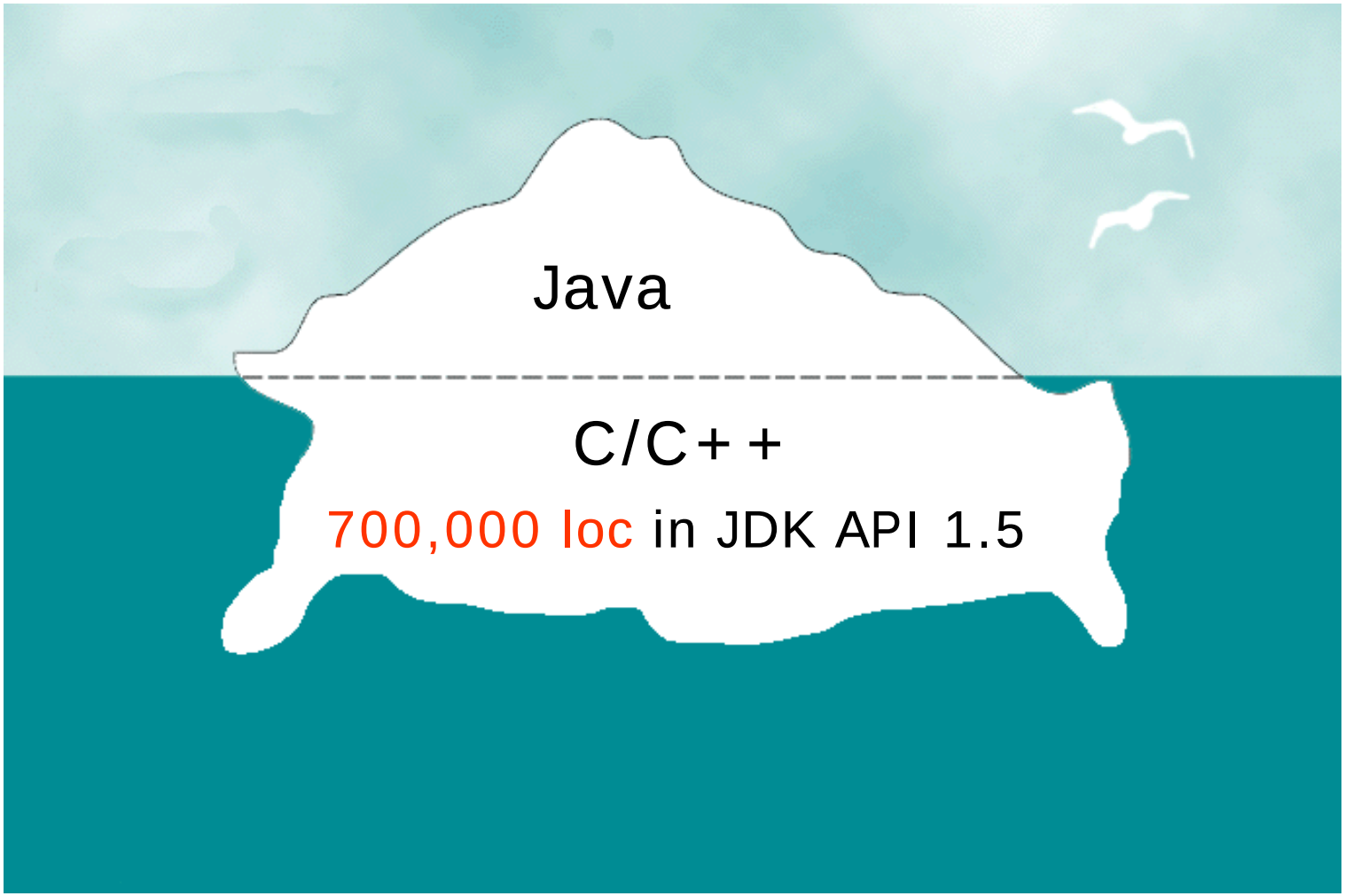


- Software components developed in different programming languages
  - e.g., Java GUI + C back end
  - e.g., Perl web scripts + Java middleware
- Reuse legacy code; mix-and-match benefits of different languages

Pure Java?

- java.io.FileInputStream
  - A Java wrapper for C code that invokes system calls
  - Java and C interacts through the Java Native Interface (JNI)
- java.util.zip.*
  - Java wrappers that invoke the Zlib C compression/decompression library

Pure Java? Not Really …



Static Analysis on Multi-Lingual Applications

- Static analysis: optimization; bug finding
- Most existing source-code analyses are limited a priori to code written in a single language
- Extending the horizon of analysis:

Intra-procedural analysis	Within a same procedure
Inter-procedural analysis	Across the procedure boundaries
Inter-language analysis	**Across the language boundaries**

Bug Finding Across Languages

- Example: Null dereferences

```
…
String s = c_fun();
int len = s.length();
…
```

Java code

```
jstring c_fun ( …) {
  if (…) {
    …
    return NULL;}
}
```

C code

s might be null

In security-critical code, high precision is needed

Our Goal

- How can we extend the horizon of existing Java analyses to cover C code?
- Basic approach
  - Build a specification of the C code
  - Extend the Java analysis to understand the specification
- Questions:
  - What specification language would be the best for all possible analyses?
  - How to generate specifications?

What Specification Language?

- Some annotation language for a particular analysis
  - Nonnull annotations:

    jstring c_fun (jobject obj)
  - Immutability annotations [Pechtchanski and Sarkar 02]
  - Annotation language by [Guyer and Lin 99]
  - \+ Very useful for a particular analysis
  - \- Any simple set of annotations will be incomplete

What Specification Language?

- Pre- and post-conditions expressed in some logic
  - ESC, Spec#, Bandera, …

    {obj ≠ null}

    jstring c_fun (jobject obj)

    {ret ≠ null}

  + Can provide a range of specifications
  - Teaching existing Java analysis to understand the specification requires a significant effort

Our Specification Language

- A slightly extended Java as the specification language for C
  - + Easy to teach existing Java analyses to understand it
    - They already understand Java
  - + Possible to build completely accurate model for some C code
    - If it can be faithfully compiled to Java
- A few primitives for approximation

choose τ

- **Return a random value of type τ**
- Example
  - A C function that invokes gettimeofday() and allocates a Java object of type

    Time {int sec; int msec;}
  - To model the C function

    new Time (choose(int), choose(int));

mutate(x:object)

- **Perform some (type-preserving) mutation to the object x**
- Example
  - A C function that mutates existing objects of any type but does not allocate

```
for (int i = 0; i < choose(int); i++) {
  Object x = choose(Object);
  mutate(x);
}
```

Extended JVML

- Formalized the semantics of the new instructions based on the $JVML_f$ model [Freund and Mitchell 03]
  - choose τ, mutate
  - top: may have any type-preserving effect on the JVM heap
- The new instructions are non-deterministic
  - They relate one state to possibly multiple states

Revisiting Our Questions

- What specification language would be the best for all possible analyses?
  - JVML + a few primitives for approximation
    - + Expressive
    - + Easy to migrate existing Java analysis
- How to generate specifications?
  - Automatic specification extractors

Specification Extractors

- Our spec. language allows a range of specification extractors
- Our spec. extractor
  - Implemented on top of the CIL (Necula et al.)
    - Use a pointer analysis provided by the CIL
    - Output is in Jasmin syntax

Our Specification Extractor

- Strategy: make common and easy cases precise, while leaving rare or difficult cases for approximation
- Support: loops; pointers; structs; unions; most of the JNI API functions; …
  - Thanks to the approximation instructions
- Assumptions
  - Faithfully capture the JVM-heap effect that the C code might have
  - Single-threaded code
- Formalization
  - The paper formalized a subset and presented theorems

Specification Extraction: Example I

```
int c-fun (int i) {
  int j;
  int *p = &i;
  if (*p > 0) j=i;
  else j = 2*i;
  return j;
}
```

C code

Type-based variable mapping

```
int c-fun (int i) {
  int j;
  if (choose boolean)
    j=i;
  else j=2*i;
  return j;
}
```

Spec.

Specification Extraction: Example II

```
void addOne (jintArray arr) {
    int len = GetArrayLength(arr);
    int * buf = malloc(len * sizeof(int));
    int i;
    GetIntArrayRegion(arr, 0, len, buf);
    for (i = 0; i < len; i++) {
      buf[i] += 1;
    }
    SetIntArrayRegion(arr, 0, len, buf);
    free(buf);
}
```

C code

Specification Extraction: Example II

```
void addOne(jintArray arr) {
  int len = GetArrayLength(arr);
  int * buf = malloc(len * sizeof(int));
  int i;
  GetIntArrayRegion(arr, 0, len, buf);
  for (i = 0; i < len; i++) {
    buf[i] += 1;
  }
  SetIntArrayRegion(arr, 0, len, buf);
  free(buf);
}
```

C code

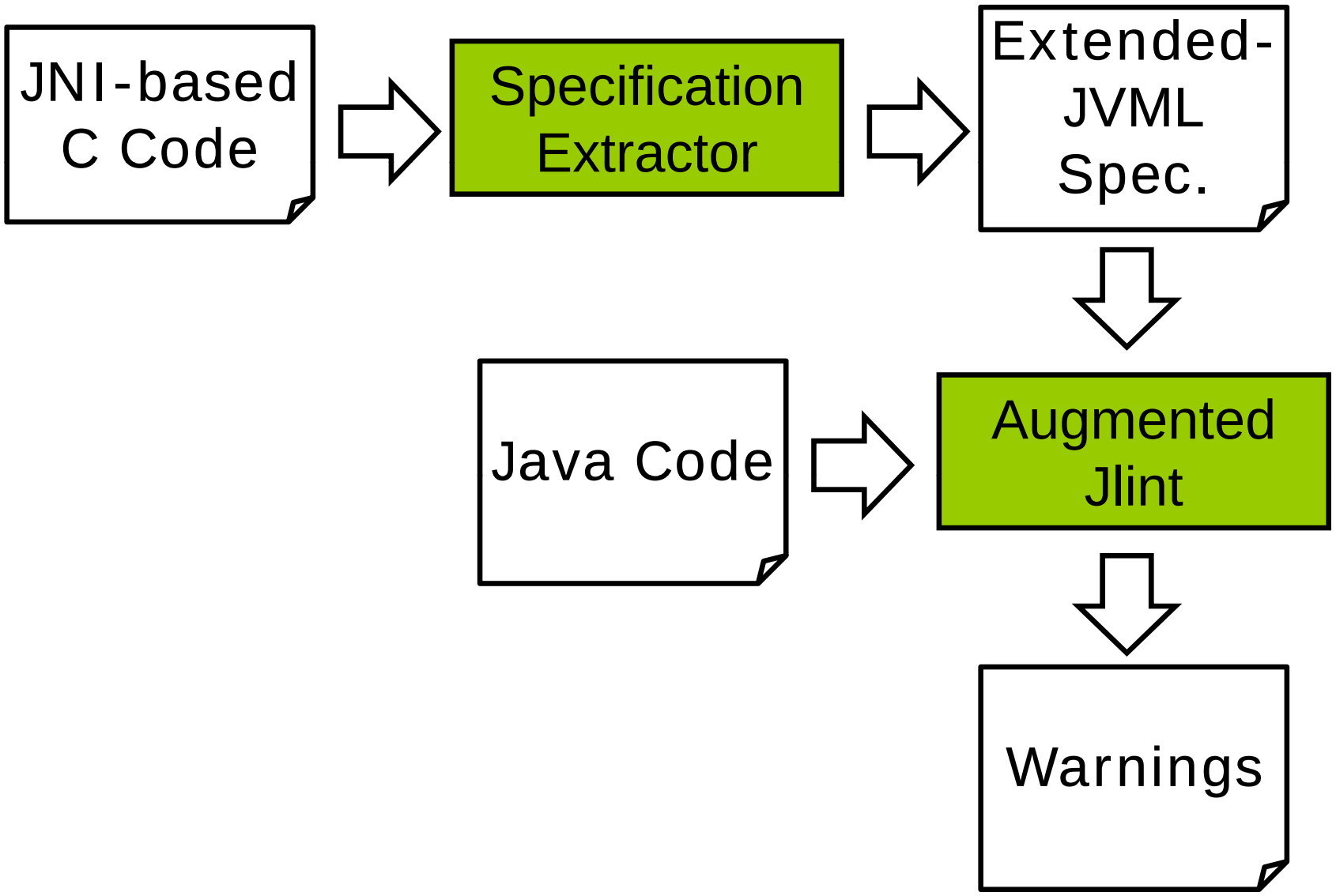
```
void sumArray (int[ ] arr) {
  int len=arr.length();
  int i;
  for (i=0; i<len; i++) {
  }
  mutate(arr);
}
```

Spec.

Utilizing Specifications

- Thesis
  - With small changes, existing Java analysis can understand the behavior of C code
- Experiment
  - Jlint: a Java bug finder
    - perform inter-procedural dataflow analysis
  - Finding null dereferences
  - JNI-based C code is prone to the error of null dereferences

Setup of Our Experiment



Preliminary Experiments

Program	C loc	Java loc	Time	Errors
java.lang.StrictMath	8658	1128	0.69s	0
java.util.zip	9195	824	0.79s	0
posix 1.0	1874	860	0.24s	15
libreadline-java-0.8.0	1810	1196	0.13s	9

* Errors = Warnings - False positives

Related Work

- Java bug finders and optimizers
  - FindBugs, Jlint, ESC/Java, Bandera, SOOT, Jikes, …
- Analysis of multilingual software
  - Saffire [Furr and Foster 05, 06]
  - SafeJNI [Tan et al. 06]
  - Interaction between statically and dynamically typed languages
    - [Gray et al. 05] [Matthews and Findler 07]
  - Better interfaces between Java and C
    - Jeannie [Hirzel and Grimm 07]
    - Janet [Budak et al. 00]
- .NET, Fortify
  - Heavy-weight

The End