Jinn: Synthesizing Dynamic Bug Detectors for Foreign Language Interfaces

Byeongcheol Lee

Ben Wiedermann

Martin Hirzel

Robert Grimm

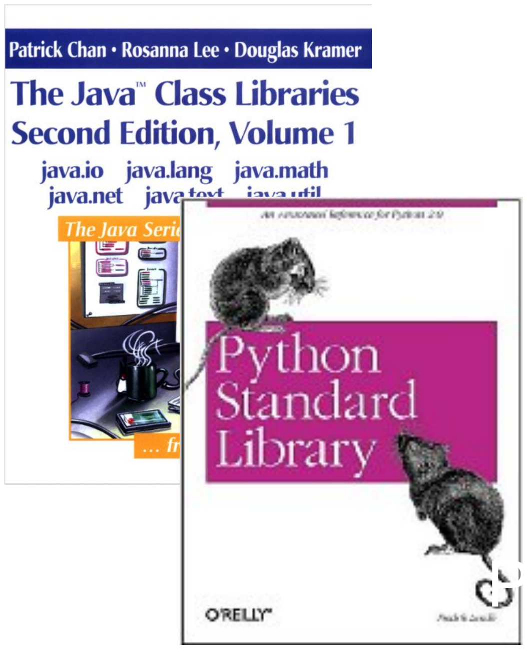
Kathryn S. McKinley

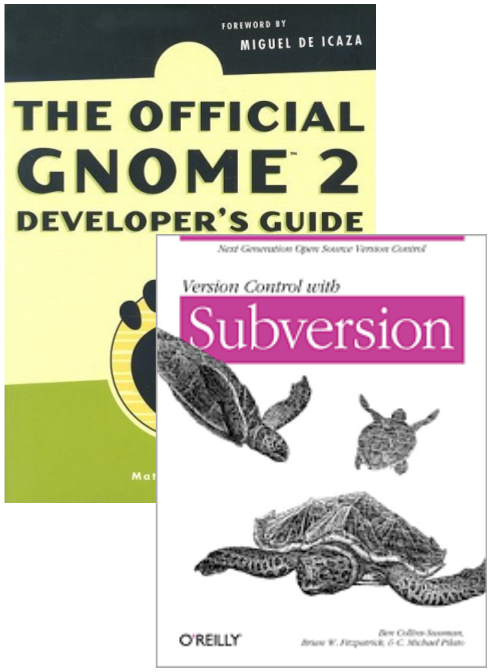
THE UNIVERSITY OF TEXAS AT AUSTIN

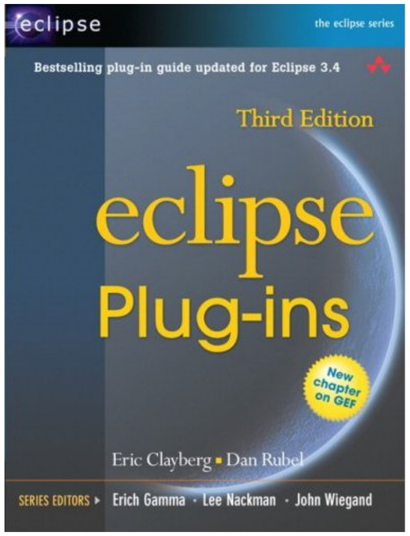
IBM

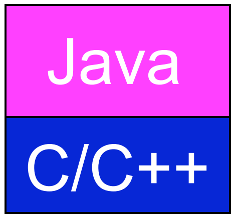
NEW YORK UNIVERSITY

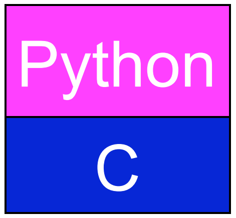
Multilingual programs are ubiquitous

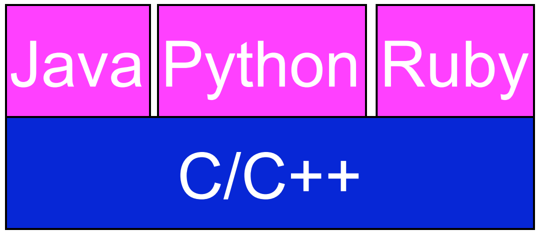

Patrick Chan • Rosanna Lee • Douglas Kramer
The Java™ Class Libraries
Second Edition, Volume 1
java.io java.lang java.math
java.net
The Java Series
Python Standard Library
O'REILLY

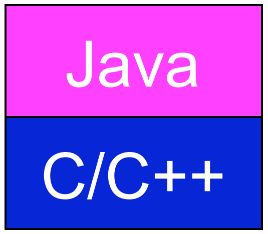

FOREWORD BY MIGUEL DE ICAZA
THE OFFICIAL GNOME 2 DEVELOPER'S GUIDE
Version Control with Subversion
O'REILLY


eclipse
the eclipse series
Bestselling plug-in guide updated for Eclipse 3.4
Third Edition
eclipse Plug-ins
New chapter on GEF
Eric Clayberg ▪ Dan Rubel
SERIES EDITORS ▸ Erich Gamma · Lee Nackman · John Wiegand







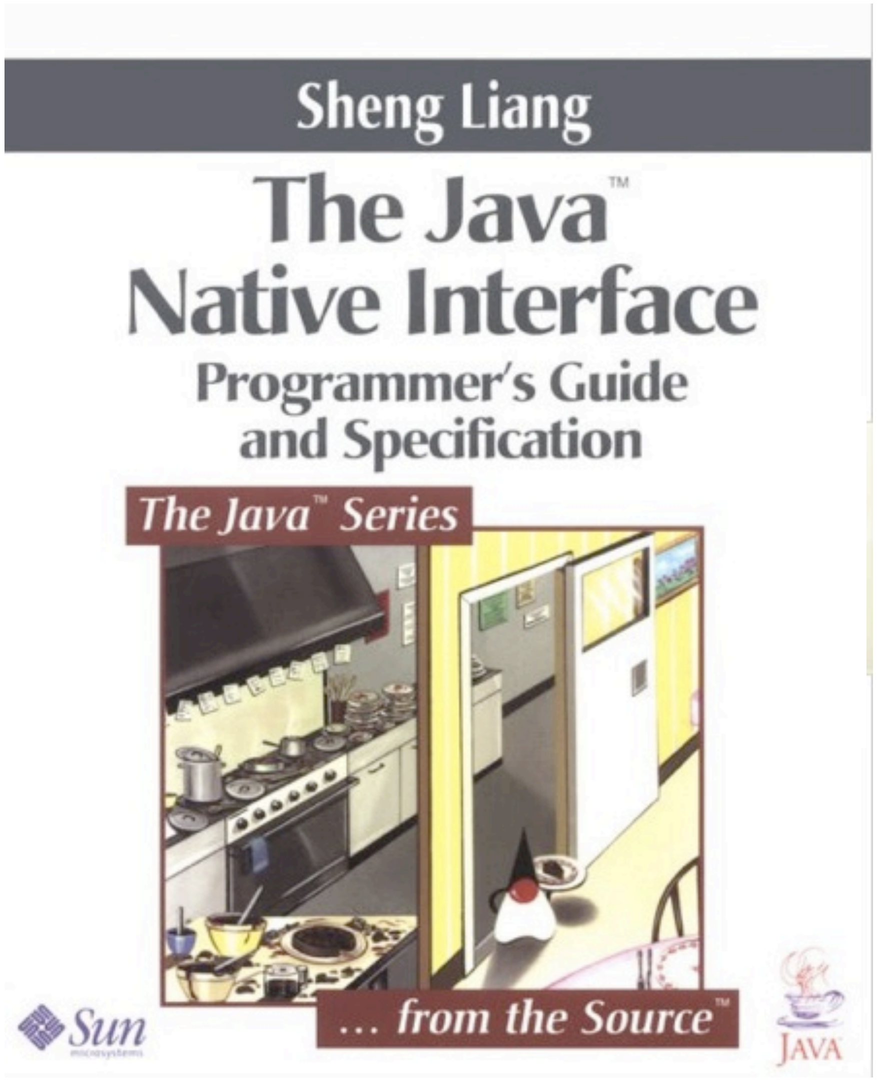


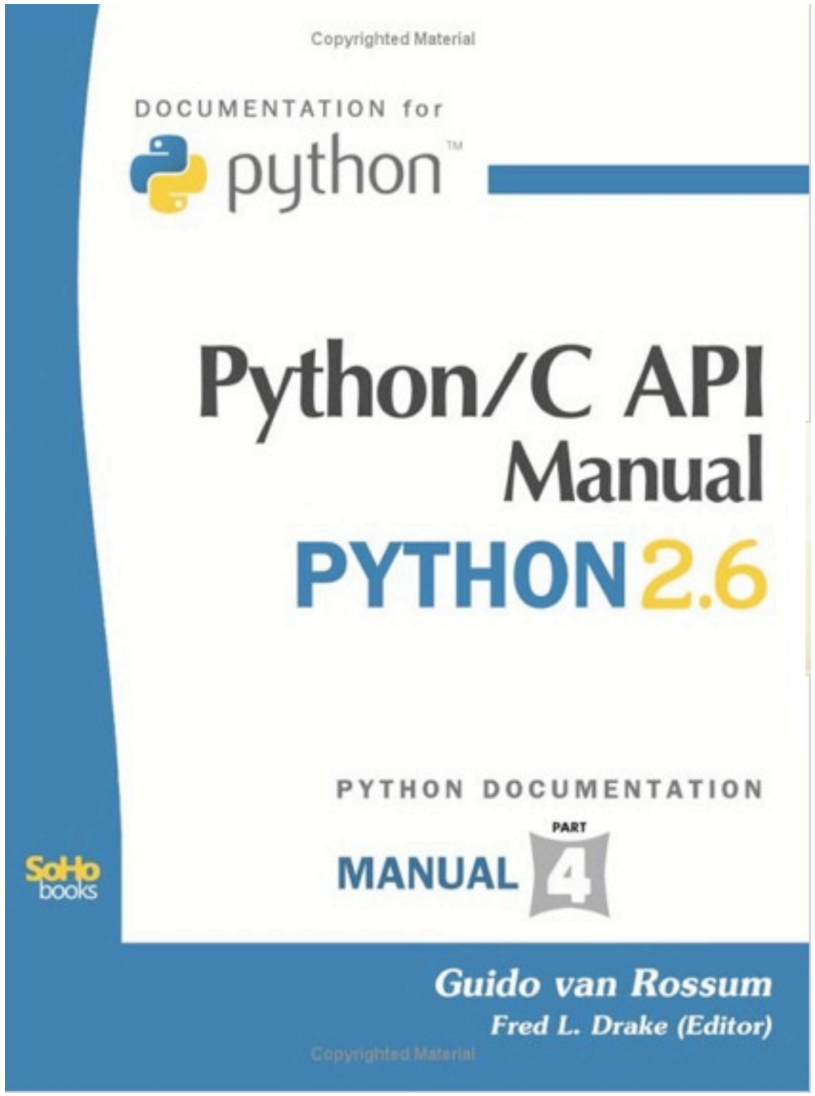
Standard libraries

Multilingual bindings

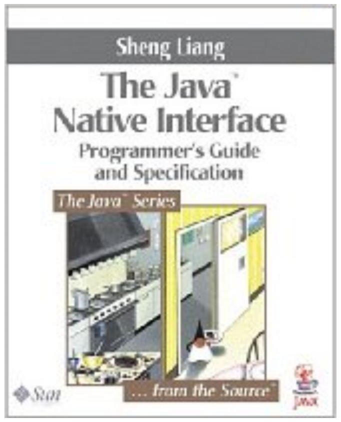
Plug-in extensions

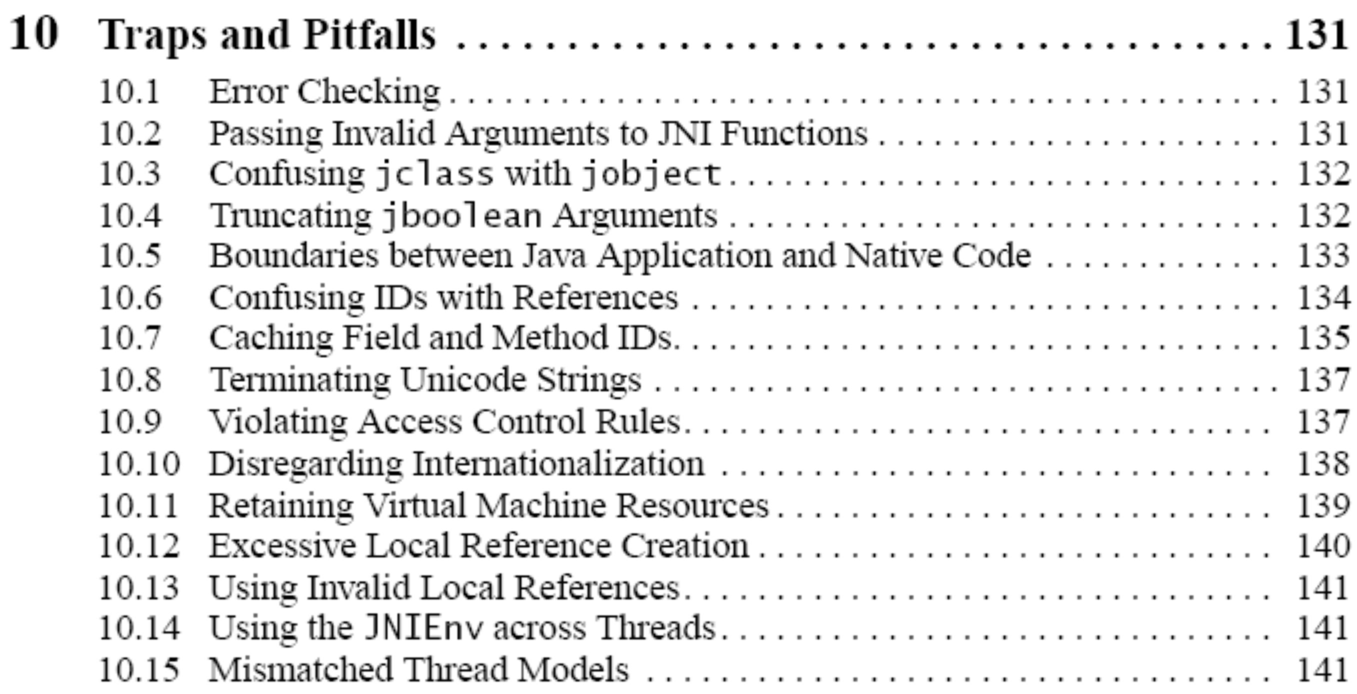
All multilingual programs use foreign function interfaces (FFIs)

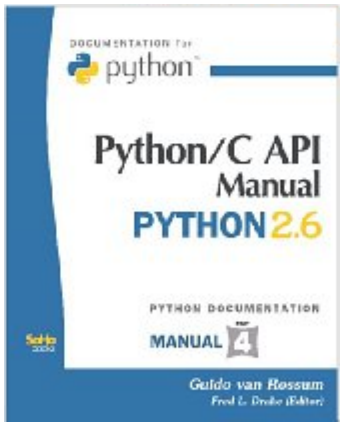


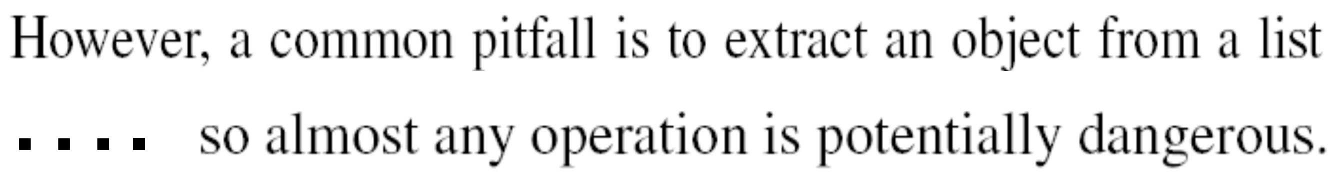


FFIs have many dangerous pitfalls



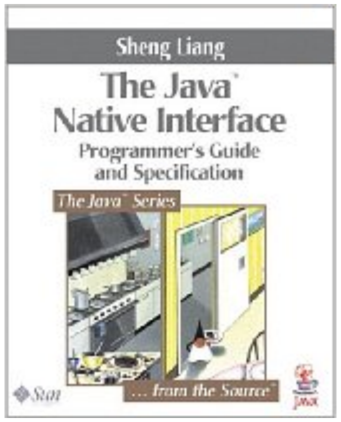


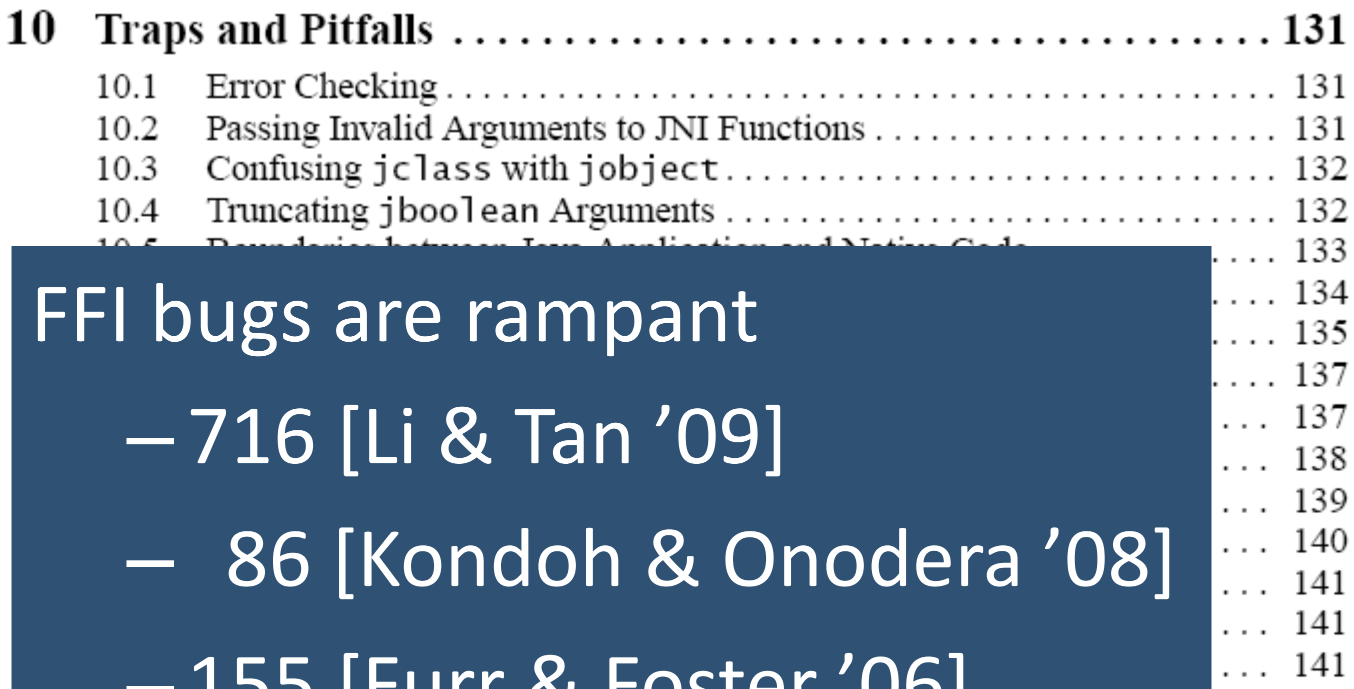



However, a common pitfall is to extract an object from a list

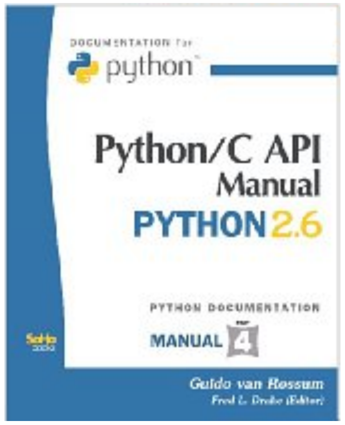
. . . . so almost any operation is potentially dangerous.

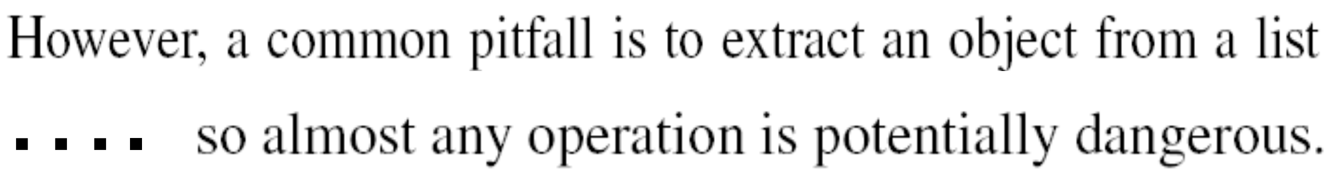
FFIs are complex and hard to program


Sheng Liang
The Java Native Interface
Programmer's Guide and Specification
The Java Series
... from the Source



FFI bugs are rampant
- 716 [Li & Tan ’09]
- 86 [Kondoh & Onodera ’08]
- 155 [Furr & Foster ’06]

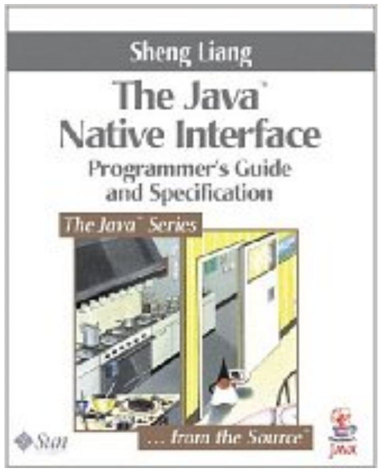



However, a common pitfall is to extract an object from a list
. . . . so almost any operation is potentially dangerous.

Multilingual programmers need dynamic bug detectors

- Static compile-time verification is hard
  - A rule of no more than 16 local references in JNI
  - False alarms in static bug finders
- Dynamic FFI checking is complementary
  - No false alarms
  - Bugs in a single program run

FFI specifications are not friendly to dynamic checking


Sheng Liang

The Java Native Interface

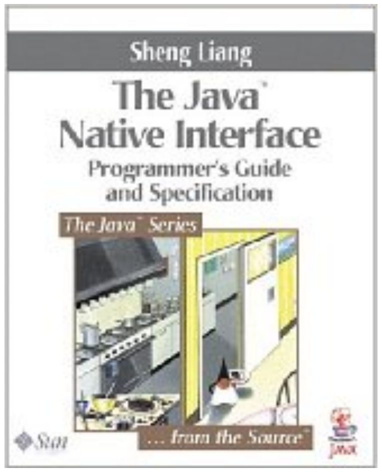
Programmer's Guide and Specification

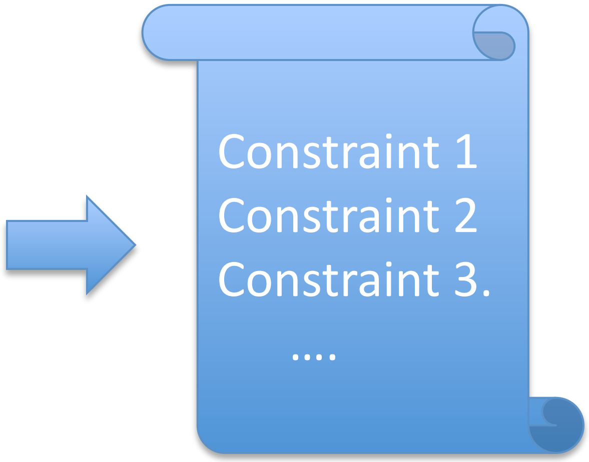
The Java Series

... from the Source

303 pages

FFI specifications are not friendly to dynamic checking

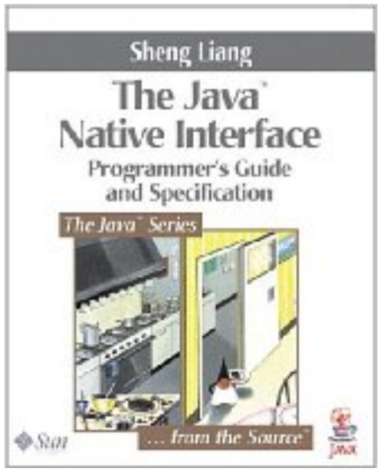

Sheng Liang
The Java Native Interface
Programmer's Guide and Specification
The Java Series
... from the Source
Sun

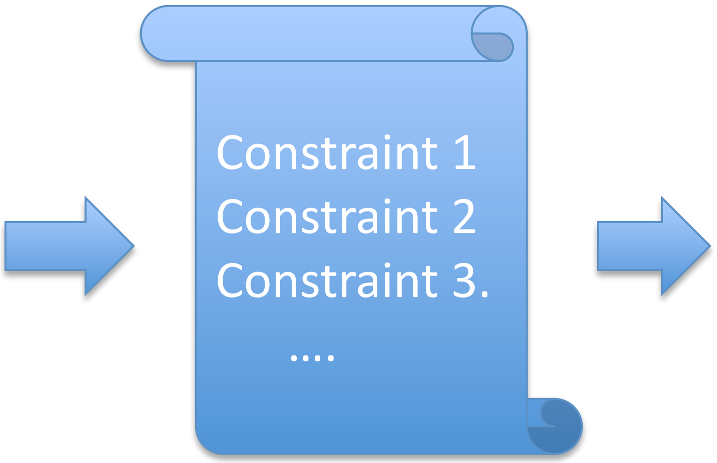


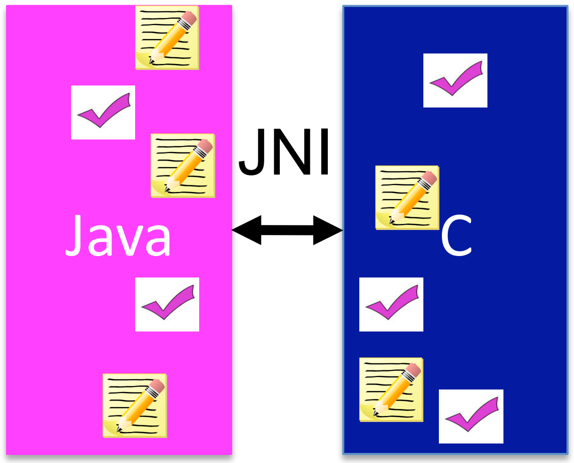
303 pages

1,500+ constraints on 229+ JNI function

FFI specifications are not friendly to dynamic checking


Sheng Liang
The Java Native Interface
Programmer's Guide and Specification
The Java Series
... from the Source





303 pages

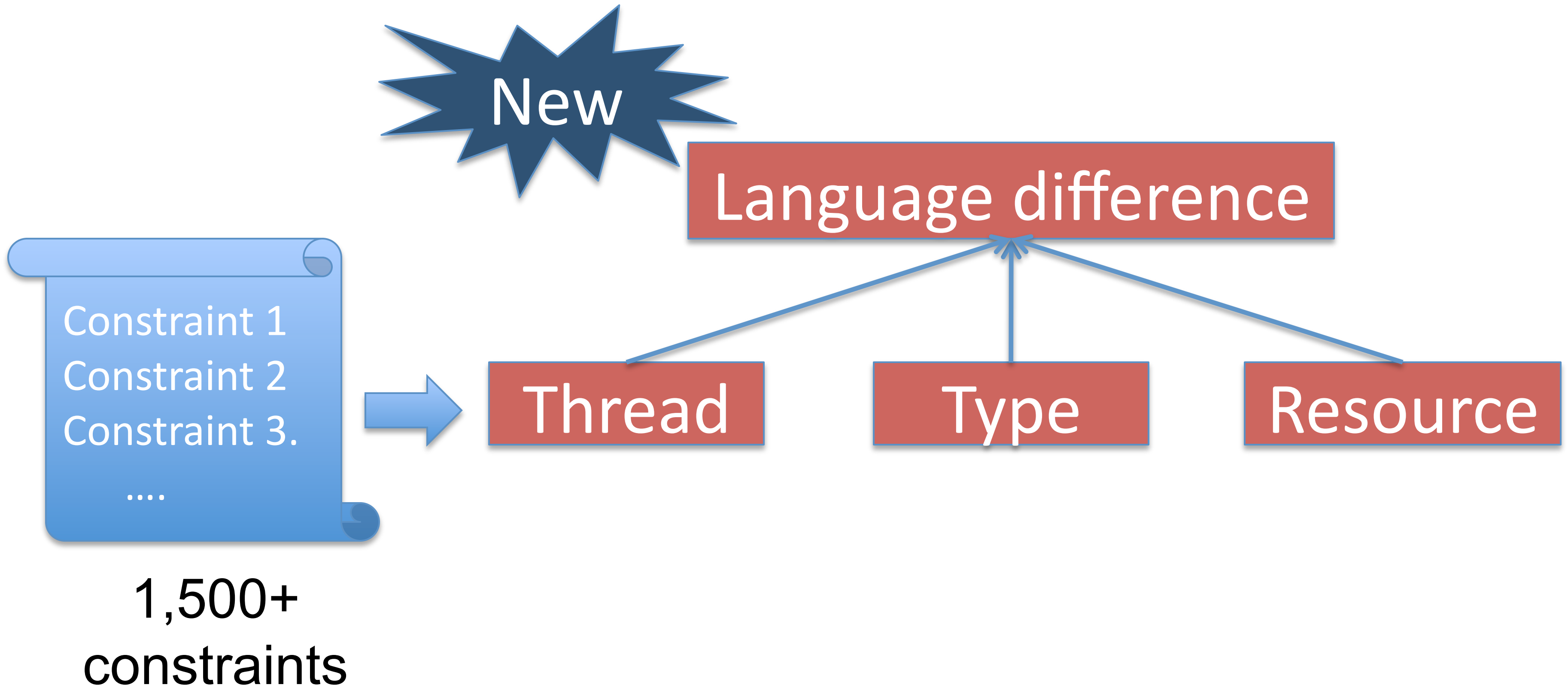
1,500+ constraints on 229+ JNI function

Every language transition requires bookkeeping & checking 1,500+ constraints

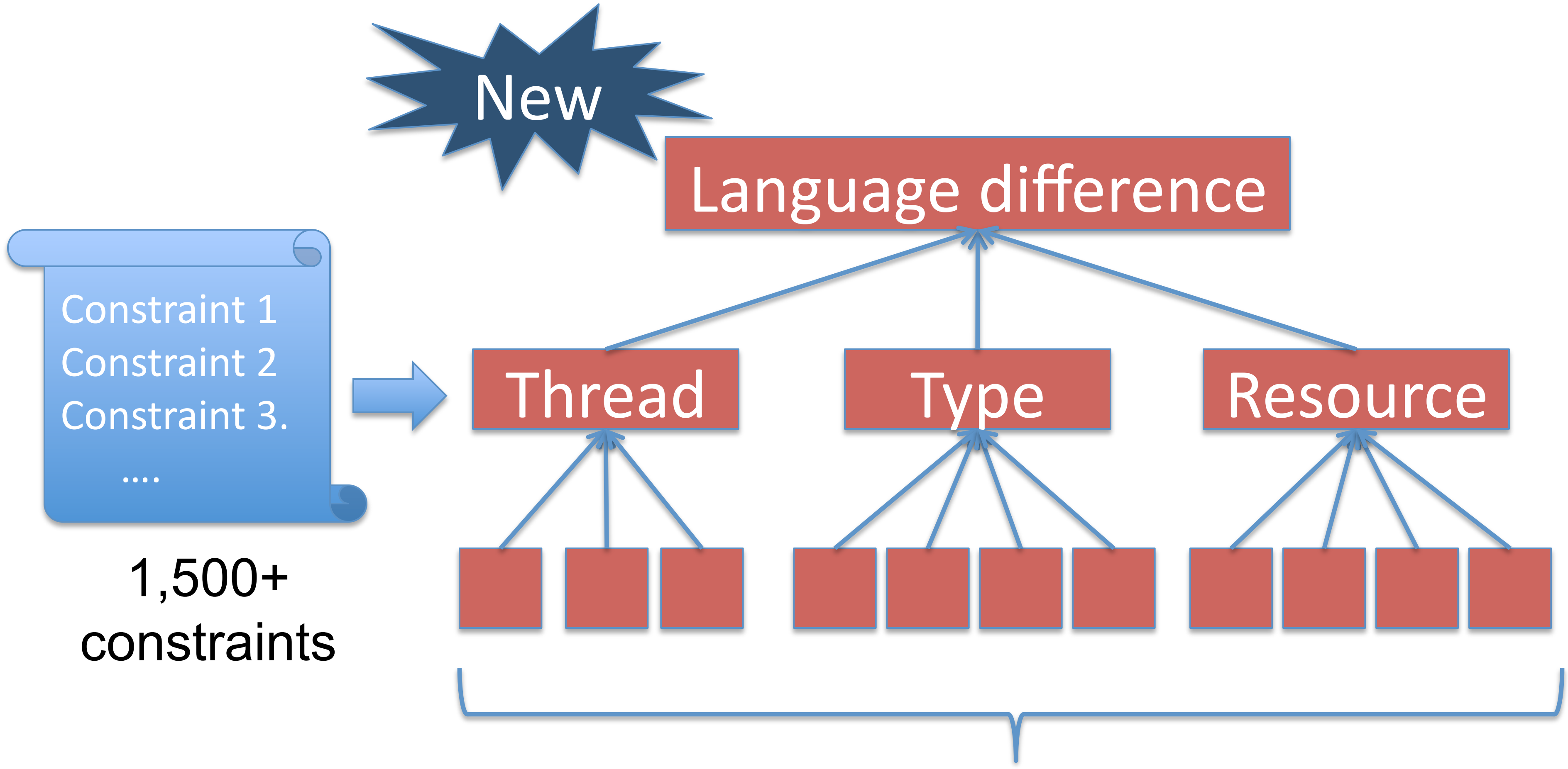
$$\underbrace{\qquad\qquad\qquad\qquad\qquad\qquad}$$

Time-consuming and error-prone

Our insight: FFI constraints have hierarchy

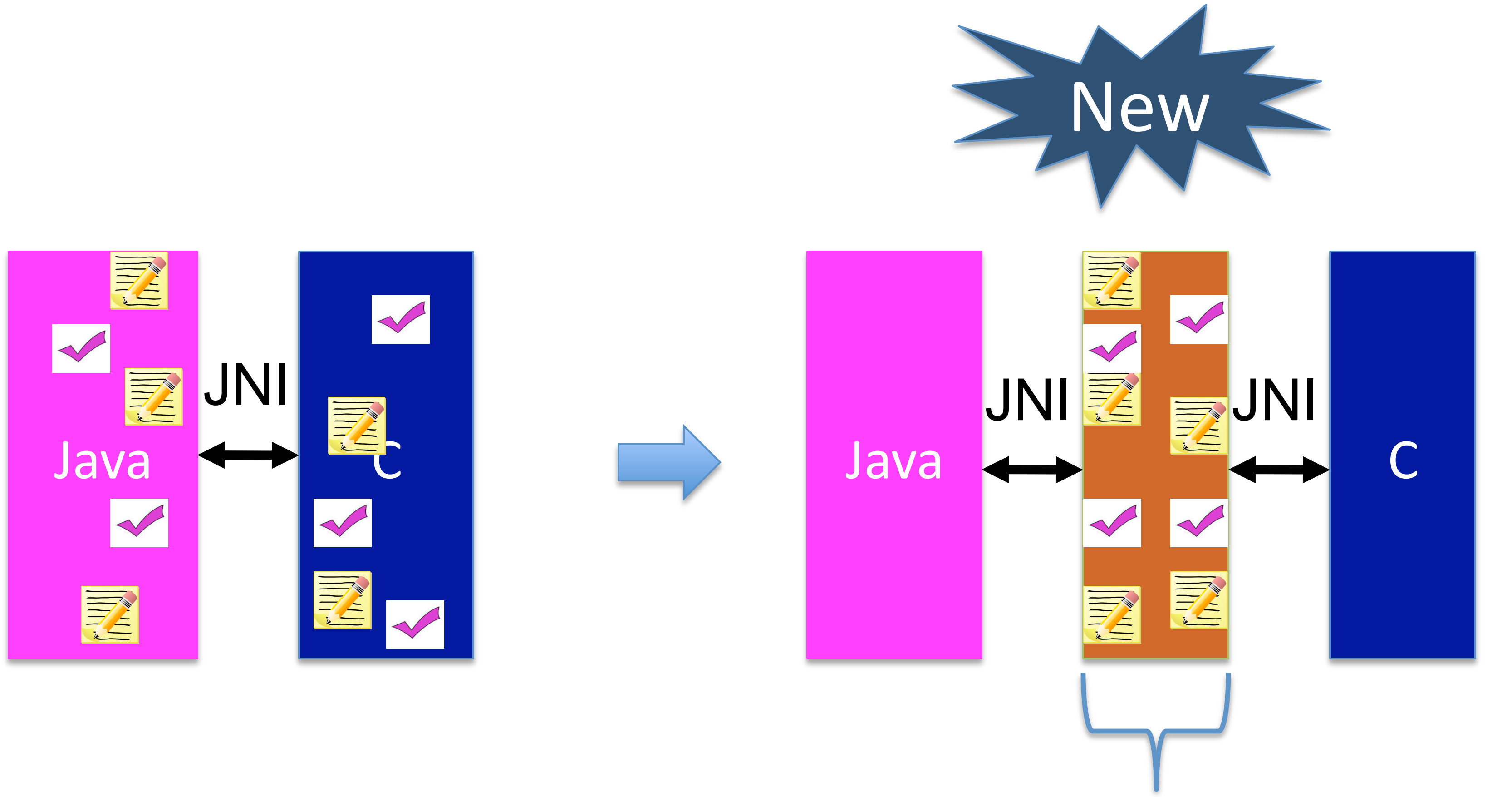


Our insight: FFI constraints have hierarchy



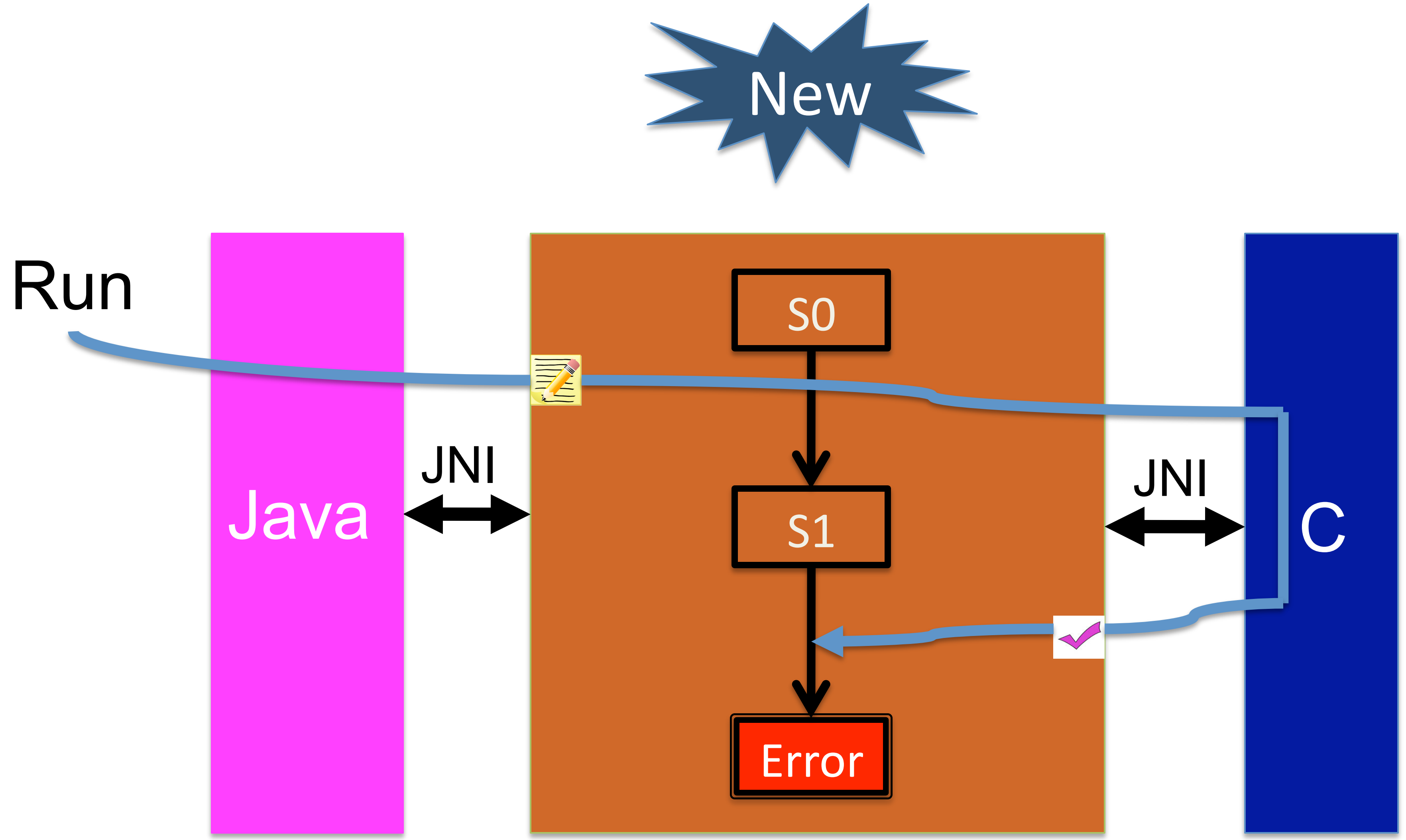
11 state machines represent 1,500+ constraints

Our insight: state machines change states at language transitions

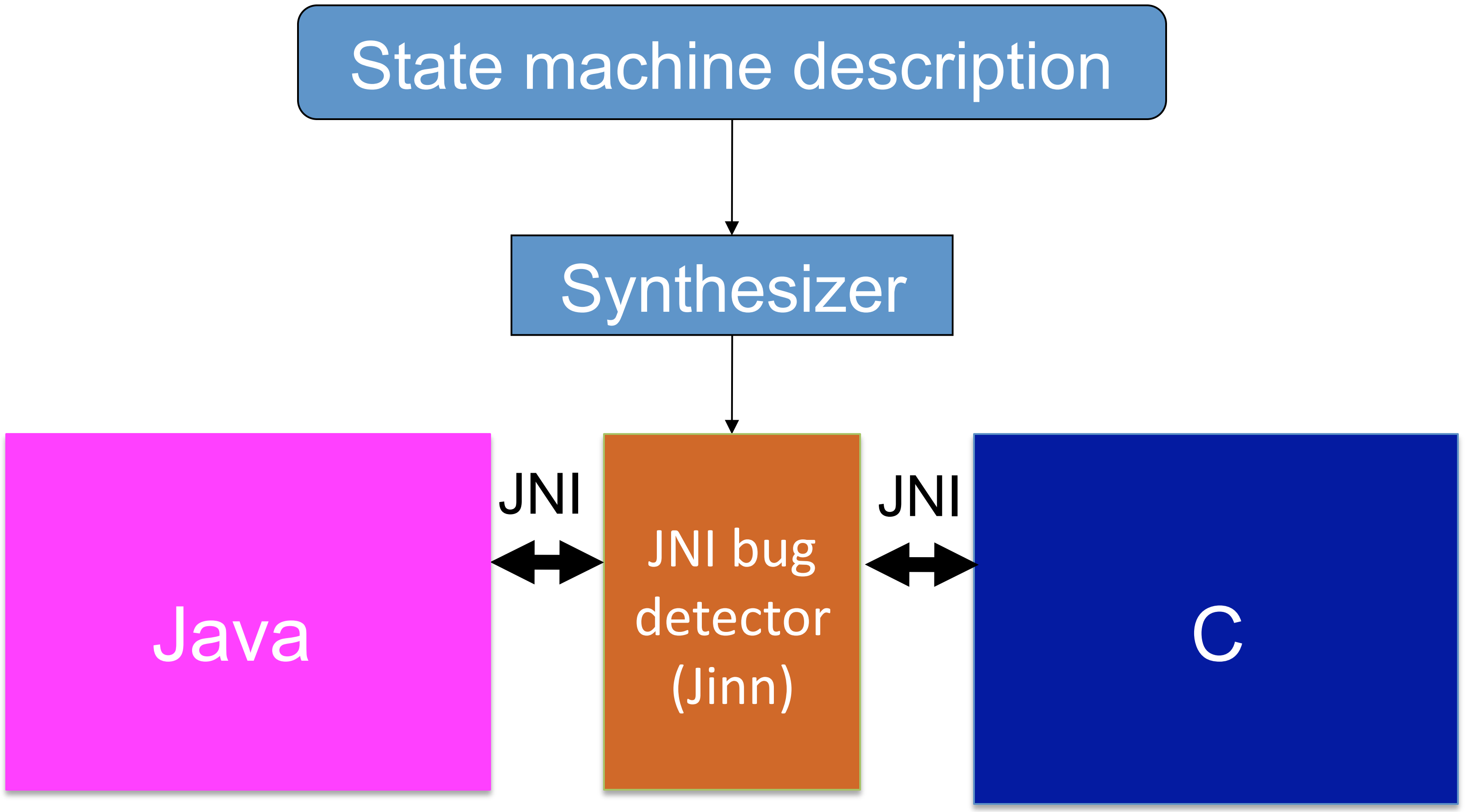


Bookkeeping and checking at language boundary

Our insight: state machines change states at language transitions



Synthesizing dynamic bug detectors



Jinn

Our synthesis approach applies to other FFIs including Python/C

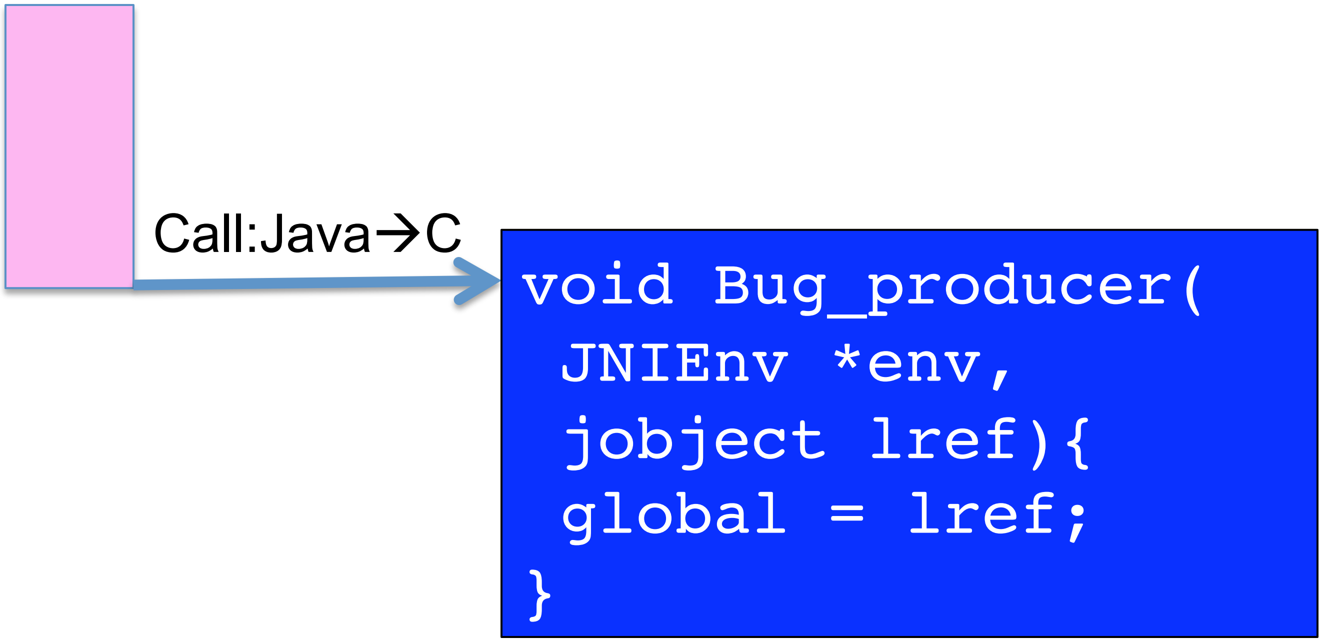
Outline

I. Classification of language semantic mismatch in FFIs
II. Synthesis of FFI bug detectors with state machines

III. State machines
   A. An example JNI bug
   B. Mapping state machines to entities
   C. Mapping state transitions to language transitions

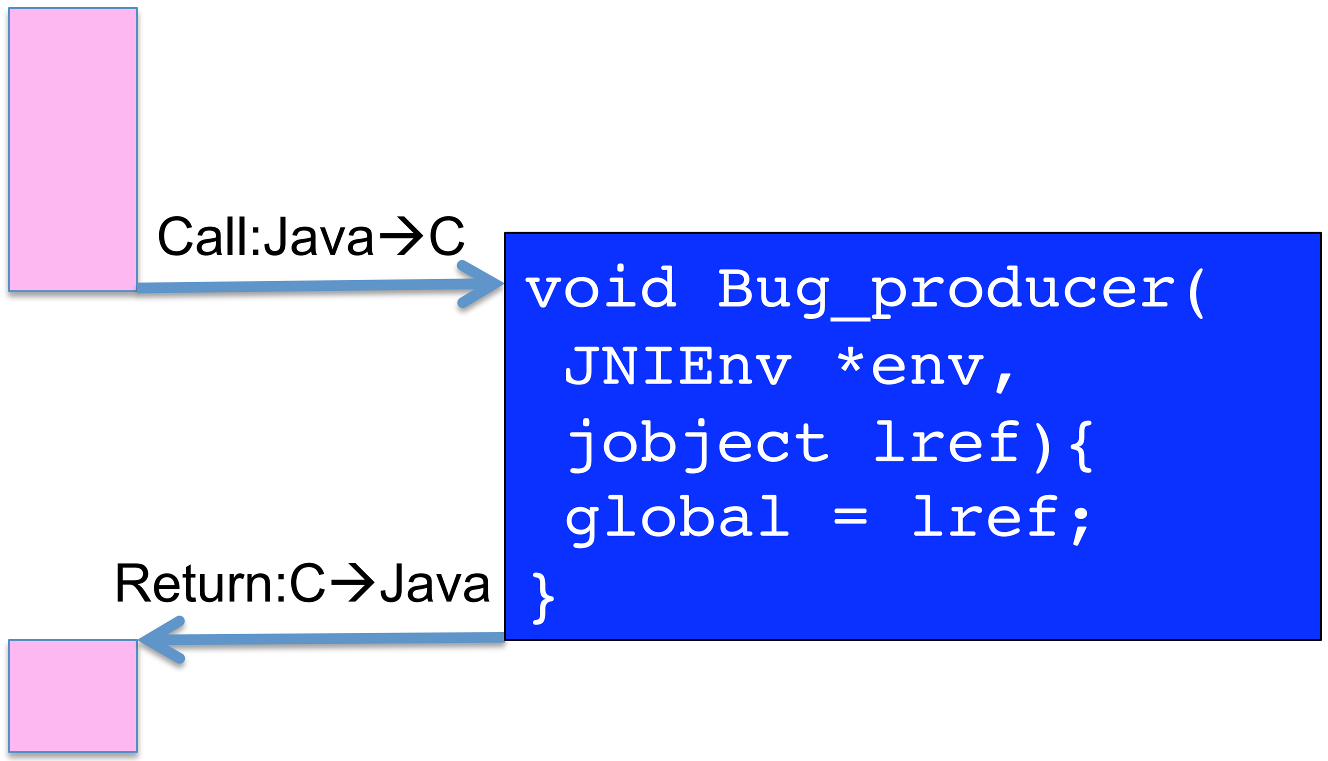
IV. Jinn: a dynamic JNI bug detector
   A. Finds more bugs than static checkers & other dynamic checkers
   B. Adds modest execution time overhead
   C. Finds lots of real-world bugs

The GNOME bug 576111 uses an invalid JNI reference



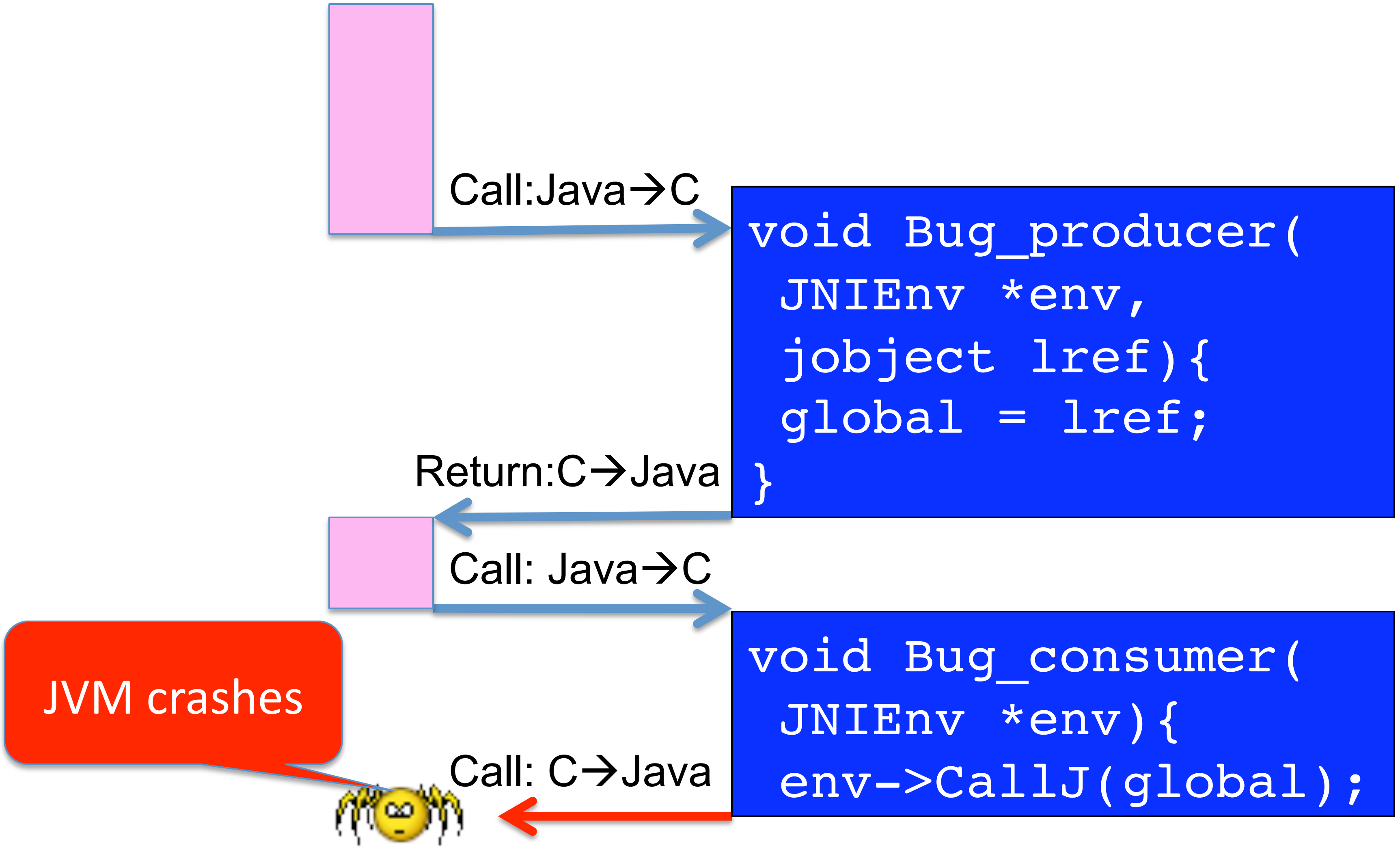
```
void Bug_producer(
 JNIEnv *env,
 jobject lref){
 global = lref;
}
```

The GNOME bug 576111 uses an invalid JNI reference



```
void Bug_producer(
 JNIEnv *env,
 jobject lref){
 global = lref;
}
```

The GNOME bug 576111 uses an invalid JNI reference



```
void Bug_producer(
 JNIEnv *env,
 jobject lref){
 global = lref;
}
```

```
void Bug_consumer(
 JNIEnv *env){
 env->CallJ(global);
```

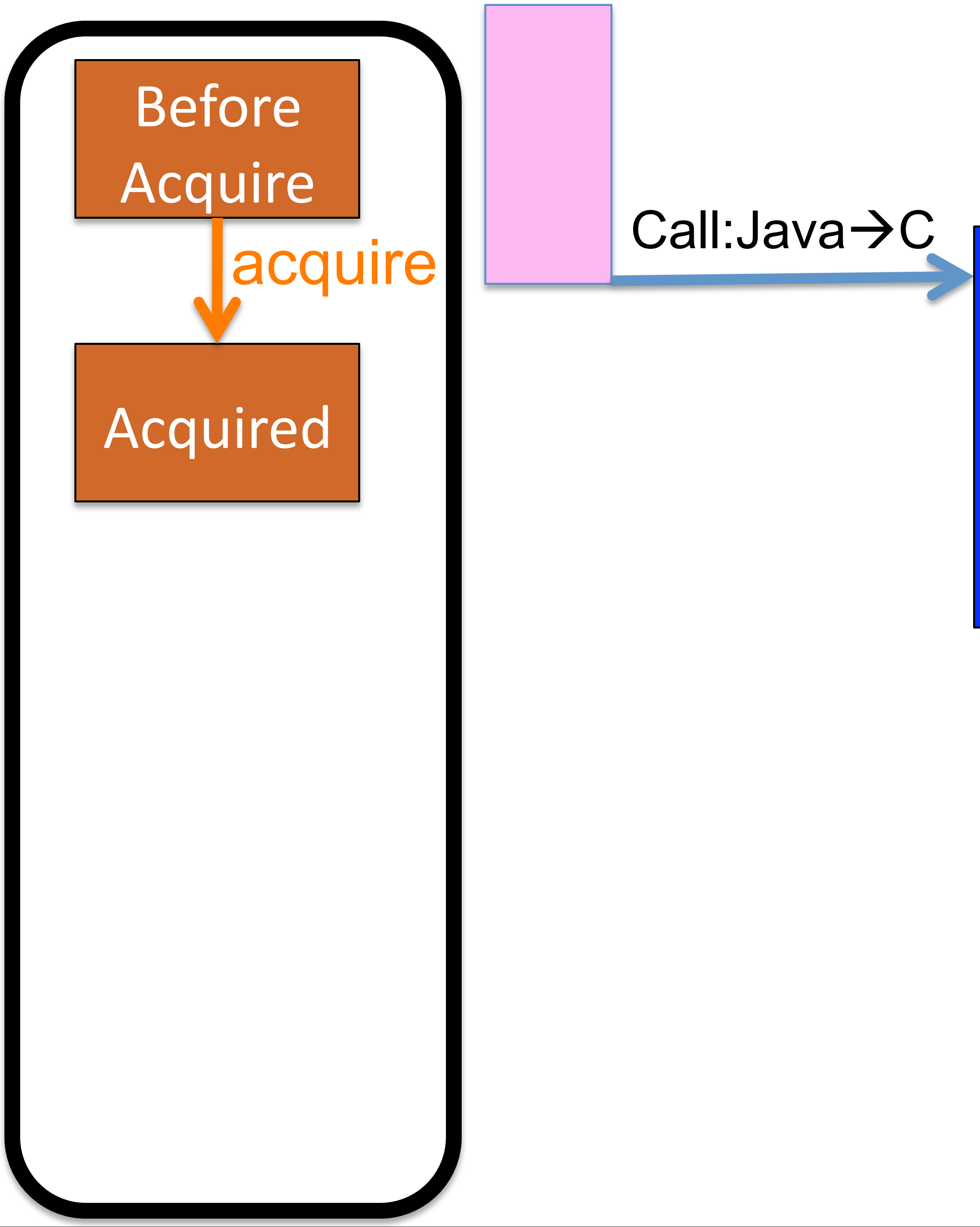
Outline

I. Classification of language semantic mismatch in FFIs
II. Synthesis of FFI bug detectors with state machines

III. State machines
   A. An example JNI bug
   B. Mapping state machines to entities
   C. Mapping state transitions to language transitions

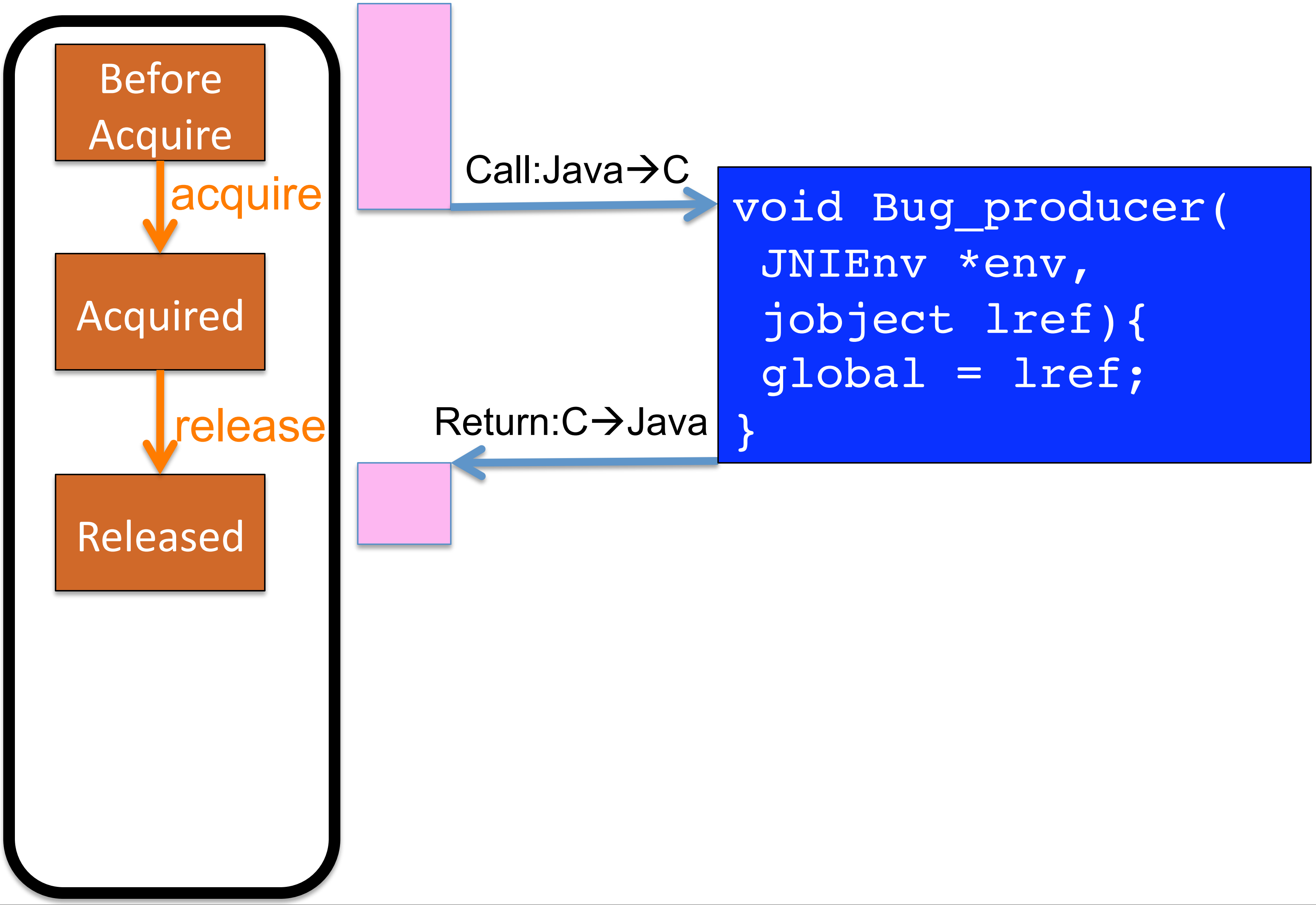
IV. Jinn: a dynamic JNI bug detector
   A. Finds more bugs than static checkers & other dynamic checkers
   B. Adds modest execution time overhead
   C. Finds lots of real-world bugs

Map a state machine to an entity



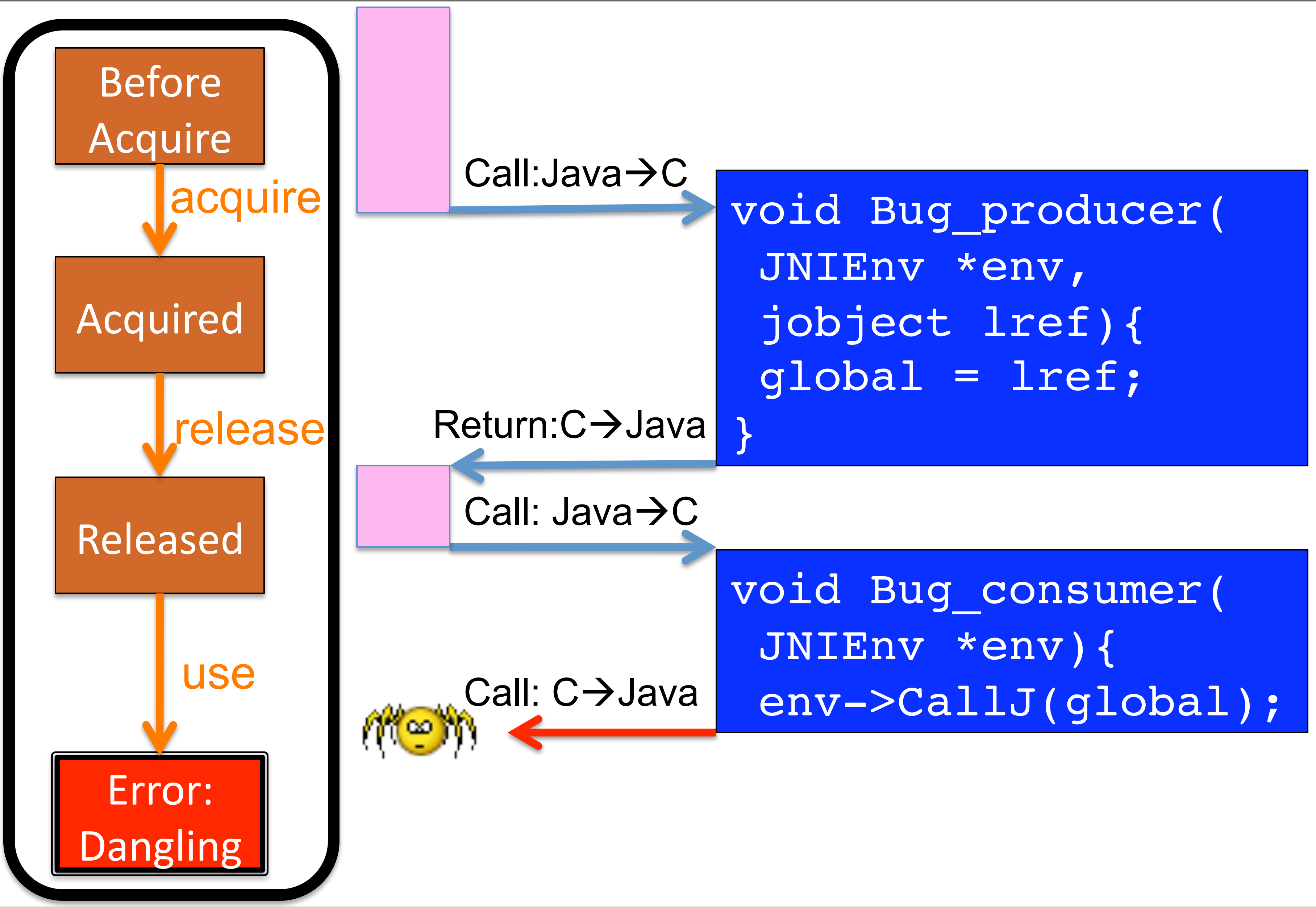
```
void Bug_producer(
 JNIEnv *env,
 jobject lref){
 global = lref;
}
```

Map a state machine to an entity



```
void Bug_producer(
 JNIEnv *env,
 jobject lref){
 global = lref;
}
```

Map a state machine to an entity



```
void Bug_producer(
 JNIEnv *env,
 jobject lref){
 global = lref;
}
```

```
void Bug_consumer(
 JNIEnv *env){
 env->CallJ(global);
```

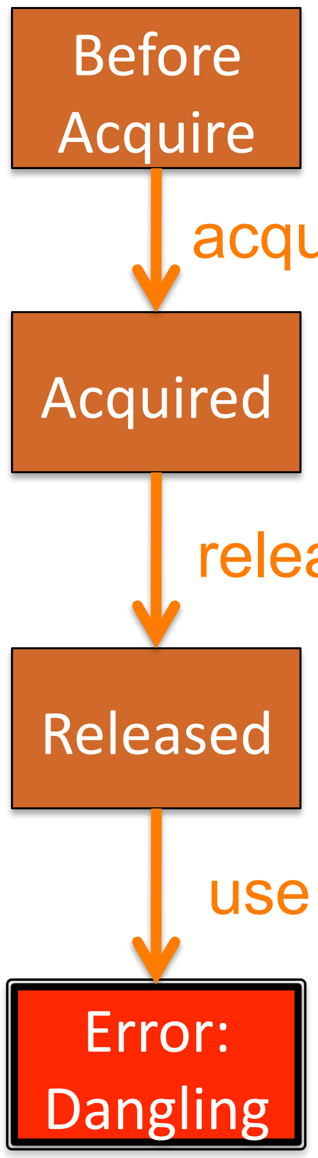
Outline

I. Classification of language semantic mismatch in FFIs
II. Synthesis of FFI bug detectors with state machines

III. State machines
  A. An example JNI bug
  B. Mapping state machines to entities
  C. Mapping state transitions to language transitions

IV. Jinn: a dynamic JNI bug detector
  A. Finds more bugs than static checkers & other dynamic checkers
  B. Adds modest execution time overhead
  C. Finds lots of real-world bugs

Map state transitions to language transitions



State transition	Language transition	Examples
Acquire	Call:Java→C	Native methods taking references
	Return:Java→C	GetObjectField
Release	Return: C→Java	Any native method
	Return: Java→C	DeleteLocalRef
Use	Call: C→ Java	CallVoidMethod
	Return: C→Java	Native methods returning reference

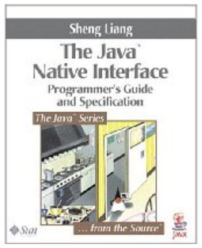
Outline

I. Classification of language semantic mismatch in FFIs
II. Synthesis of FFI bug detectors with state machines

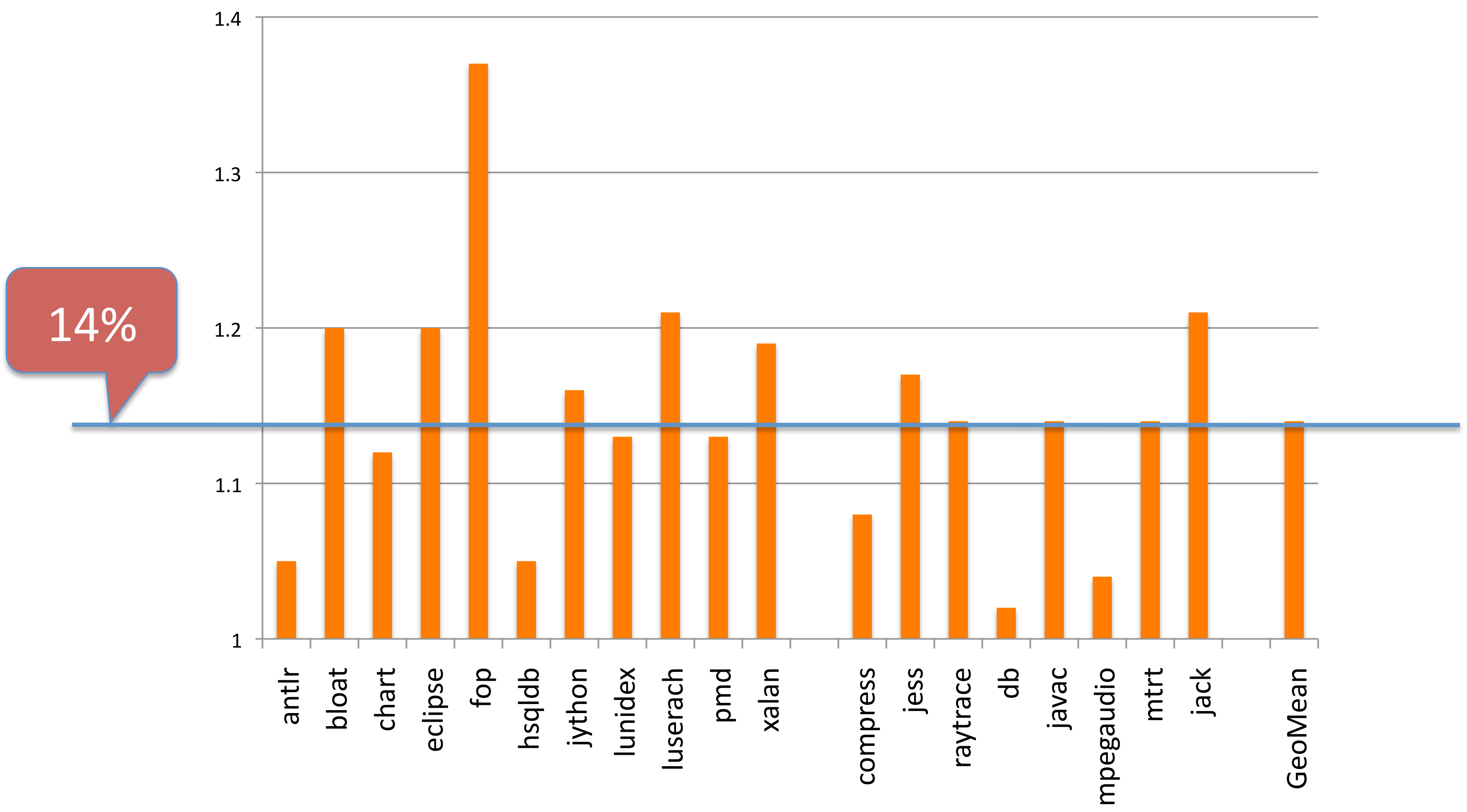
III. State machines
   A. An example JNI bug
   B. Mapping state machines to entities
   C. Mapping state transitions to language transitions

IV. Jinn: a dynamic JNI bug detector
   A. Finds more bugs than static checkers & other dynamic checkers
   B. Adds modest execution time overhead
   C. Finds lots of real-world bugs

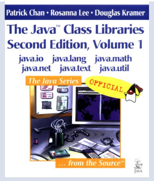
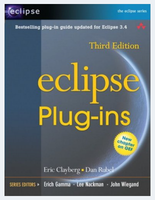
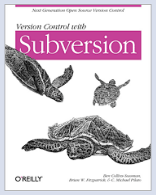
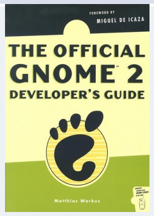
Jinn covers more bugs than JVM internal checkers

JNI Pitfall  Sheng Liang The Java Native Interface Programmer's Guide and Specification The Java Series ...from the Source	JVM checking		Jinn
	Hotspot	J9	
Error checking	Warning	Error	Exception
Invalid Arguments to JNI functions	Running	Crash	Exception
Confusing jclass with jobject	Error	Error	Exception
Confusing IDs with references	Error	Error	Exception
Violating access control rules	NPE	NPE	Exception
Retaining virtual machine resources	Crash	Error	Exception
Excessive local reference creation	Running	Error	Exception
Using invalid local references	Error	Error	Exception
Using the JNIEnv across threads	Error	Crash	Exception

Jinn adds modest time overhead



Jinn finds JNI bugs in real world applications

Programs	bug reports	Community response
	1	Confirmed: bug 69510896
	1	To be reported
	5	Fixed: r949842, r946181, r944525, r947006, r946518
	2	Fixed: r676 Confirmed: bug 576111

Related work

How about legacy JNI programs?

Hirzel & Grimm ’07

Tan et al. ’06

Safe interface languages

Related work

How about false alarms?

Li & Tan '09

Hirzel & Grimm '07

Kondoh & Onodera '08

Tan et al. '06

Furr & Foster '06

Safe interface languages

Static bug finders

Related work

How about low coverage?

	Li & Tan ’09	
Hirzel & Grimm ’07	Kondoh & Onodera ’08	J9
Tan et al. ’06	Furr & Foster ’06	Hotspot
Safe interface languages	Static bug finders	Dynamic checking in JVMs

Summary

- FFI has many programming constraints and bugs.
- Synthesis of dynamic FFI bug detectors
  - Classification system for characterizing language semantic mismatches
  - State machine transitions in terms of language transitions.
- Jinn: An effective dynamic bug detector for JNI
  - High coverage
  - Modest overhead
  - Finds bugs in real-world JNI programs

Thank you