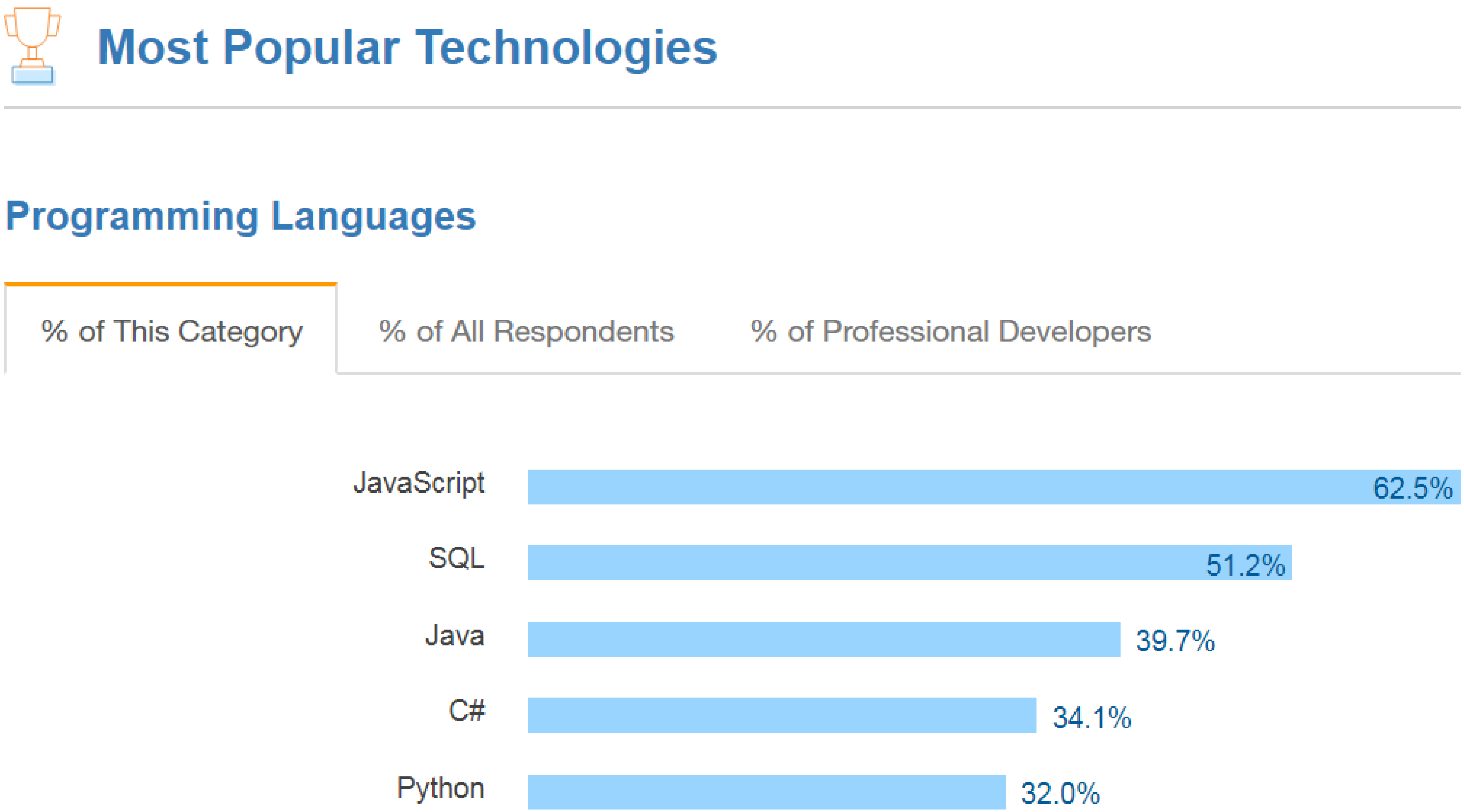
Graph-based analysis of JavaScript source code repositories

Gábor Szárnyas

Graph Processing devroom @ FOSDEM 2018

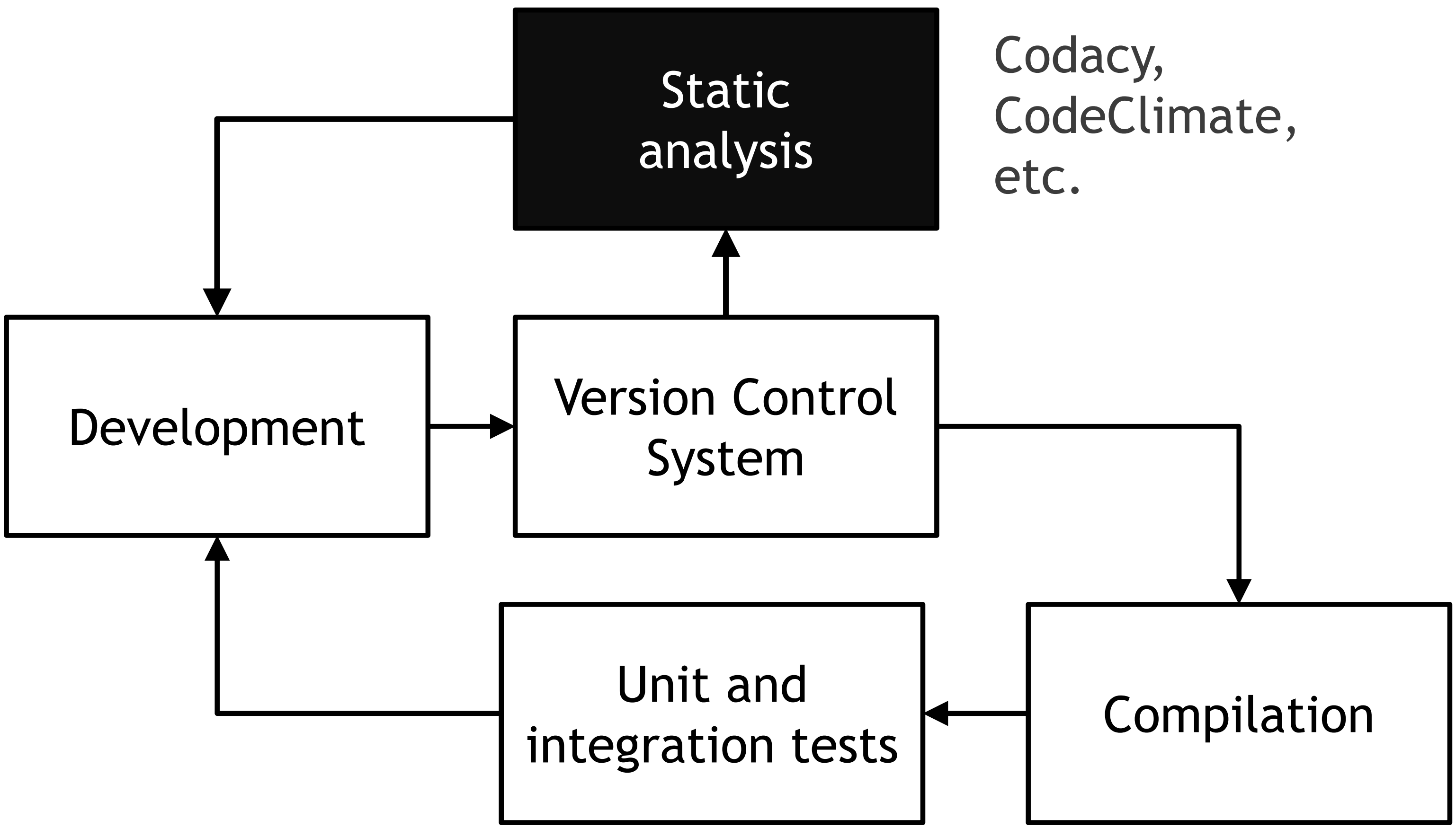
JAVASCRIPT



Latest standard: **ECMAScript 2017**

STATIC ANALYSIS

- *Static source code analysis* is a software testing approach performed without compiling and executing the program itself.



STATIC ANALYSIS TOOLS

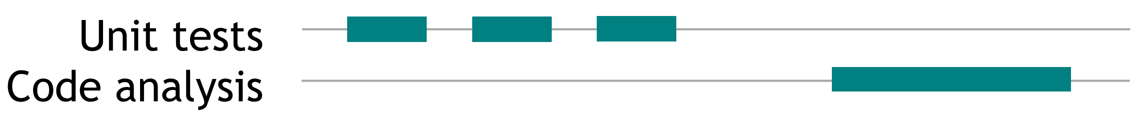
- C
  - lint -> *linters*
- Java
  - FindBugs
  - PMD
- JavaScript
  - ESLint
  - Facebook Flow
  - Tern.js
  - TAJS

1. **NO GLOBAL RULES OR**
2. **DIFFICULT TO EXTEND**

PERFORMANCE CONSIDERATIONS

- Checking global rules is computationally expensive
- Slow for large projects, difficult to integrate even to CI

- Workaround #1: no global rules (ESLint)
- Workaround #2: batching (e.g. 1/day)



- Workaround #3: custom algorithms (e.g. Flow)

PROJECT GOALS

Goal

- Static analysis for JavaScript applications

Design considerations

- Custom analysis rules
  - Both global and local
  - Extensible
- High-performance
  - “real-time” responses

ARCHITECTURE AND WORKFLOW

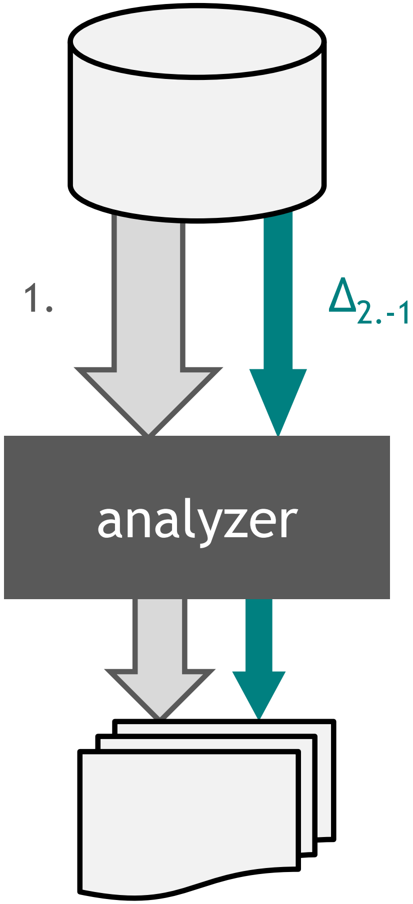
PROPOSED APPROACH

Design considerations

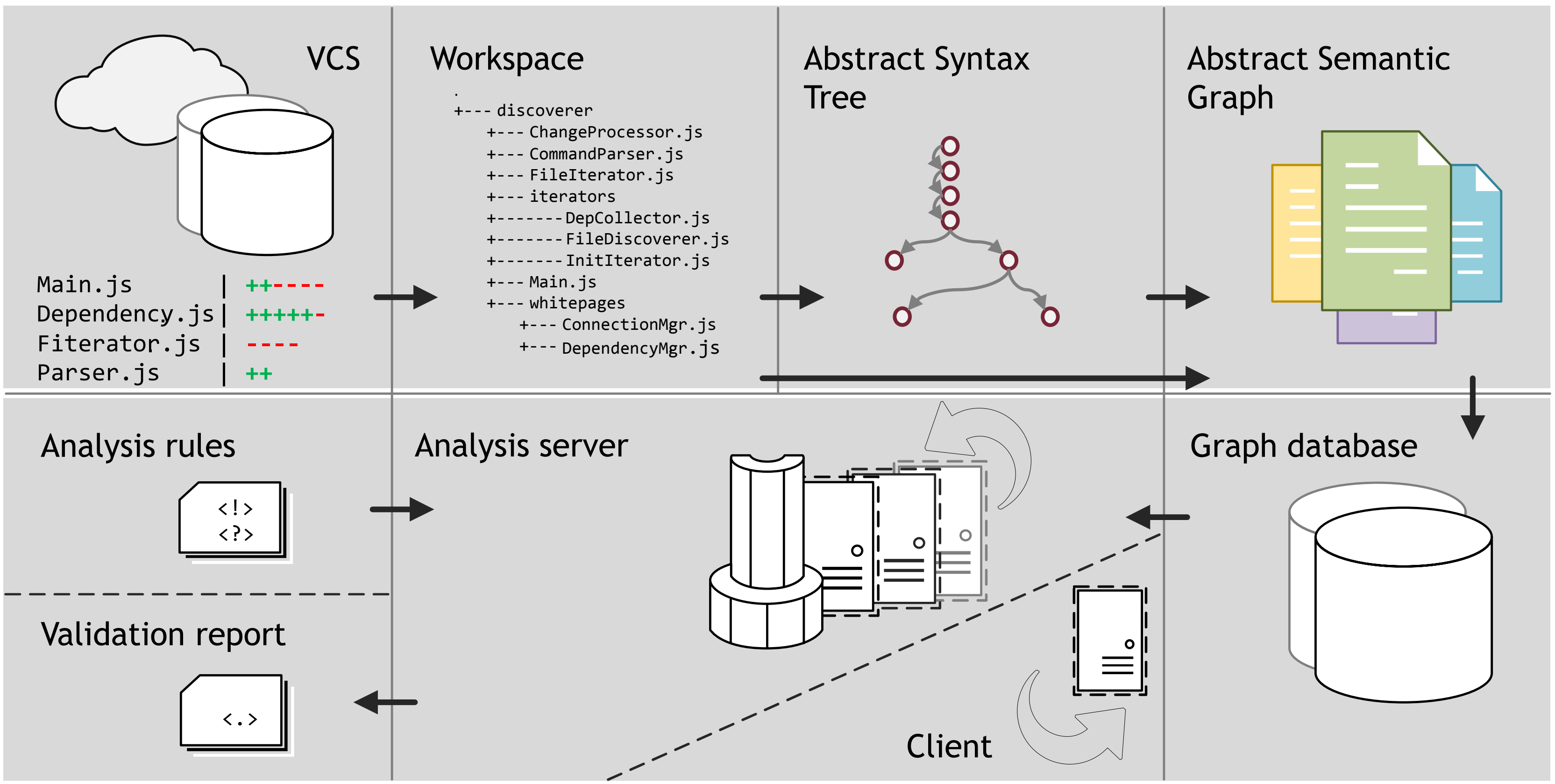
- Custom analysis rules
- High-performance

Approach

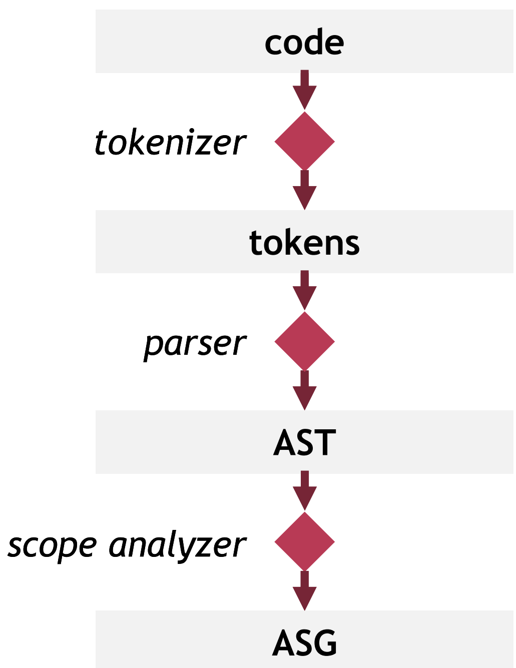
- Use a declarative query language
- Use incremental processing
  - in lieu of batch execution
  - file-granularity
  - maintain results



ARCHITECTURE



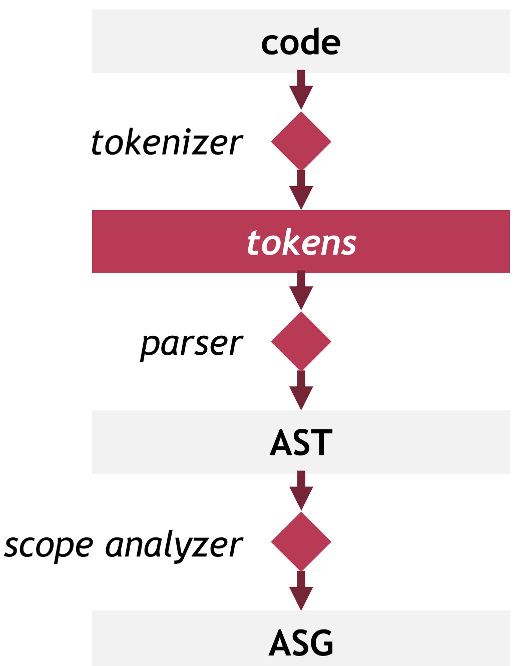
CODE PROCESSING STEPS



a sequence of statements:

```
var foo = 1 / 0
```

CODE PROCESSING STEPS — TOKENS



tokens: the shortest meaningful character sequence

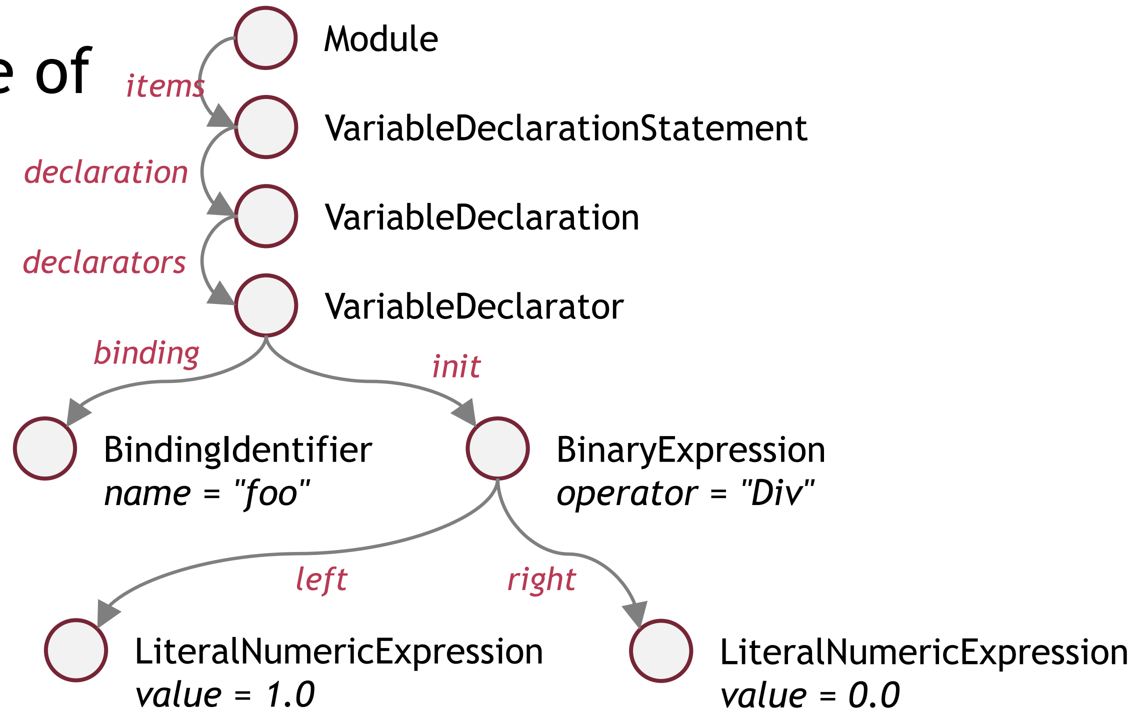
```
var foo = 1 / 0
```

Token	Token type
var	VAR (Keyword)
foo	IDENTIFIER (Ident)
=	ASSIGN (Punctuator)
1	NUMBER (NumericLiteral)
/	DIV (Punctuator)
0	NUMBER (NumericLiteral)

CODE PROCESSING STEPS AST

Abstract Syntax Tree

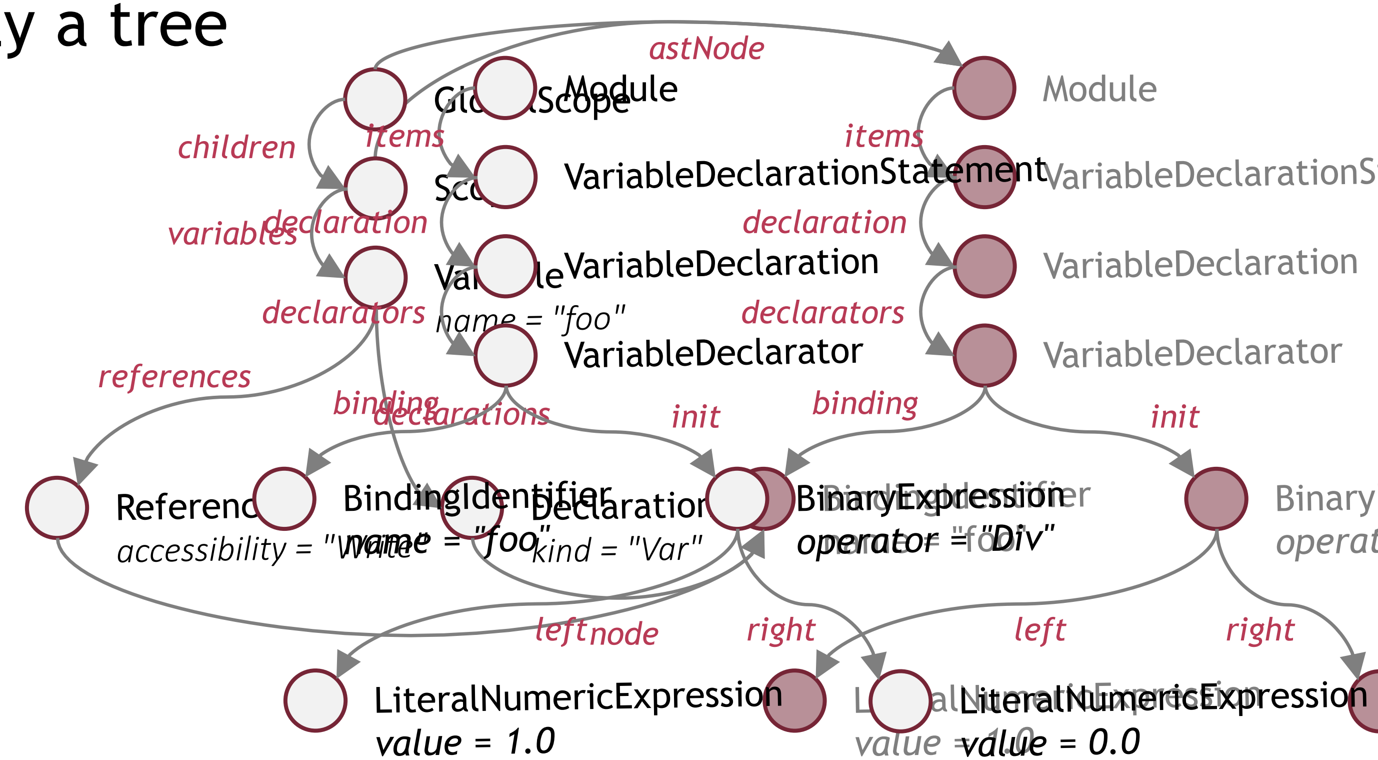
- Tree representation of
- the grammar structure of
- sequence of tokens.



CODE PROCESSING STEPS

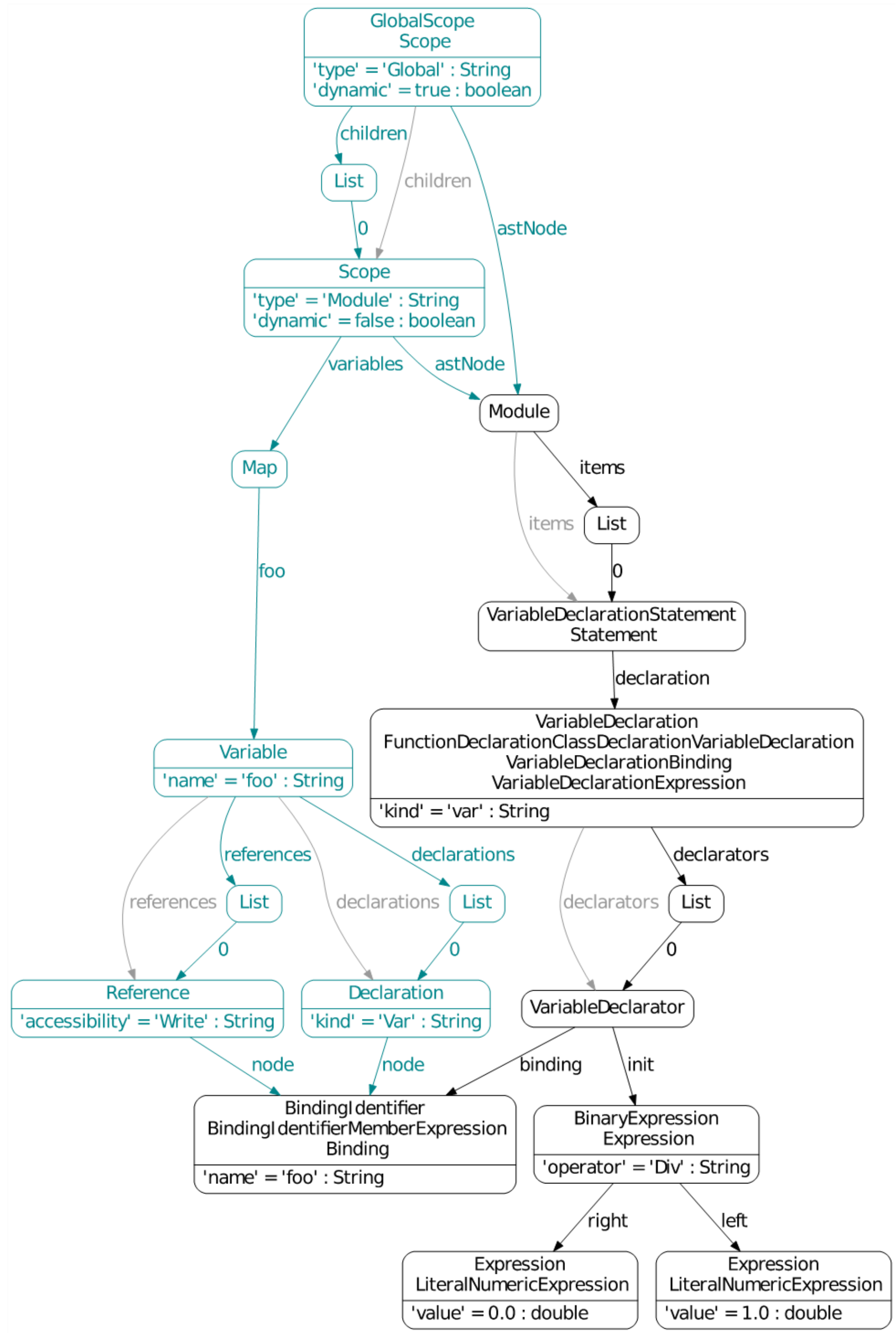
Abstract Semantic Graph

- Not necessarily a tree
- Has scopes & semantic info
- Cross edges



AST VS. ASG

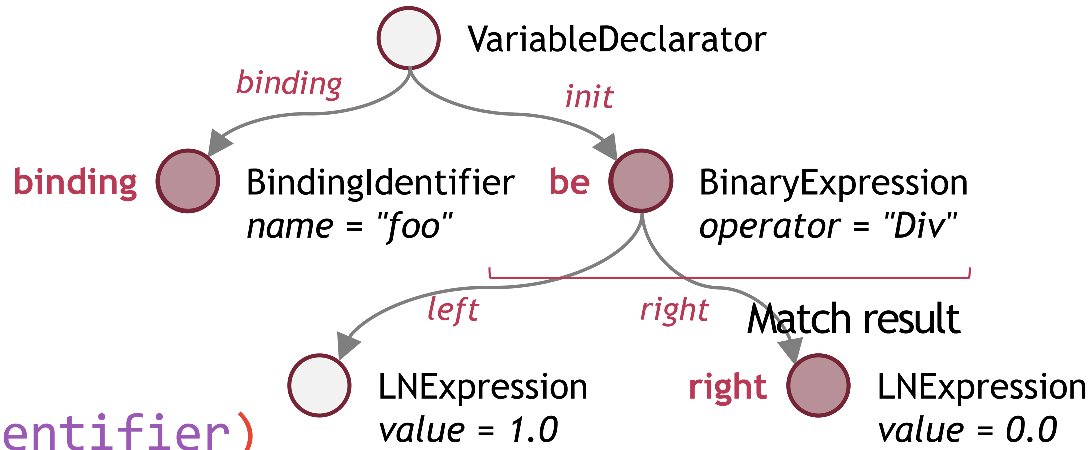
```
var foo = 1 / 0
```



1 LOC -> 20+ nodes

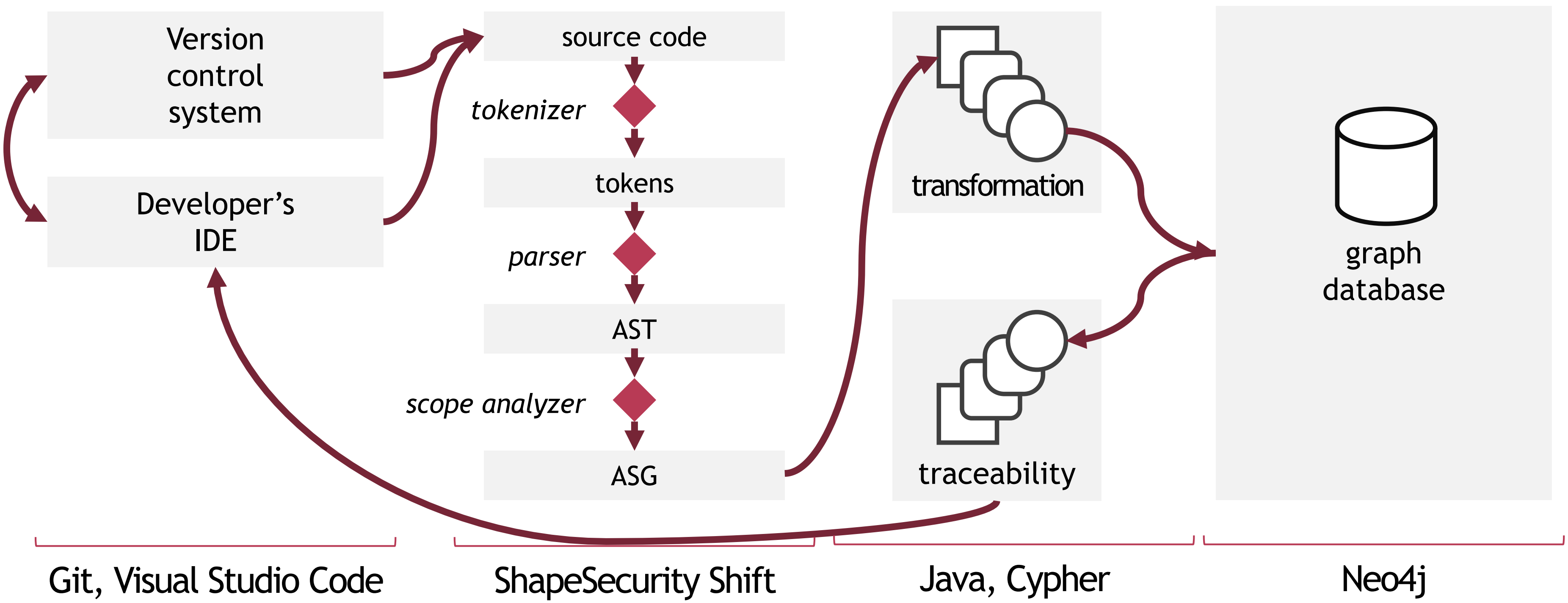
PATTERN MATCHING

- Declarative graph patterns with Cypher



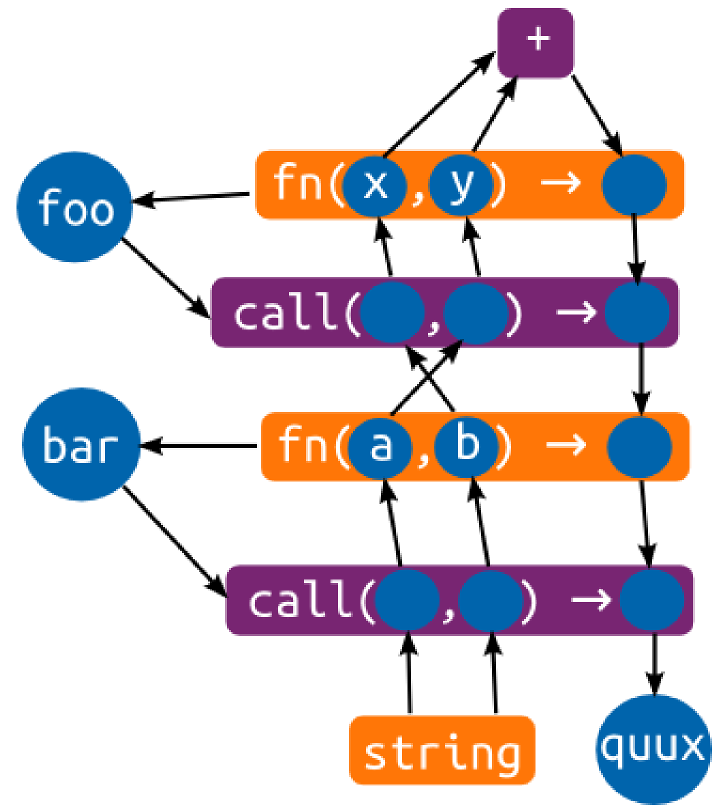

```
MATCH  (binding:BindingIdentifier)
       <-[:binding]-()-->
       (be:BinaryExpression)
       -[:right]->(right:LNExpression)

WHERE be.operator = 'Div'
  AND right.value = 0.0

RETURN binding
```

WORKFLOW


Version control system
Developer's IDE
source code
tokenizer
tokens
parser
AST
scope analyzer
ASG
transformation
traceability
graph database
Git, Visual Studio Code
ShapeSecurity Shift
Java, Cypher
Neo4j

USE CASES

TYPE INFERENCING



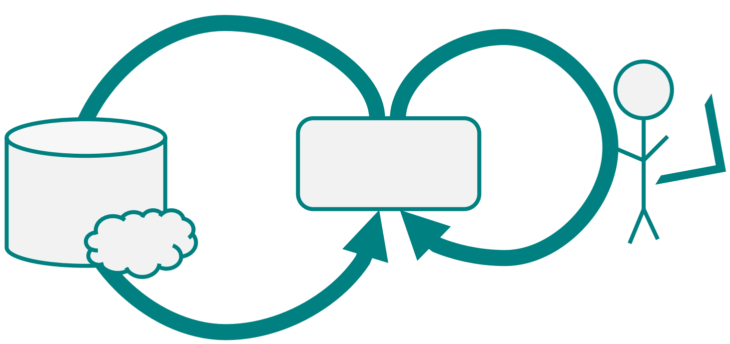

```
function foo(x, y) {
  return (x + y);
}
function bar(a, b) {
  return foo(b, a);
}
var quux = bar("goodbye", "hello");
```

Source: http://marijnhaverbeke.nl/blog/tern.html

USE CASES

Reachability:

- dead code detection
- async/await (ECMAScript 2017)
- potential division by zero



TECH DETAILS

IMPORTS AND EXPORTS

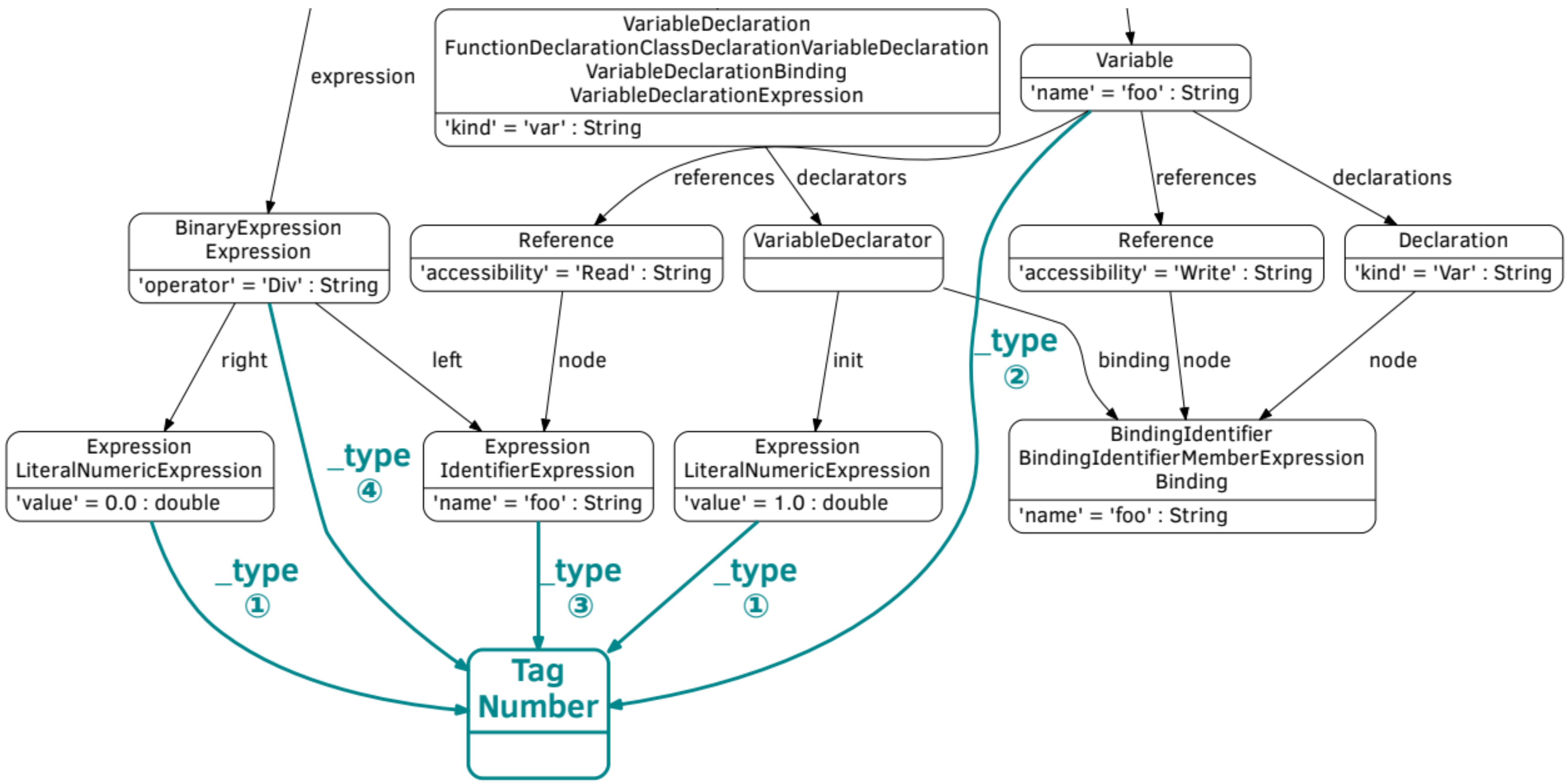
```
// exportName
export { name1, ... };
// exportDefaultName
export default name1;
// exportAlias
export { name1 as exportedName1, ... };
// exportAsDefault
export { name1 as default, ... };
// exportEmptyLetDeclaration
export let name1, ... ;
// exportEmptyVarDeclaration
export var name1, ... ;
// exportLetDeclaration
export let name1 = ..., ... ;
// exportVarDeclaration
export var name1 = ..., ... ;
// exportConstDeclaration
export const name1 = ..., ... ;
// exportClass
export class name1 { ... }
```

	importName	importAlias	importDefault	importNamespace	importModule
exportName	●	●	○	●	●
exportDefaultName	●	●	●	●	●
exportAlias	●	●	○	●	●
exportAsDefault	○	○	●	○	●
exportEmptyLetDeclaration	●	●	○	●	●
exportEmptyVarDeclaration	●	●	○	●	●
exportLetDeclaration	●	●	○	●	●

```
MATCH
// exporter.js: export { name1 as exportedName1 };
    (exporter:CompilationUnit)-[:contains]->(:ExportLocals)-[:namedExports]->(exportLocalSpecifier:ExportLocalSpecifier)
        -[:name]->(:IdentifierExpression)<-[:node]-(:Reference)<-[:references]-(:Variable)
        -[:declarations]->(declarationToMerge:Declaration)-[:node]->(:BindingIdentifier),

// importer.js: import { exportedName1 } from "exporter";
    (importer:CompilationUnit)-[:contains]->(import:Import)-[:namedImports]->(:ImportSpecifier)
        -[:binding]->(importBindingIdentifierToMerge:BindingIdentifier)<-[:node]-(declarationToDelete:Declaration)
        <-[:declarations]-(importedVariable:Variable)

    WHERE exporter.parsedFilePath CONTAINS import.moduleSpecifier
      AND exportLocalSpecifier.exportedName = importBindingIdentifierToMerge.name

MERGE
    (importedVariable)-[:declarations]->(declarationToMerge)-[:node]->(importBindingIdentifierToMerge)

DETACH DELETE
    declarationToDelete
;
```

FIXPOINT ALGORITHMS

- Lots of propagation algorithms
- „Run to completion" scheduling
  - Mix of Java code and Cypher



EFFICIENT INITIALIZATION

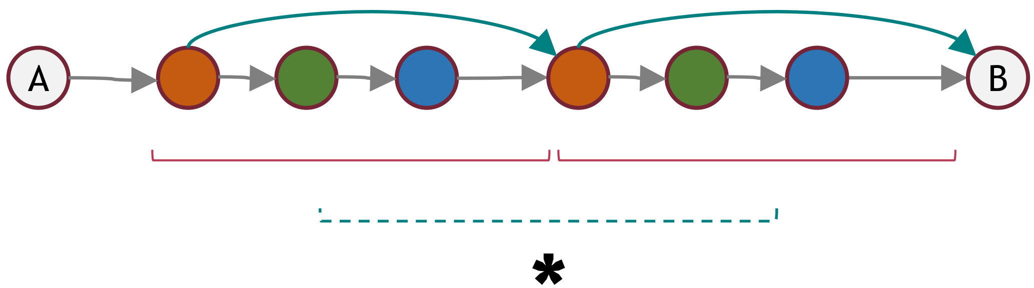
- Initial build of the graph with Cypher was slow
- Generate CSV and bulk load
- Two files: nodes, relationships

```
$NEO4J_HOME/bin/neo4j-admin import
  --database=db
  --nodes=nodes.csv
  --relationships=relationships.csv
```

- 10× speedup

REGULAR PATH QUERIES

- Transitive closure on certain combinations
- Workaround:
  - Start transaction
  - Add proxy relationships
  - Calculate transitive closure
  - Rollback transaction



- openCypher proposal for *path patterns*

```
(:A)-/[:R1 :R2 :R3]+/->(:B)
```

INCREMENTAL QUERIES

OPENCYPHER SYSTEMS

openCypher

- „The openCypher project aims to deliver a full and open specification of the industry's most widely adopted graph database query language: Cypher." (late 2015)
- Research prototypes
  - Graphflow (Univesity of Waterloo)
  - ingraph (incremental graph engine)

incremental processing

(Source: Keynote talk @ GraphConnect NYC 2017)

FOSDEM 2017: INGRAPH

Incremental Graph Queries with openCypher

Gábor Szárnyas

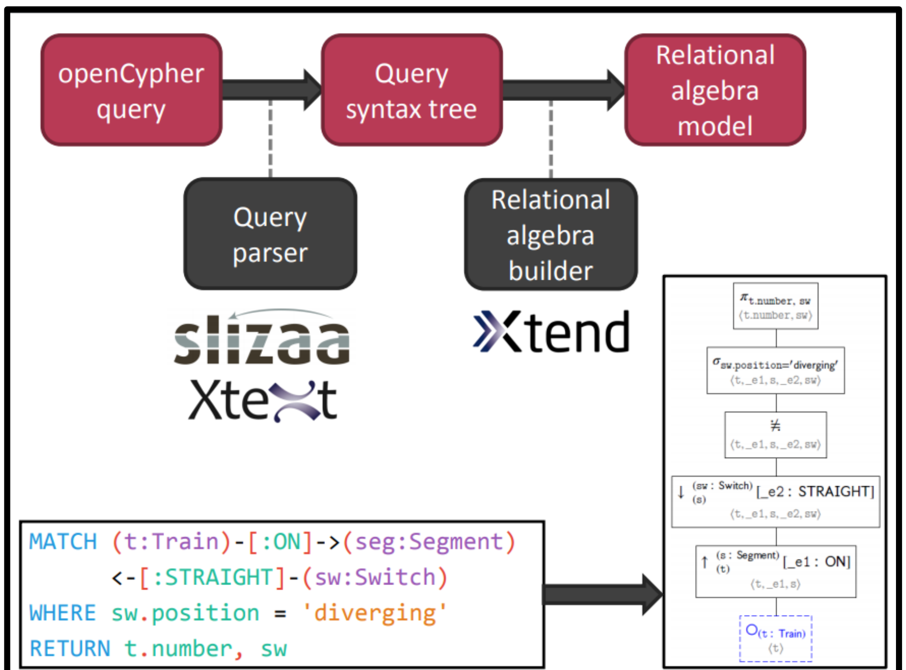
GraphDevroom @ FOSDEM 2017

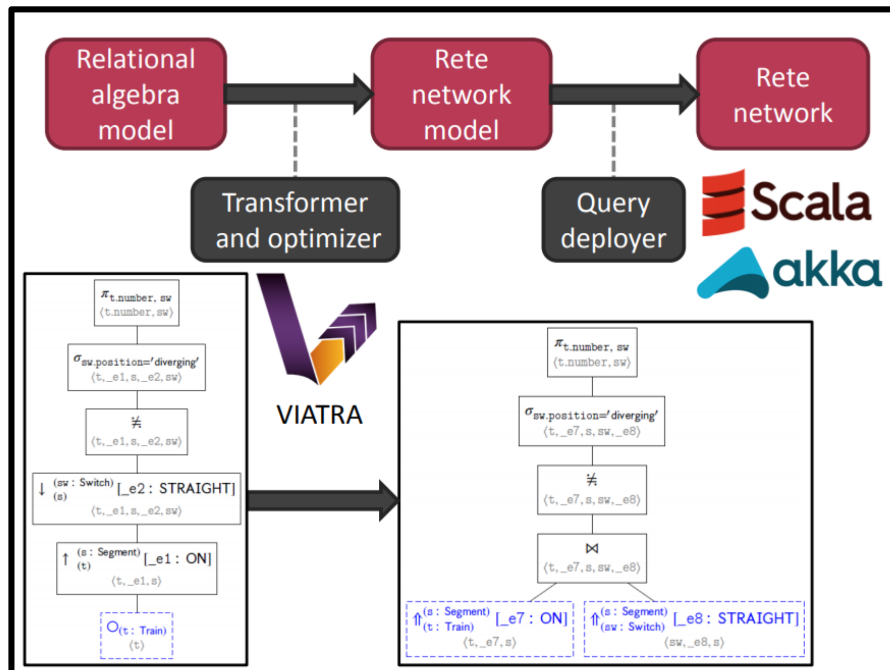
Budapest University of Technology and Economics
McGill University, Montréal

McGill

Mapping openCypher to relational algebra

Combining and filtering pattern matches	
`MATCH «p»`	$\not\equiv_{\text{edges of } p}(p)$
`MATCH «p1», «p2»`	$\not\equiv_{\text{edges of } p1 \text{ and } p2}(p_1 \bowtie p_2)$
`MATCH «p1»` `MATCH «p2»`	$\not\equiv_{\text{edges of } p1}(p_1) \bowtie \not\equiv_{\text{edges of } p2}(p_2)$
`MATCH «p1»` `OPTIONAL MATCH «p2»`	$\not\equiv_{\text{edges of } p1}(p_1) ⟕ \not\equiv_{\text{edges of } p2}(p_2)$
`MATCH «p»` `WHERE «condition»`	$\sigma_{\text{condition}}(r)$, where condition may specify patterns and arithmetic constraints on existing variables
Result and sub-result operations. Rules for `RETURN` also apply to `WITH`.	
`RETURN «variables»`	$\pi_{\text{variables}}(r)$
`RETURN «v1» AS «alias1» ...`	$\pi_{\text{v1}\to\text{alias1},\ldots}(r)$
`RETURN DISTINCT «variables»`	$\delta\left(\pi_{\text{variables}}(r)\right)$
`RETURN «variables», «aggregates»`	$\gamma_{\text{variables,aggregates}}(r)$
List operations	
`ORDER BY «v1» [ASC|DESC] ...`	$\tau_{\uparrow/\downarrow \text{v1},\ldots}(r)$
`LIMIT «l»`	$\lambda_l(r)$





STATE OF INGRAPH IN 2018

- Cover a substantial fragment of openCypher
  - `MATCH`, `OPTIONAL MATCH`, `WHERE`
  - `WITH`, functions, aggregations
  - `CREATE`, `DELETE`
- Features on the roadmap
  - `MERGE`, `REMOVE`, `SET`
  - List comprehensions

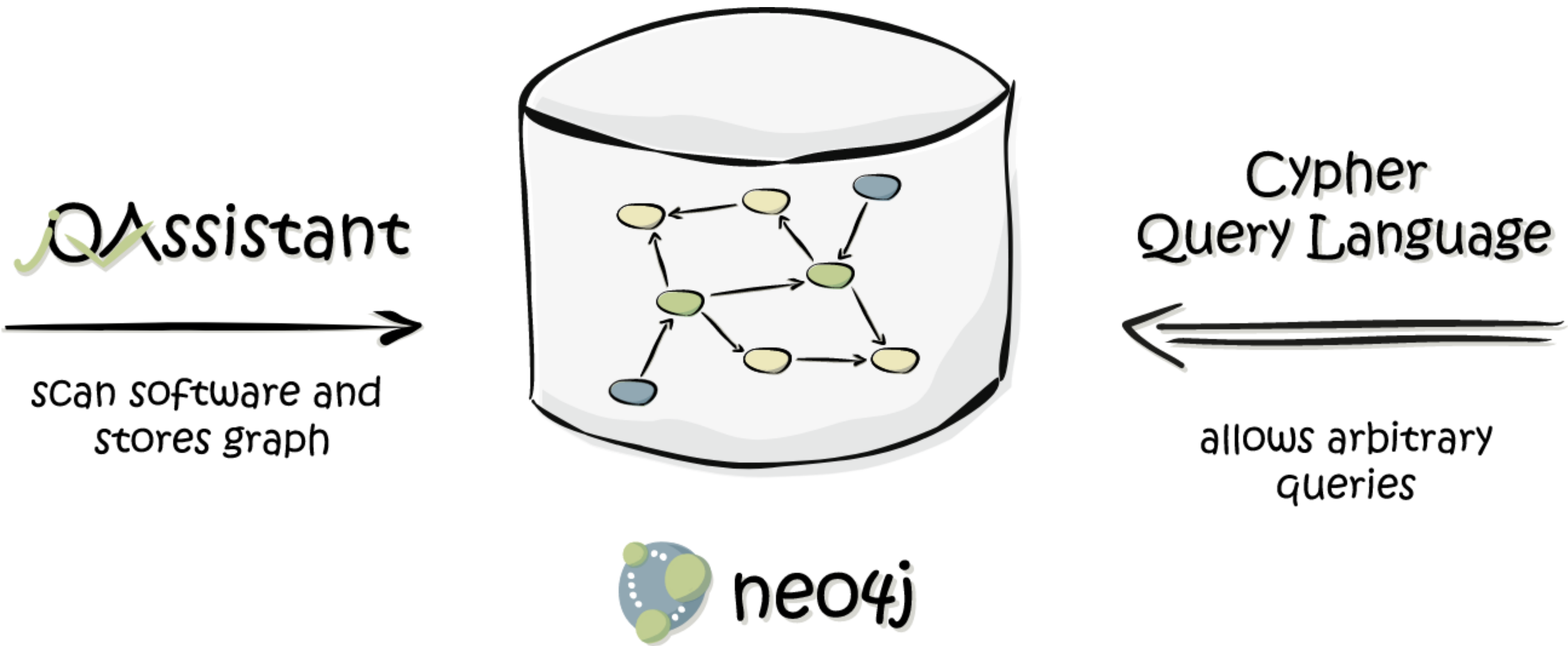
J. Marton, G. Szárnyas, D. Varró:
Formalising openCypher Graph Queries in Relational Algebra,
ADBIS, Springer, 2017

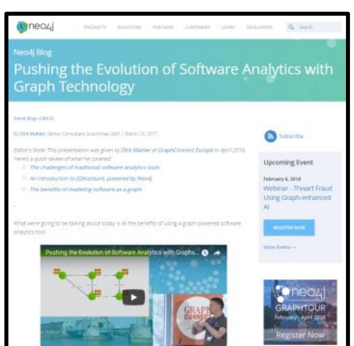
G. Szárnyas:
Incremental View Maintenance for Property Graph Queries,
SIGMOD SRC, 2018

RELATED PROJECTS

JQASSISTANT

Code comprehension: software to graph

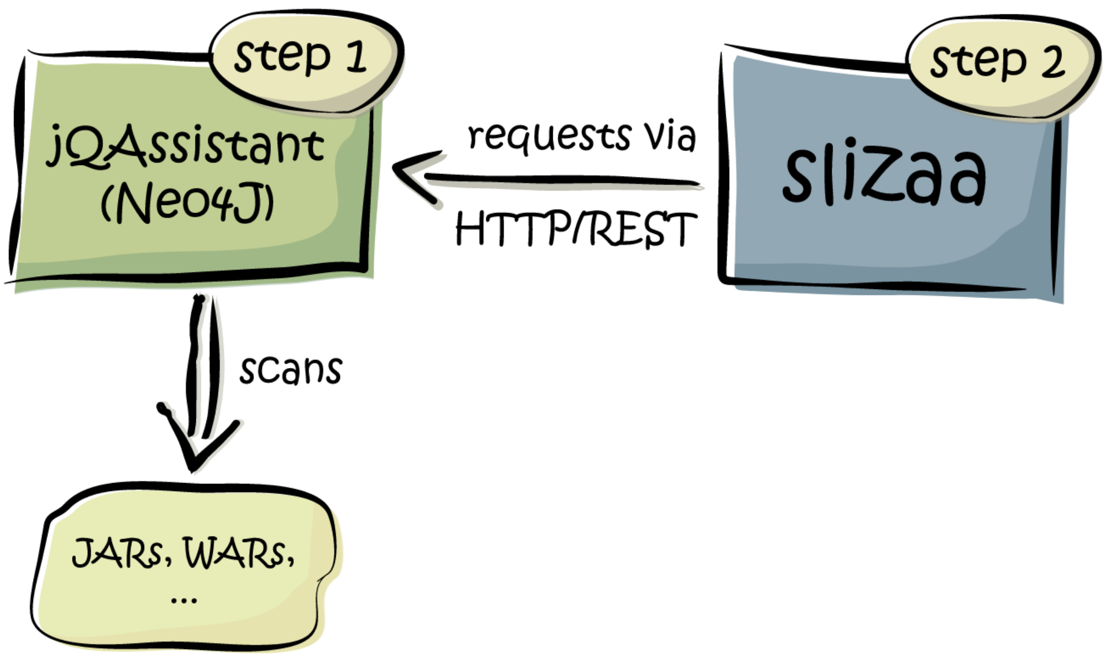


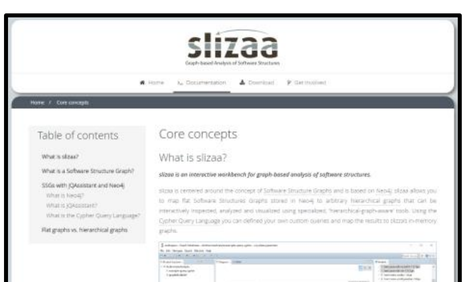


Dirk Mahler,
Pushing the evolution of software analytics with graph technology,
Neo4j blog, 2017

SLIZAA

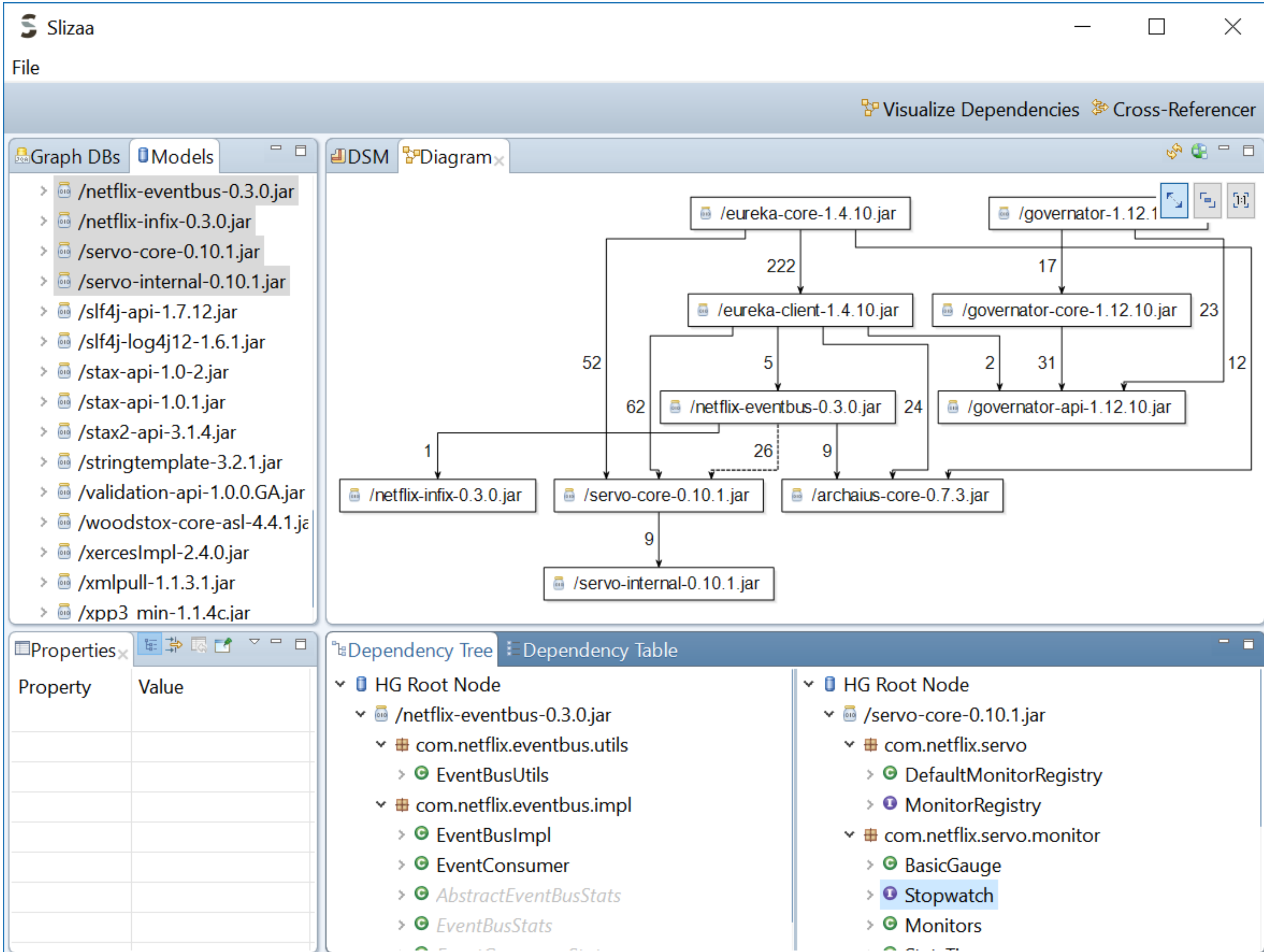
slizaa uses Neo4j/jQAssistant and provides a front end with a bunch of specific tools and viewers to provide an easy-to-use in-depth insight of your software's architecture.





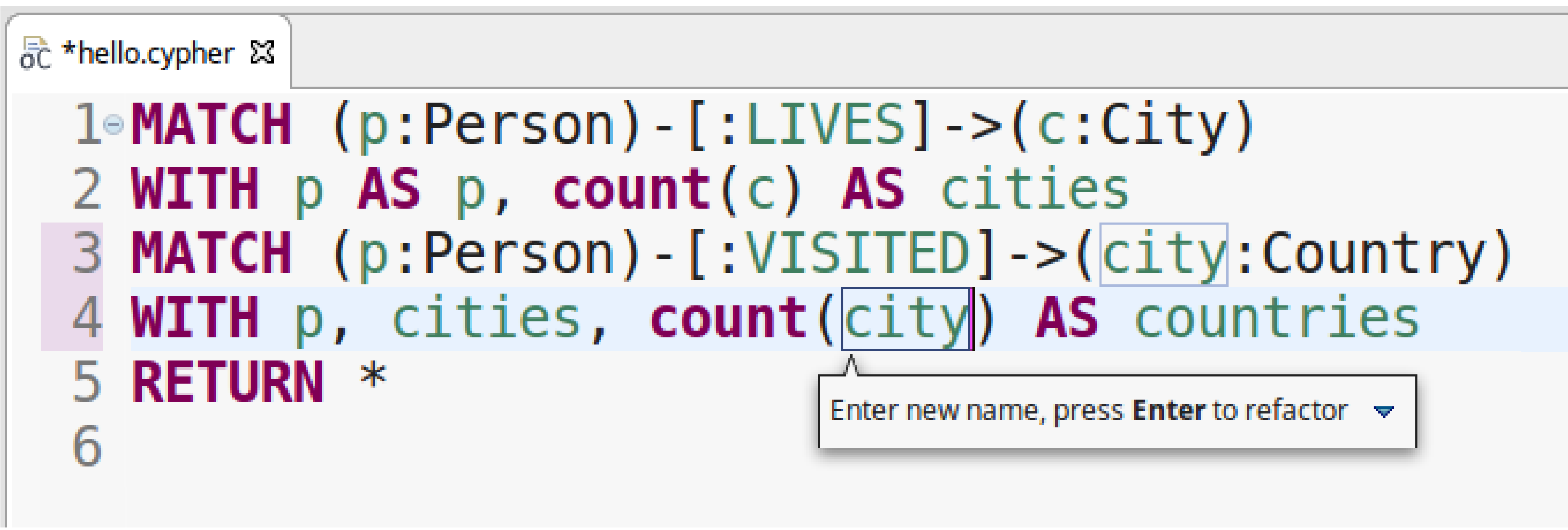
Gerd Wütherich, *Core concepts*, slizaa

SLIZAA: ECLIPSE IDE



SLIZAA: XTEXT OPENCYPHER

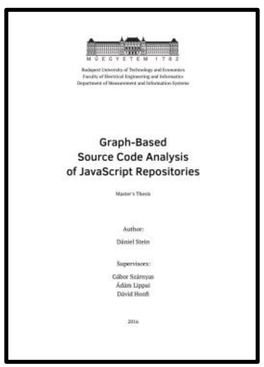
- Xtext-based grammar
- Used in the ingraph compiler
- Now has a scope analyzer
- Works in the Eclipse IDE and web UI



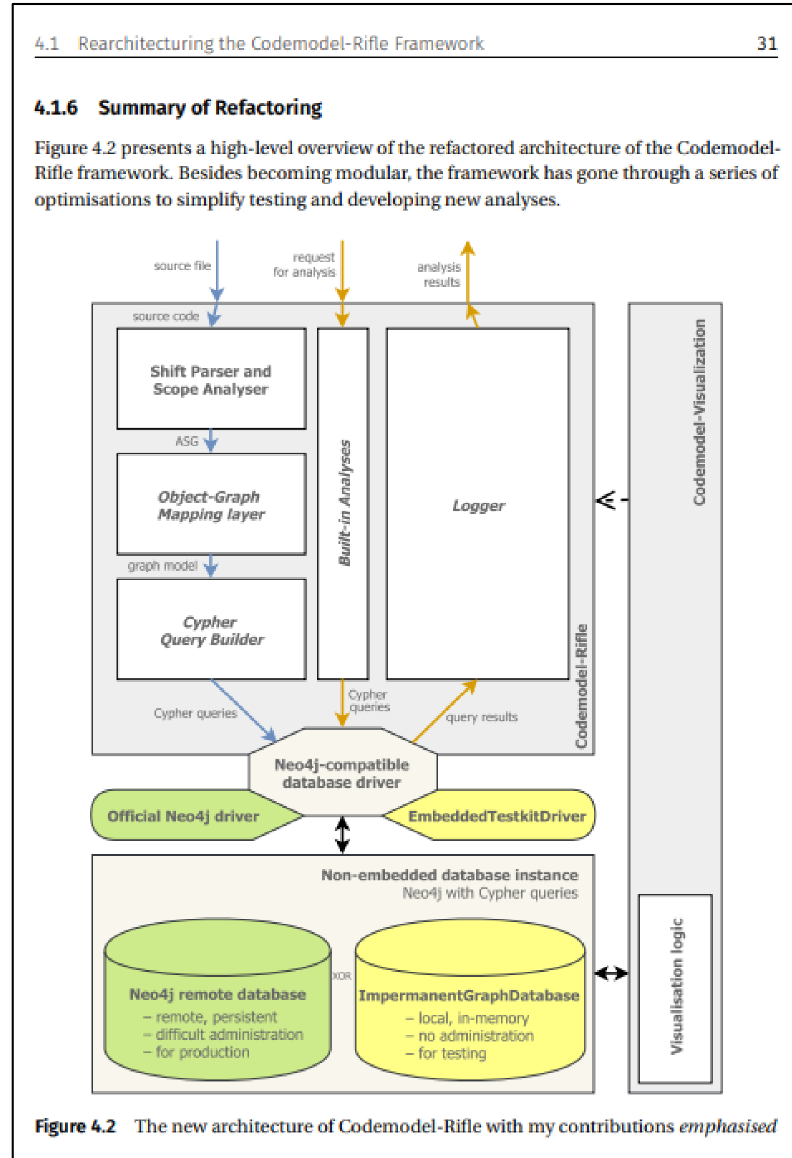
WRAPPING UP

PUBLICATIONS

Dániel Stein:
Graph-based source code analysis of JavaScript repositories,
Master’s thesis, 2016



Soma Lucz:
Static analysis algorithms for JavaScript,
Bachelor’s thesis, 2017


4.1 Rearchitecturing the Codemodel-Rifle Framework 31

4.1.6 Summary of Refactoring

Figure 4.2 presents a high-level overview of the refactored architecture of the Codemodel-Rifle framework. Besides becoming modular, the framework has gone through a series of optimisations to simplify testing and developing new analyses.

Figure 4.2 The new architecture of Codemodel-Rifle with my contributions *emphasised*

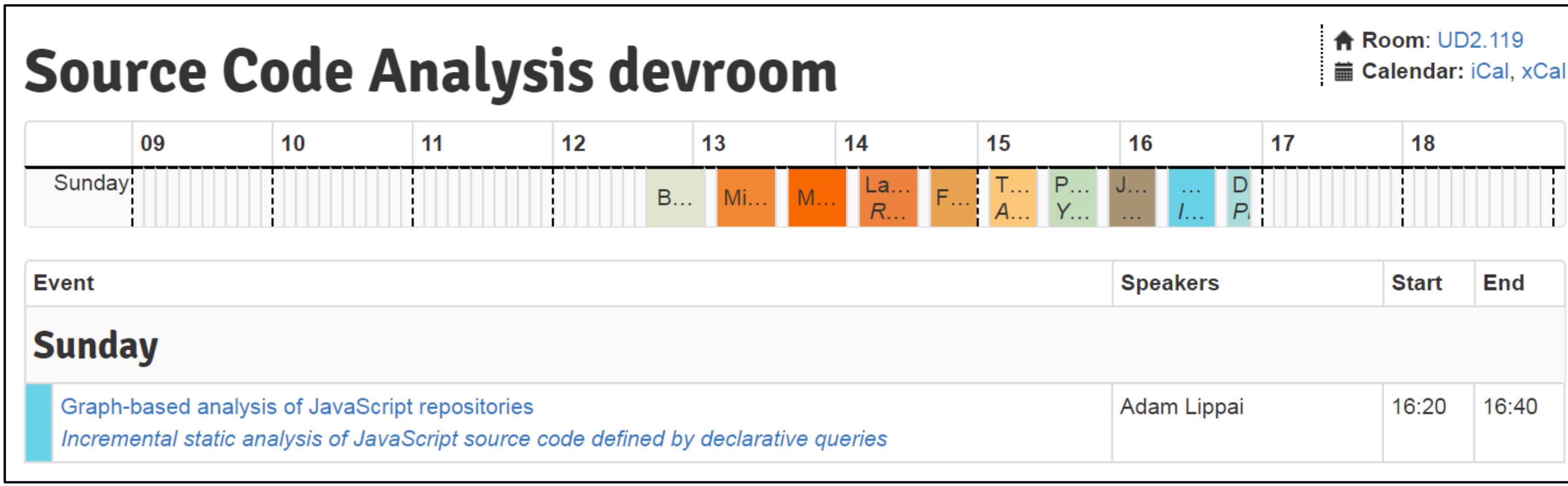
CONCLUSION

- Some interesting analysis rules require a global view of the code
- Good use case for graph databases
  - Property graph
  - Cypher language
- Very good use case for incremental queries
  - Incrementality on multiple levels

RELATED RESOURCES

Codemodel-Rifle	github.com/ftsrg/codemodel-rifle
ingraph engine	github.com/ftsrg/ingraph
Shape Security's Shift parser	github.com/shapesecurity/shift-java
Slizaa openCypher Xtext	github.com/slizaa/slizaa-opencypher-xtext

Thanks to Ádám Lippai, Soma Lucz, Dániel Stein, Dávid Honfi and the ingraph team.



Ω

VISUAL STUDIO CODE INTEGRATION

- Language Server Protocol (LSP) allows portable implementation

```
function jpegParsingError(code) {
  return {
    type: 'JpegParsingError',
    code: code
  };
}
This function is never used or not accessible.
(parameter) len: any
function buf_ensure(bytes, ptr, len) {
  if (!buf_has(bytes, ptr, len)) throw jpegParsingError('RangeOutOfBounds');
}

function buf_has(bytes, ptr, len) {
  return ptr + len < bytes.length;
}
```

USE CASES

- Control Flow Graph
  - graph representation of
  - every possible statement sequence
- Basis for type inferencing and test generation