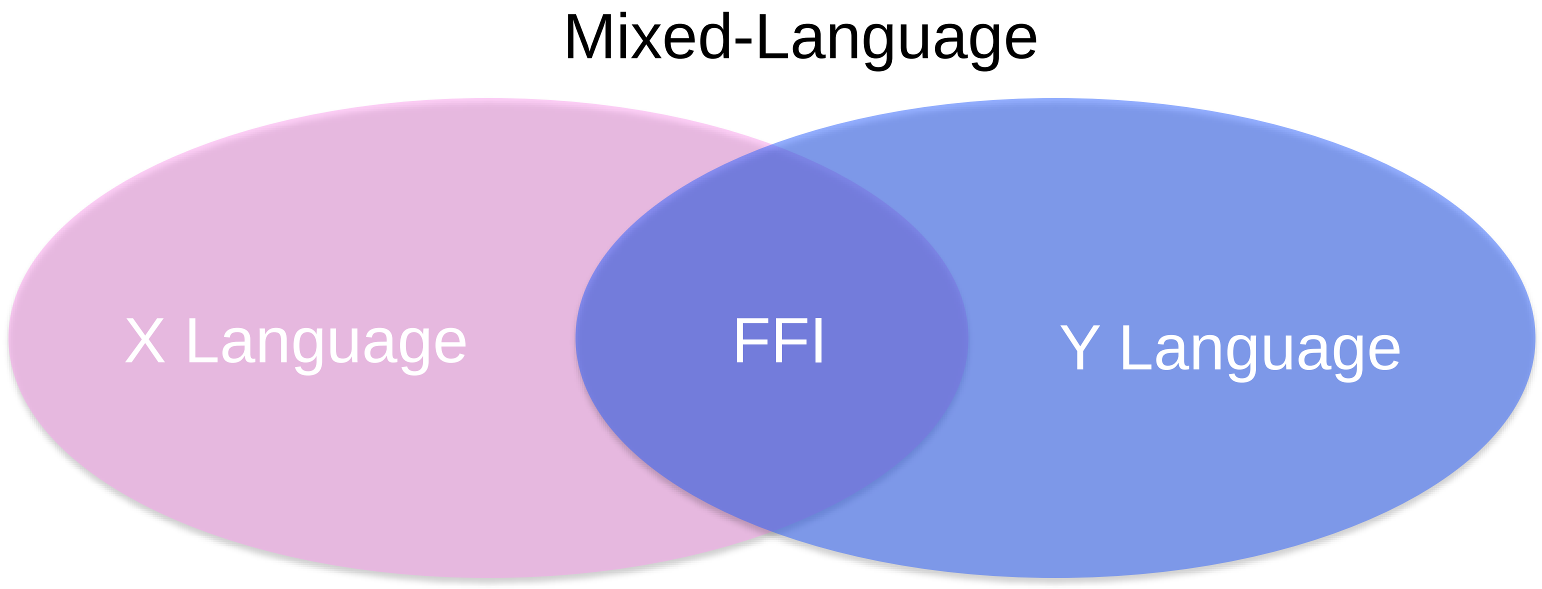
Debug All Your Code: Portable Mixed-Environment Debugging

Byeongcheol Lee
Martin Hirzel
Robert Grimm
Kathryn McKinley

Portable mixed-environment debugging

Programmers build systems in multiple languages.

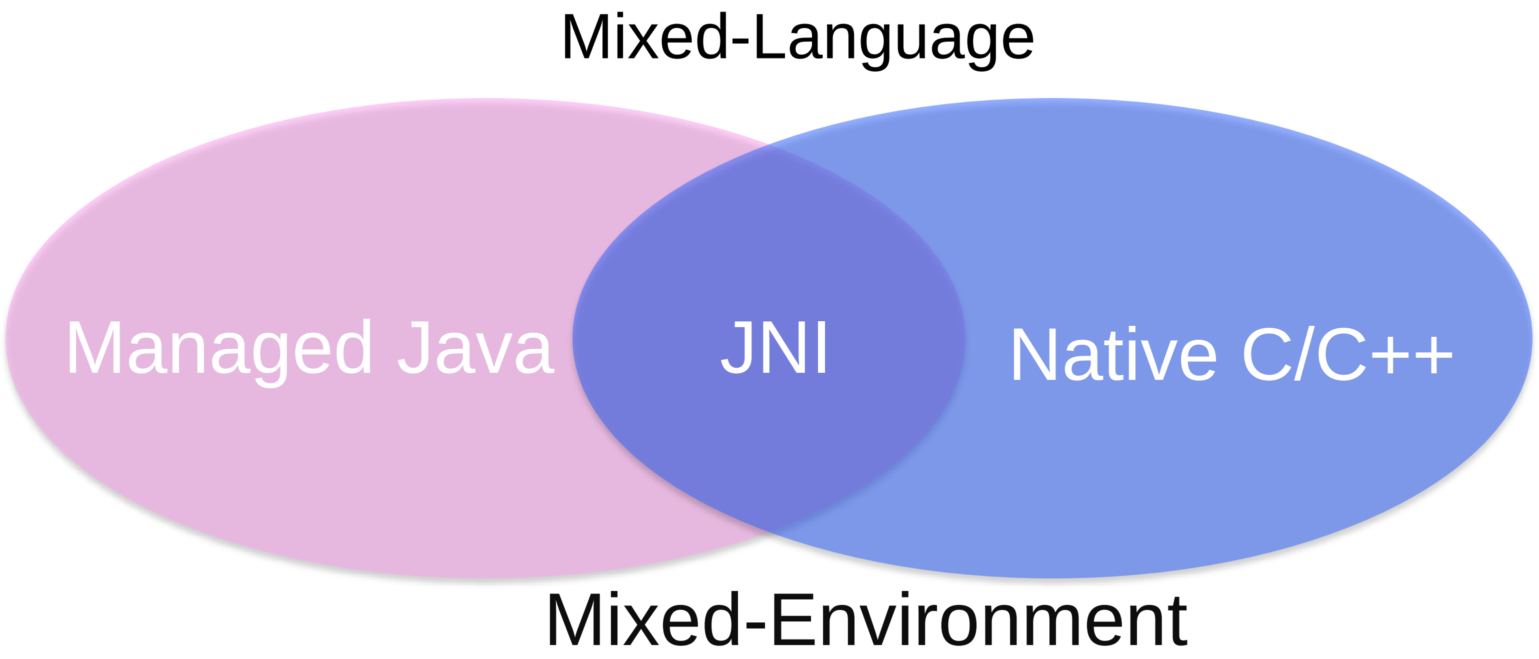
1. Leverage legacy code and existing libraries.
2. Match language features to a task.



Portable mixed-environment debugging

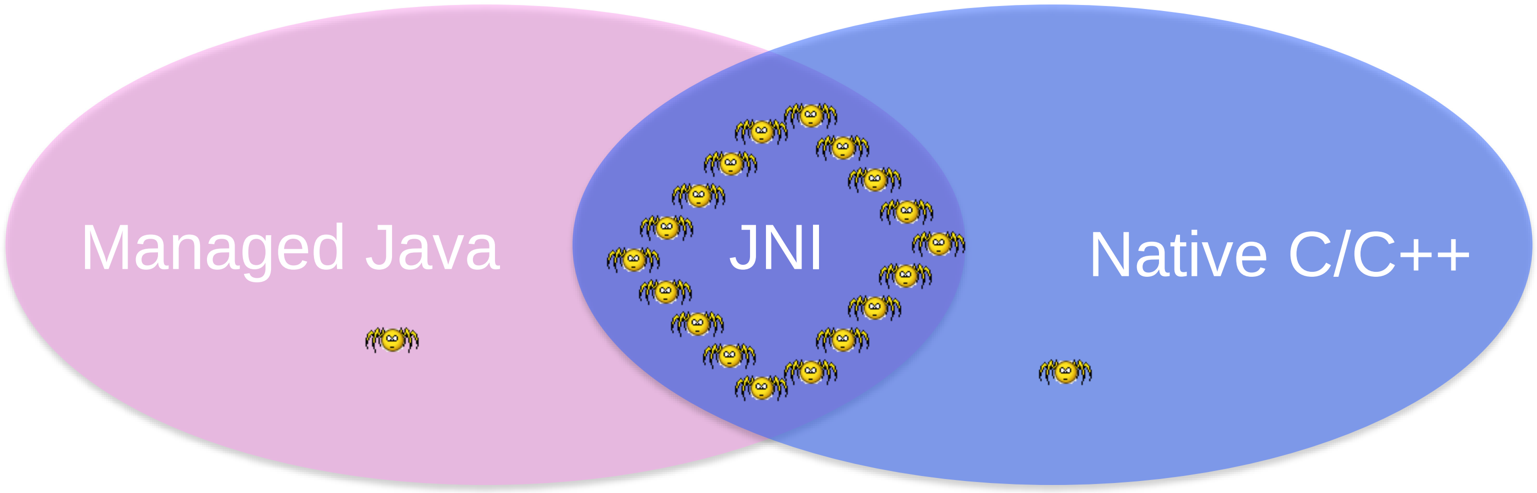
Programmers build systems in multiple languages.

1. Leverage legacy code and existing libraries.
2. Match language features to a task.



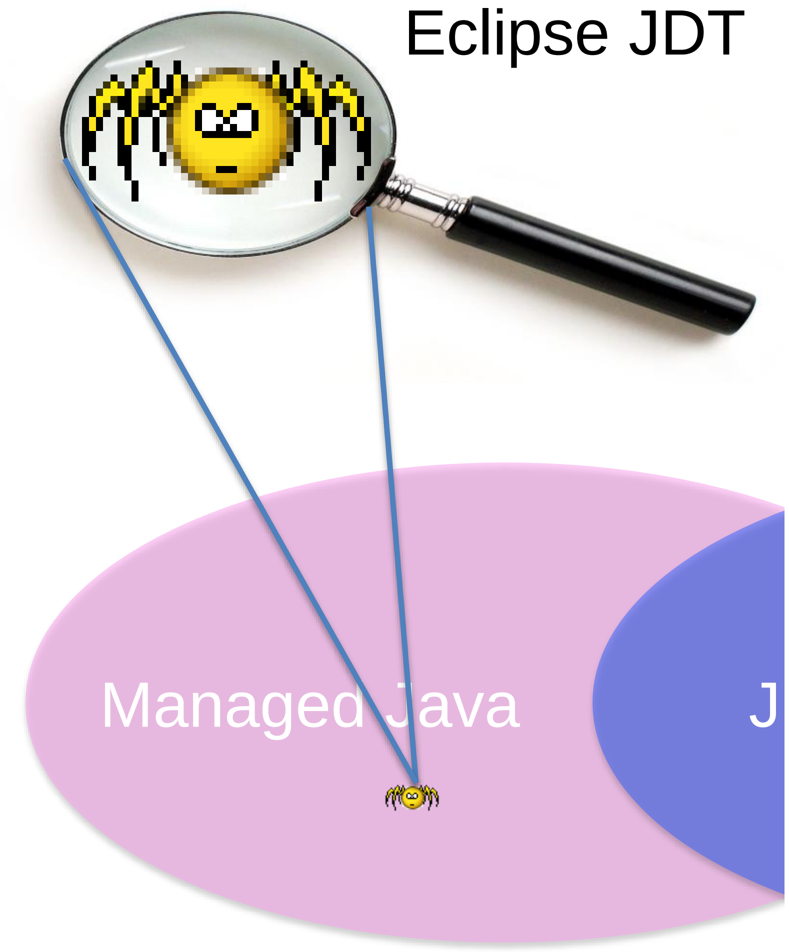
The problem

Bugs appear in all your code!



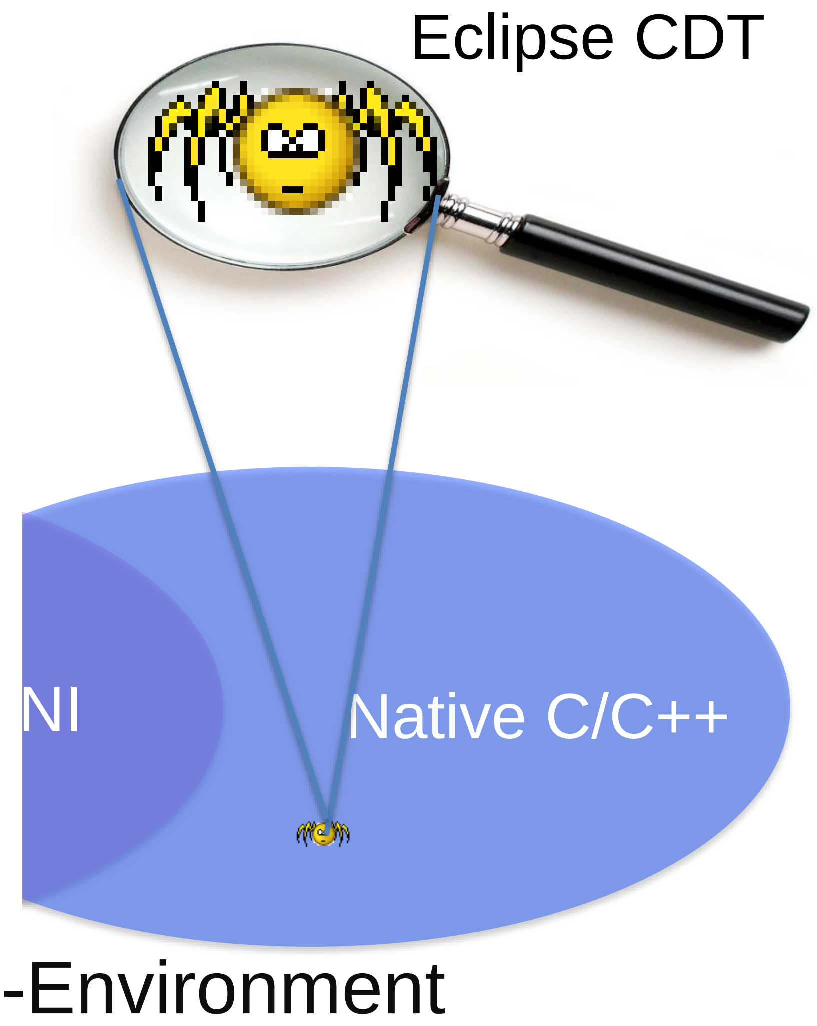
Mixed-Environment

The problem with single-environment debugging



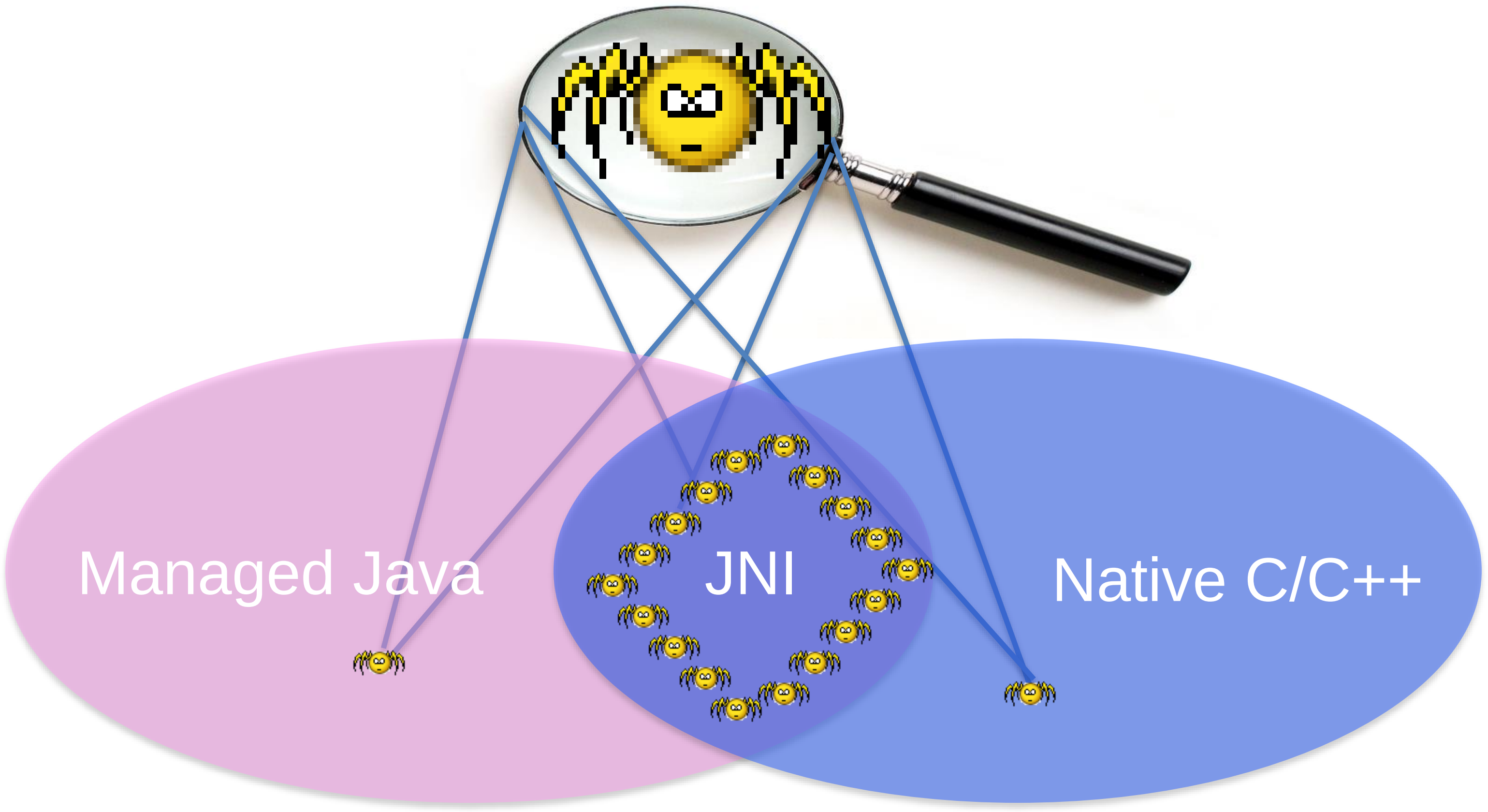
Mixed-Environment

The problem with single-environment debugging



Our goal

Portable mixed-environment debugging



Mixed-Environment

Our contribution and results

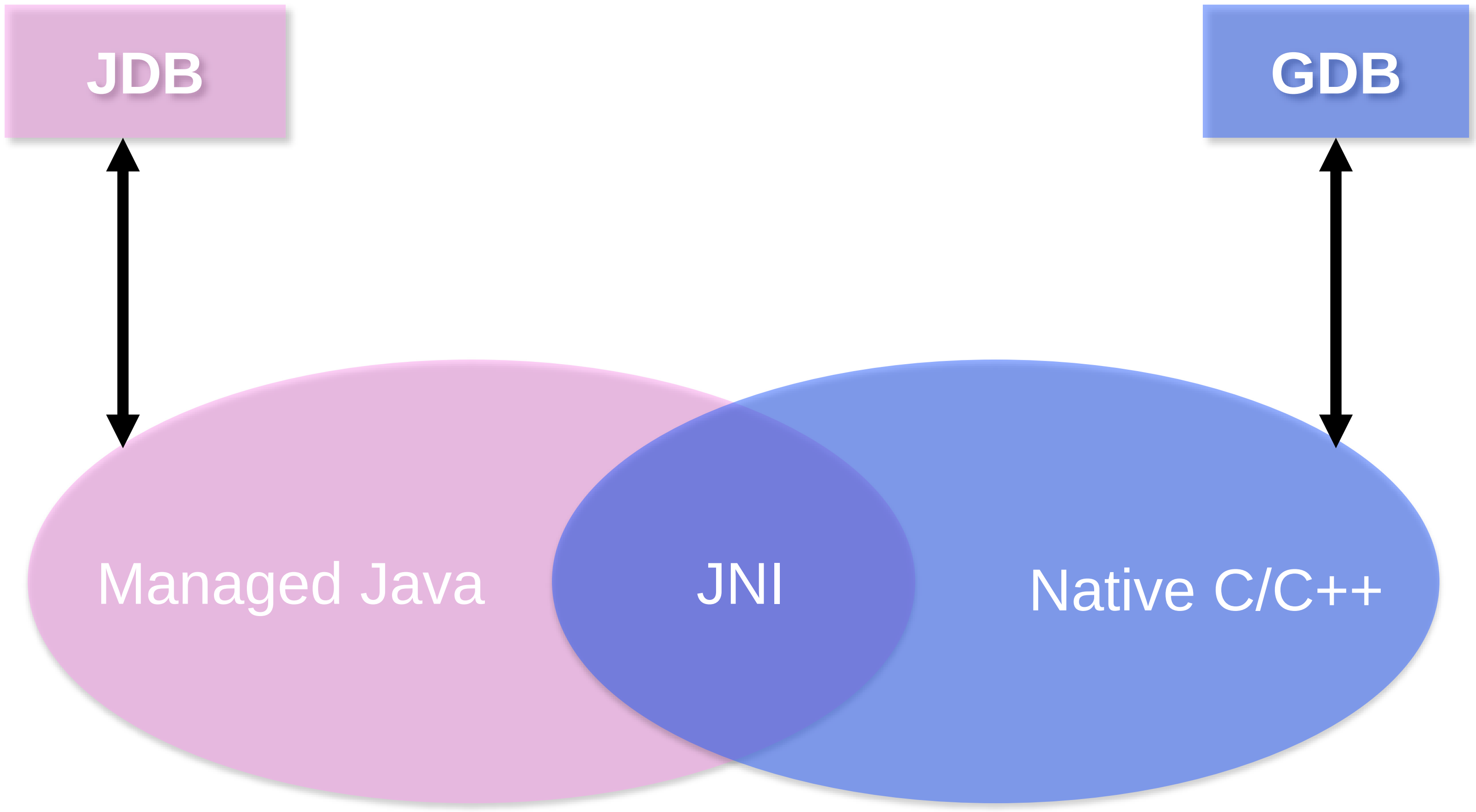
Composition

1. Add an intermediate agent.
2. Attach single-environment debuggers.
3. Dispatch debuggers dynamically.

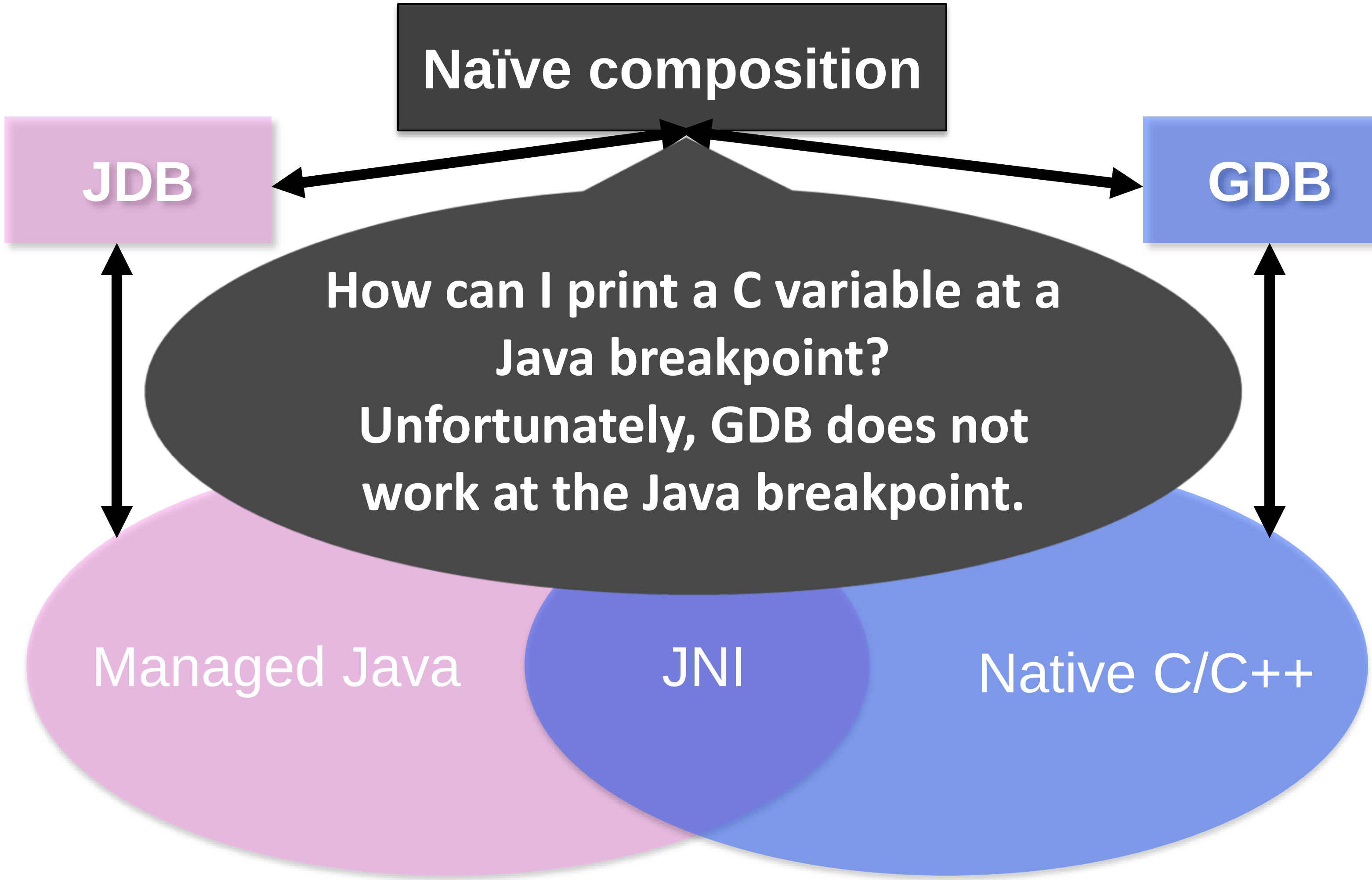
Blink results

1. **Simple**: Add 10 K SLOC of new code.
2. **Portable**: Support Linux, Windows, Hotspot, J9, GCC, Microsoft C++.
3. **Powerful**: Catch FFI bugs.

Problem: GDB does not work at a Java breakpoint.

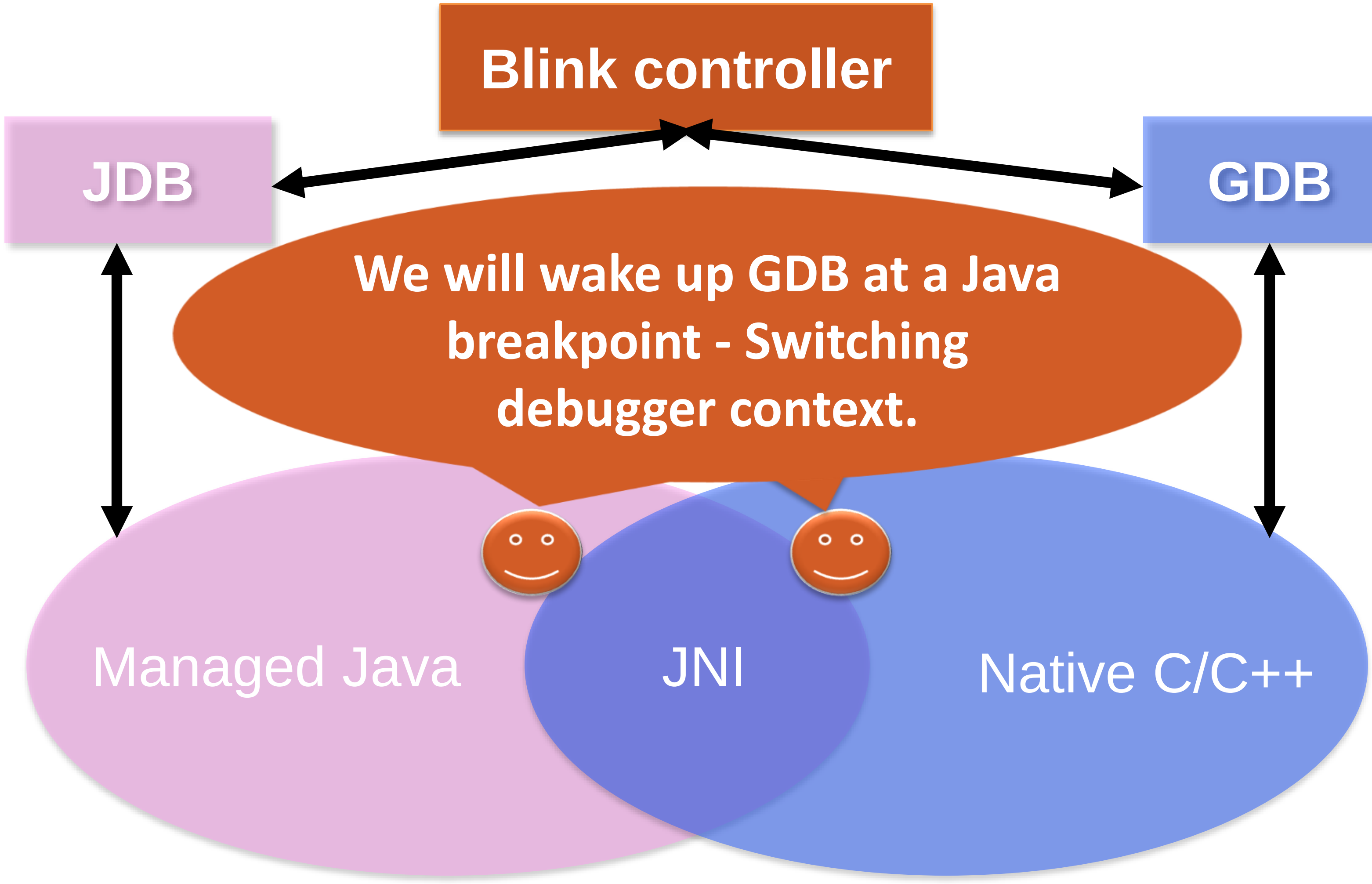


Problem: GDB does not work at a Java breakpoint.

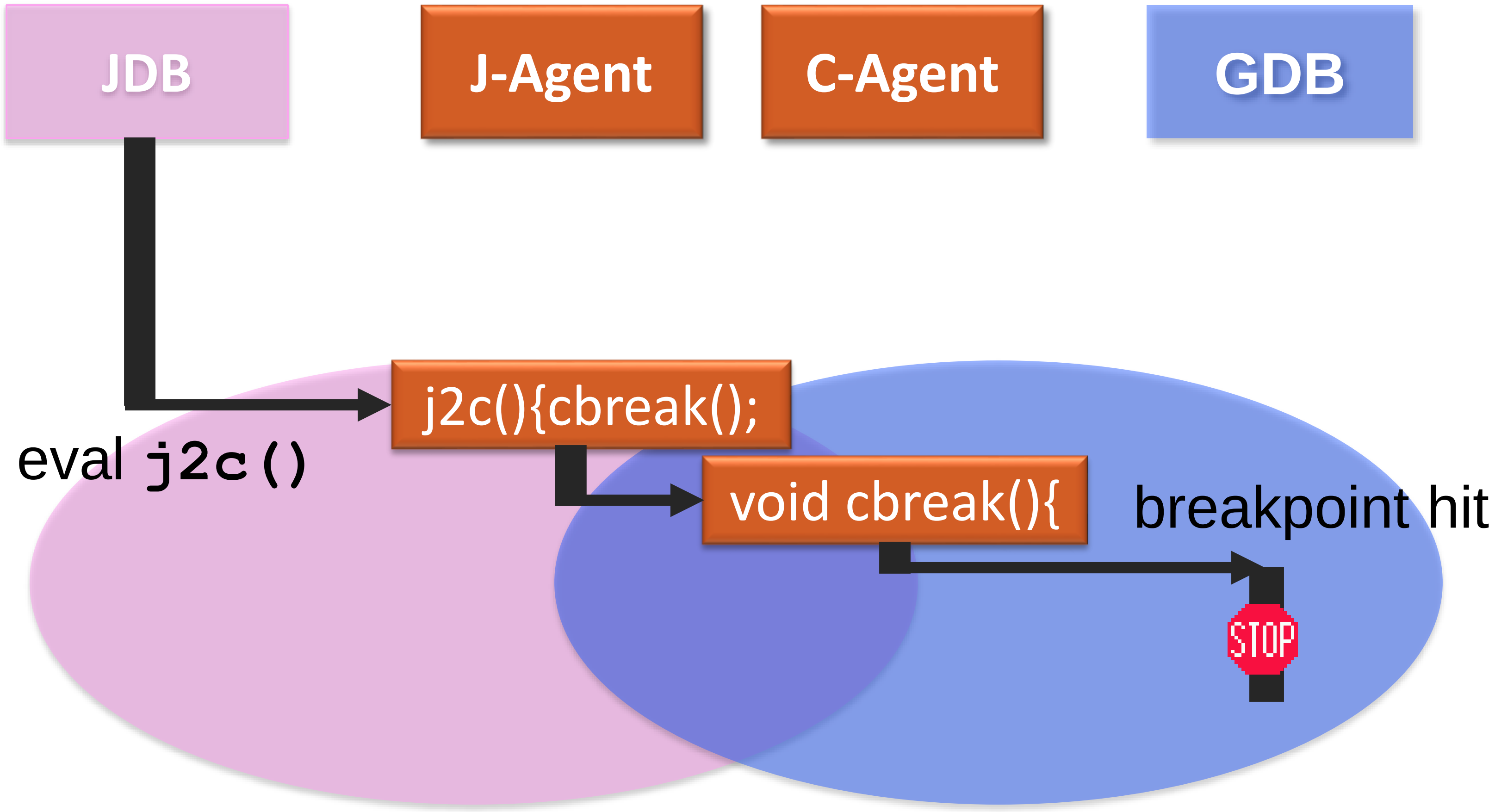


How can I print a C variable at a Java breakpoint?
Unfortunately, GDB does not work at the Java breakpoint.

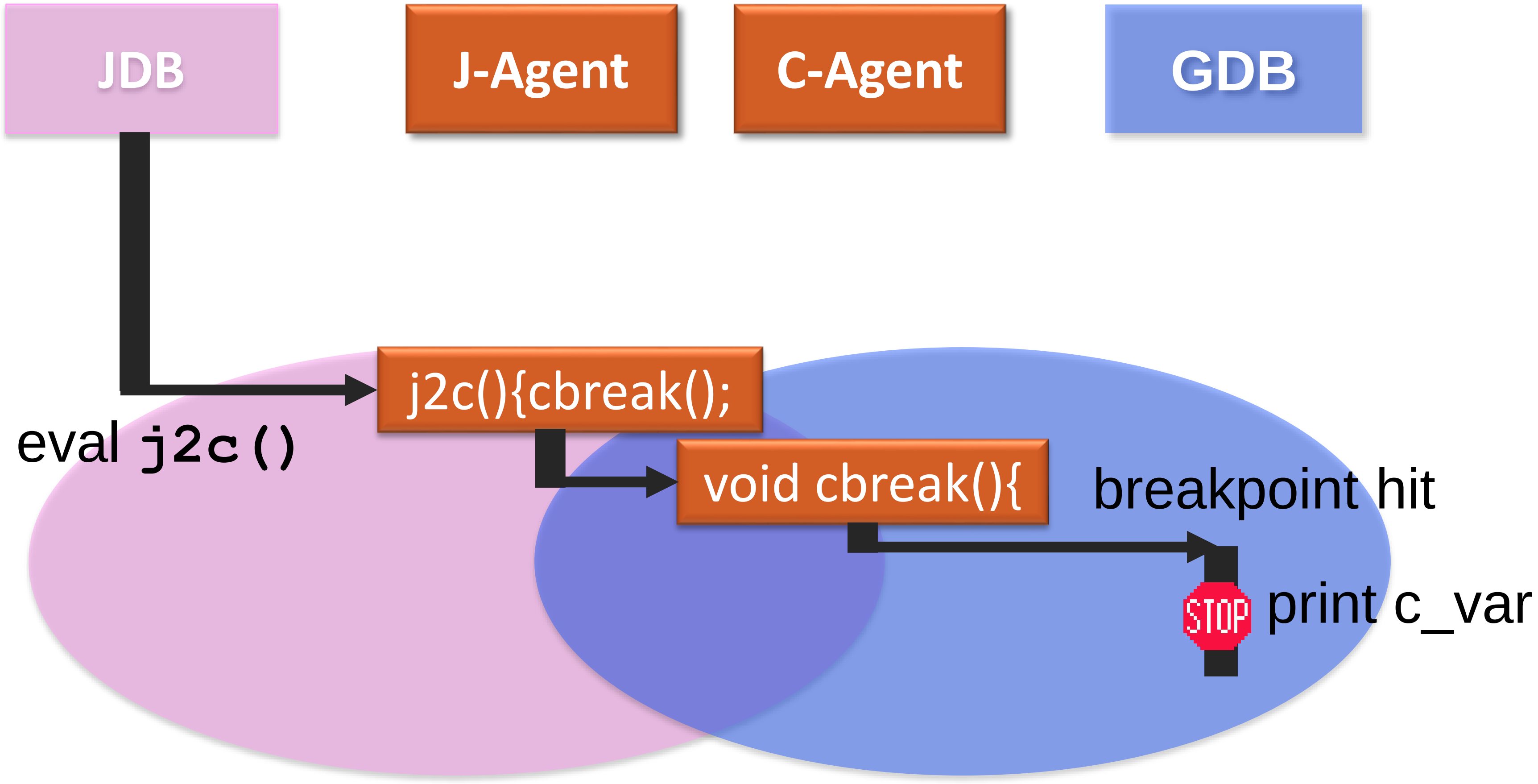
Our solution: the intermediate agent switches debugger context.


Blink controller
JDB
GDB
We will wake up GDB at a Java breakpoint - Switching debugger context.
Managed Java
JNI
Native C/C++

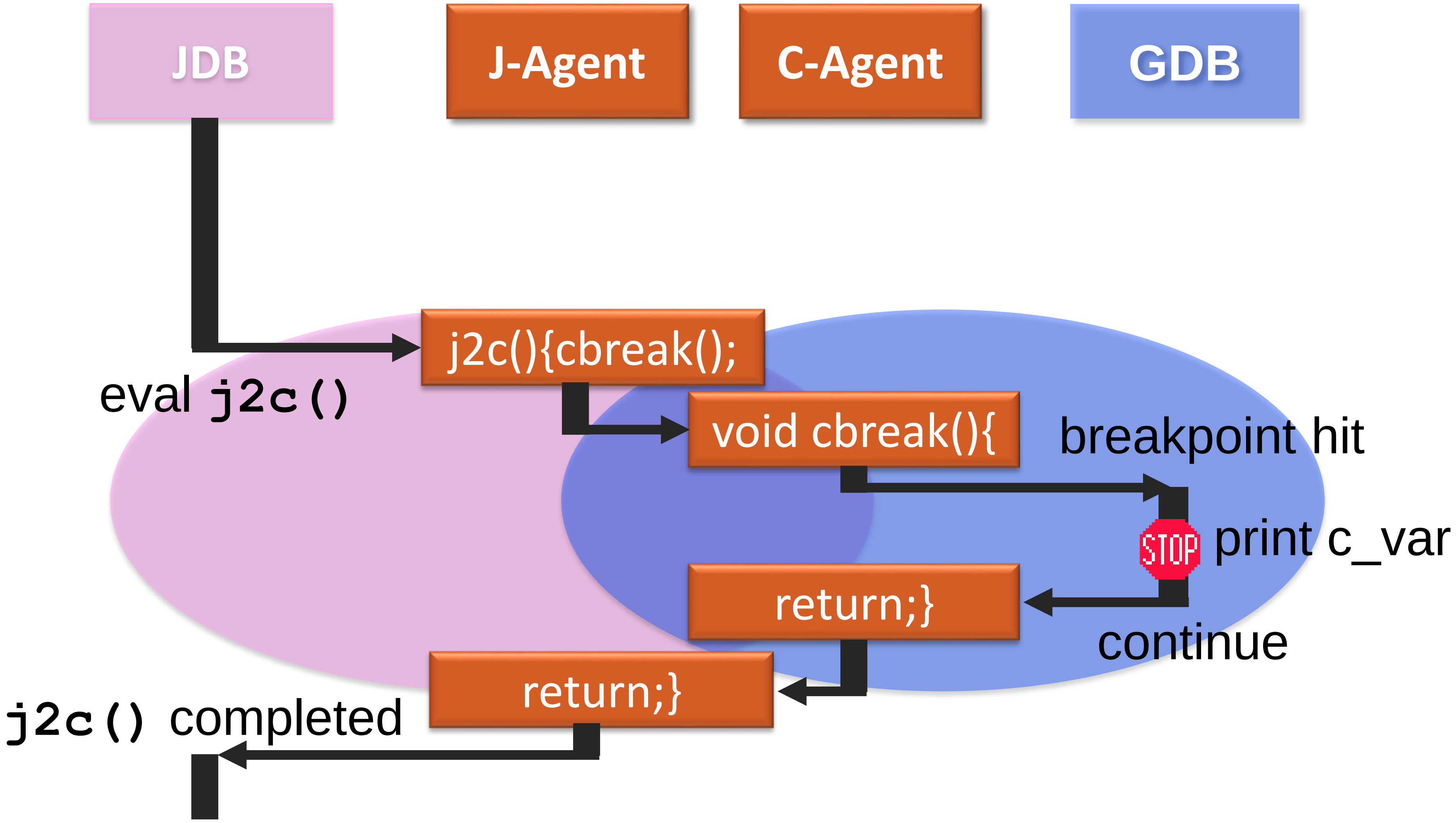
Our solution: switch debugger context from Java to C.



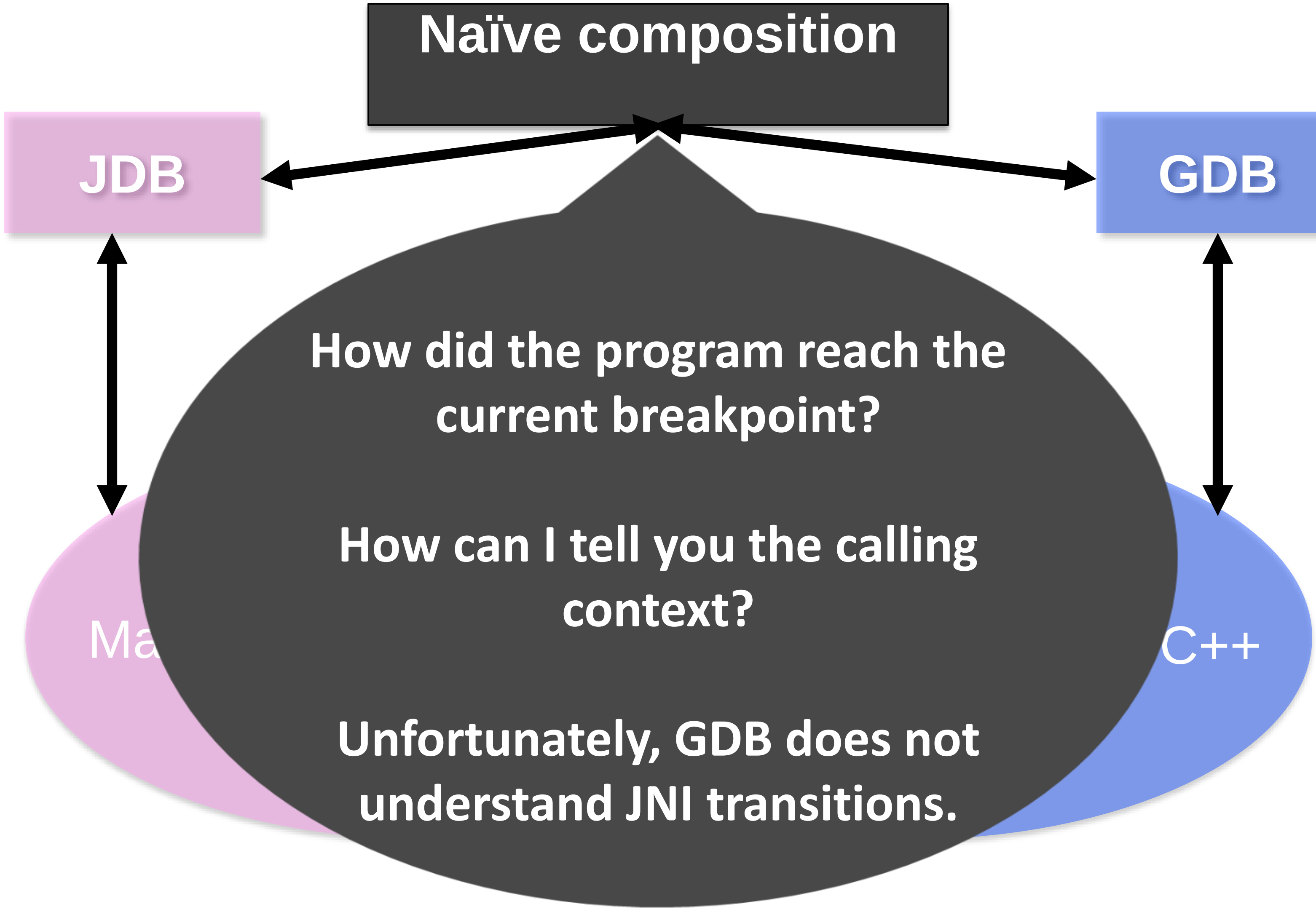
Our solution: switch debugger context from Java to C.



Our solution: switch debugger context from Java to C.



Problem: GDB does not understand JNI calling conventions.

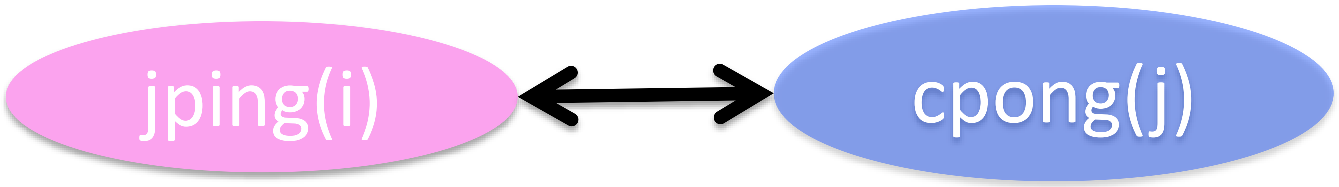


How did the program reach the current breakpoint?

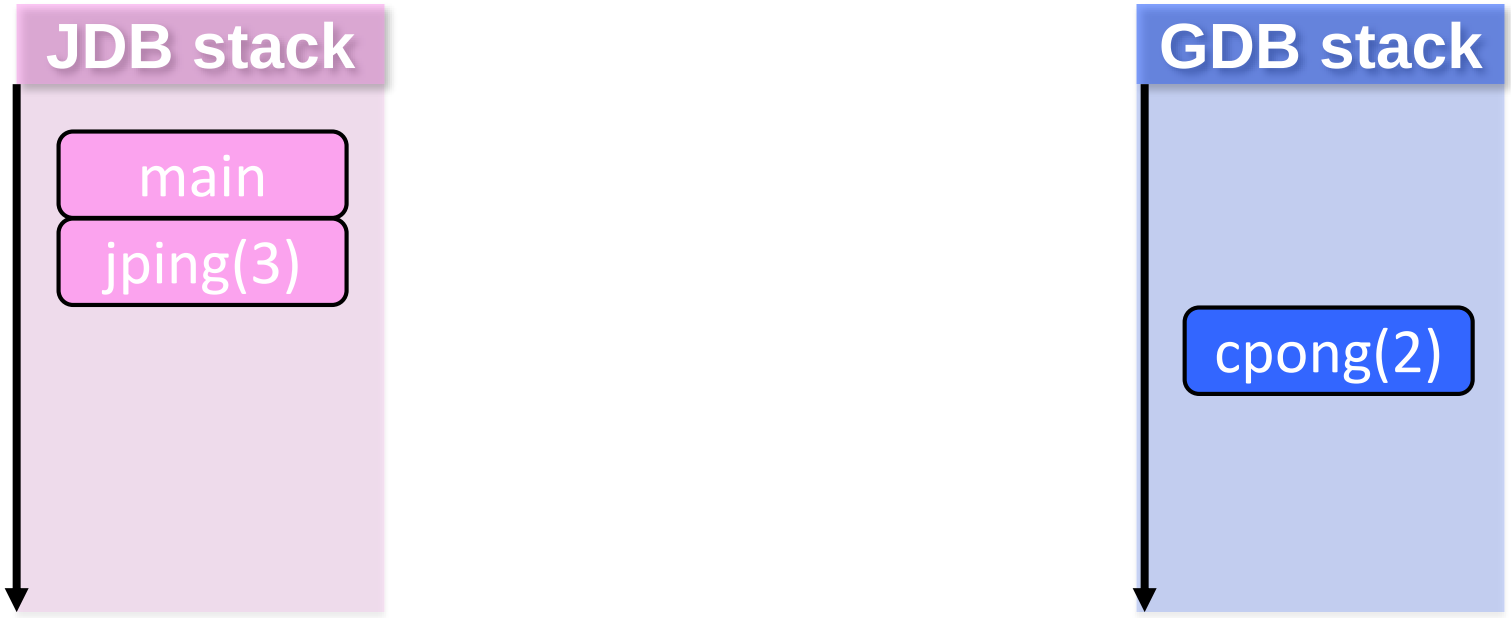
How can I tell you the calling context?

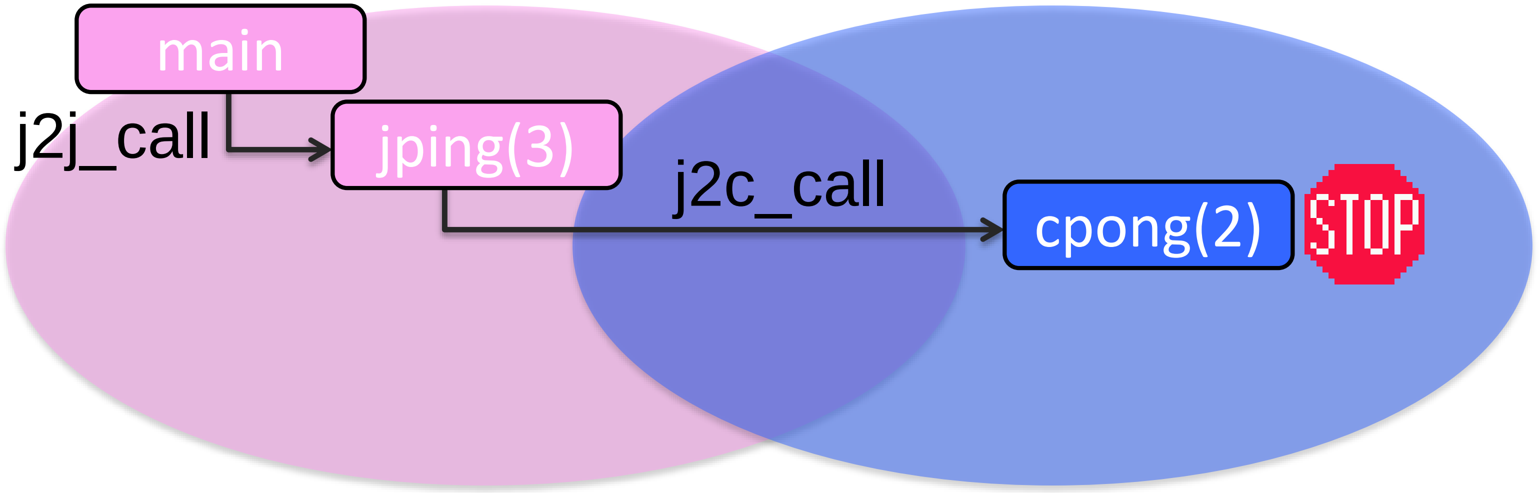
Unfortunately, GDB does not understand JNI transitions.

Problem: GDB does not understand JNI calling conventions.

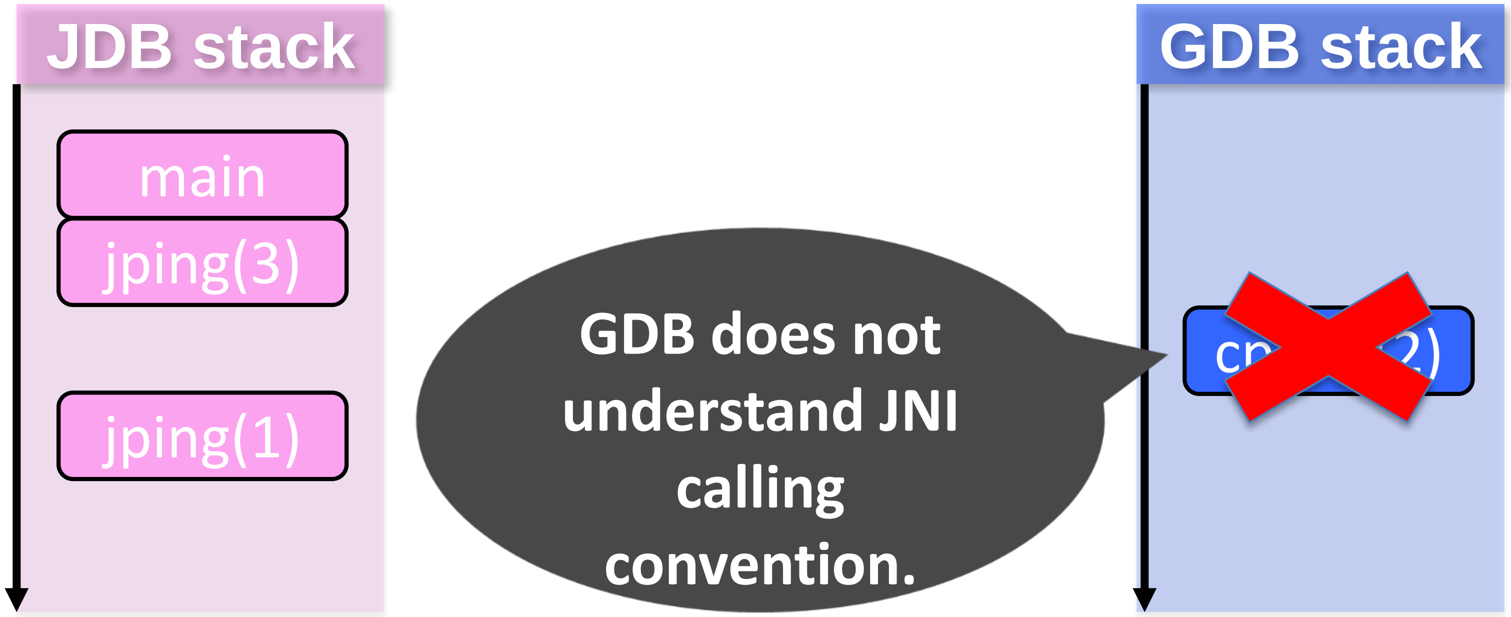


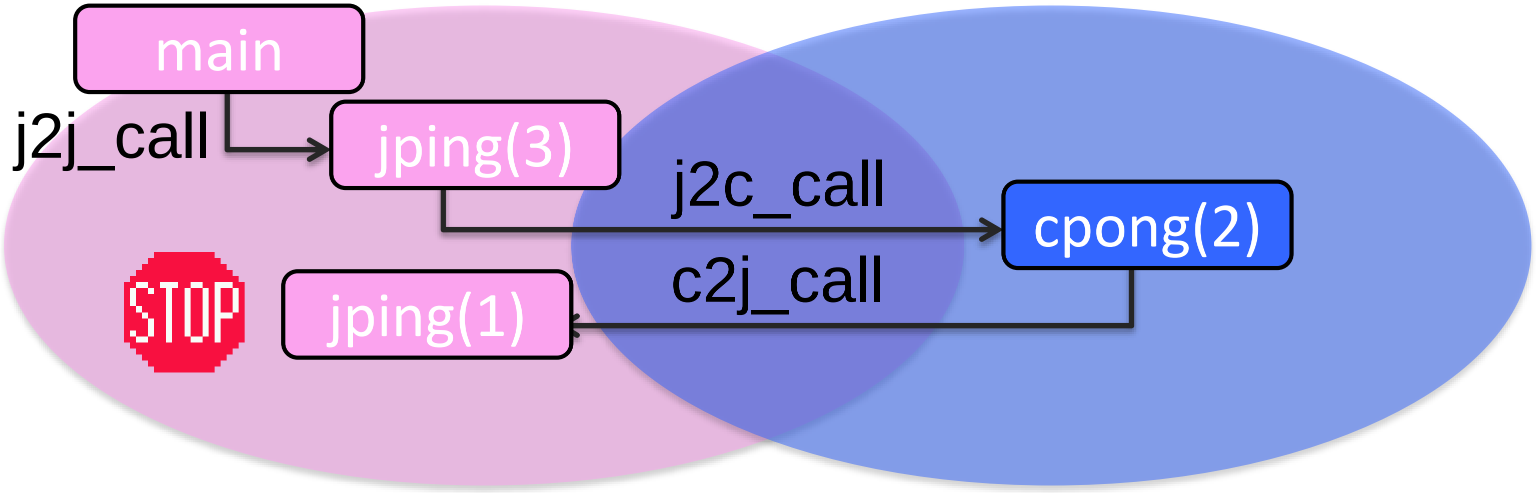
Problem: GDB does not understand JNI calling conventions.



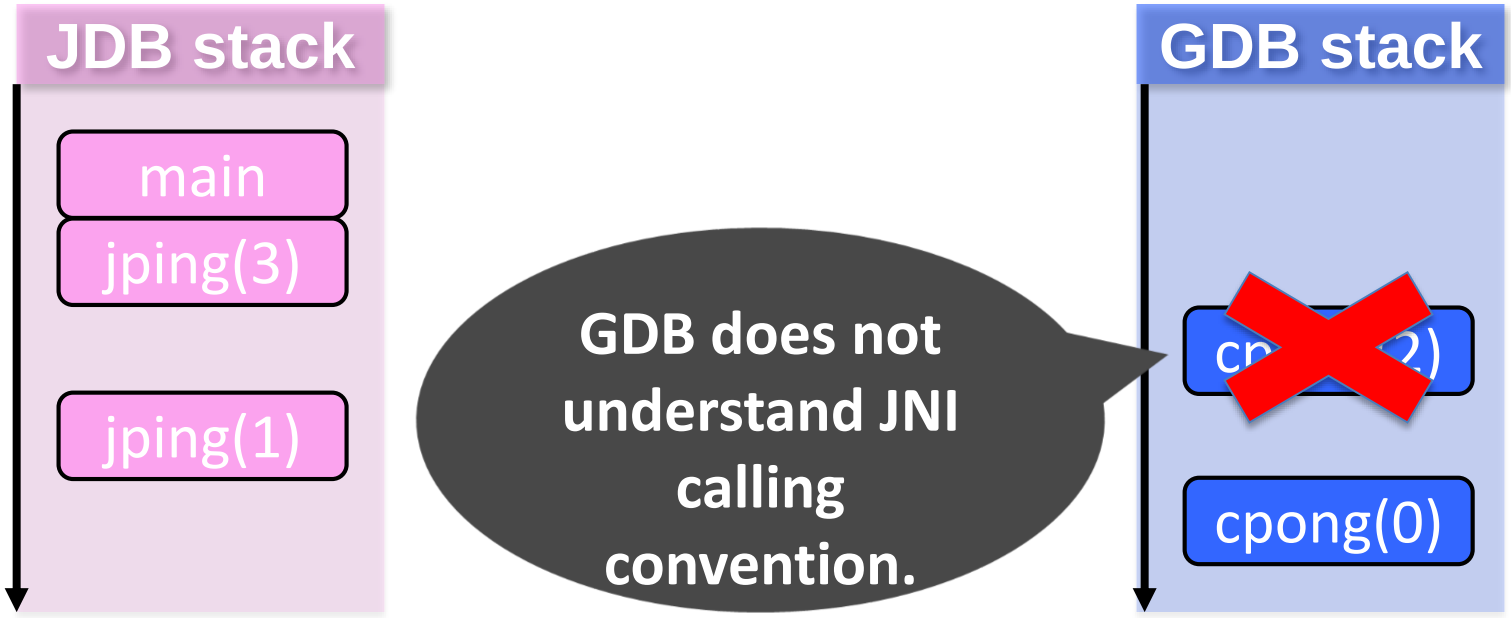


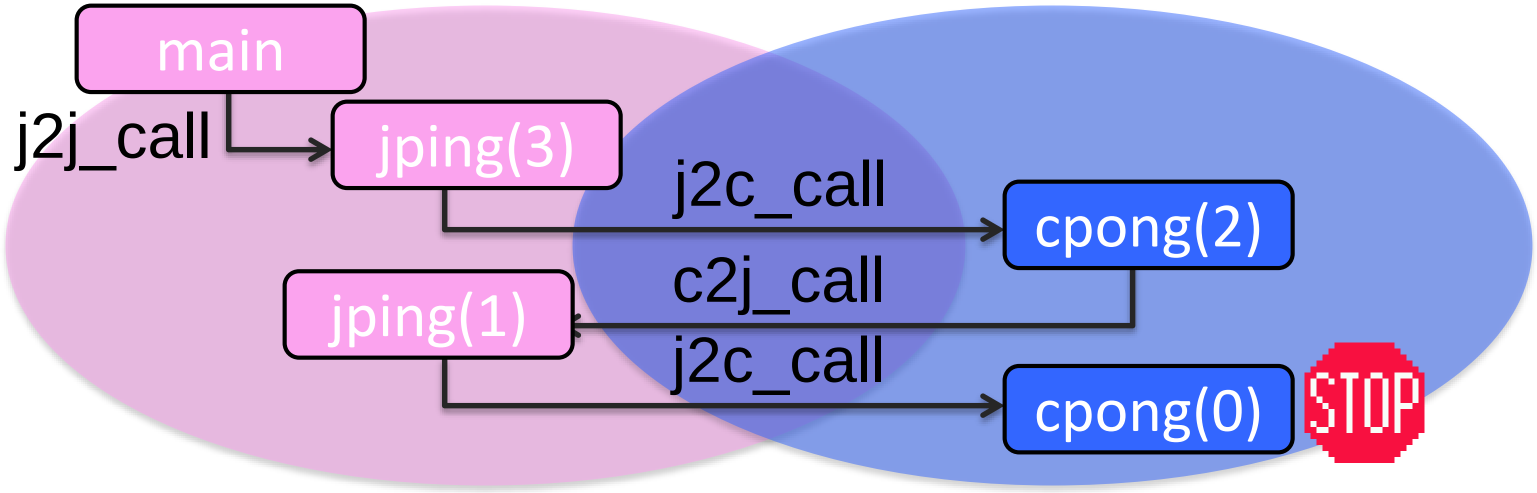
Problem: GDB does not understand JNI calling conventions.



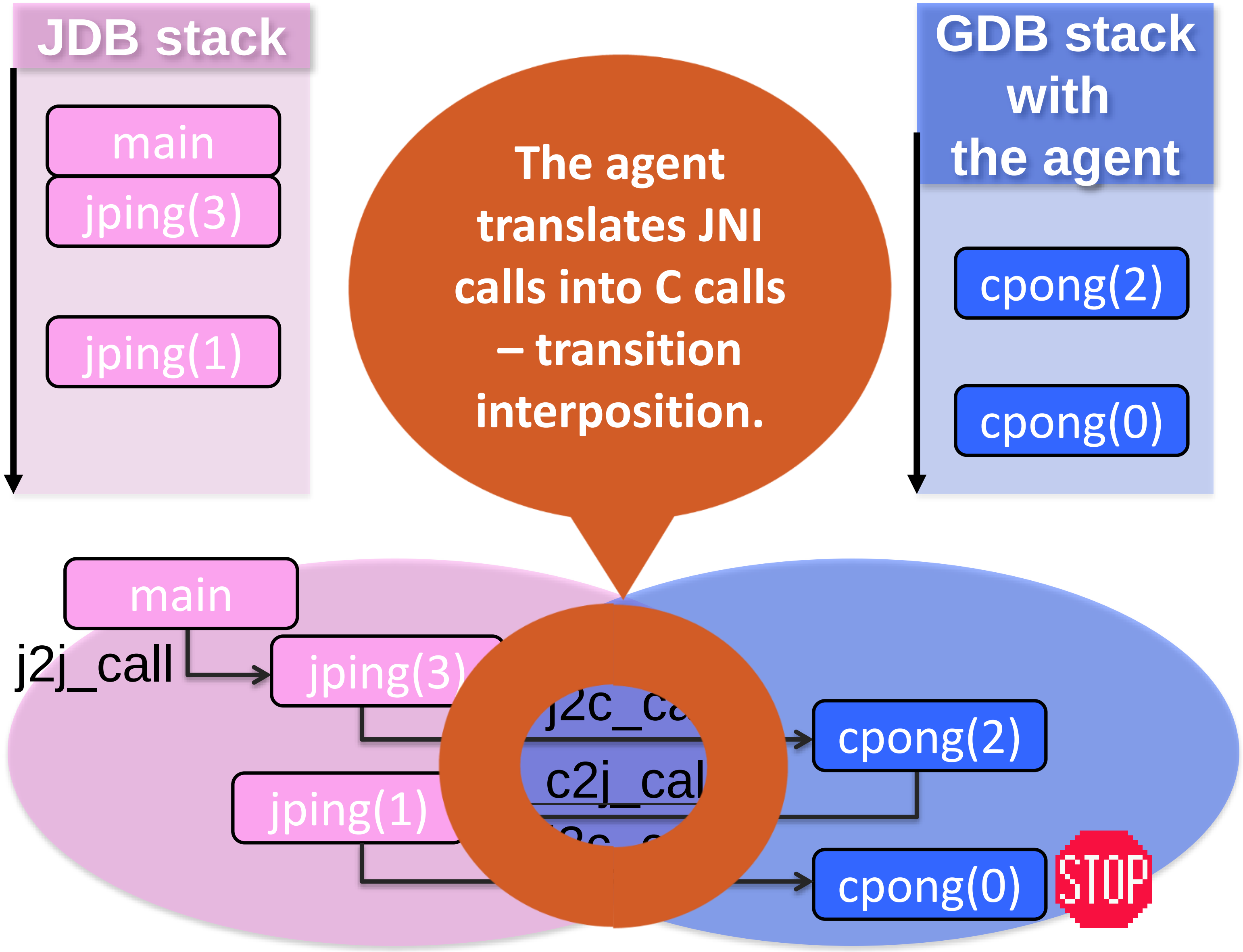


Problem: GDB does not understand JNI calling conventions.

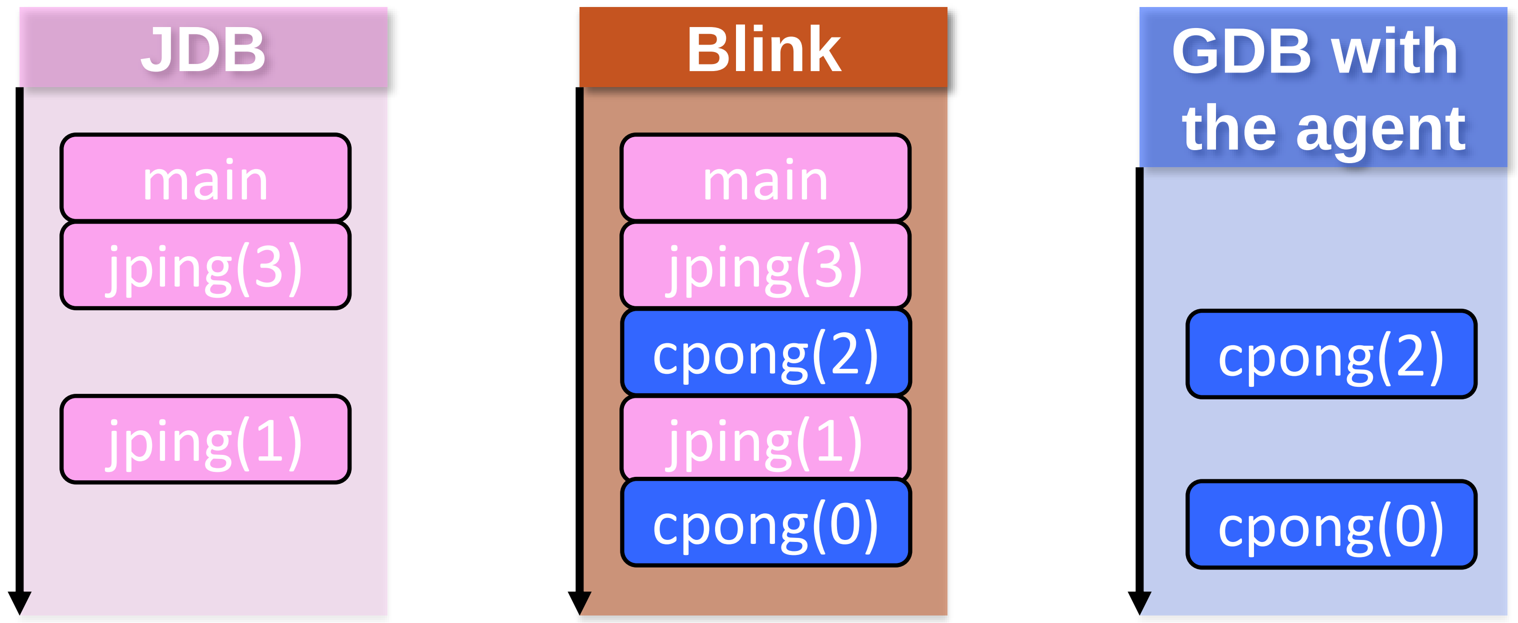


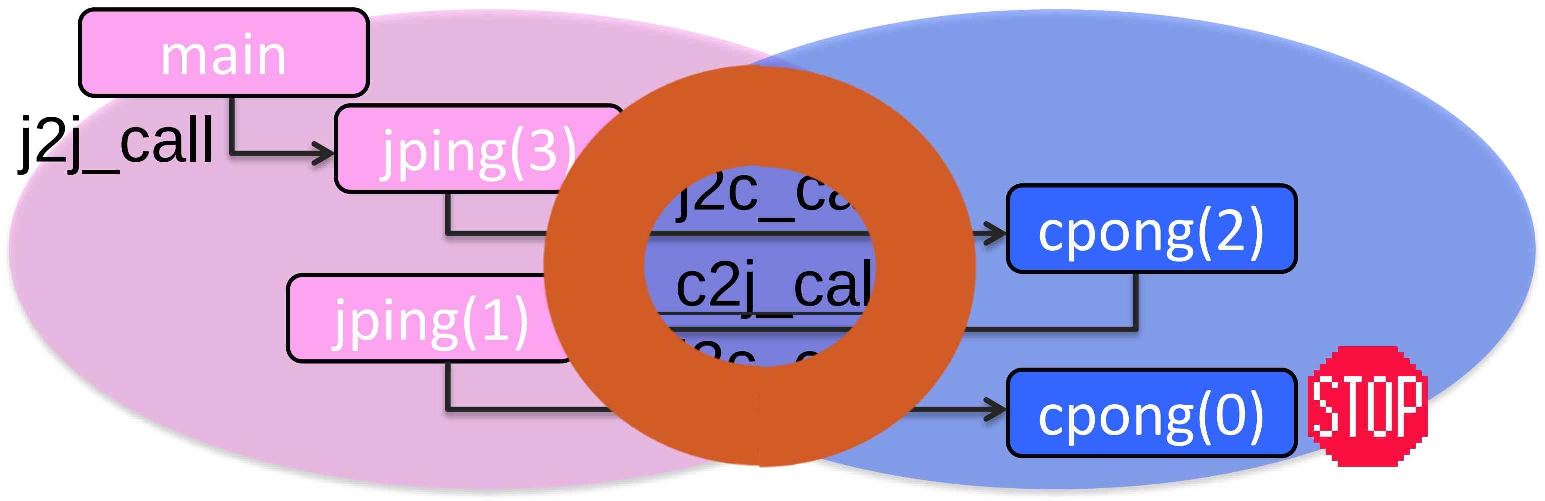


Our solution: translate JNI transitions into C transitions.

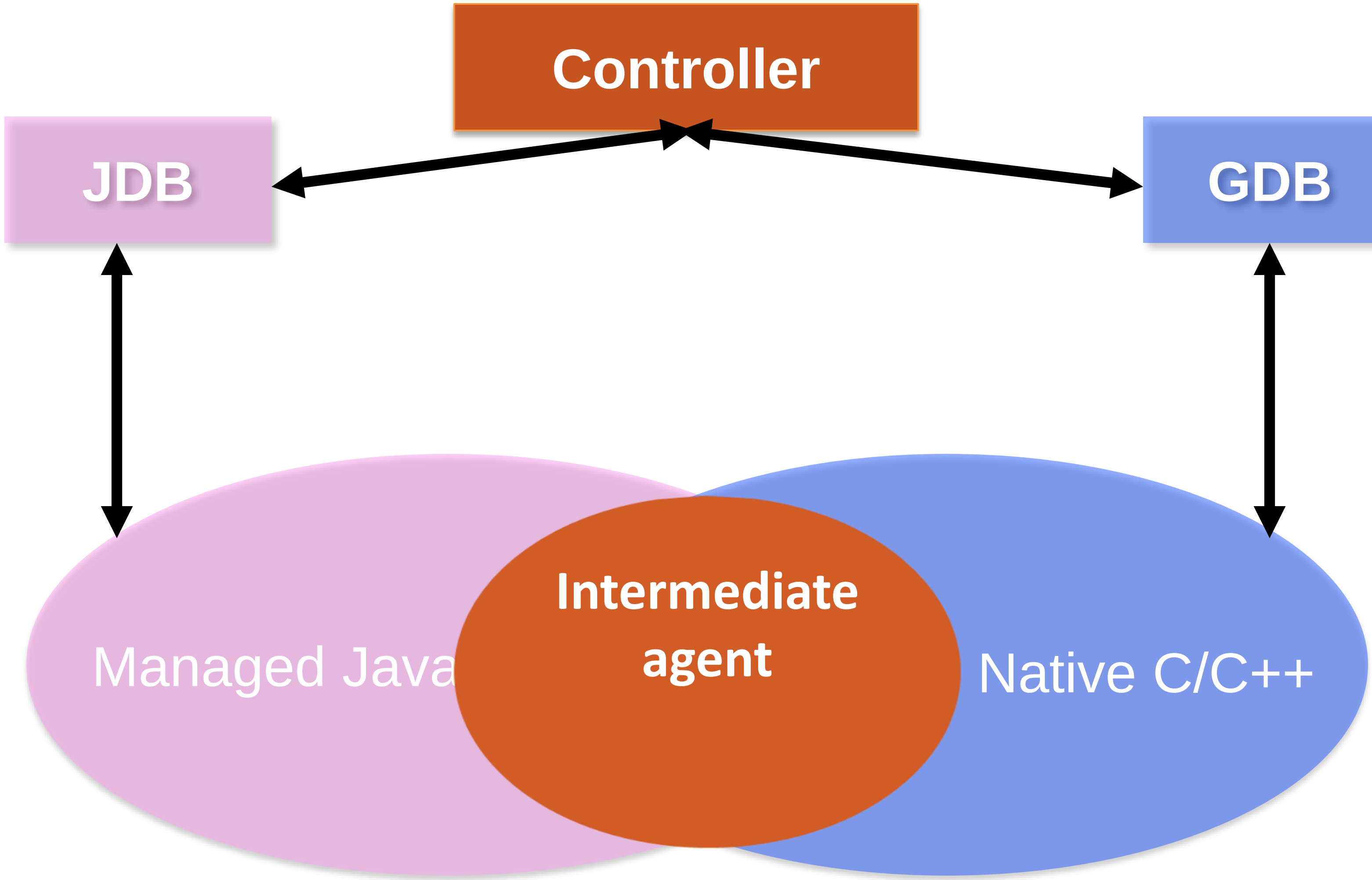


Our solution: compose a calling context.





Our solution: controller and intermediate agent

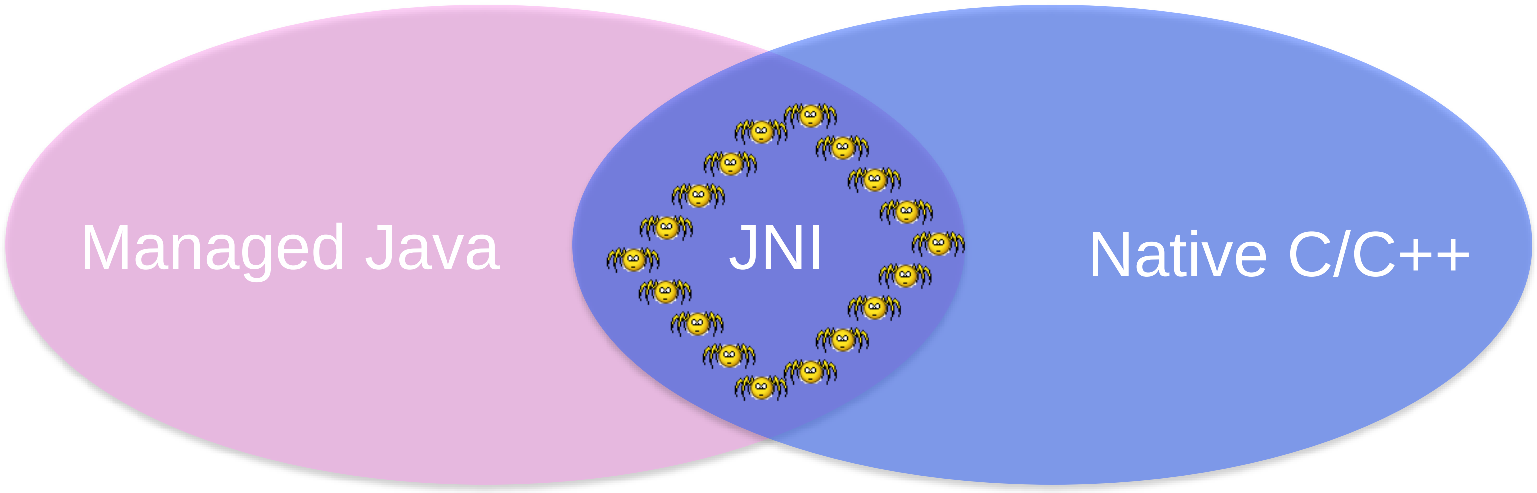


Outline

I. Problem
II. Debugger composition
    A. Switching debugger context
    B. Interposing transitions
→ III. Advanced features
    A. Evaluating Jeannie mixed-environment expressions
    B. Detecting FFI bugs
IV. Evaluation

Debugging boundary code

What do I need to debug boundary code?

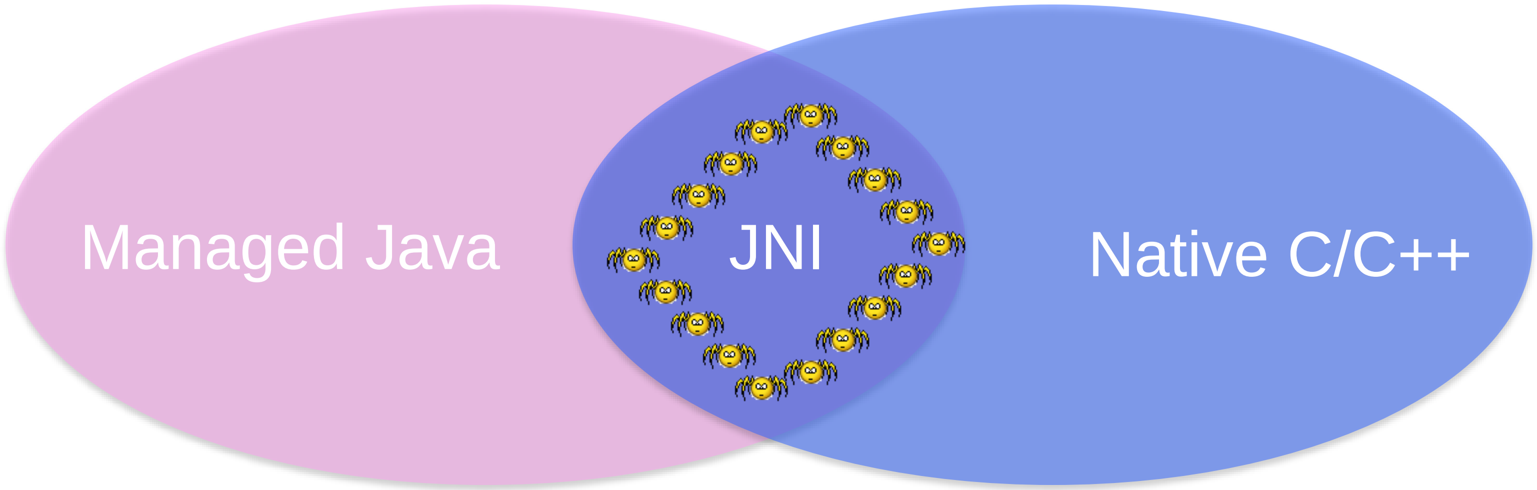


Mixed-Environment

Advanced features to debug boundary code

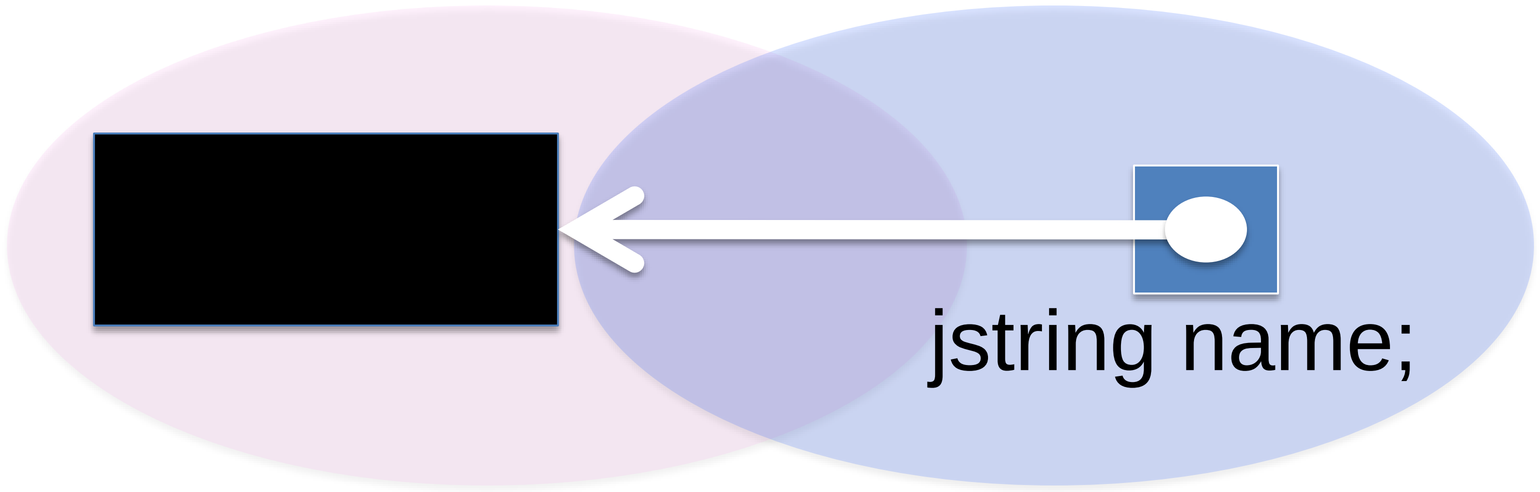
1. Evaluating Jeannie Expressions

2. Detecting FFI bugs

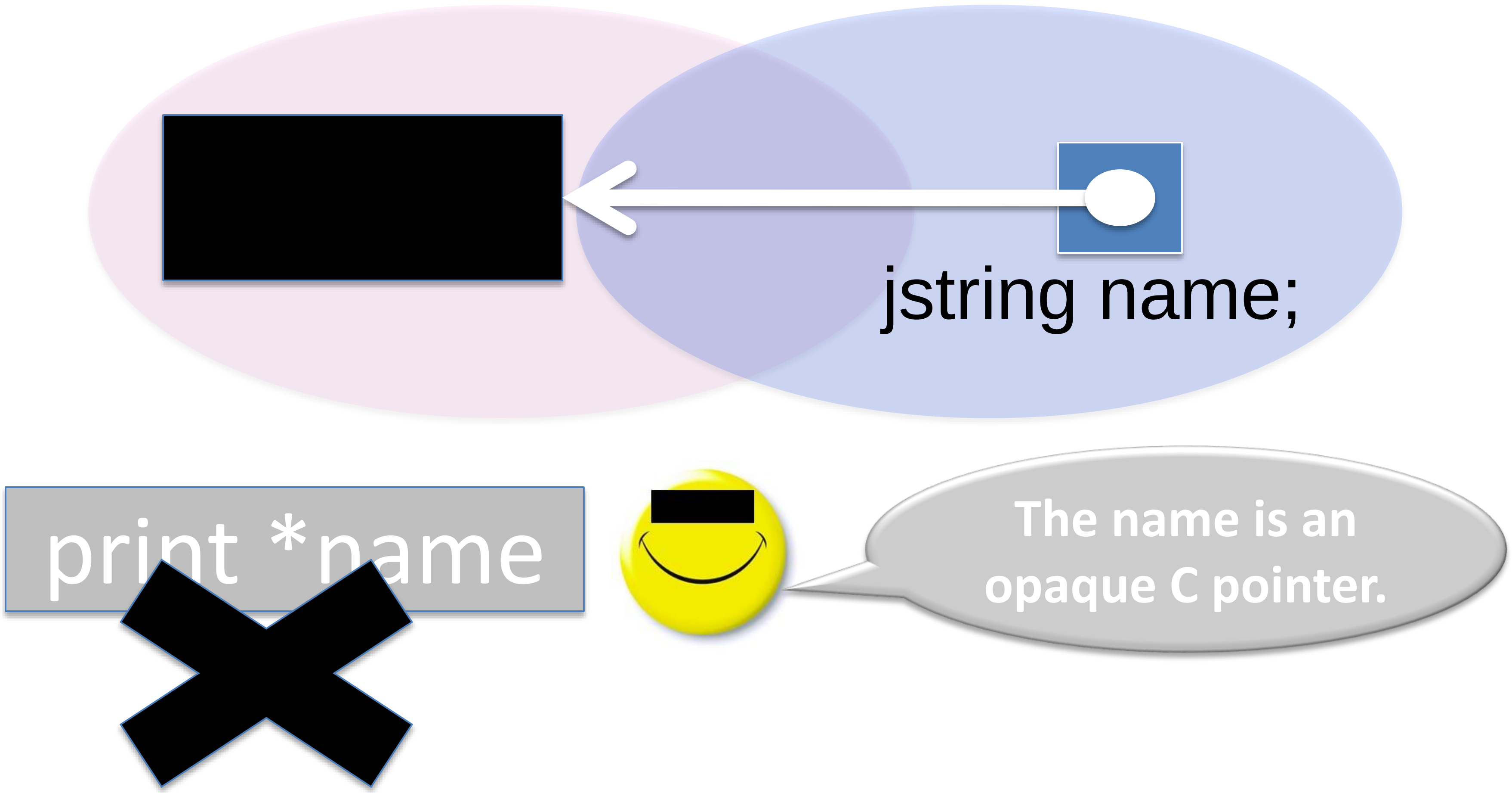


Mixed-Environment

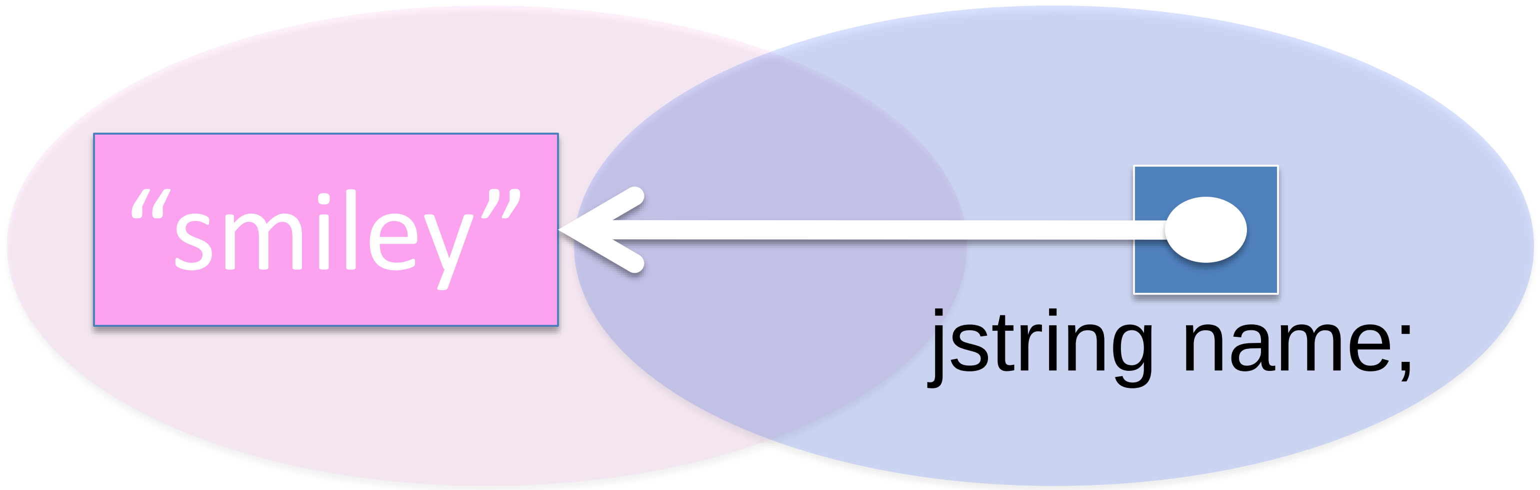
Problem: you can not de-reference opaque pointers in C.

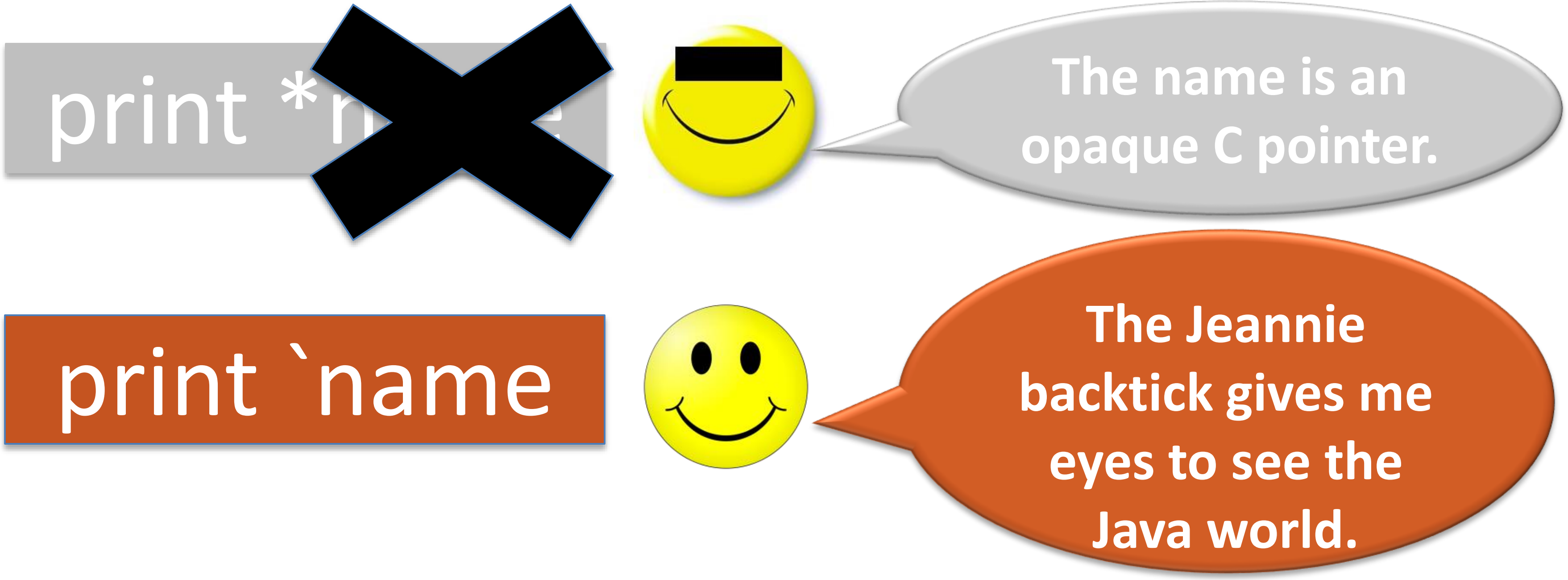


Problem: you can not de-reference opaque pointers in C.

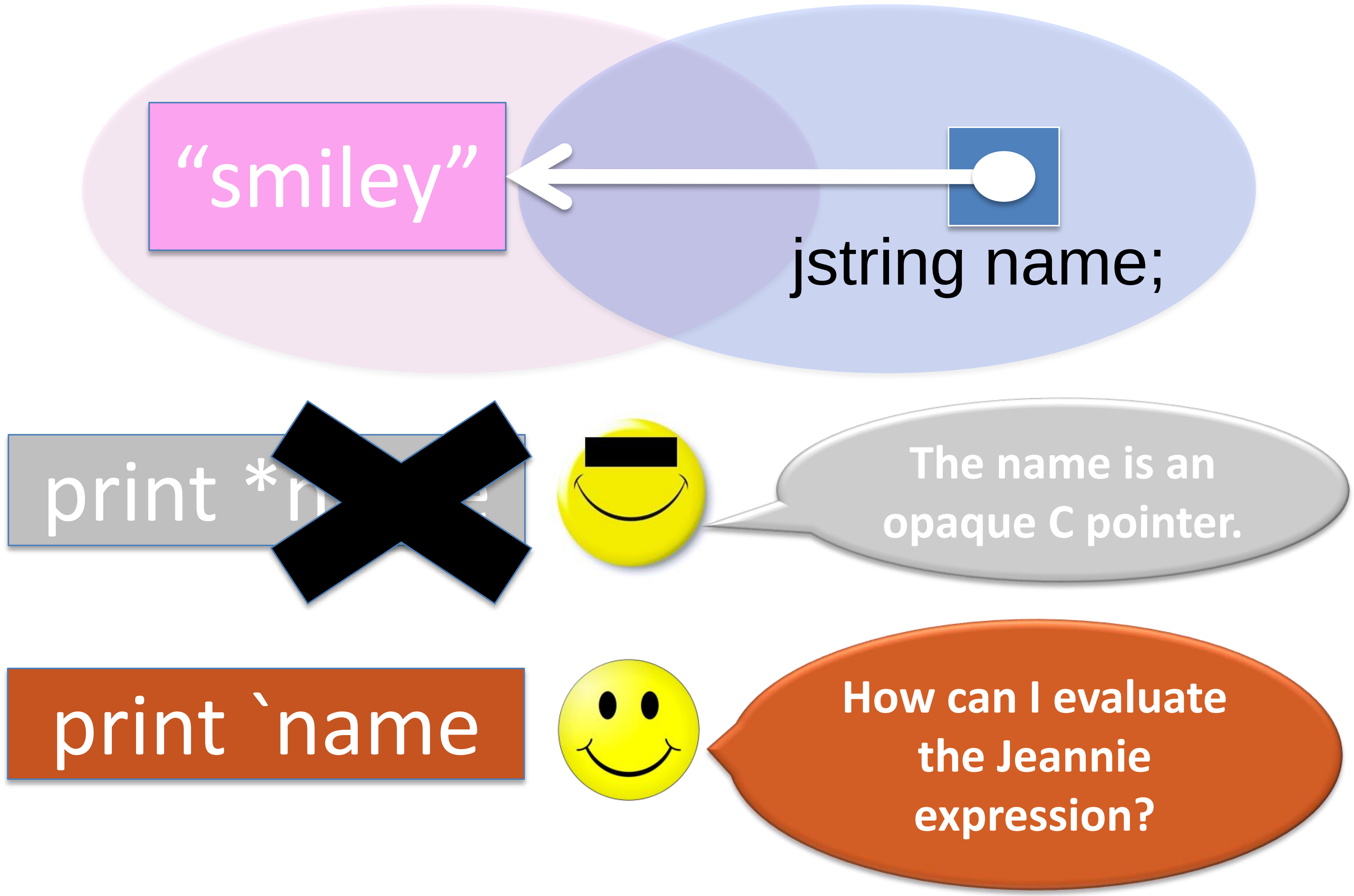


Our solution: use Jeannie expression



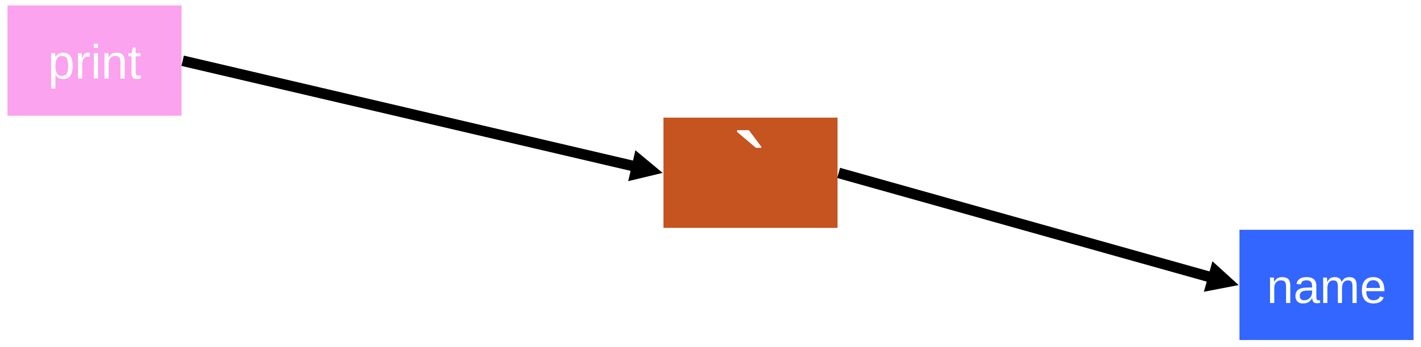

~~print *n e~~
The name is an opaque C pointer.
print `name
The Jeannie backtick gives me eyes to see the Java world.

Our solution: use Jeannie expression.

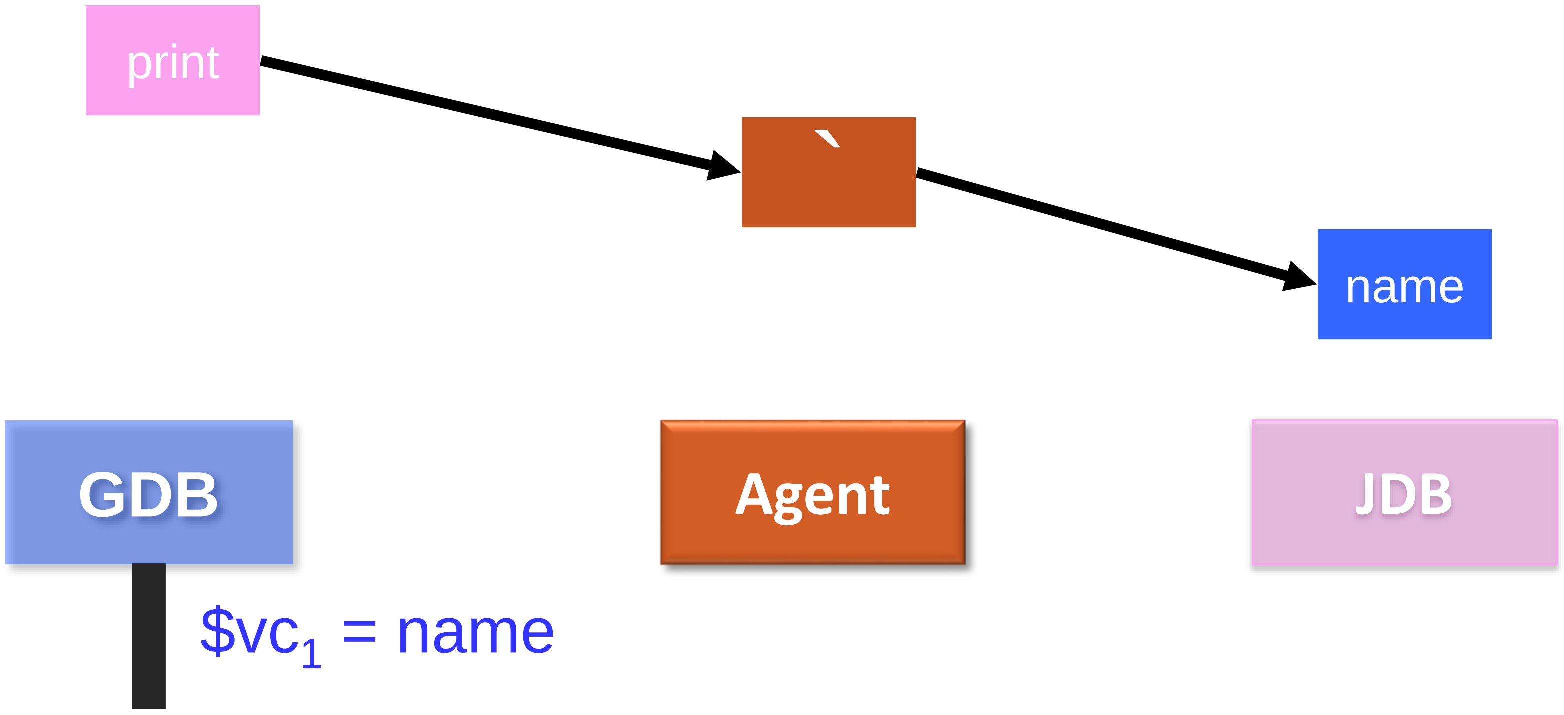


"smiley"
jstring name;
print *n e
The name is an opaque C pointer.
print `name
How can I evaluate the Jeannie expression?

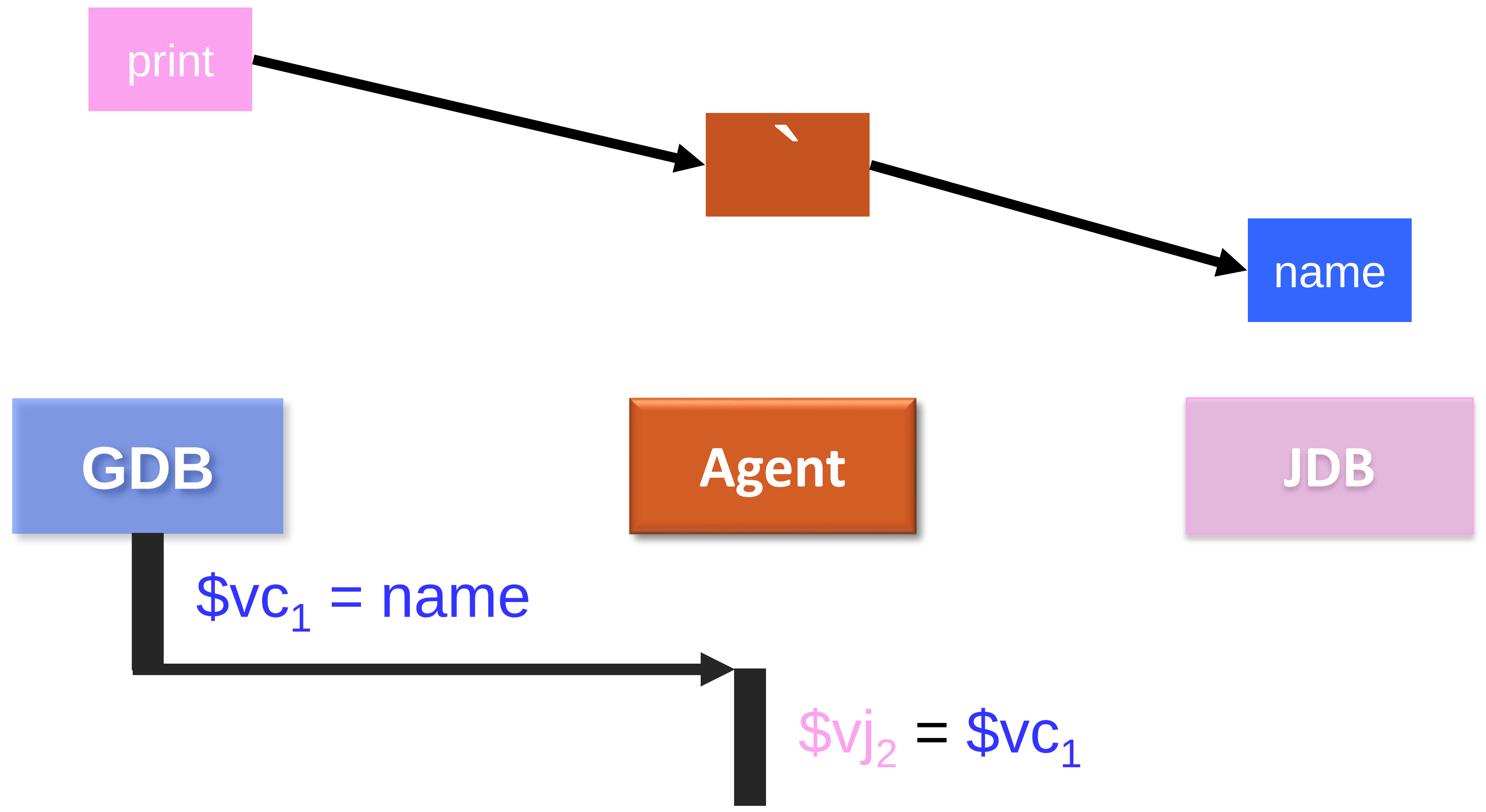
Build abstract syntax tree



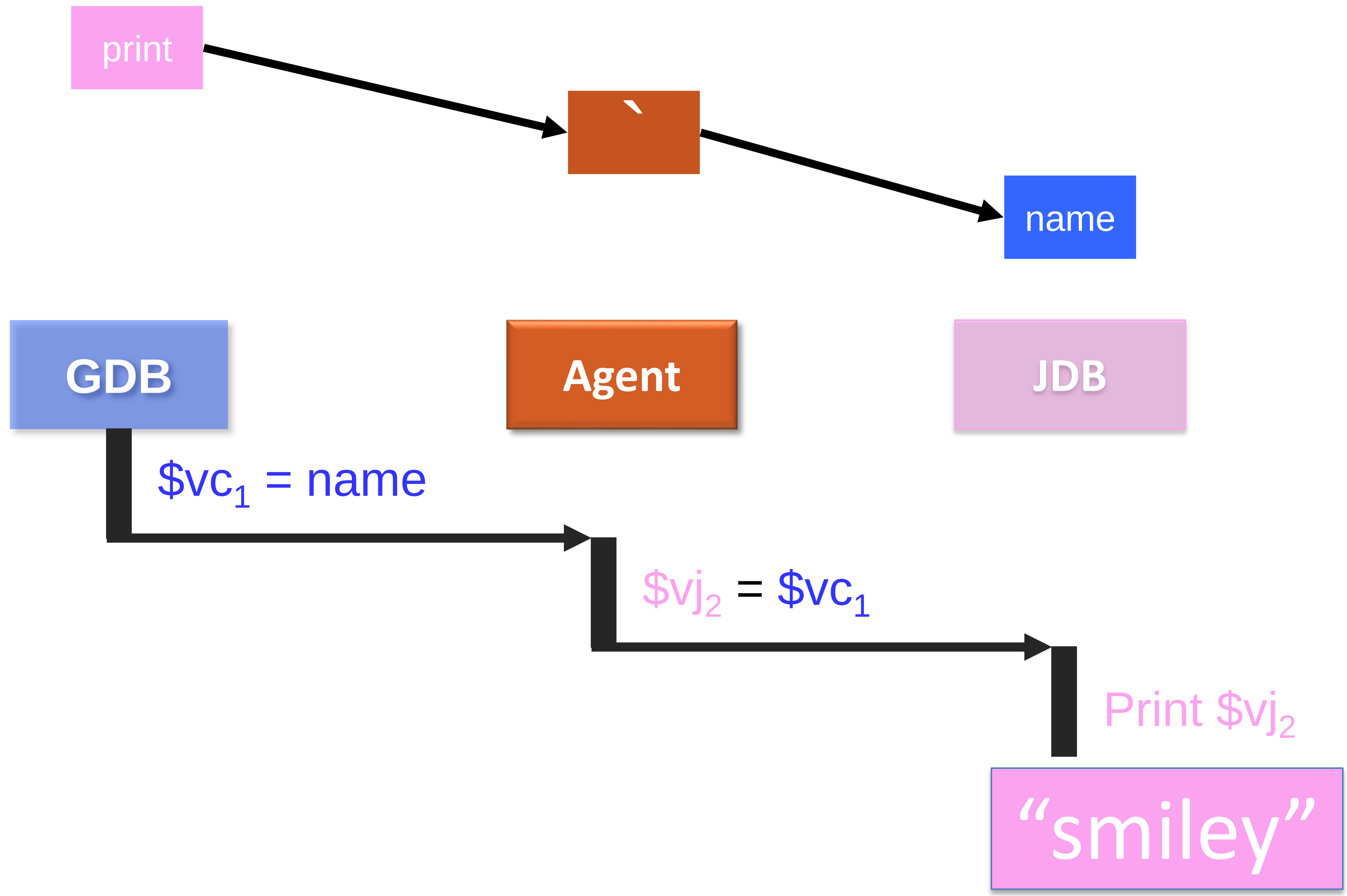
Evaluate AST in Bottom-up order.



Evaluate AST in Bottom-up order.

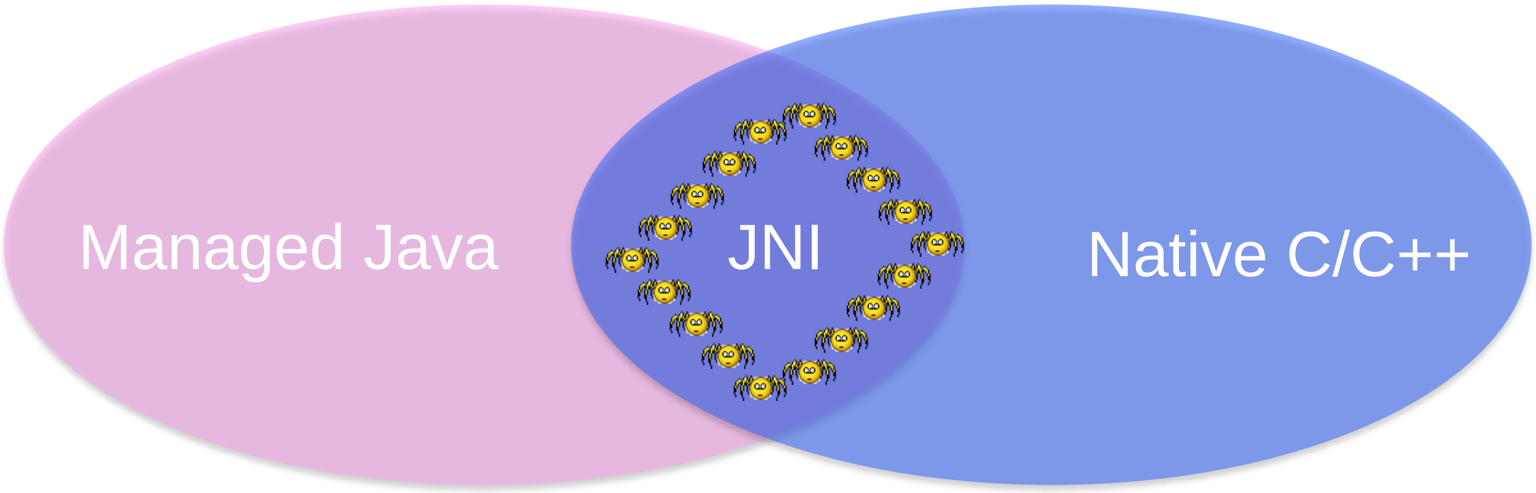


Evaluate AST in Bottom-up order.

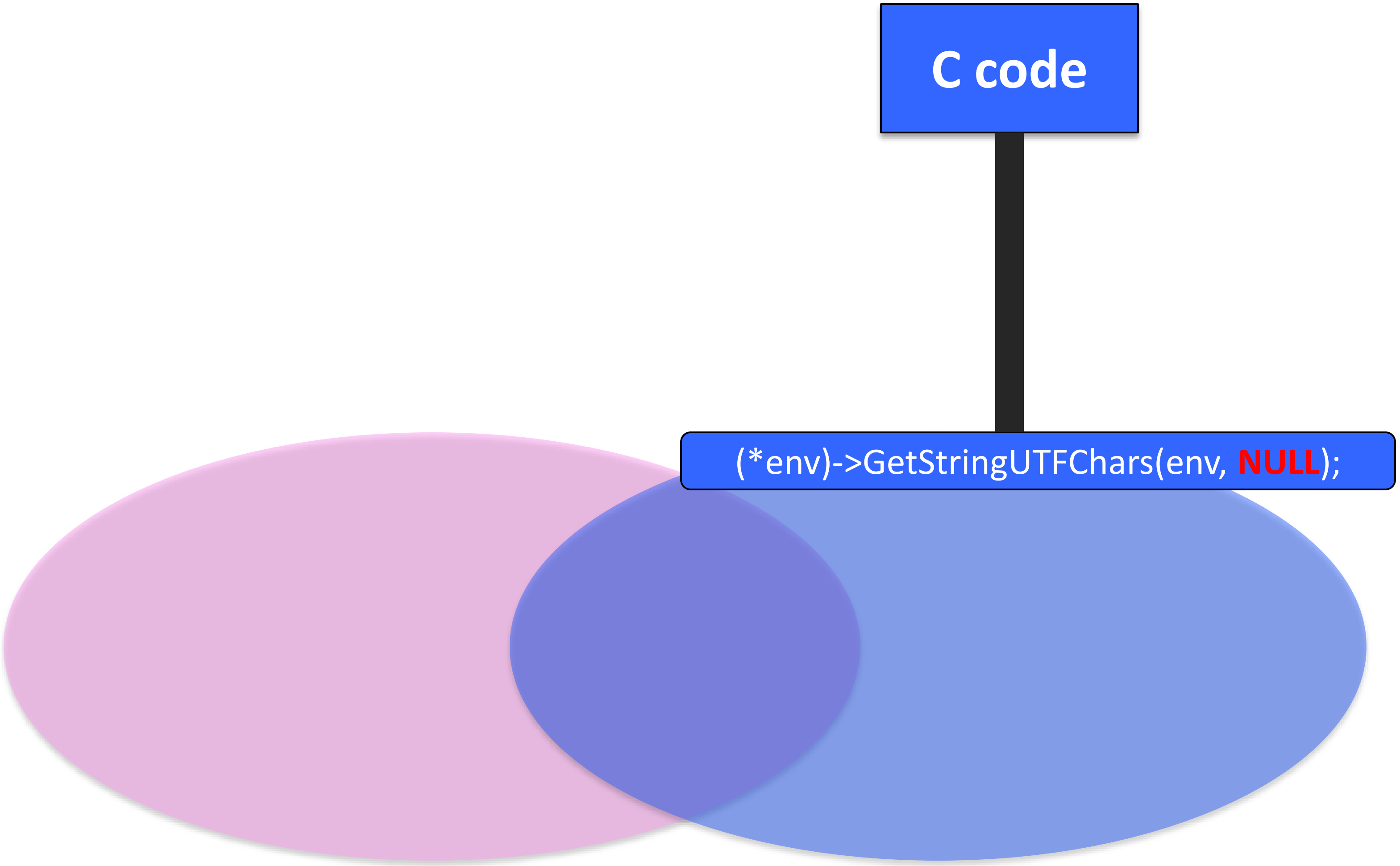


A. Evaluating Jeannie Expressions

B. Detecting FFI bugs



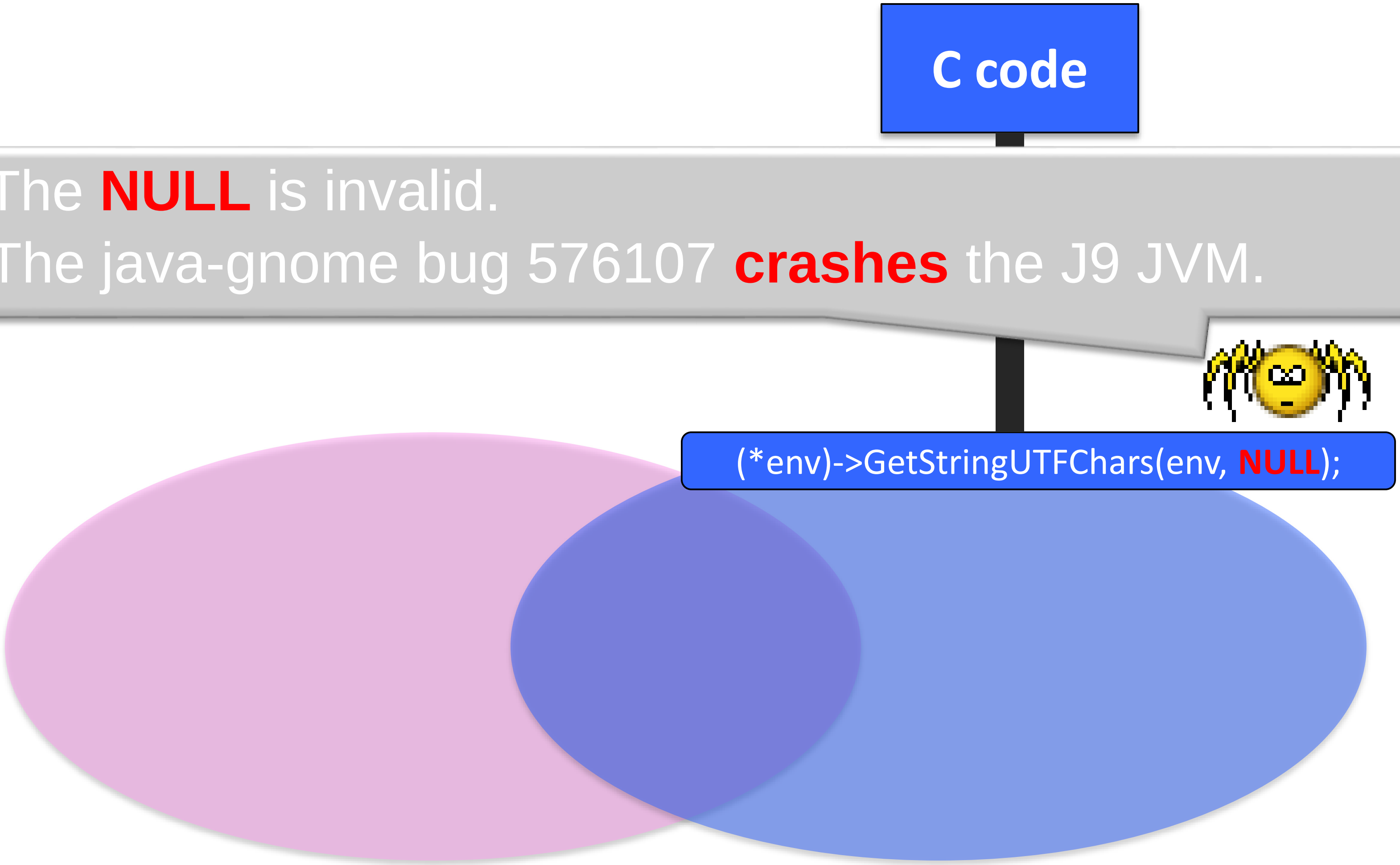
Problem: you may misuse foreign function interface.



```
(*env)->GetStringUTFChars(env, NULL);
```

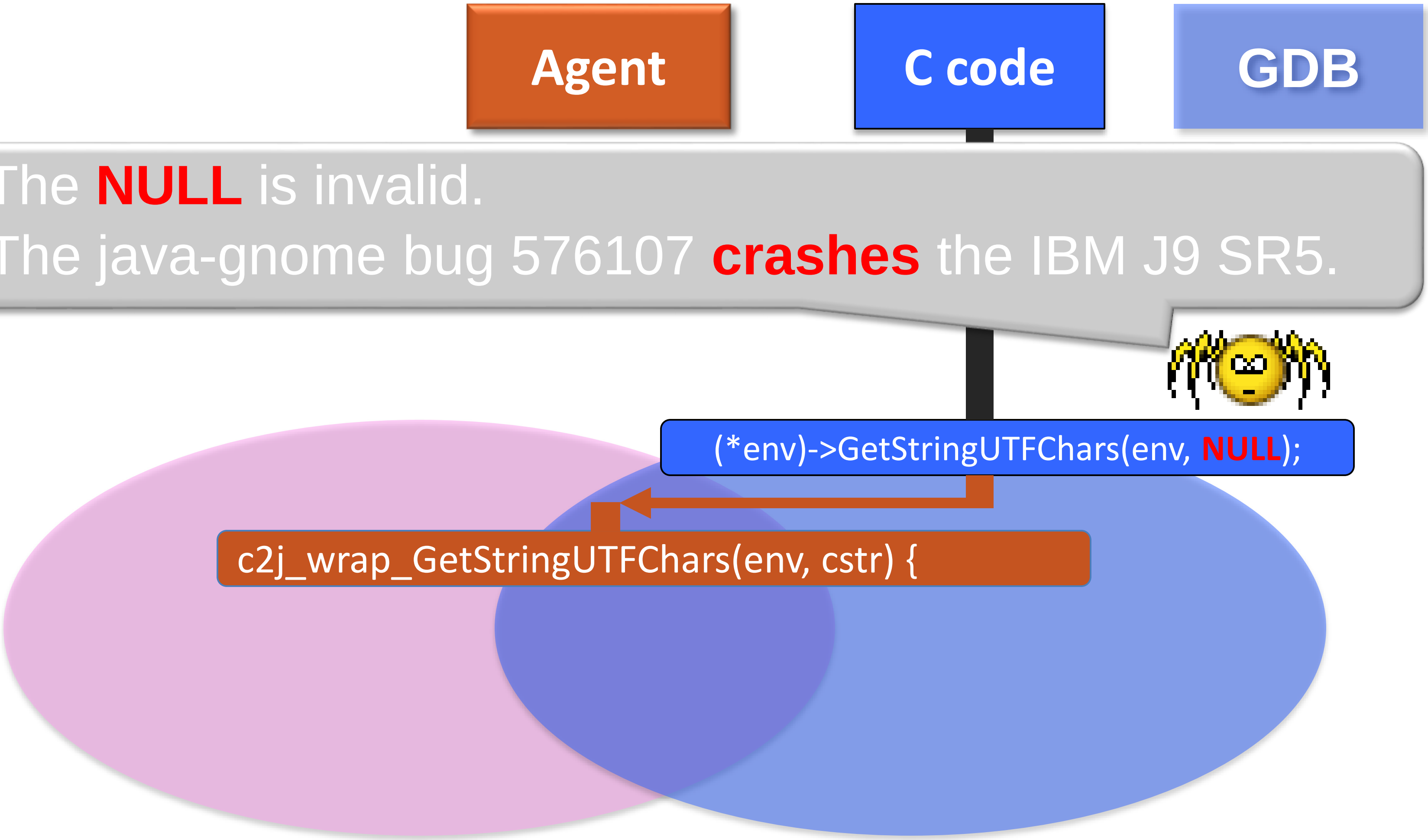
Problem: you may misuse foreign function interface.

The **NULL** is invalid.
The java-gnome bug 576107 **crashes** the J9 JVM.



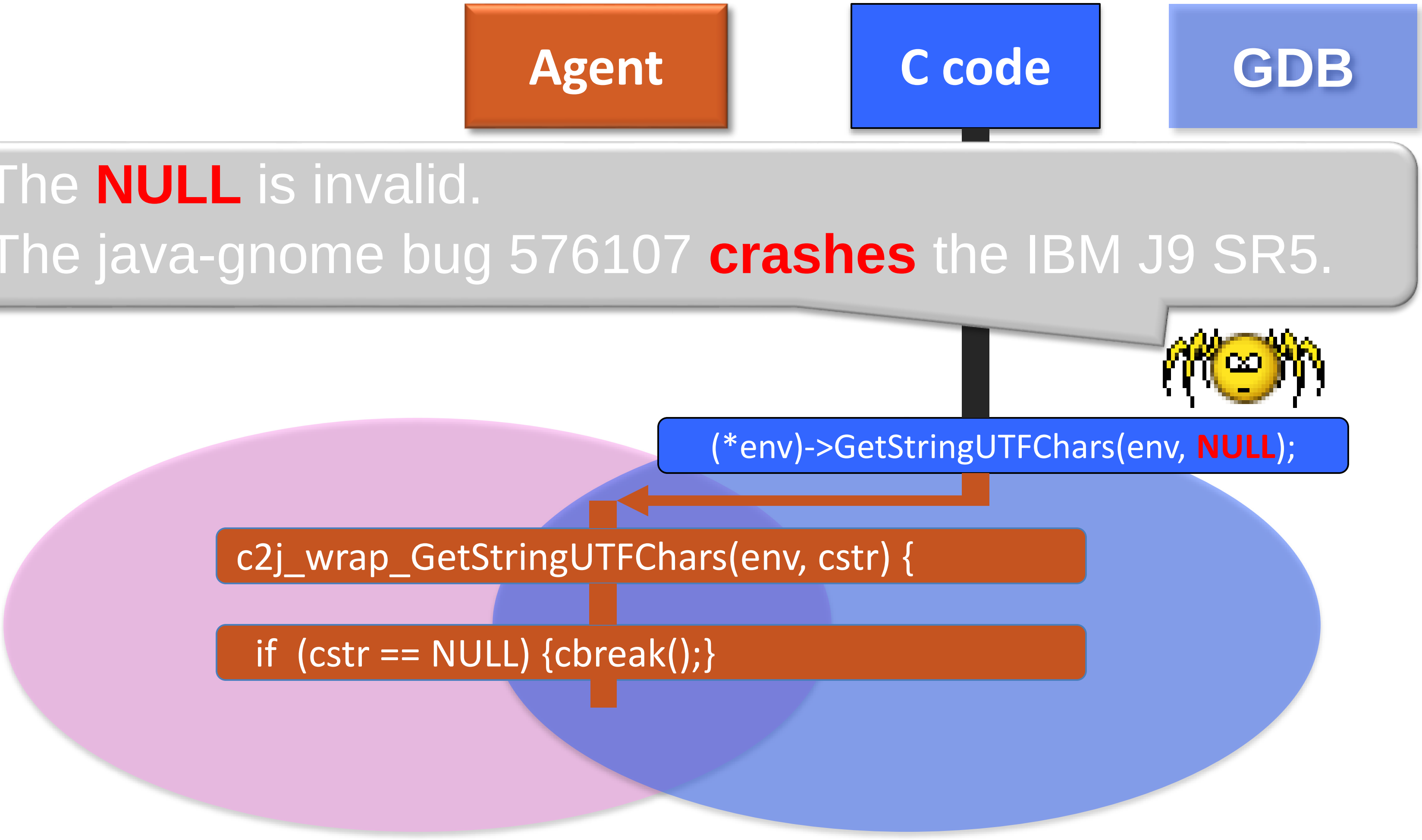
```
(*env)->GetStringUTFChars(env, NULL);
```

Our solution: detect FFI bugs


Agent
C code
GDB
(*env)->GetStringUTFChars(env, NULL);
c2j_wrap_GetStringUTFChars(env, cstr) {

The **NULL** is invalid.
The java-gnome bug 576107 **crashes** the IBM J9 SR5.

Our solution: detect FFI bugs.

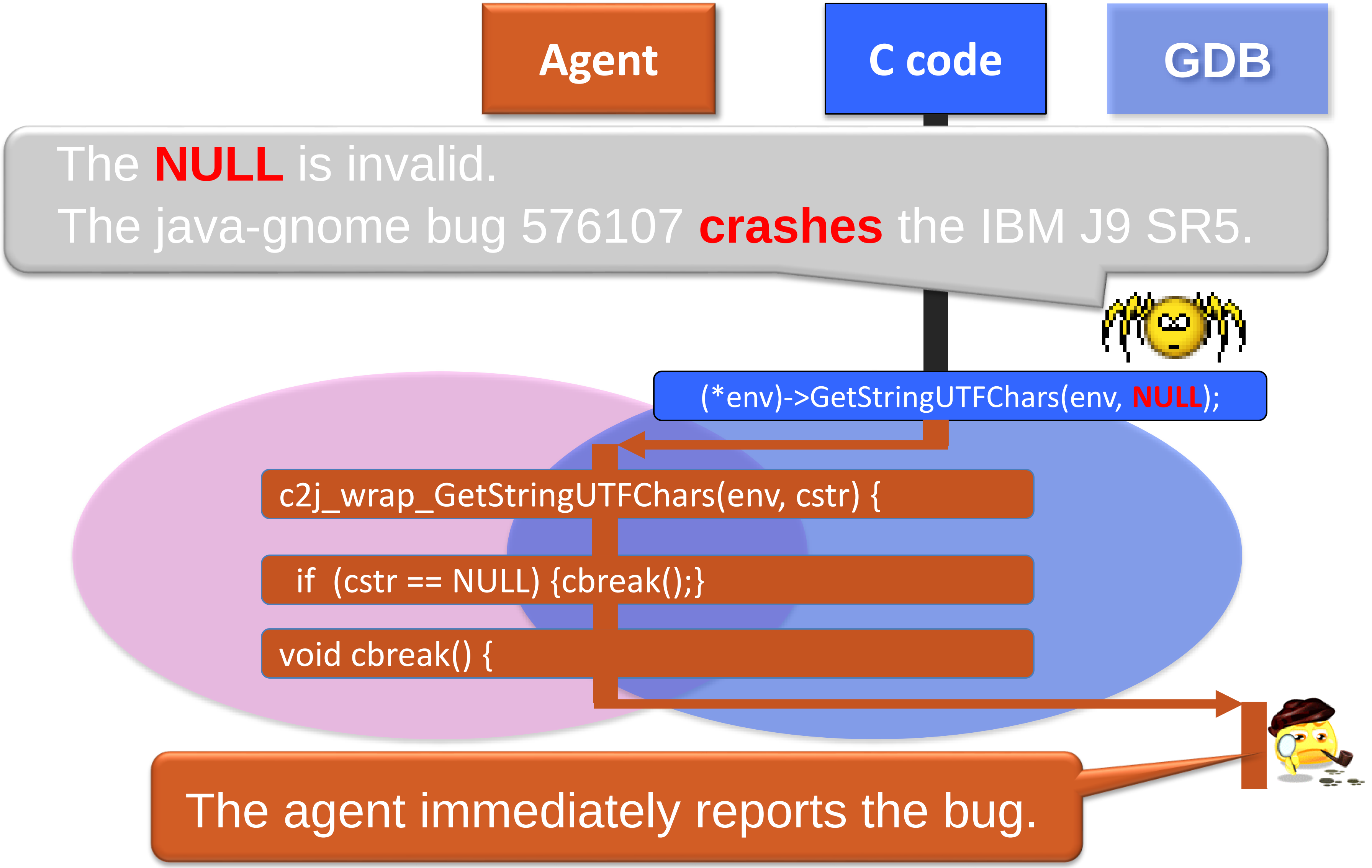


The **NULL** is invalid.

The java-gnome bug 576107 **crashes** the IBM J9 SR5.

```
(*env)->GetStringUTFChars(env, NULL);
c2j_wrap_GetStringUTFChars(env, cstr) {
  if  (cstr == NULL) {cbreak();}
```

Our solution: detect FFI bugs.



The NULL is invalid.
The java-gnome bug 576107 crashes the IBM J9 SR5.

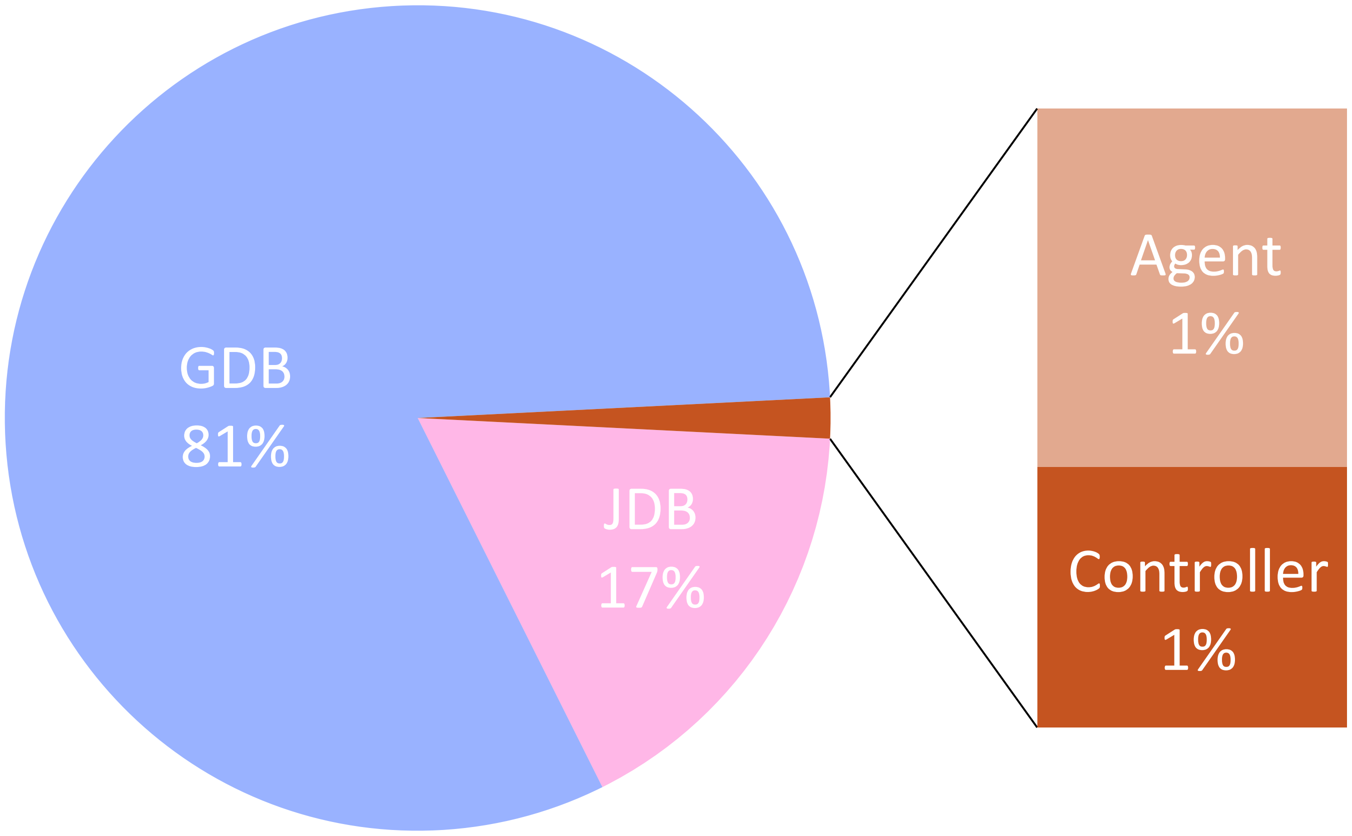
```
(*env)->GetStringUTFChars(env, NULL);
```

```
c2j_wrap_GetStringUTFChars(env, cstr) {
  if  (cstr == NULL) {cbreak();}
void cbreak() {
```

The agent immediately reports the bug.

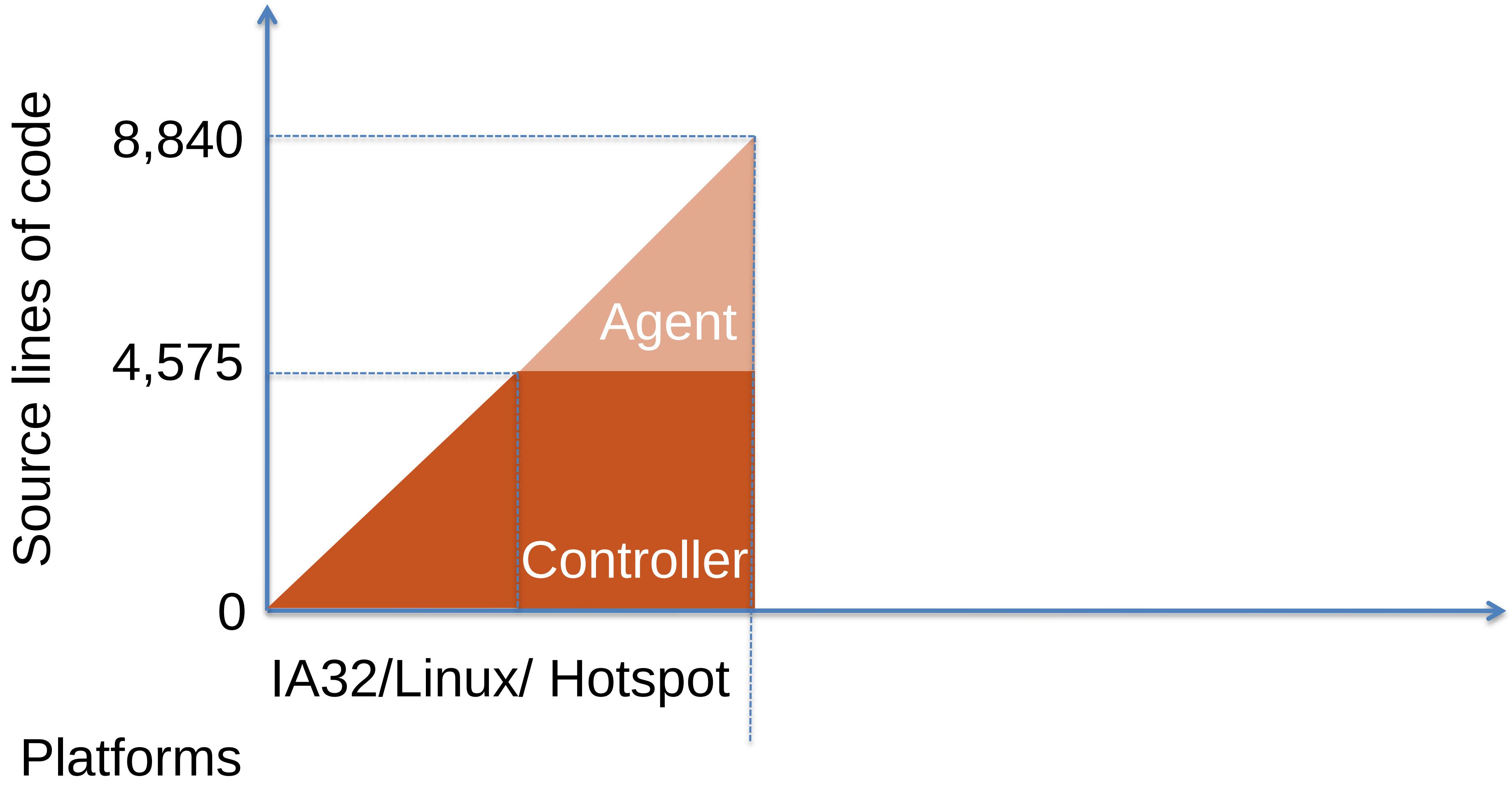
Outline

Composition is simple.

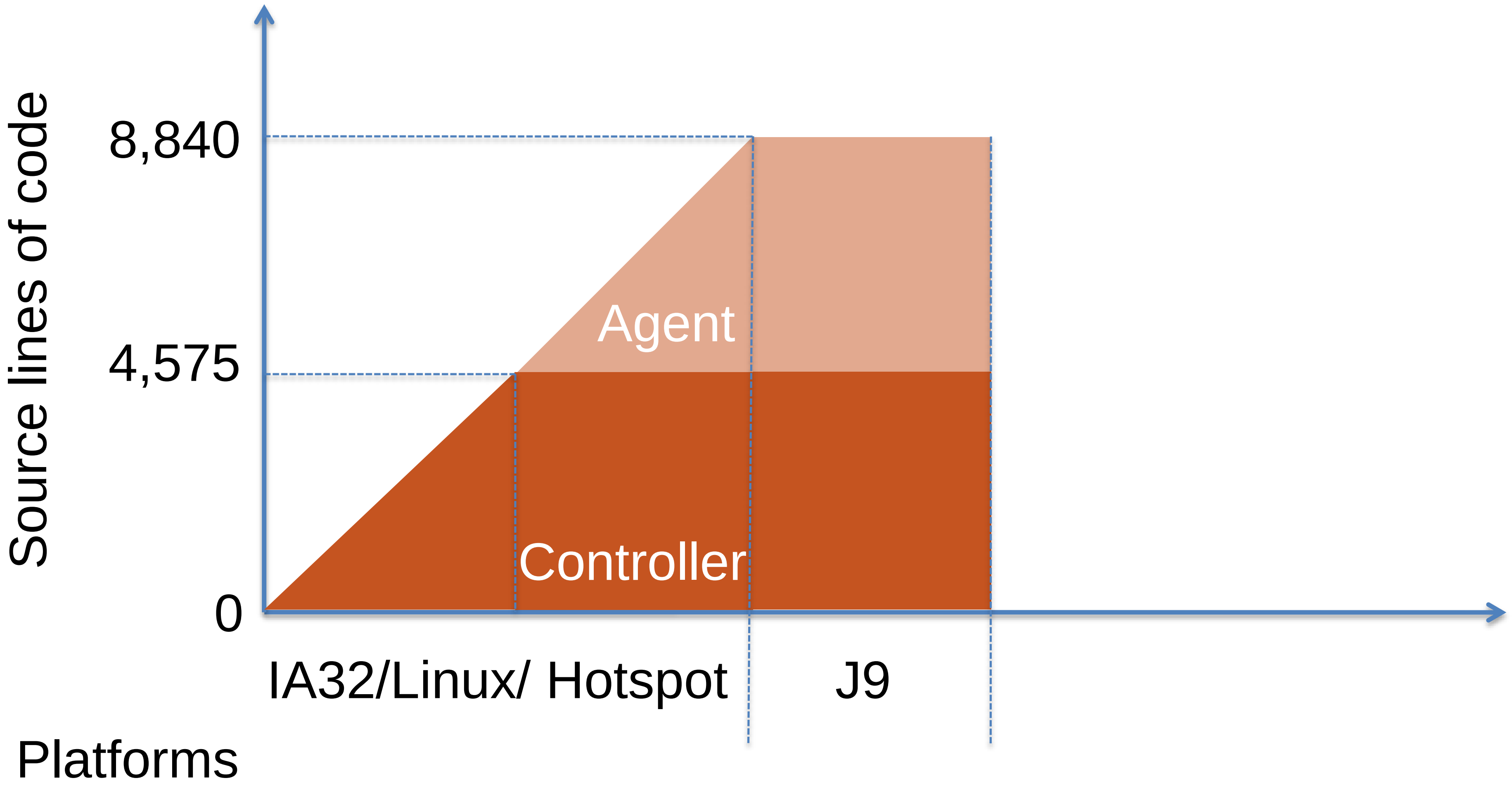


515K source lines of code in total

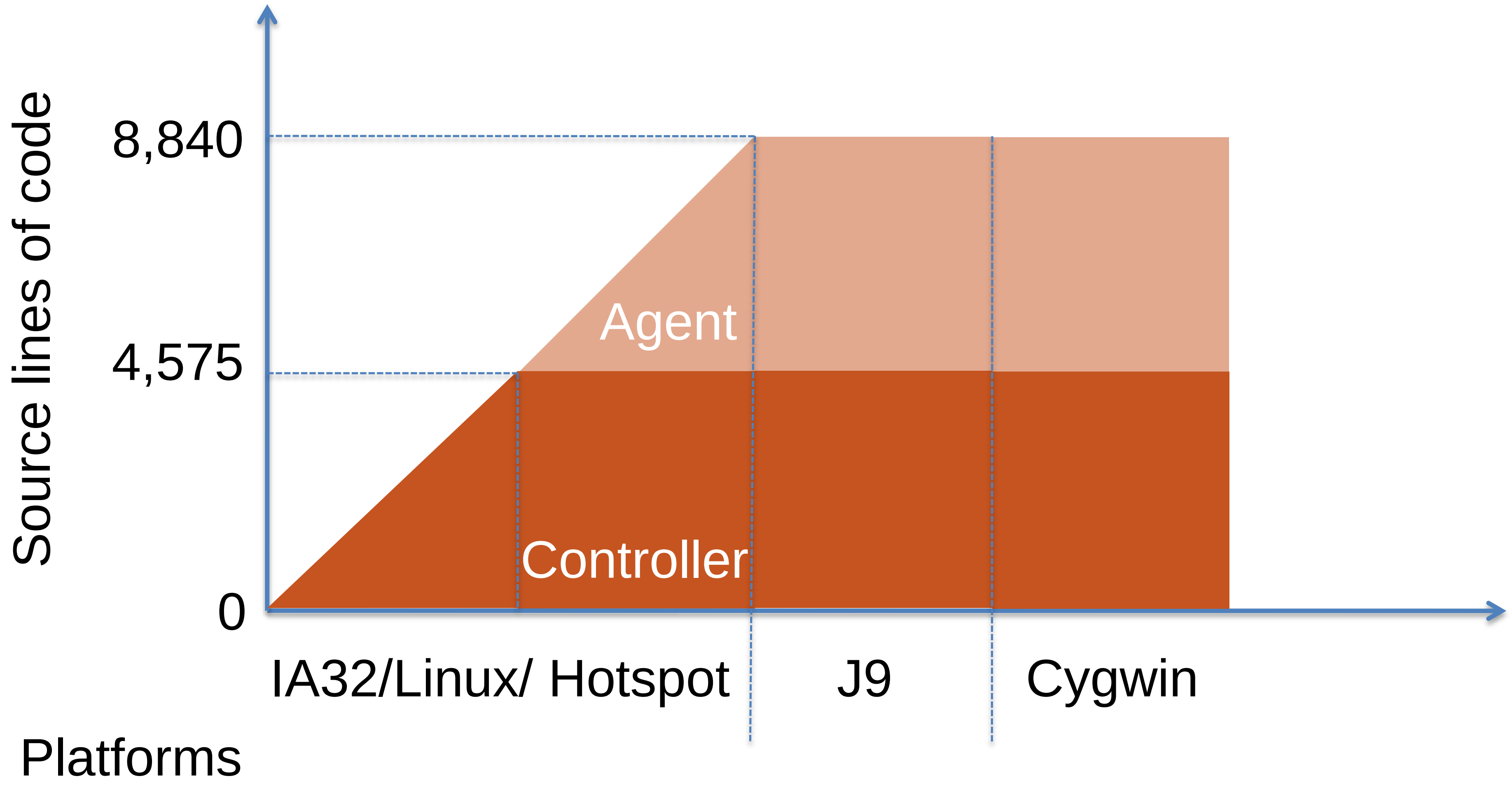
Composition is portable.



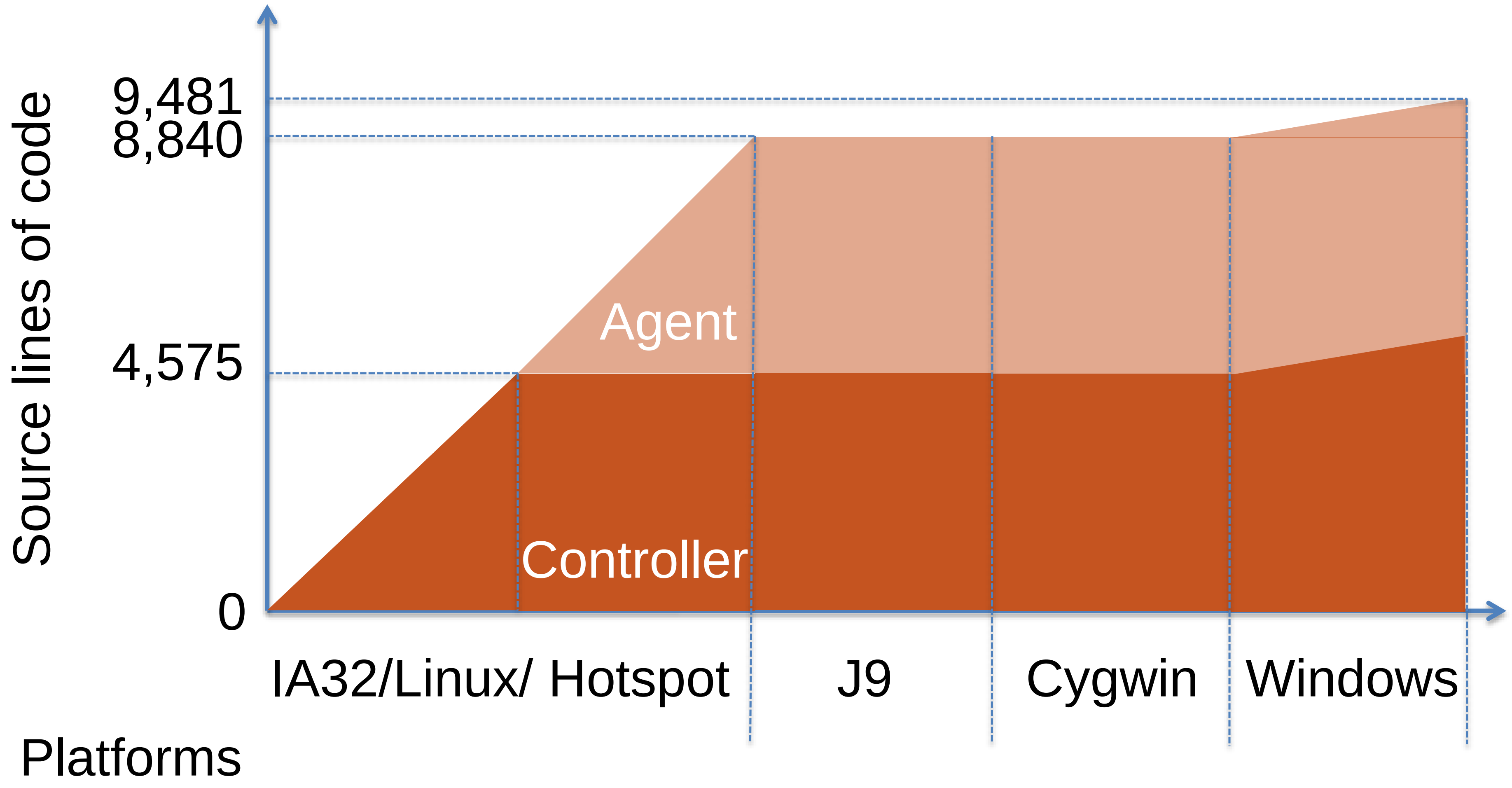
Composition is portable.



Composition is portable.



Composition is portable.



Composition is powerful.

The bug 576107 in java-gnome 4.0.10

	Hotspot VM 1.6.0_10	J9 VM SR5
Production run	running	crash

Composition is powerful.

The bug 576107 in java-gnome 4.0.10

	Hotspot VM 1.6.0_10	J9 VM SR5
Production run	running	crash
Runtime checking (-Xcheck:jni)	warning	warning

Composition is powerful.

The bug 576107 in java-gnome 4.0.10

	Hotspot VM 1.6.0_10	J9 VM SR5
Production run	running	crash
Runtime checking (-Xcheck:jni)	warning	warning
jdb	running	crash
gdb	running	fault

Composition is powerful.

The bug 576107 in java-gnome 4.0.10

	Hotspot VM 1.6.0_10	J9 VM SR5
Production run	running	crash
Runtime checking (-Xcheck:jni)	warning	warning
jdb	running	crash
gdb	running	fault
Blink	**breakpoint**	**breakpoint**

Related work

- Mixed-environment debuggers
  - Intel XDI for Harmony JVM
  - SUN dbx
  - Microsoft .NET debuggers
- Advance debugging features
  - Static analyses
    - BEAM [Kondoh & Onodera ’08]
    - J-Saffire [Furr & Foster ’06]
    - ILEA [Tan & Morrisett ‘07]
  - Language designs
    - Jeannie [Hirzel & Grimm ‘07]
    - SafeJNI [Tan et al. ‘06]
  - Wrapper generators
    - Automatic binding generator [Ravitch ‘09]
    - SWIG [Beazley ‘96]

Summary

- Portable mixed-environment debugging
- Composition with an intermediate agent
  1. Switching debugger context
  2. Interposing transitions
- Results
  1. Simple
  2. Portable
  3. Powerful

Thank you