Contrast SECURITY

Static Code Analysis

A behind-the-scenes look

Arno Haase

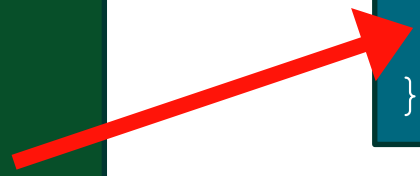
File and text utilities

```
$ grep -r InvoiceService
```

```
$ find . -name *.java | grep /test/ | xargs cat | wc
```

Challenge: Cross Referencing

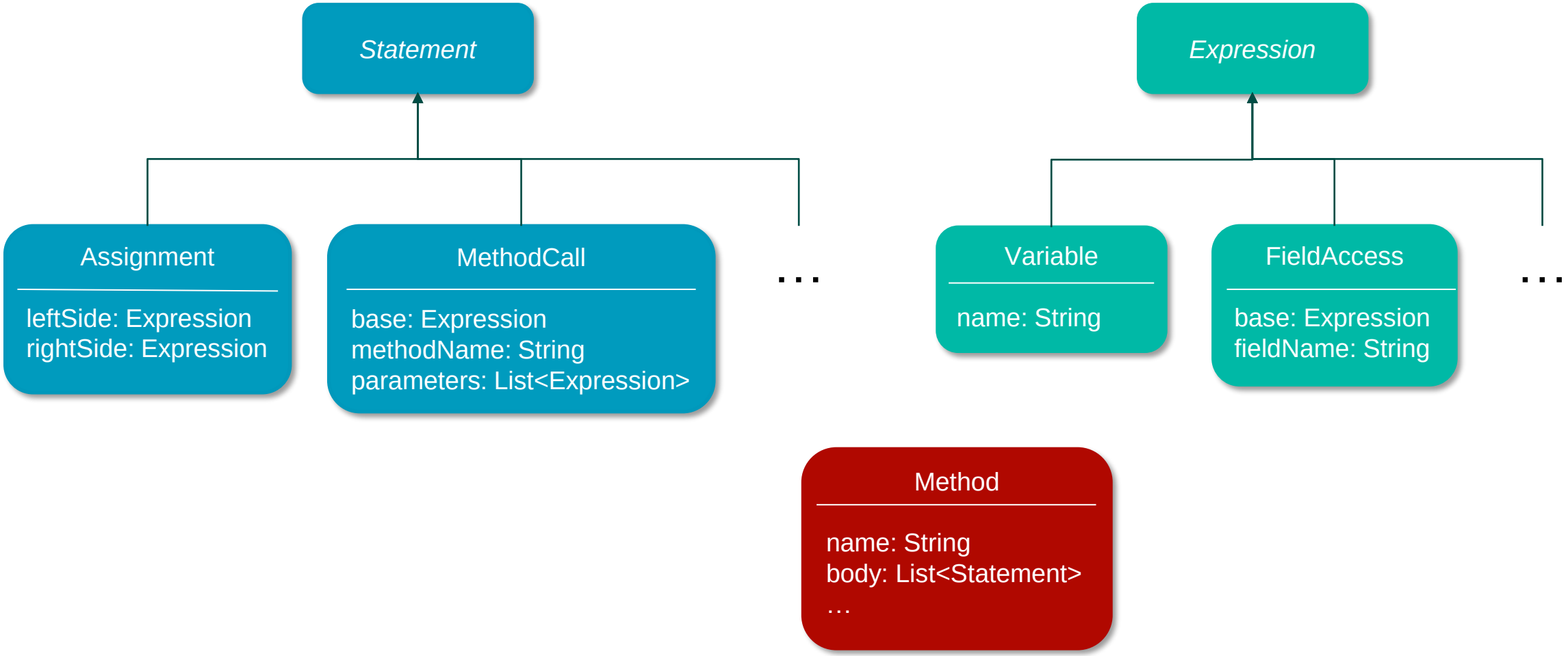
```
void myMethod(Person p) {
   …
   System.out.println(p.name);
   …
}
```



```
class Person {
   String name;
}
```

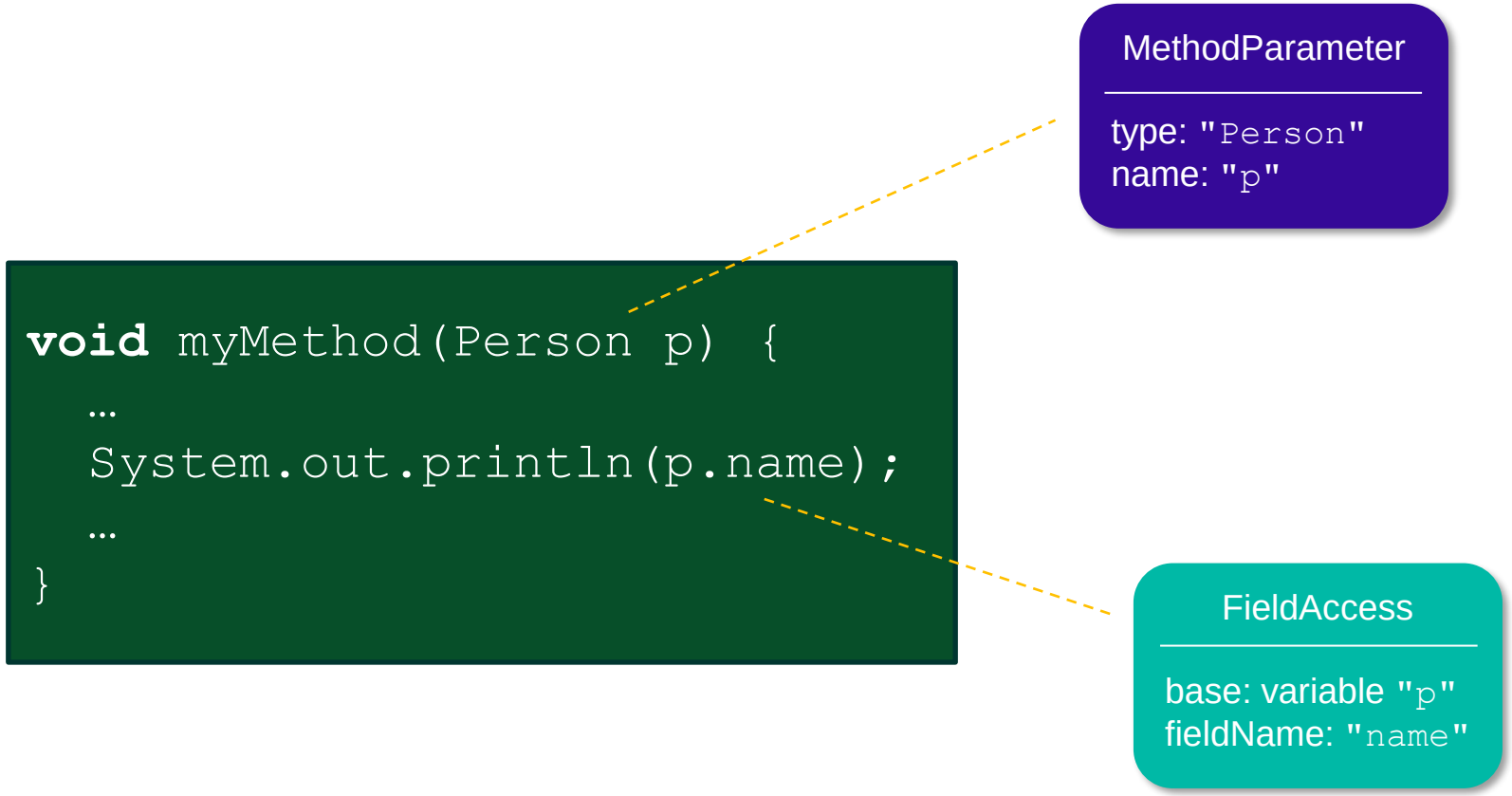
```
class City {
   String name;
}
```

Abstract Syntax Tree



Contrast SECURITY

Which identifier refers to what?



Explicit support for all language features

```
String lambda(Person p) {
  Supplier<Person> s=()->p;
  return s.get().name;
}
```

```
void typeInference(Person p) {
  var x = p;
  System.out.println(x.name);
}
```

```
sealed class A {}
non-sealed class B extends A {}
```

Byte Code

```
void myMethod(Person p) {
  …
  System.out.println(p.name);
  …
}
```

```
class Person {
  String name;
}
```

```
GETSTATIC java/lang/System.out : Ljava/io/PrintStream;
ALOAD 1
GETFIELD org/demo/Person.name : Ljava/lang/String;
INVOKEVIRTUAL java/io/PrintStream.println (Ljava/lang/String;)V
```

Which is the better choice?

Source Code Parser

- Analysis of comments, layout, language features
- Fault tolerant parsing
- IDE plugin
- Smoother result presentation
- (btw – requires byte code analysis of third party libraries)

Byte Code Parser

- Efficiency
- Leverage compiler
- Robustness against new language features

Tracking flows

```
@GetMapping
void myMethod(HttpServletRequest request) {
  String s = request.getHeader("abc");
  jdbcConnection.prepareStatement(s).update();
}
```

Assignment

```
@GetMapping
void myMethod(HttpServletRequest request) {
  String s = request.getHeader("abc");
  String x = s;
  Person p1 = new Person();
  p1.name = x;
  y = p.name;
  jdbcConnection.prepareStatement(y).update();
}
```

Propagators

```
@GetMapping
void myMethod(HttpServletRequest request) {
  String raw = request.getHeader("abc");
  String sql = "delete PERSON where id=" + raw;
  jdbcConnection.prepareStatement(sql).update();
}
```

Conditional Flow

```
@GetMapping
void myMethod(HttpServletRequest request) {
  String s;
  if(request.getHeader("xyz" == null) {
    s = SAFE_JDBC_STRING;
  }
  else {
    s = request.getHeader("abc");
  }
  jdbcConnection.prepareStatement(s).update();
}
```

Conditional Flow

```
@GetMapping
void myMethod(HttpServletRequest request) {
  String s = request.getHeader("abc");
  if(System.currentTimeMillis() > 0) {
    s = SAFE_JDBC_STRING;
  }
  if(request.getHeader("abc") == null) {
    jdbcConnection.prepareStatement(s).update();
  }
  if(s == null) {
    jdbcConnection.prepareStatement(s).update();
  }
}
```

Combinatorial Explosion

```
@GetMapping
void myMethod(HttpServletRequest request) {
  String s = request.getHeader("abc");
  if(b()) System.out.println("Hi");     // 2 paths
  if(b()) System.out.println("Hi");     // *2
  if(b()) System.out.println("Hi");     // *2
  if(b()) System.out.println("Hi");     // *2
  if(b()) System.out.println("Hi");     // *2
  if(b()) System.out.println("Hi");     // *2
  if(b()) System.out.println("Hi");     // *2
  if(b()) System.out.println("Hi");     // *2
  if(b()) System.out.println("Hi");     // *2
  if(b()) System.out.println("Hi");     // *2 = 1024 paths!
  jdbcConnection.prepareStatement(s).update();
}
```

Contrast SECURITY

Merging

```
@GetMapping
void myMethod(HttpServletRequest request) {
  String s;
  if(request.getHeader("xyz" == null) {
    s = SAFE_JDBC_STRING;
  }
  else {
    s = request.getHeader("abc");
  }
  jdbcConnection.prepareStatement(s).update();
}
```

Combinatorial Explosion - Revisited

```
@GetMapping
void myMethod(HttpServletRequest request) {
  String s = request.getHeader("abc");
  if(b()) System.out.println("Hi");     // 2 paths
  if(b()) System.out.println("Hi");     // +1
  if(b()) System.out.println("Hi");     // +1
  if(b()) System.out.println("Hi");     // +1
  if(b()) System.out.println("Hi");     // +1
  if(b()) System.out.println("Hi");     // +1
  if(b()) System.out.println("Hi");     // +1
  if(b()) System.out.println("Hi");     // +1
  if(b()) System.out.println("Hi");     // +1
  if(b()) System.out.println("Hi");     // +1= 11 paths!
  jdbcConnection.prepareStatement(s).update();
}
```

Contrast SECURITY

Merging - Limitations

```
@GetMapping
void myMethod(HttpServletRequest request) {
  String s = request.getHeader("abc");
  if(condition)
    s = "safe";
  …
  if(condition)
    jdbcConnection.prepareStatement(s).update();
}
```

Loops

```
@GetMapping
void myMethod(HttpServletRequest request) {
  String a,b,c = request.getHeader("abc");
  while(check()) {
    a=b;
    b=c;
    jdbcConnection.prepareStatement(a).update();
  }
}
```

Impossible to be precise

```
@GetMapping
void myMethod(HttpServletRequest request) {
  String s = request.getHeader("abc");
  long n = new SecureRandom().nextLong()
  while(n!=1) {
    if(n%2 == 0) n=n/2;
    else n=3*n+1;
  }
  jdbcConnection.prepareStatement(s).update();
}
```

Function Calls

```
@GetMapping
void myMethod(HttpServletRequest request) {
  String s = source(request);
  sink(s)
}

static String source(HttpServletRequest r) {
  return request.getHeader("abc");
}

static void sink(String s) {
  jdbcConnection.prepareStatement(s).update();
}
```

Recursion

```
@GetMapping
void myMethod(HttpServletRequest request) {
  String s = request.getHeader("abc");
  f("", "", s);
}

static void f(String a, String b, String c) {
  jdbcConnection.prepareStatement(a).update();
  if(…)
    f(b, c, "safe");
}
```

Virtual Method Calls

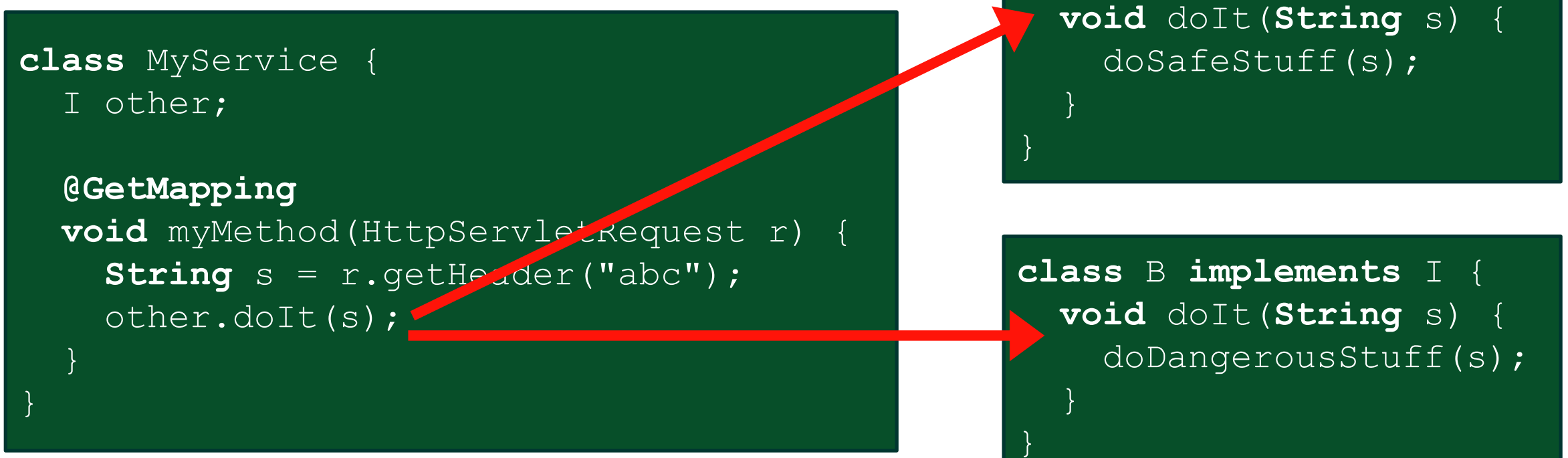
```
class MyService {
  I other;

  @GetMapping
  void myMethod(HttpServletRequest r) {
    String s = r.getHeader("abc");
    other.doIt(s);
  }
}
```

```
interface I {
  void doIt(String s);
}
```

```
class A implements I {
  void doIt(String s) {
    doSafeStuff(s);
  }
}
```

```
class B implements I {
  void doIt(String s) {
    doDangerousStuff(s);
  }
}
```



Call Graph

- List of possible call targets per function call statement

```
String s = request.getHeader("abc");
Consumer<> c = new Safe();
c.accept(s);
```

```
class Safe
  implements Consumer<String>
{
  void accept(String s) {
    System.out.println(s);
  }
}
```

```
class NotSafe
  implements Consumer<String>
{
  void accept(String s) {
    conn.prepareStatement(s)
      .update();
  }
}
```

Flow sensitivity

```
if(…) {
  s = request.getHeader("abc");
  c = new Safe();
}
else {
  s = "a string literal";
  c = new NotSafe()
}
c.accept(s);
```

Aliasing

```
StringBuilder a = new StringBuilder();
var b=a;
…
a.append(request.getHeader("abc"));
…
jdbcConnection
   .prepareStatement(b.toString())
   .update();
```

Framework / Library knowledge

Start point for scanning

All fields of ‘data‘ are untrustworthy

Not vulnerable for XSS

```
@PutMapping("/api/person/{id}")
@Produces("application/json")
Person getPerson(
  @PathParam("id") String id,
  @RequestBody Person data,
) {
  jdbcConnection.prepareStatement(
    "update PERSON set name='" +
    data.name + "' WHERE id='" + id + "'"
  .update();
  …
}
```

Identifying unique findings

```
String s = getHeader("a");
if(…)
  sqlUpdate(SQL_1 + s);
else
  sqlUpdate(SQL_2 + s;
```

```
String sql;
String s = getHeader("a");
if(…)
  sql = SQL_1 + s;
else
  sql = SQL_2 + s;
sqlUpdate(sql);
```

```
var sql = new StringBuilder(QUERY);
if(…)
  sql.append(" and name='" + person.name + "'");
if(…)
 sql.append(" and street='" + person.street + "'");
if(…)
 sql.append(" and city='" + person.city + "'");
sqlQuery(sql);
```

Contrast SECURITY

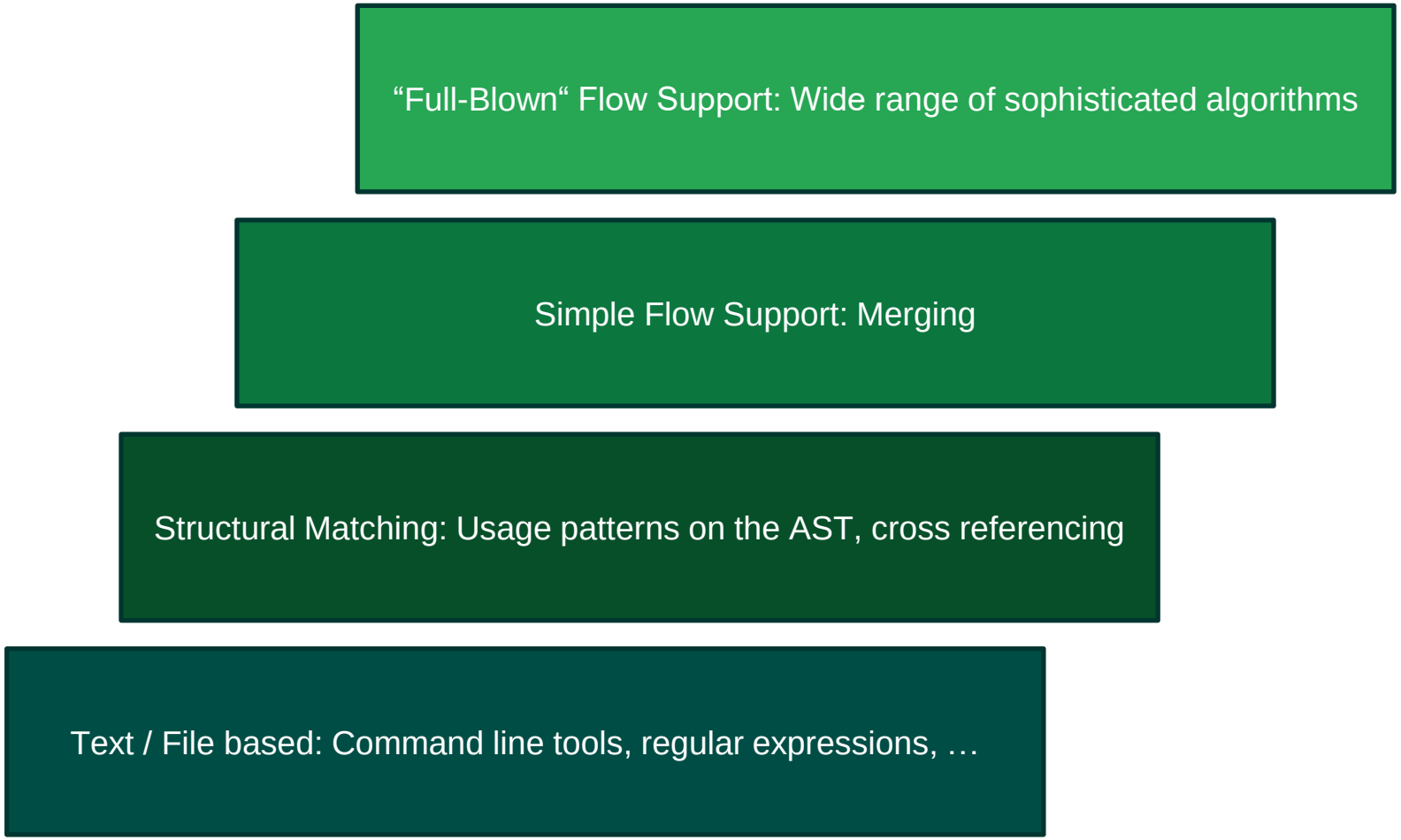
Quality of results

“If we don‘t report a problem, there is no problem“

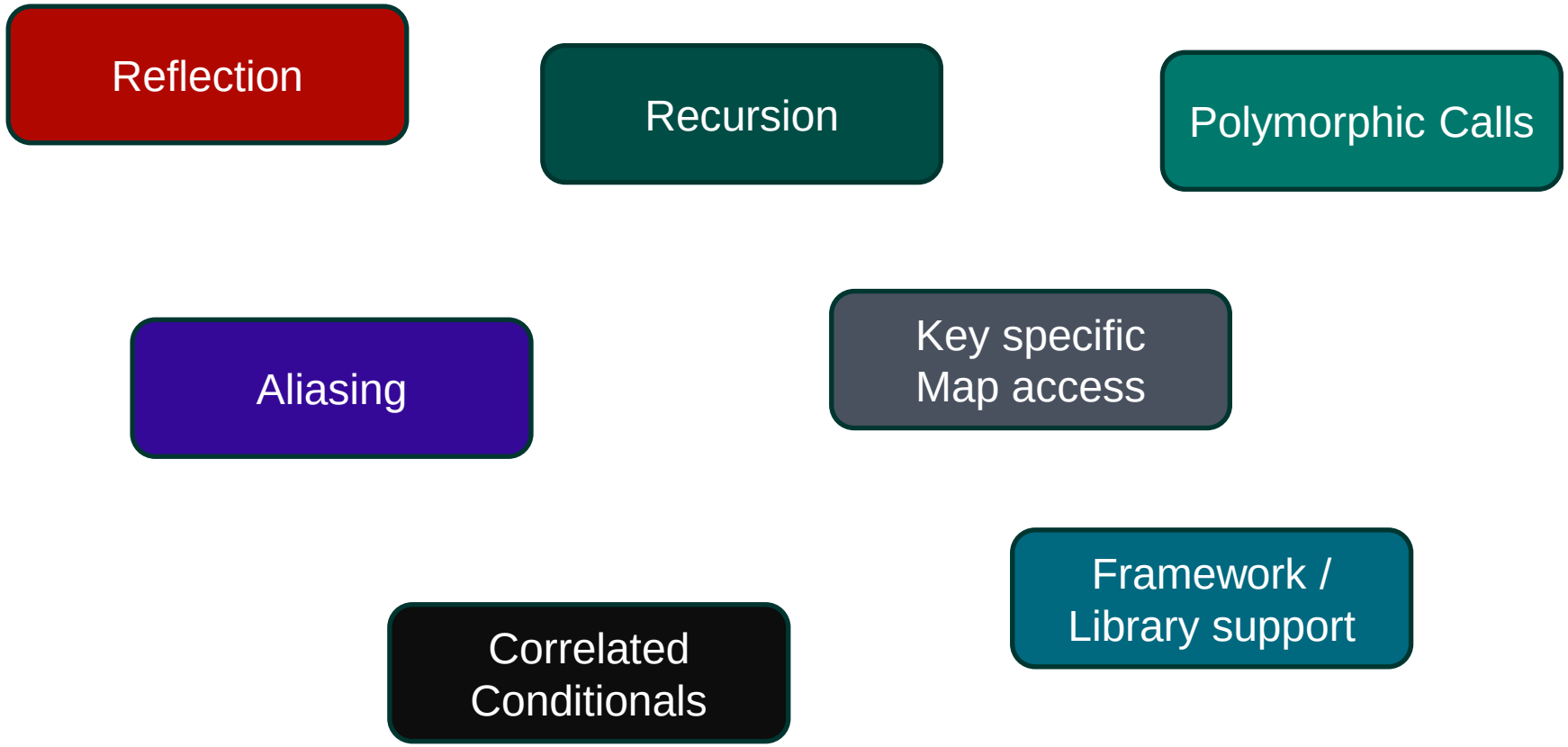
vs.

“Everything we report is worth looking at“

Levels of Sophistication



Stuff that‘s tricky to analyze statically



Thank you

Contrast SECURITY