The Checker Framework: pluggable static analysis for Java

http://CheckerFramework.org/

@CheckerFrmwrk on Twitter
CheckerFramework on Facebook & Google+

Werner Dietl
University of Waterloo
https://ece.uwaterloo.ca/~wdietl/

Joint work with Michael D. Ernst and many others.

Outline

- Pluggable type-checking
- Architecture overview
  - Java 8 syntax for Type Annotations
  - Dataflow Framework
  - Checker Framework
- Our experience
  - Case Studies
  - SPARTA: Android Security
  - VeriGames: Crowd-sourced Verification Games
- Writing your own type system

Java's type system is too weak

Type checking prevents many errors

```
int i = “hello”; // error
```

Type checking doesn't prevent enough errors

```
System.console().readLine();
Collections.emptyList().add(“one”);
dbStatement.executeQuery(userData);
```

Java's type system is too weak

Type checking prevents many errors

```
int i = “hello”; // error
```

Type checking c ors

NullPointerException

```
System.console().readLine();
Collections.emptyList().add(“one”);
dbStatement.executeQuery(userData);
```

Java's type system is too weak

Type checking prevents many errors

```
int i = “hello”; // error
```

Type checking doesn't prevent enough errors

```
System
Collections.emptyList().add(“one”);
dbStatement.executeQuery(userData);
```

UnsupportedOperationException

Java's type system is too weak

Type checking prevents many errors

```
int i = “hello”; // error
```

Type checking doesn't prevent enough errors

```
System.console().readLine();

Collections

dbStatement.executeQuery(userData);
```

SQL Injection Attacks!

Static types: not expressive enough

Null pointer exceptions

```
String op(Data in) {
  return “transform: ” + in.getF();
}
...
String s = op(null);
```

Many other properties can't be expressed

Prevent null pointer exceptions

Type system that statically guarantees that the program only dereferences known non-null references

Types of data

`@NonNull`	reference is never null
`@Nullable`	reference may be null

Null pointer exception

```
String op(Data in) {
  return “transform: ” + in.getF();
}
...
String s = op(null);
```

Where is the error?

Solution 1: Restrict use

```
String op(@NonNull Data in) {
  return “transform: ” + in.getF();
}
…
String s = op(null);      // error
```

Solution 2: Restrict implementation

```
String op(@Nullable Data in) {
  return “transform: ” + in.getF();
                        // error
}
…
String s = op(null);
```

Benefits of type systems

- **Find bugs** in programs
- Guarantee the **absence of errors**
- **Improve documentation**
- Improve code structure & maintainability
- Aid compilers, optimizers, and analysis tools
- Reduce number of run-time checks

- Possible negatives:
  - Must write the types (or use type inference)
  - False positives are possible (can be suppressed)

Extended type systems

- Null pointer exceptions (ICSE SEIP'11)
- Energy consumption (PLDI'11)
- Unwanted side effects (OOPSLA'04, '05, '12, ECOOP'08, ESEC/FSE’07)
- Unstructured heaps (ECOOP'07, '11, '12)
- Malformed input (FTfJP'12)
- UI actions not on event thread (ECOOP'13)
- Information leakage (CCS’14)

Extended type systems

- Null pointer exceptions (ICSE SEIP'11)
- Energy consumption (PLDI'11)
- Unwanted side effects (OOPSLA'04, '05, '12, ECOOP'08, ESEC/FSE'07)
- Unstructured heaps (ECOOP'07, '11, '12)
- Malformed input (FTfJP'12)
- UI actions not on event thread (ECOOP'13)
- Information leakage (CCS'14)

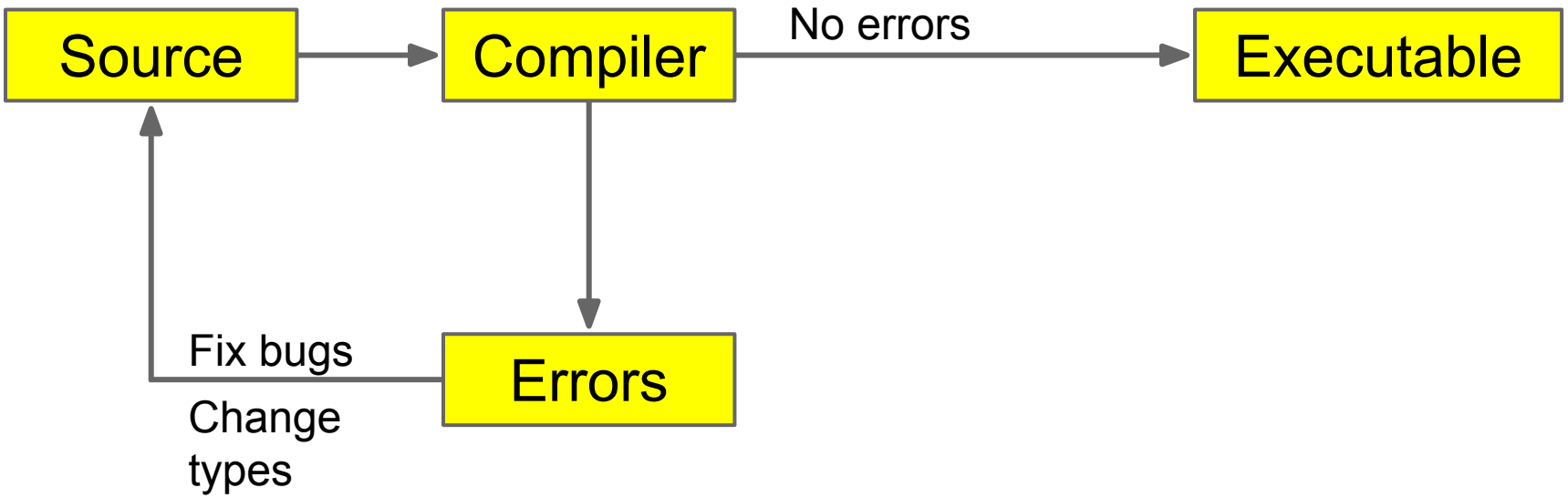
The Checker Framework

A framework for pluggable type checkers
“Plugs” into the OpenJDK compiler

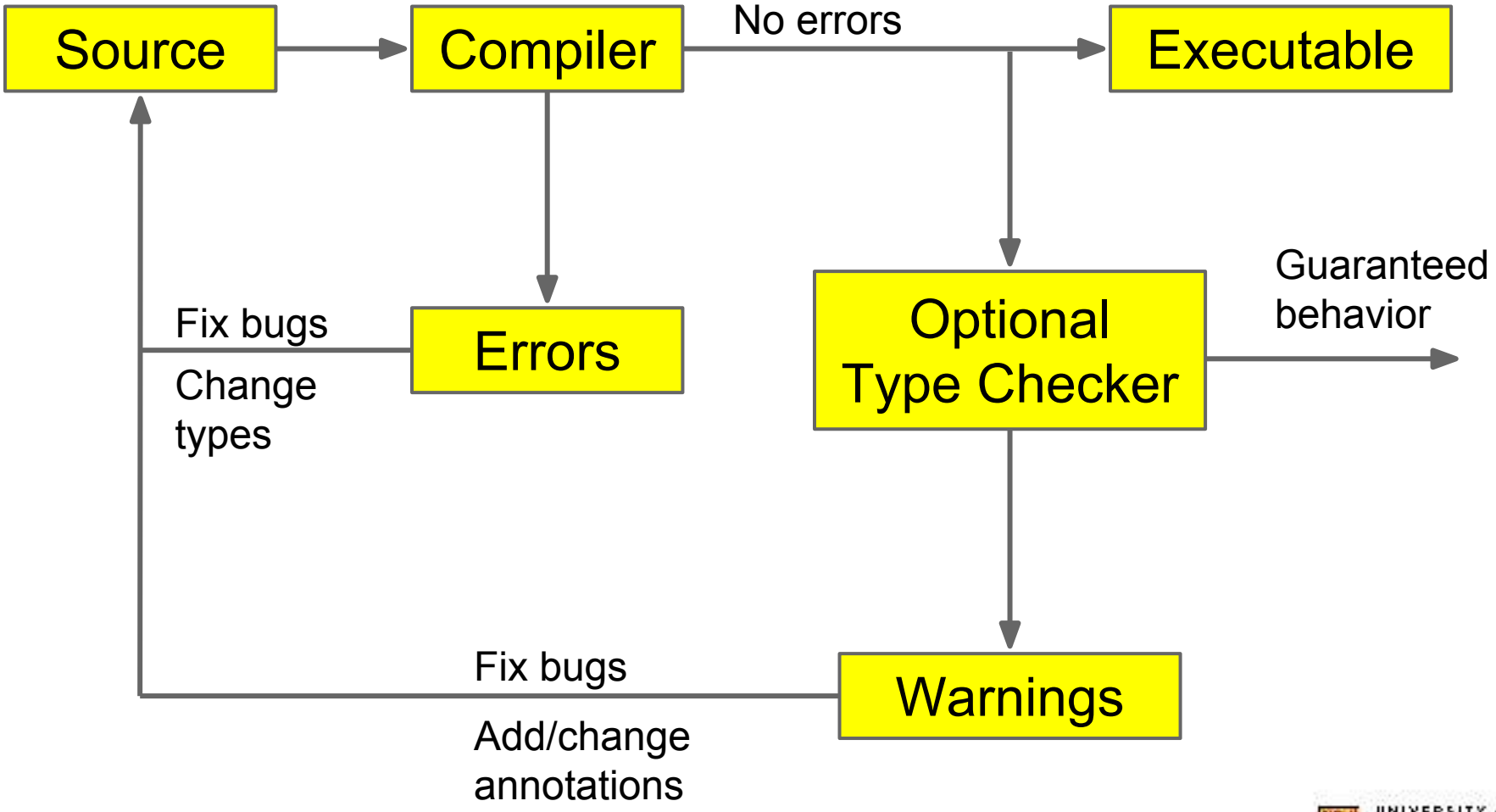
```
javac -processor MyChecker ...
```

Eclipse plug-in, Ant and Maven integration

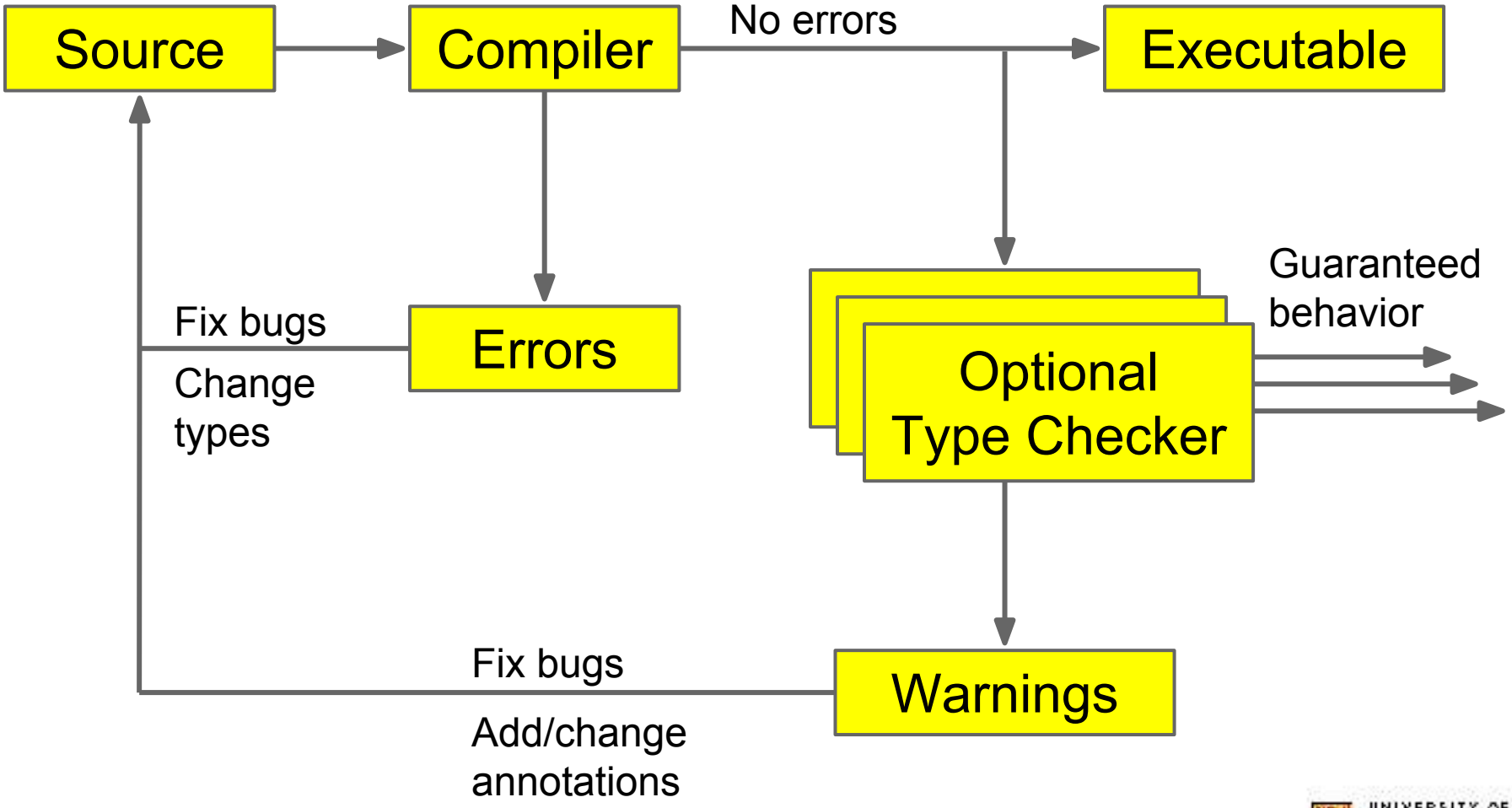
Type Checking



Optional Type Checking



Optional Type Checking



What bugs can you detect & prevent?

The property you care about:	The annotation you write:
Null dereferences	`@NonNull`
Mutation and side-effects	`@Immutable`
Concurrency: locking	`@GuardedBy`
Security: encryption,	`@Encrypted`
tainting	`@Untainted`
Aliasing	`@Linear`
Equality tests	`@Interned`
Strings: localization,	`@Localized`
Regular expression syntax	`@Regex`
Signature format	`@FullyQualified`
Enumerations	`@Fenum`
Typestate (e.g., open/closed files)	`@State`

Users can write their own checkers!

UNIVERSITY OF WATERLOO

Checker Framework experience

Type checkers reveal important latent defects

Ran on >3 million LOC of real-world open-source code as of early 2011

Found hundreds of user-visible failures

Annotation overhead is low

Mean 2.6 annotations per kLOC

    20 annotations per kLOC for Nullness Checker

[Dietl et al. ICSE'11]

Formalizations

$$
\begin{array}{rcll}
P & \in & \text{Program} & ::= \overline{\text{Class}},\ \text{ClassId},\ \text{Expr} \\
\text{Cls} & \in & \text{Class} & ::= \text{class ClassId}\langle\overline{\text{TVarI}}\rangle \\
 & & & \text{extends ClassId}\langle{}^{\text{s}}\text{Ty}\rangle \\
 & & & \{\ \overline{\text{FieldId } {}^{\text{s}}\text{Type};}\ \text{Met} \\
{}^{\text{s}}\text{T} & \in & {}^{\text{s}}\text{Type} & ::= {}^{\text{s}}\text{NType} \mid \text{TVarId} \\
{}^{\text{s}}\text{N} & \in & {}^{\text{s}}\text{NType} & ::= \text{OM ClassId}\langle\overline{{}^{\text{s}}\text{Type}}\rangle \\
\text{u} & \in & \text{OM} & ::= \\
\text{mt} & \in & \text{Meth} & ::= \\
 & & \text{MethSig} & ::= \\
\text{w} & \in & \text{Purity} & ::= \\
\text{e} & \in & \text{Expr} & ::= \\
 & & & \text{Expr.MethId}\langle{}^{\text{s}}\text{Type}\rangle(\text{Expr}) \mid \\
 & & & \text{new } {}^{\text{s}}\text{Type} \mid ({}^{\text{s}}\text{Type})\ \text{Expr} \\
{}^{\text{s}}\Gamma & \in & {}^{\text{s}}\text{Env} & ::= \overline{\text{TVarId } {}^{\text{s}}\text{NType}};\ \overline{\text{ParId } {}^{\text{s}}\text{Type}}
\end{array}
$$

$$
\begin{array}{rcll}
\text{h} & \in & \text{Heap} & = \text{Addr} \to \text{Obj} \\
\iota & \in & \text{Addr} & = \text{Set of Addresses} \cup \{\text{null}_a\} \\
\text{o} & \in & \text{Obj} & = {}^{\text{r}}\text{Type},\ \text{Fields} \\
{}^{\text{r}}\text{T} & \in & {}^{\text{r}}\text{Type} & = \text{OwnerAddr ClassId}\langle\overline{{}^{\text{r}}\text{Type}}\rangle \\
\text{Fs} & \in & \text{Fields} & = \text{FieldId} \to \text{Addr} \\
\iota & \in & \text{OwnerAddr} & = \text{Addr} \cup \{\text{any}_a\} \\
{}^{\text{r}}\Gamma & \in & {}^{\text{r}}\text{Env} & = \overline{\text{TVarId } {}^{\text{r}}\text{Type}};\ \overline{\text{ParId Addr}}
\end{array}
$$

$$\text{OS-Upd}\ \frac{\begin{array}{c}\text{h}, {}^{\text{r}}\Gamma, \text{e}_0 \rightsquigarrow \text{h}_0, \iota_0 \\ \iota_0 \neq \text{null}_a \\ \text{h}_0, {}^{\text{r}}\Gamma, \text{e}_2 \rightsquigarrow \text{h}_2, \iota \\ \text{h}' = \text{h}_2[\iota_0.\text{f} := \iota]\end{array}}{\text{h}, {}^{\text{r}}\Gamma, \text{e}_0.\text{f} = \text{e}_2 \rightsquigarrow \text{h}',}$$

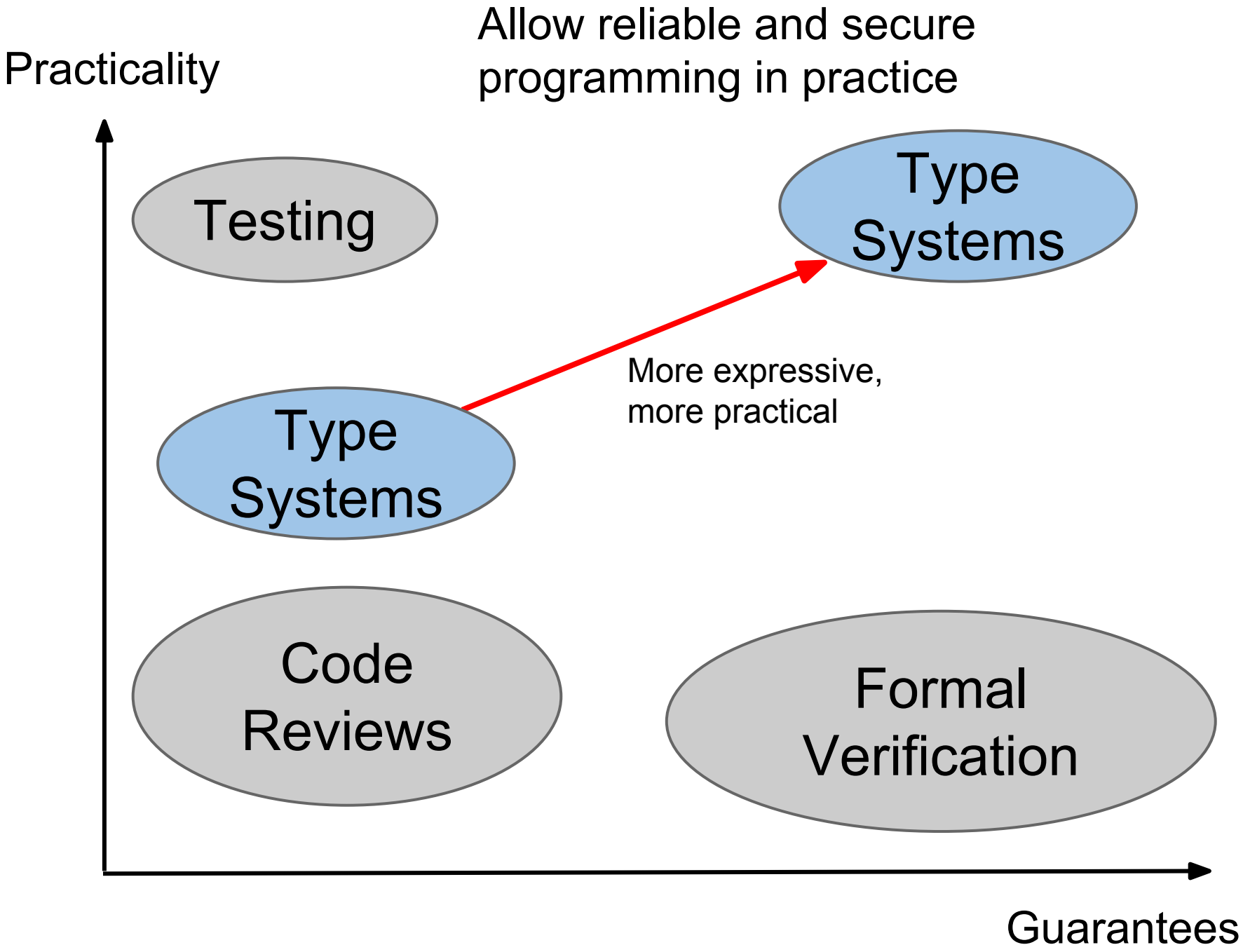
$$\text{OS-Read}\ \frac{\begin{array}{c}\text{h}, {}^{\text{r}}\Gamma, \text{e}_0 \rightsquigarrow \text{h}', \iota_0 \\ \iota_0 \neq \text{null}_a \\ \iota = \text{h}'(\iota_0)\downarrow_2(\text{f})\end{array}}{\text{h}, {}^{\text{r}}\Gamma, \text{e}_0.\text{f} \rightsquigarrow \text{h}', \iota}$$

$$\text{GT-Read}\ \frac{\Gamma \vdash \text{e}_0 : \text{N}_0 \qquad \text{N}_0 = \_\langle}{\Gamma \vdash \text{e}_0.\text{f} : \text{N}_0 \rhd fType(\text{C}_0, \text{f})}$$

$$\text{GT-Upd}\ \frac{\begin{array}{c}\Gamma \vdash \text{e}_0 : \text{N}_0 \qquad \text{N}_0 = \text{u}_0\ \text{C}_0\langle\_\rangle \\ \text{T}_1 = fType(\text{C}_0, \text{f}) \\ \Gamma \vdash \text{e}_2 : \text{N}_0 \rhd \text{T}_1 \\ \text{u}_0 \neq \text{any} \qquad rp(\text{u}_0, \text{T}_1)\end{array}}{\Gamma \vdash \text{e}_0.\text{f} = \text{e}_2 : \text{N}_0 \rhd \text{T}_1}$$

$$\text{DYN}\ \frac{\begin{array}{l}\text{h} \vdash {}^{\text{r}}\Gamma : {}^{\text{s}}\Gamma \\ \text{h} \vdash \iota_1 : dyn({}^{\text{s}}\text{N}, \text{h}, {}^{\text{r}}\iota_1) \\ \text{h} \vdash \iota_2 : dyn({}^{\text{s}}\text{T}, \iota_1, \text{h}(\iota_1)\downarrow_1) \\ {}^{\text{s}}\text{N} = \text{u}_N\ \text{C}_N\langle\_\rangle \\ \text{u}_N = \text{this}_u \Rightarrow {}^{\text{r}}\Gamma(\text{this}) \\ free({}^{\text{s}}\text{T}) \subseteq dom(\text{C}_N)\end{array}}{\begin{array}{l}\Big\} \implies \text{h} \vdash \iota_2 : dyn({}^{\text{s}}\text{N} \rhd {}^{\text{s}}\text{T}, \text{h}, {}^{\text{r}}\Gamma) \\ {}^{\text{r}}\text{T} = \iota'\ \_\langle\_\rangle \qquad \iota \vdash {}^{\text{r}}\text{T}\ {}^{\text{r}}\!<: \iota'\ \text{C}\langle\overline{{}^{\text{r}}\text{T}}\rangle \qquad \iota \vdash {}^{\text{r}}\text{T}\ {}^{\text{r}}\!<: \iota'\ \text{C}\langle\overline{{}^{\text{r}}\text{T}_a}\rangle \Rightarrow \iota \vdash \overline{{}^{\text{r}}\text{T}}\ {}^{\text{r}}\!<: \overline{{}^{\text{r}}\text{T}_a} \\ \text{dom}(\text{C}) = \overline{\text{X}} \qquad free({}^{\text{s}}\text{T}) \subseteq \overline{\text{X}} \circ \overline{\text{X}'}\end{array}}{dyn({}^{\text{s}}\text{T}, \iota, {}^{\text{r}}\text{T}, (\overline{\text{X}'\ {}^{\text{r}}\text{T}'}; \_)) = {}^{\text{s}}\text{T}[\iota'/\text{this}, \iota'/\text{peer}, \iota/\text{rep}, \text{any}_a/\text{any}_u, \overline{{}^{\text{r}}\text{T}/\text{X}}, \overline{{}^{\text{r}}\text{T}'/\text{X}'}]}$$

UNIVERSITY OF WATERLOO


Allow reliable and secure programming in practice
Practicality
Testing
Type Systems
More expressive, more practical
Type Systems
Code Reviews
Formal Verification
Guarantees

Outline

- Pluggable type-checking
- Architecture overview
  - Java 8 syntax for Type Annotations
  - Dataflow Framework
  - Checker Framework
- Our experience
  - Case Studies
  - SPARTA: Android Security
  - VeriGames: Crowd-sourced Verification Games
- Writing your own type system

Project overview

Java 8 type annotation feature

jsr308-langtools extensions

annotation-tools

checker-framework

- javacutils
- dataflow
- stubparser
- framework
- checker: 18 default type systems

checker-framework-inference

Project overview

Java 8 type annotation feature

jsr308-langtools extensions

annotation-tools

checker-framework

- javacutils
- **dataflow**
- stubparser
- **framework**
- checker: 18 default type systems

checker-framework-inference

Java 8 extends annotation syntax

Annotations on all occurrences of types:

```
@Untainted String query;
List<@NonNull String> strings;
myGraph = (@Immutable Graph) tmp;
class UnmodifiableList<T>
  implements @Readonly List<T> {}
```

Stored in classfile

Handled by javac, javap, javadoc, …

Explicit method receivers

```
class MyClass {
  int foo(@TParam String p) {…}
  int foo(@TRecv MyClass this,
          @TParam String p) {…}
```

No impact on method binding and overloading

Constructor return & receiver types

Every constructor has a return type

```
class MyClass {
  @TReturn MyClass(@TParam String p) {...}
```

Inner class constructors also have a receiver

```
class Outer {
  class Inner {
    @TReturn Inner(@TRecv Outer Outer.this,
                   @TParam String p) {...}
```

Array annotations

A read-only array of non-empty arrays of English strings:

```
@English String @ReadOnly [] @NonEmpty [] a;
```

CF has invariant arrays:

- `@NonNull String[]` is unrelated to `@Nullable String[]`

Compare to Java's unsound covariant arrays:

- `String[]` is a subtype of `Object[]`

CF: Java 6 & 7 Compatibility

Annotations in comments

```
List</*@NonNull*/ String> strings;
```

Voodoo comments for arbitrary source code

```
/*>>> import myquals.TRecv; */
...
  int foo(/*>>> @TRecv MyClass this,*/
          @TParam String p) {…}
```

CF: Annotating external libraries stub files and annotated JDK

Allows annotating external libraries

- As separate text file (stub file)

  checker-framework/checker/src/…/jdk.astub

- Within its .jar file (from annotated partial source code)

  checker-framework/checker/jdk/...

Dataflow Framework

Initially for flow-sensitive type refinement

Now a project of its own right

1. Translate AST to CFG
   Standard multipass visitor over AST
2. Perform dataflow analysis over CFG with user-provided
   a. Abstract value — What are we tracking?
   b. Transfer functions — What do operations do?
   c. Store — What are intermediate results?
3. Allow queries about result

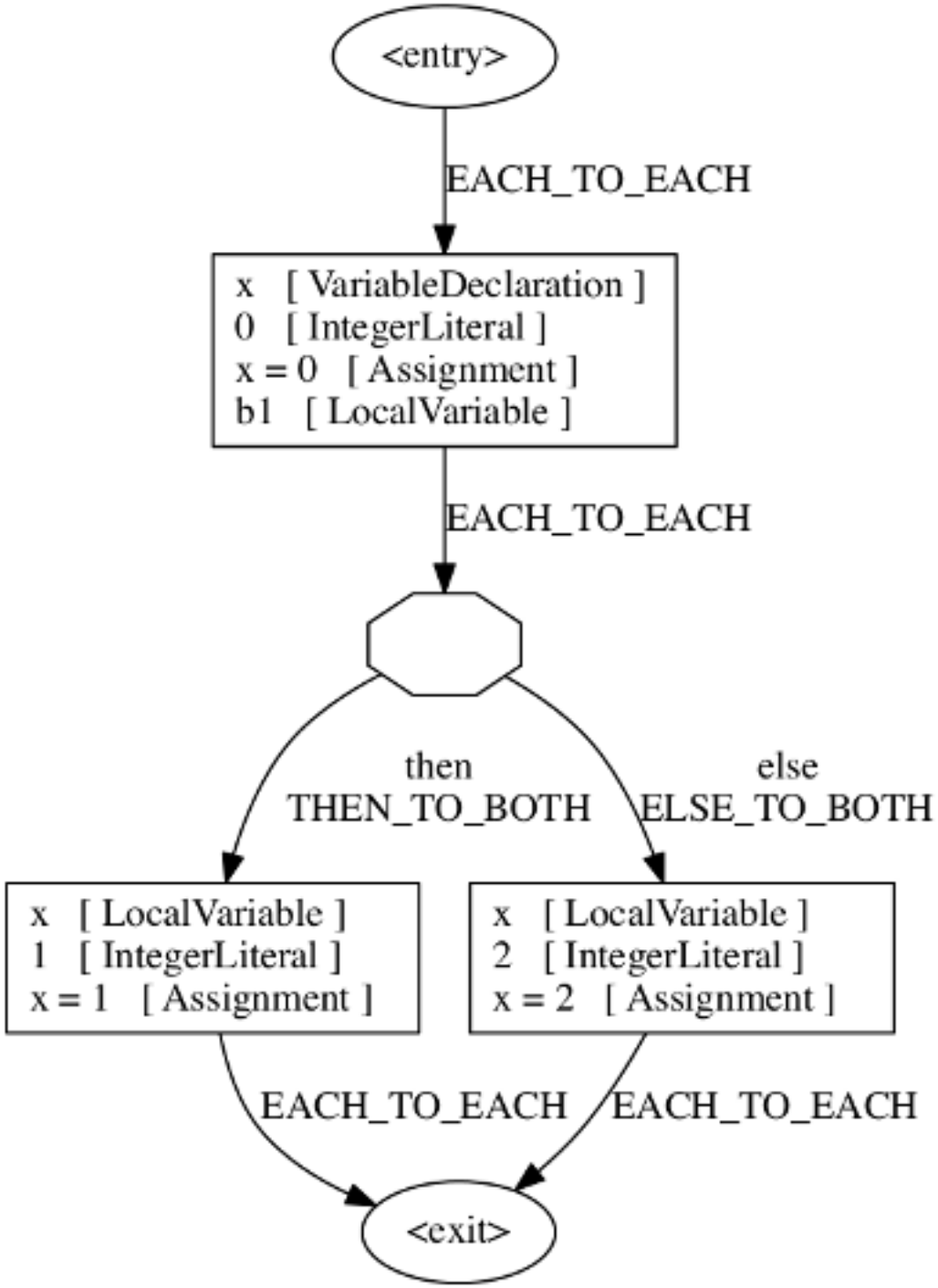
Control Flow Graph

CFG is a graph of basic blocks

- Conditional basic blocks to model conditional control flow
- Exceptional edges

Use type Node for all Java operations and expressions, e.g.,

- StringLiteralNode, FieldAccessNode, etc.
- Make up the content of basic blocks



Properties of the CFG

- Explicit representation of implicit Java constructs
  - Unboxing, implicit type conversions, etc.
  - Analyses do not need to worry about these things
  - All control flow explicitly modeled (e.g. exceptions on field access)
- High-level constructs
  - Close to source language
- Different from other approaches
  - Not three-address-form
  - Analysis is not performed over the AST

Checker Framework - “Framework”

- Full type systems: inheritance, overriding, ...
- Generics (type polymorphism)
  - Also qualifier polymorphism
- Flow-sensitive type qualifier inference
  - Infers types for local variables
  - reusable component
- Qualifier defaults
- Pre-/Post-conditions
- Warning suppression
- Testing infrastructure

Outline

- Pluggable type-checking
- Architecture overview
  - Java 8 syntax for Type Annotations
  - Dataflow Framework
  - Checker Framework
- Our experience
  - Case Studies
  - SPARTA: Android Security
  - VeriGames: Crowd-sourced Verification Games
- Writing your own type system

Evaluations

- Checkers reveal important latent bugs
  - Ran on >3 million LOC of real-world code
  - Found 40 user-visible bugs, hundreds of mistakes
- Annotation overhead is low
  - Mean 2.6 annotations per kLOC
- Learning their usage is easy
  - Used successfully by first-year CS majors
- Building checkers is easy
  - New users developed 3 new realistic checkers

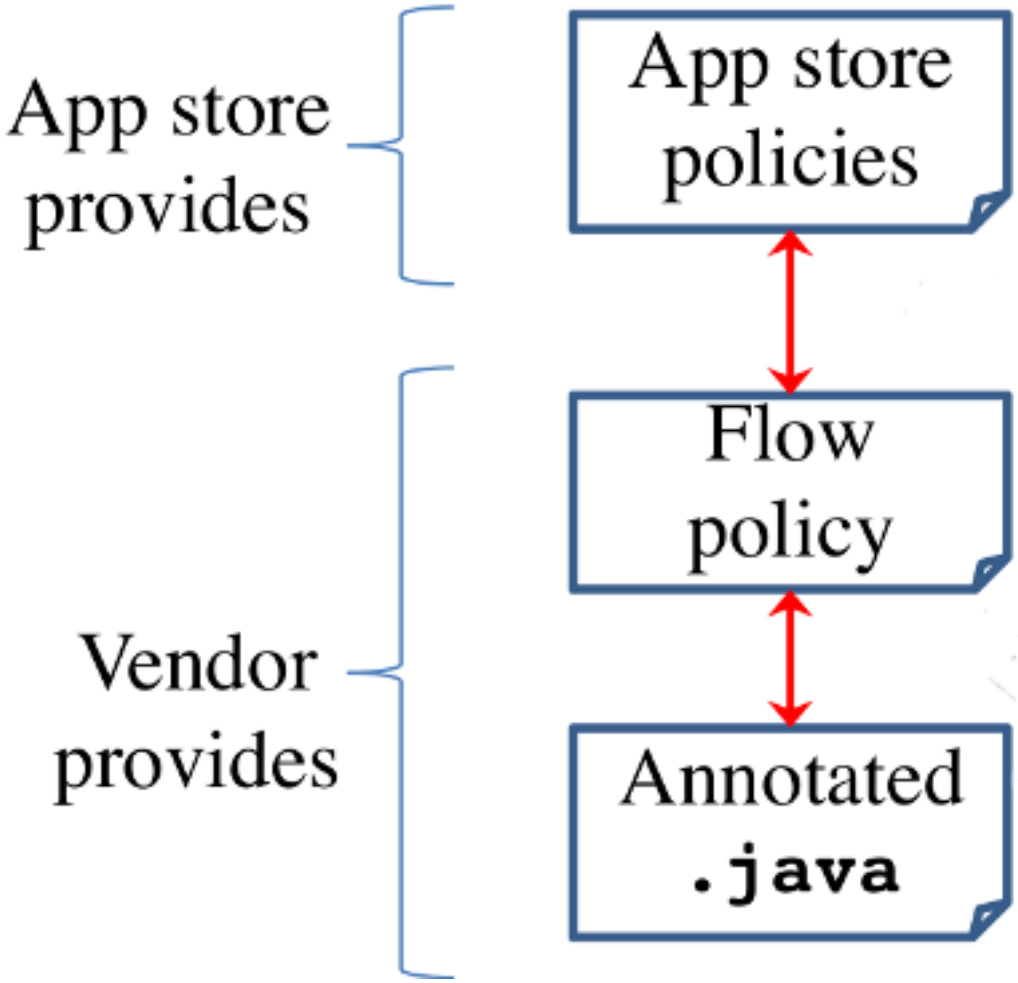
SPARTA: Static Program Analysis for Reliable Trusted Apps

Security type system for Android apps
Guarantees no leakage of private information

Part of DARPA's
Automated Program Analysis
for Cybersecurity (APAC)
program

UNIVERSITY OF WATERLOO

Collaborative Verification



SPARTA: Static Program Analysis for Reliable Trusted Apps

- Software vendor and the app store auditor collaborate on verification task
- Flow-sensitive, context-sensitive information-flow type system
- Red Team provided 72 apps (576 kLOC)
- Detected 96% of information flow malware and 82% of all malware

See ”Collaborative Verification of Information Flow for a High-Assurance App Store” paper at CCS’14

Crowd-sourced verification

Make software verification easy and fun

Make the game accessible to everyone

Harness the power of the crowd

Goal: Verify software while waiting

`http://verigames.com/`

FTfJP'12 paper

Building checkers is easy

Example: Ensure encrypted communication

```
void send(@Encrypted String msg) {…}
@Encrypted String msg1 = ...;
send(msg1);   // OK
String msg2 = ....;
send(msg2);   // Warning!
```

Building checkers is easy

Example: Ensure encrypted communication

```
void send(@Encrypted String msg) {…}
@Encrypted String msg1 = ...;
send(msg1);   // OK
String msg2 = ....;
send(msg2);   // Warning!
```

The complete checker:

```
@Target(ElementType.TYPE_USE)
@SubtypeOf(Unqualified.class)
public @interface Encrypted {}
```

Defining a type system

1. Qualifier hierarchy
   - defines subtyping
2. Type introduction rules
   - types for expressions
3. Type rules
   - checker-specific errors
4. Flow-refinement
   - checker-specific flow properties

Testing Infrastructure

jtreg-based testing as in OpenJDK

Light-weight tests with in-line expected errors:

```
String s = "%+s%";
//:: error: (format.string.invalid)
f.format(s, "illegal");
```

Conclusions

Java 8 syntax for type annotations

Dataflow Framework

Checker Framework for creating type checkers

- Featureful, effective, easy to use, scalable

Prevent bugs at compile time

Create custom type-checkers

Learn more, or download at:

`http://CheckerFramework.org/`