01

SHIFTING LEFT '21

ShiftLeft

GRAPH DATABASES FOR CODE ANALYSIS

Michael Pollmeier
ShiftLeft
Principal Engineer, Code Science Team

WHO AM I?

- Michael Pollmeier
- Principal engineer in the ShiftLeft Code Science team
  - Part of the initial crew @ShiftLeft
- Scala, Graph Databases and Open Source fan
  - gremlin-scala: a tinkerpop language variant for Scala
  - overflowdb: in-memory graph database with a low memory footprint
  - Apache committer, contributor to various OSS projects
- @pollmeier

WHAT YOU'LL LEARN IN THIS SESSION

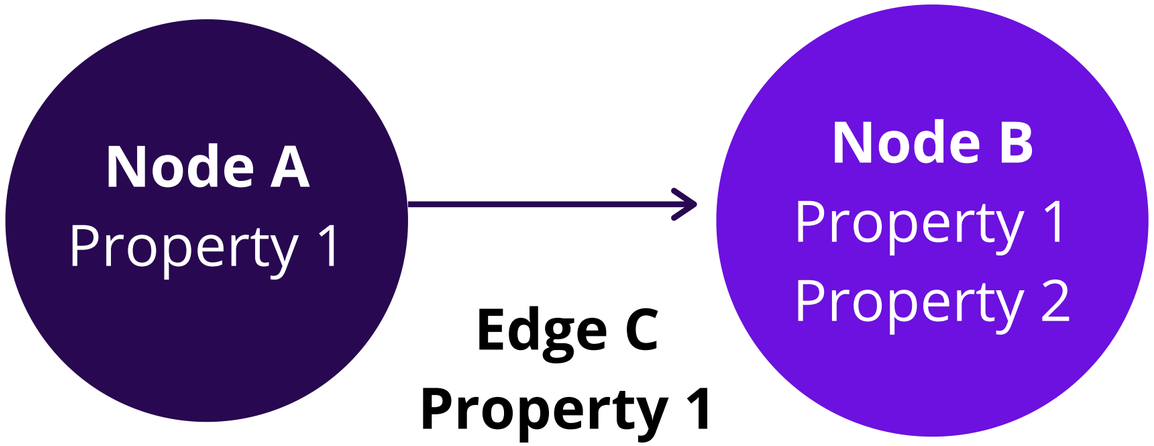
03

- What's a Graph Database?
- Code analysis domain model: graph and relational
- SQL injection: building up a traversal in small increments
- Summary

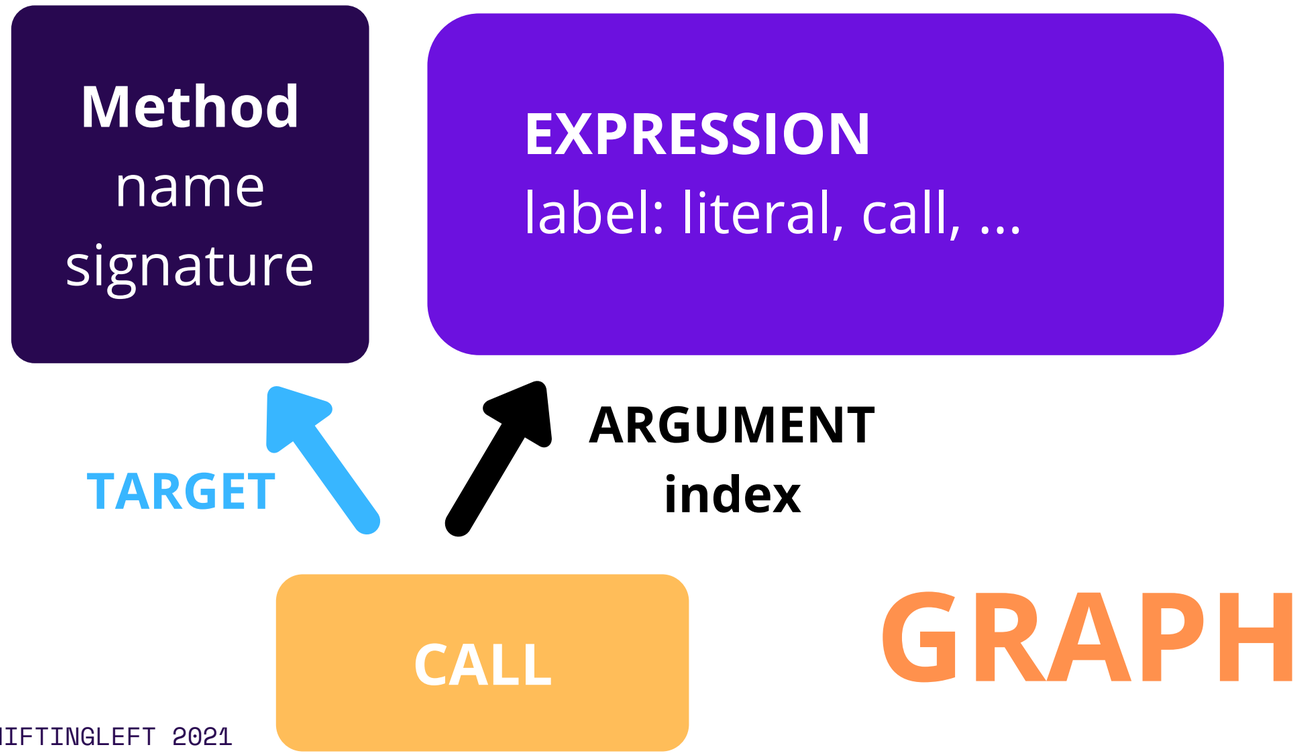
ShiftLeft

WHAT'S A GRAPH DATABASE

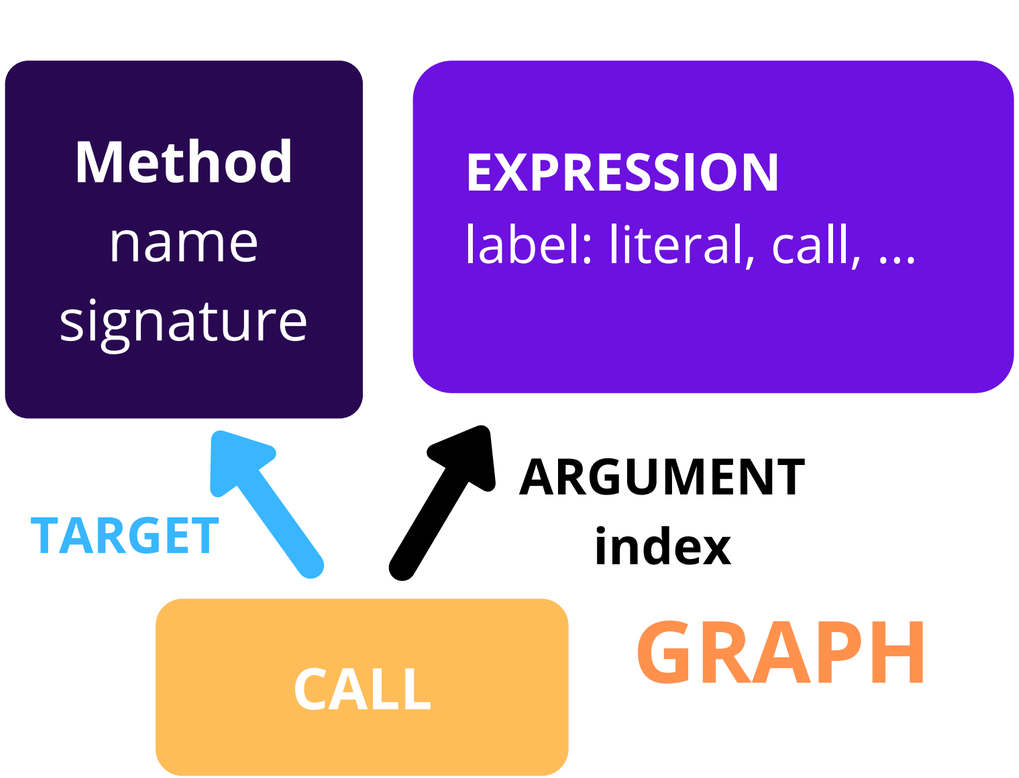
- Nodes with properties
- Directed edges with properties - first class citizens
- Modelling compatible with human mind



CODE ANALYSIS DOMAIN MODEL: GRAPH MODEL



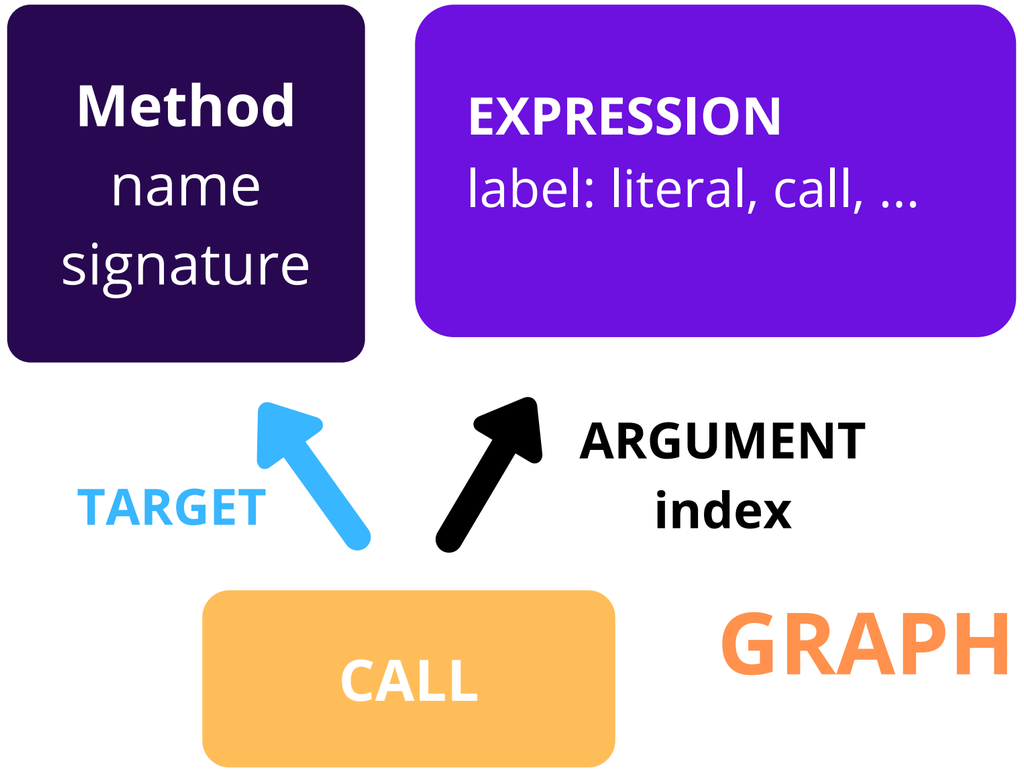
CODE ANALYSIS DOMAIN MODEL



1: METHOD CALL, PASSING A LITERAL AS ARGUMENT

```
String sql = "delete from users";
statement.execute(sql);
```

```
graph.nodes("METHOD")
     .has("name", "Statement.execute")
     .in("TARGET")     // -> CALL
     .out("ARGUMENT")
```



1: METHOD CALL, PASSING A LITERAL AS ARGUMENT

```
String sql = "delete from users";
statement.execute(sql);
```

```
graph.nodes("METHOD")
     .has("name", "Statement.execute")
     .in("TARGET")    // -> CALL
     .out("ARGUMENT")
```

```
SELECT * FROM METHOD
JOIN CALL ON METHOD.id = CALL.target_method_id
JOIN CALL_ARG_IDX ON CALL_ARG_IDX.call_id = CALL.id
JOIN EXPRESSION ON CALL_ARG_IDX.expression_id = EXPRESSION.id
WHERE METHOD.name = 'java.sql.Statement.execute';
```

Method
name
signature

EXPRESSION
label: literal, call, ...
literal_id
call_id

CALL
target_method_id

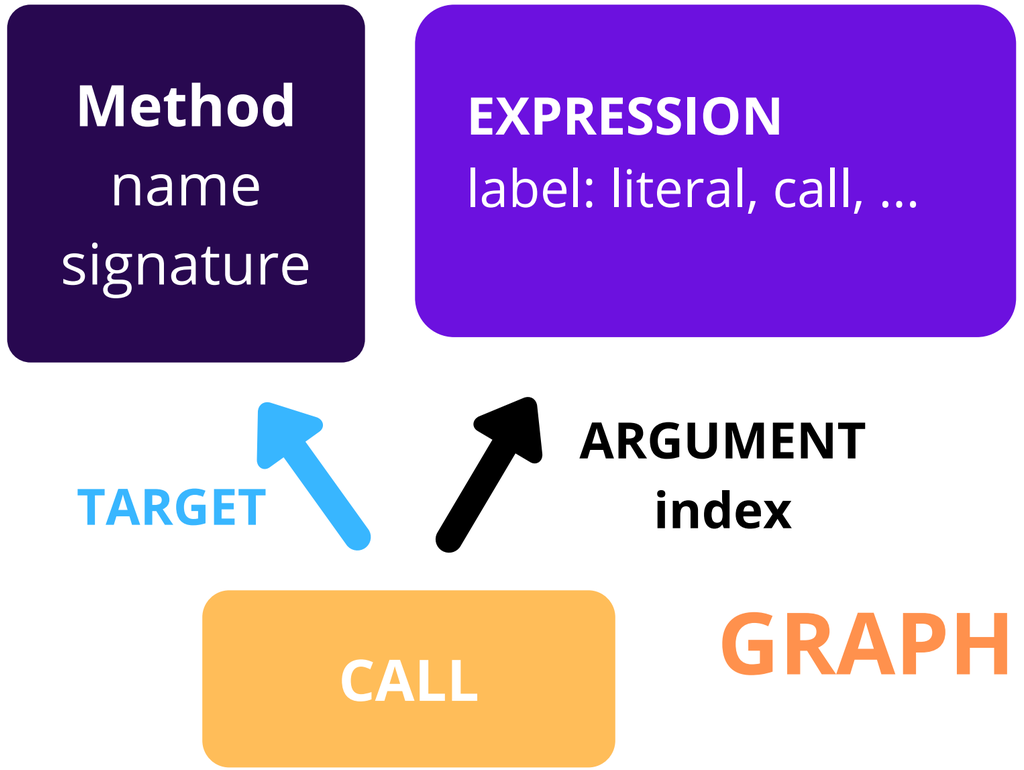
CALL_ARG_IDX
call_id
expression_id
argument_index

RELATIONAL

2: CALL ARGUMENT IS ANOTHER METHOD CALL

```
String sql = servlet.inputParameter("sql");
statement.execute(sql);
```

```
graph.nodes("METHOD")
     .has("name", "Statement.execute")
     .in("TARGET")      // -> CALL
     .out("ARGUMENT")
     .hasLabel("CALL") // only CALLs
     .out("TARGET")     // -> METHOD
```



2: CALL ARGUMENT IS ANOTHER METHOD CALL

```
String sql = servlet.inputParameter("sql");
statement.execute(sql);
```

```
graph.nodes("METHOD")
     .has("name", "Statement.execute")
     .in("TARGET")     // -> CALL
     .out("ARGUMENT")
     .hasLabel("CALL") // only CALLs
     .out("TARGET")    // -> METHOD
```

```
SELECT * FROM METHOD method1
JOIN CALL call1 ON method1.id = call1.target_method_id
JOIN CALL_ARG_IDX ON CALL_ARG_IDX.call_id = call1.id
JOIN EXPRESSION
  ON CALL_ARG_IDX.expression_id = EXPRESSION.id
  AND EXPRESSION.type = 'CALL'
JOIN CALL call2 ON EXPRESSION.call_id = call2.id
JOIN METHOD method2 ON call2.target_method_id = method2
WHERE method1.name = 'java.sql.Statement.execute';
```

Method
name
signature

EXPRESSION
label: literal, call, ...
literal_id
call_id

CALL
target_method_id

CALL_ARG_IDX
call_id
expression_id
argument_index

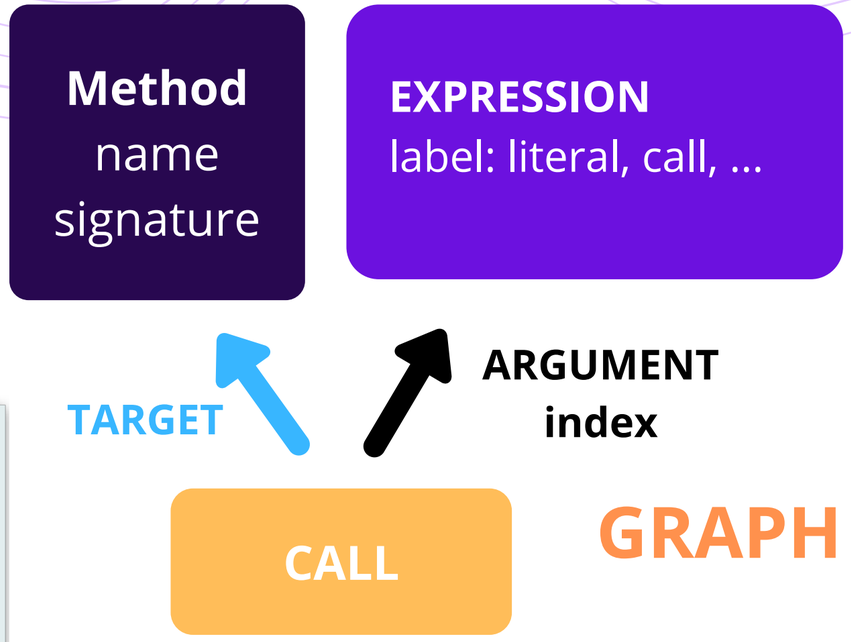
RELATIONAL

3: CHAIN OF METHOD CALLS OF ARBITRARY LENGTH

```
public String concatenate(String a, String b) {
  return a + b;
}
String select = "select * from user where name = ";
String user   = servlet.inputParameter("user");
String sql    = concatenate(select, user);
statement.execute(sql);
```

```
graph.nodes("METHOD")
  .has("name", "Statement.execute")
  .in("TARGET") // -> CALL
  .repeat(
    _.out("ARGUMENT").hasLabel("CALL"))
  (_.until(_.out("TARGET").has("name", "Servlet.inputParameter")))
```

```
SELECT 'giving up';
```



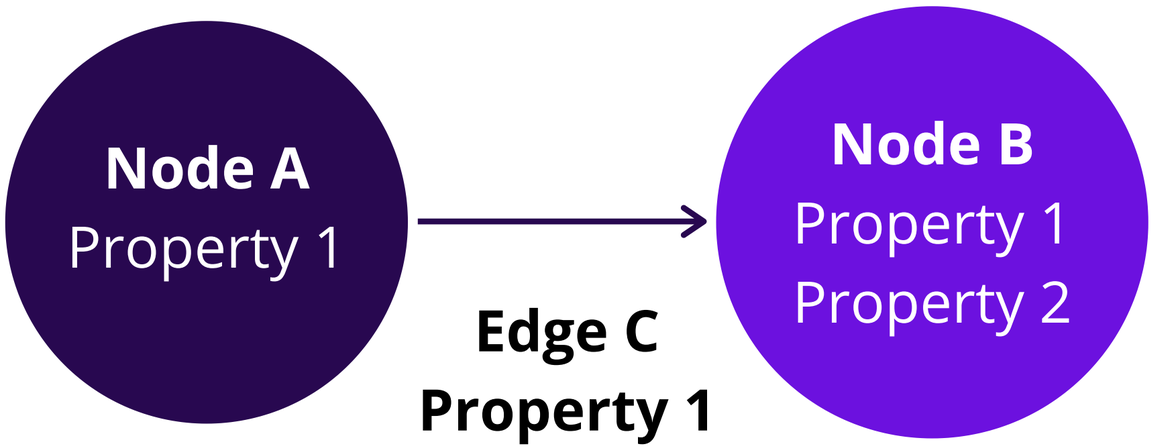
ShiftLeft

THE SILVER BULLET?

- For code analysis: maybe
- Not as mature as Relational DBs
- https://github.com/ShiftLeftSecurity/overflowdb

SUMMARY

- Nodes and directed edges
  - relationships are first class citizens
  - Modelling compatible with human mind
- Traversals very expressive
  - no need to reify IDs in joins
  - Repeat step
- Still a niche technology



QUESTIONS & ANSWERS

Please feel free to ask any questions in the chat interface!

THANK YOU!

QUESTIONS? CLARIFICATIONS?
LET US KNOW!

TWITTER

@pollmeier