



FRAPPÉ

Querying and managing evolving code dependency graphs

David Meibusch & Nathan Hawes

Oracle Labs Australia
June 2016

ORACLE



Safe Harbour

The following is intended to provide some insight into a line of research in Oracle Labs. It is intended for information purposes only, and may not be incorporated into any contract.

It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. Oracle reserves the right to alter its development plans and practices at any time, and the development, release, and timing of any features or functionality described in connection with any Oracle product or service remains at the sole discretion of Oracle.

Any views expressed in this presentation are my own and do not necessarily reflect the views of Oracle.

```
The input file name and line number are omitted if the shell
currently interactive. If the shell is not currently intera
the input file name is inserted only if it is different from
shell name. */
d
defined (PREFER_STDARG)
ser_error (int lineno, const char *format, ...)
se
ser_error (lineno, format, va_alist)
int lineno;
const char *format;
va_dcl
dif

a_list args;
i
Pattern/action structure for CASE_COM. */
#define struct pattern_list {
struct pattern_list *next; /* Clause to try in case this one
ORD_LIST *patterns; /* Linked list of patterns to test. *
COMMAND *action; /* Thing to execute if a pattern matches.
at flags;
PATTERN_LIST *patterns;

the CA
#define struct case_com {
at flags;
or li
ORD_DESC *word; /* The thing to test.
PATTERN_LIST *clauses; /* The clauses to test against, or NU
CASE_COM;

Spaces: 2
Define if you have a working `mmap' system call. */
#define HAVE_MMAP 1

Define if you have the `munmap' function. */
#define HAVE_MUNMAP 1

Define if you have the `nl_langinfo' function. */
#undef HAVE_NL_LANGINFO */

Define if you have the `copy' function. */
212 COND_COM *cond;
213 {
214 if (cond)
215 {
216 if (cond->left)
217 dispose_cond_node (cond->left);
218 if (cond->right)
219 dispose_cond_node (cond->right);
220 if (cond->op)
221 dispose_word (cond->op);
222 free (cond);
223 }
224 }
225 #endif /* COND_COMMAND */
226
227 void
228 dispose_function_def_contents (c)
229 FUNCTION_DEF *c;
230 {
231 dispose_word (c->name);
232 dispose_command (c->command);
233 FREE (c->source_file);
234 }
235
236 void
237 dispose_function_def_contents (c)
238 FUNCTION_DEF *c;
239 {
240 dispose_word (c->name);
241 dispose_command (c->command);
242 FREE (c->source_file);
243 }
244 /* How to free a WORD_DESC. */
245 void
246 dispose_word (w)
247 WORD_DESC *w;
248 {
249 FREE (w->word);
250 ocache_free (wdcache, WORD_DESC, w);
251 }
252
253 /* Free a WORD_DESC, but not the word conta
254 void
255 dispose_word_desc (w)
256 WORD_DESC *w;
65 {
66 hash_flush (hash, 0);
67 }
68
69 int
70 assoc_insert (hash, key, value)
71 HASH_TABLE *hash;
72 char *key;
73 char *value;
74 {
75 BUCKET_CONTENTS *b;
76
77 b = hash_search (key, hash, HASH_CREATE);
78 if (b == 0)
79 return -1;
80 /* If we are overwriting an existing element's val
81 use the key. Nothing in the array assignment co
82 string, so we can free it here to avoid a memory
83 if (b->key != key)
84 free (key);
85 FREE (b->data);
86 b->data = value ? savestring (value) : (char *)0;
87 return (0);
88 }
89
90 /* Like assoc_insert, but returns b->data instead of
91 PTR_T
92 assoc_replace (hash, key, value)
93 HASH_TABLE *hash;
94 char *key;
95 char *value;
96 {
97 BUCKET_CONTENTS *b;
98 PTR_T t;
99
100 b = hash_search (key, hash, HASH_CREATE);
101 if (b == 0)
102 return (PTR_T)0;
103 /* If we are overwriting an existing element's val
104 use the key. Nothing in the array assignment co
105 string, so we can free it here to avoid a memory
106 if (b->key != key)
107 free (key);
108 t = b->data;
109 b->data = value ? savestring (value) : (char *)0;
```

The Truth is in the Source!

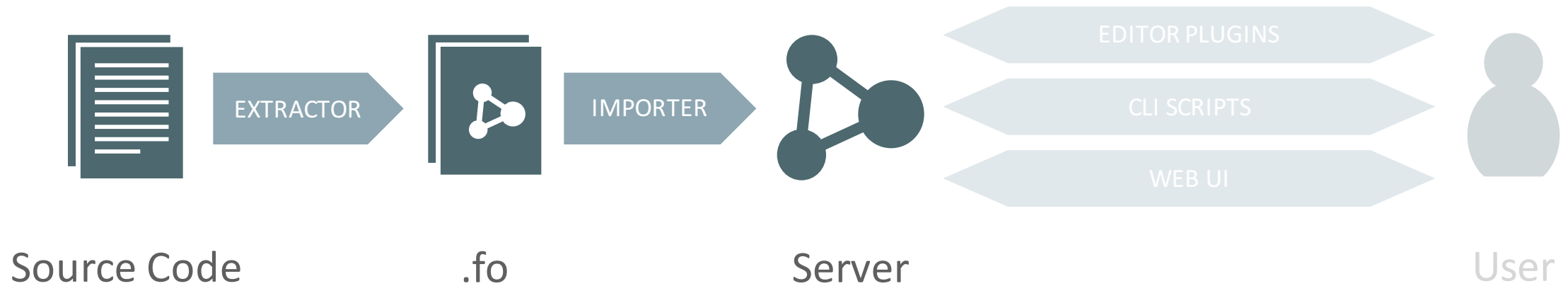
But the source is often complicated, multi-language and really really big



Frappé: code as a property graph

- Graph natural in this domain
 - Call graph, directory hierarchy, type hierarchy, data/control flow graphs
- Overlay data from different spaces
 - File system, build, preprocessor, AST, cross-language
- Lets users specify queries in terms of this graph

How it works



Graph model example

```
int bar(int);
```

foo.h

```
#include "foo.h"  
int bar(int *input) {  
    return *input * 2;  
}
```

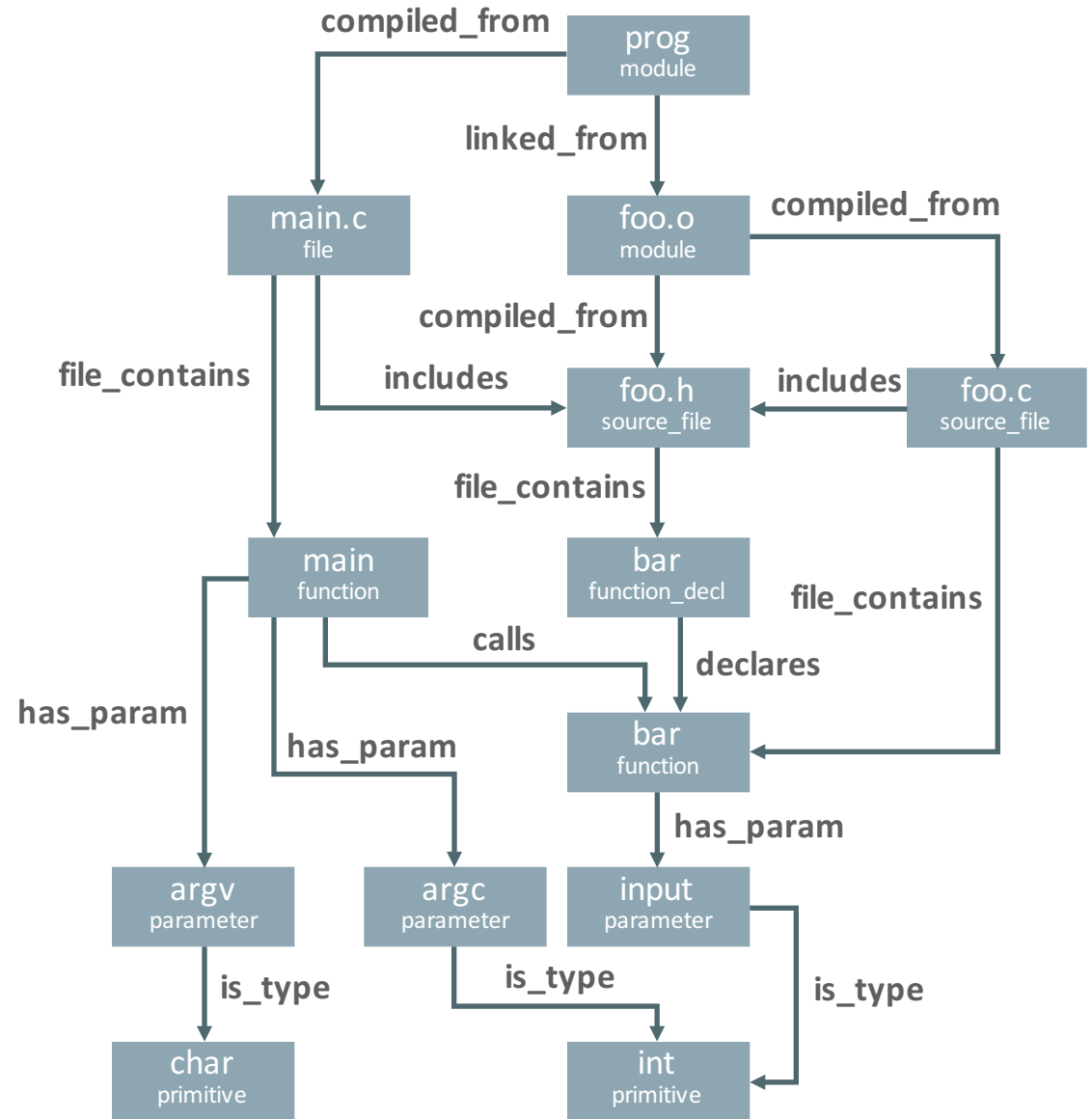
foo.c

```
#include "foo.h"  
int main(int argc, char **argv) {  
    return bar(&argc);  
}
```

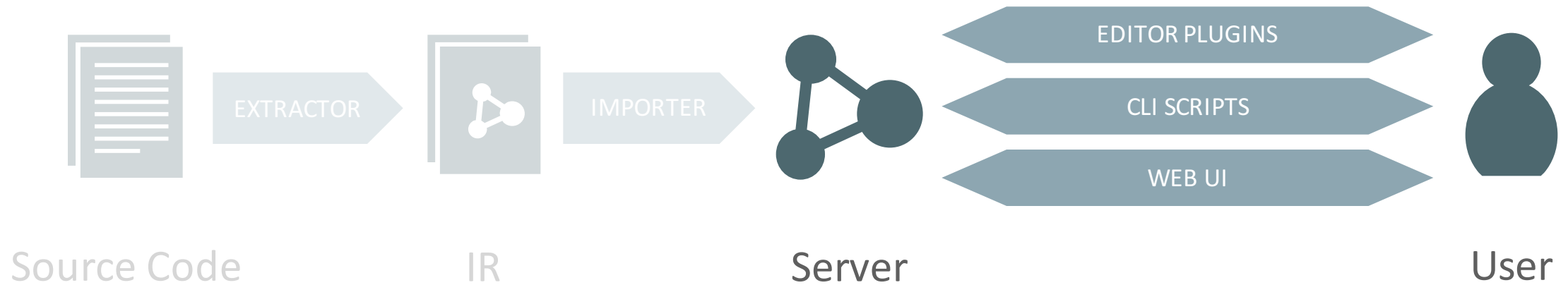
main.c

```
gcc foo.c -c -o foo.o  
gcc main.c foo.o -o prog
```

build



How it works



Use cases

- Code search
- Code navigation
 - Go to definition
 - Find references
- Code comprehension
 - Visualization
 - Transitive closure calls, includes, etc.
 - Shortest path queries

How it looks

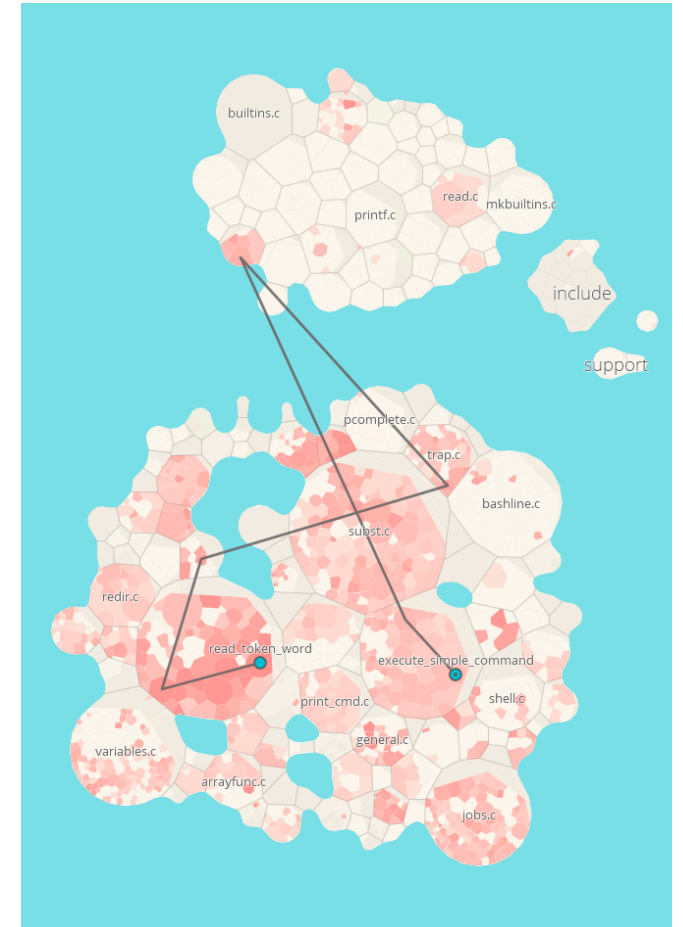
```
357
358 /* Return the working directory for the current
359 job_working_directory, this does not call
360 of the functions it calls. This is so that
361 from a signal handler. */
362 static char *
363 current_working_directory ()
364 {
365     char *dir;
366     static char d[PATH_MAX];
367
368     dir = get_string_value ("PWD");
369
370     if (dir == 0 && the_current_working_directory
371         dir = the_current_working_directory;
372
373     if (dir == 0)
374     {
375         dir = getcwd (d, sizeof(d));
376         if (dir)
377             dir = d;
378     }
379 }
```

```
jobs.c
References to "current_working_directory" (node
declared by Function... current_working...
called by Function print_pipeline at jobs.c
called by Function start_job at jobs.c
called by Function notify_of_job_s... at jobs.c
called by Function notify_of_job_s... at jobs.c
~
~
~
```

```
F rl_rubout
lib/readline/text.c

1058 /* Rubout the character behind point. */

1059 int
1060 rl_rubout (count, key)
1061     int count, key;
1062 {
1063     if (count < 0)
1064         return (rl_delete (-count, key));
1065
1066     if (!rl_point)
1067     {
1068         rl_ding ();
1069         return -1;
1070     }
1071
1072     if (rl_insert_mode == RL_IM_OVERWRITE)
1073         return (_rl_overwrite_rubout (count, key));
1074
1075     return (_rl_rubout_char (count, key));
1076 }
```



Use cases

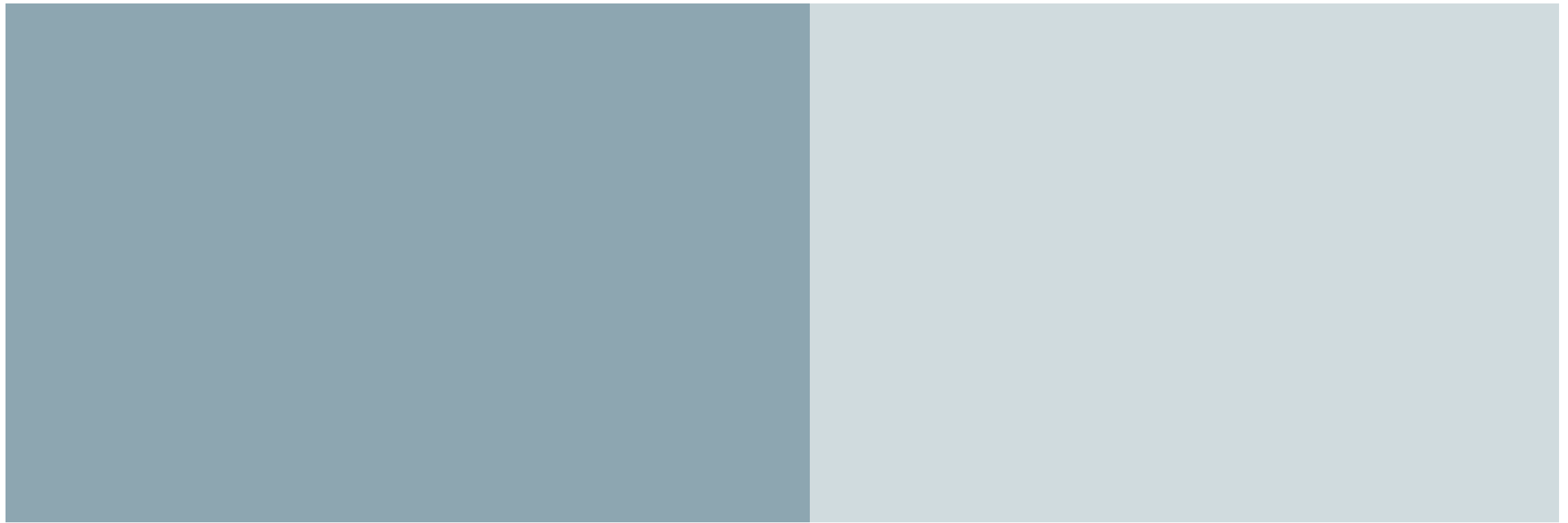
- Code search
- Code navigation
 - Go to definition
 - Find references
- Code comprehension
 - Visualization
 - Transitive closure calls, includes, etc.
 - Reachability queries

Queries and Neo4j performance
detailed in GRADES'15 paper:

“Frappé: Querying the Linux Kernel
dependency graph”

Code search

 foo.c



Code search

 foo.c

```
(n with name='foo.c')
```

Code search

🔍 foo.c

(n with name='foo.c')

```
struct foo {  
    int c;  
}
```

Code search

🔍 foo.c

```
(c with name='foo')  
  -[:contains]->  
  (n with name='c')  
    UNION  
  (n with name='foo.c')
```

```
struct foo {  
    int c;  
}
```

Code search

🔍 foo.c

```
(c with name='foo')  
  -[:contains]->  
  (n with name='c')  
      UNION  
(n with name='foo.c')
```

```
struct bar {  
    int c;  
}  
  
typedef struct bar foo;
```

Code search

🔍 foo.c

```
(c with name 'foo')  
-[:aliases*]->()-[:contains]->  
  (n with name='c')  
      UNION  
  (n with name='foo.c')
```

```
struct bar {  
    int c;  
}  
  
typedef struct bar foo;
```


Code search

🔍 foo.c

```
(c with name 'foo')  
-[:aliases*]->()-[:contains]->  
  (n with name='c')  
      UNION  
  (n with name='foo.c')
```

```
struct bar {  
    int c;  
}  
  
typedef struct bar foo;
```

Use cases

- Code search
- Code navigation
 - Go to definition
 - Find references
- Code comprehension
 - Visualization
 - Transitive closure calls, includes, etc.
 - Reachability queries

Queries and Neo4j performance
detailed in GRADES'15 paper:

“Frappé: Querying the Linux Kernel
dependency graph”

Use cases

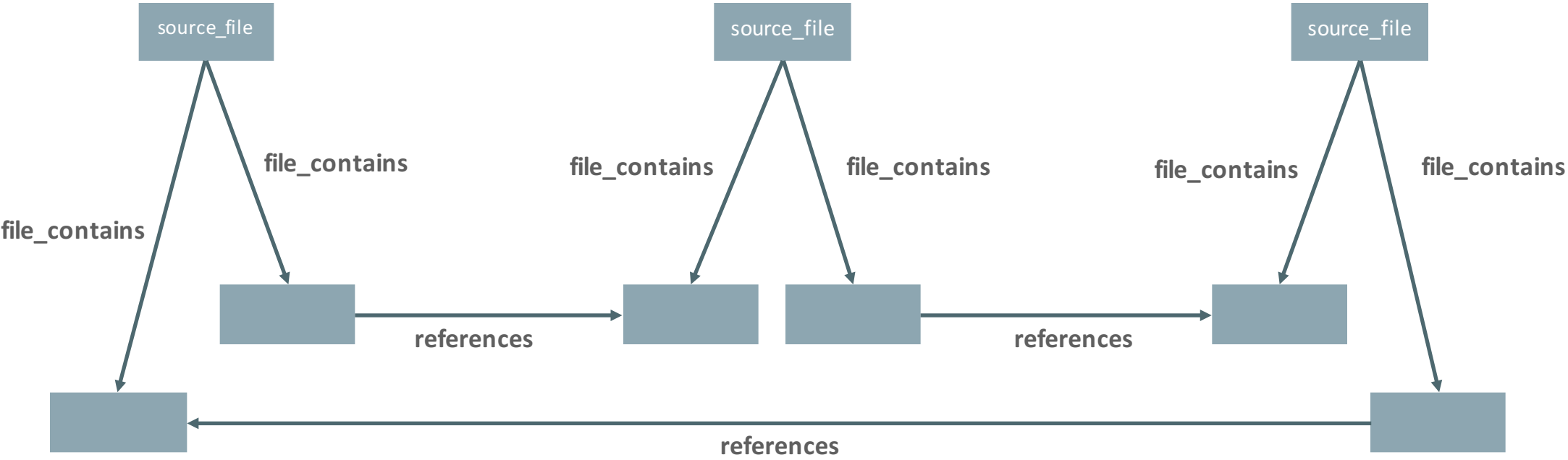
- Code search
- Code navigation
 - Go to definition
 - Find references
- Code comprehension
 - Visualization
 - Transitive closure calls, includes, etc.
 - Reachability queries

- Custom user queries
 - (Anti) pattern detection
 - *Expose query language*

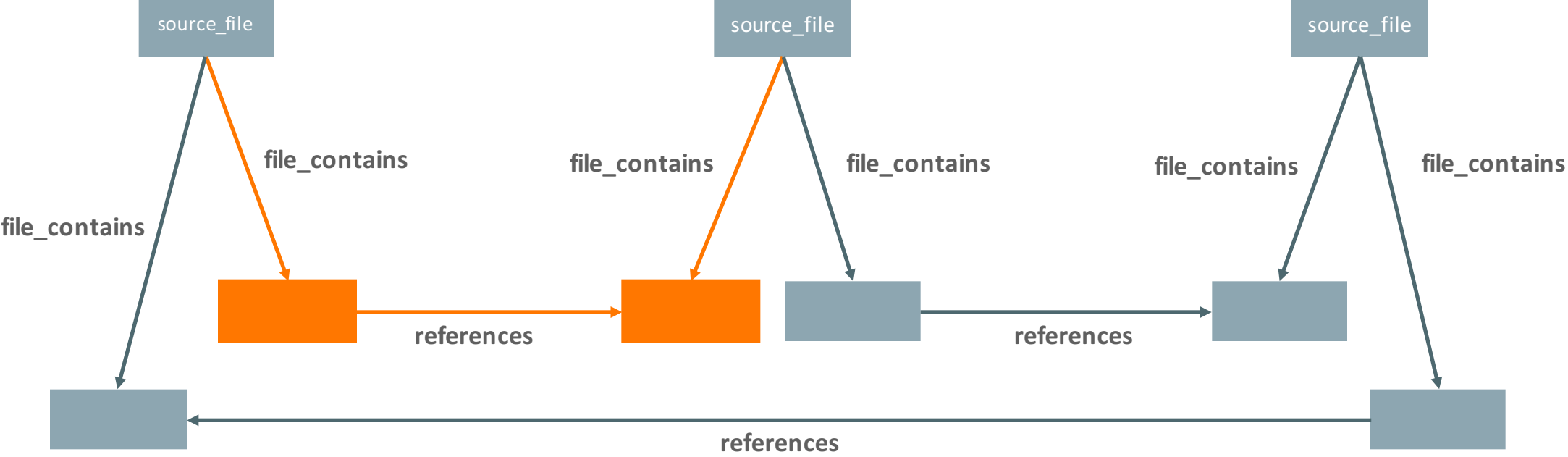
Queries and Neo4j performance
detailed in GRADES'15 paper:

“Frappé: Querying the Linux Kernel
dependency graph”

Dependency cycle



Dependency cycle



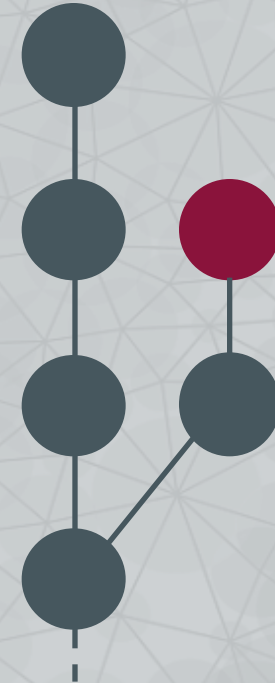
Dependency cycle



```
( )-[
:includes | uses_namespace | expands | interrogates | undefined | aliases |
  uses_enumerator | has_friend | isa_type | extends | uses_type | throws |
has_ret_type | has_param_type | calls | may_call | overridden_by | declares
| reads | writes | dereferences | address_of | type_of | size_of | align_of
| casts_to
]->( )
```

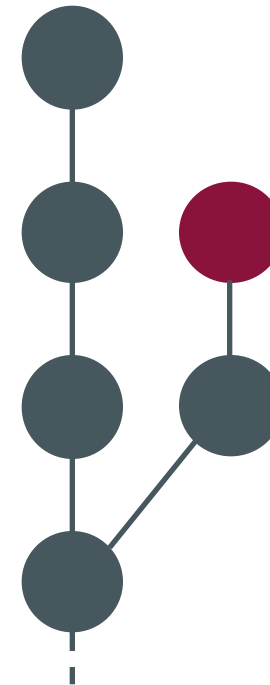
BUT

Developers working off of different versions of the code



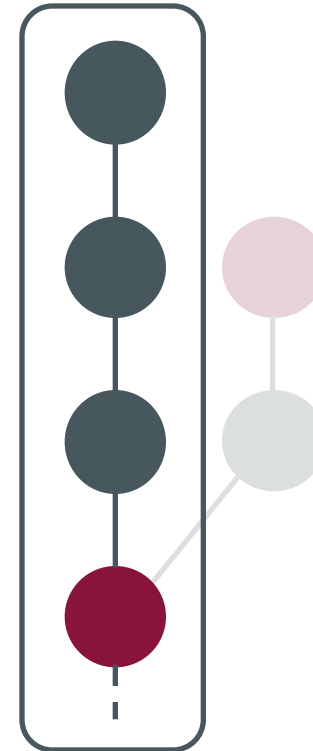
Target Scenario

- 1000s of developers working from main branch
- Changes merged regularly
- Most developers working off of versions from the past **30** days



Current Deployment

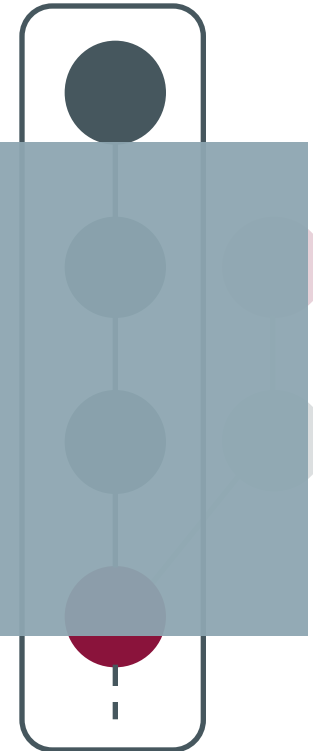
- Separate Neo4j instance for the most recent **5** versions of the code
- New graph generated in nightly regression
- Script on the client to determine which server to connect to
- Deployment effort and complexity
- Inefficient use of resources
 - Redundant data
 - Memory requirements



Current Deployment

- Separate Neo4j instance for the most recent **5** versions of the code
- New graph generated in nightly regression
- Script on the client to determine which server to connect to
- Deployment effort and complexity
- Inefficient use of resources
 - Redundant data
 - Memory requirements

Single logical server
with multiple versions
and efficient storage



Use cases for multiple versions

- Single version use cases per version

Select version

- Code review: 2 versions
 - Are there any architectural constraints being newly violated?
 - Are there any new usages of deprecated methods?
 - Are any methods now unused?

Version selection

```
PATH vcalls := () -[:call WITH 1013 between fromV and toV]-> ()
SELECT path
FROM freebsd
WHERE path =
  (:function WITH name='source', 1013 between fromV and toV)
  -/:vcalls*/->
  (:function WITH name='sink', 1013 between fromV and toV)
```

VS

```
SELECT path
FROM freebsd@1013
WHERE path =
  (:function WITH name='source') -/:calls*/-> (:function WITH name='sink')
```

Use cases for multiple versions

- Single version use cases per version
- Code review: 2 versions
 - Are there any architectural constraints being newly violated?
 - Are there any new usages of deprecated methods?
 - Are any methods now unused?

Match in each version
+
Compare results

Compare results

```
SELECT path
FROM freebsd@1013
WHERE path =
  (:function WITH name='source') -/:calls*/-> (:function WITH name='sink')
```

DIFFERENCE

```
SELECT path
FROM freebsd@1014
WHERE path =
  (:function WITH name='source') -/:calls*/-> (:function WITH name='sink')
```

VS

```
SELECT DIFFERENCE path
FROM freebsd@1013, freebsd@1014
WHERE path =
  (:function WITH name='source') -/:calls*/-> (:function WITH name='sink')
```

Open Questions

- Node/Edge identity
 - Supplied or derived using graph information
- Query language expressiveness
 - Regular path expressions
 - Multiple edge labels
- Efficient storage and querying for multiple graph versions:
 - List or ideally DAG of versions
 - Union, intersection, difference of results from different versions
 - Query language that abstracts away versioning

FreeBSD dataset available on OTN

- Includes graphs of 10.1, 10.2, 10.3 kernel + documentation
- Each graph
 - Extracted from 10M LOC
 - 2 million vertices
 - 10 million edges
- Try it out and get in touch:
 - nathan.hawes@oracle.com
 - david.meibusch@oracle.com
 - ben.barham@oracle.com

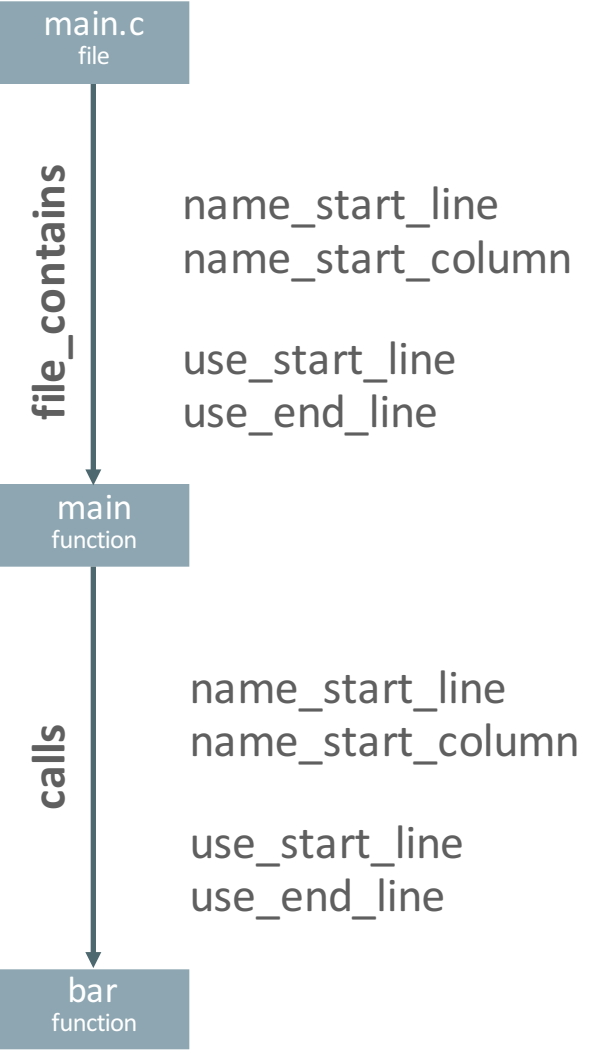
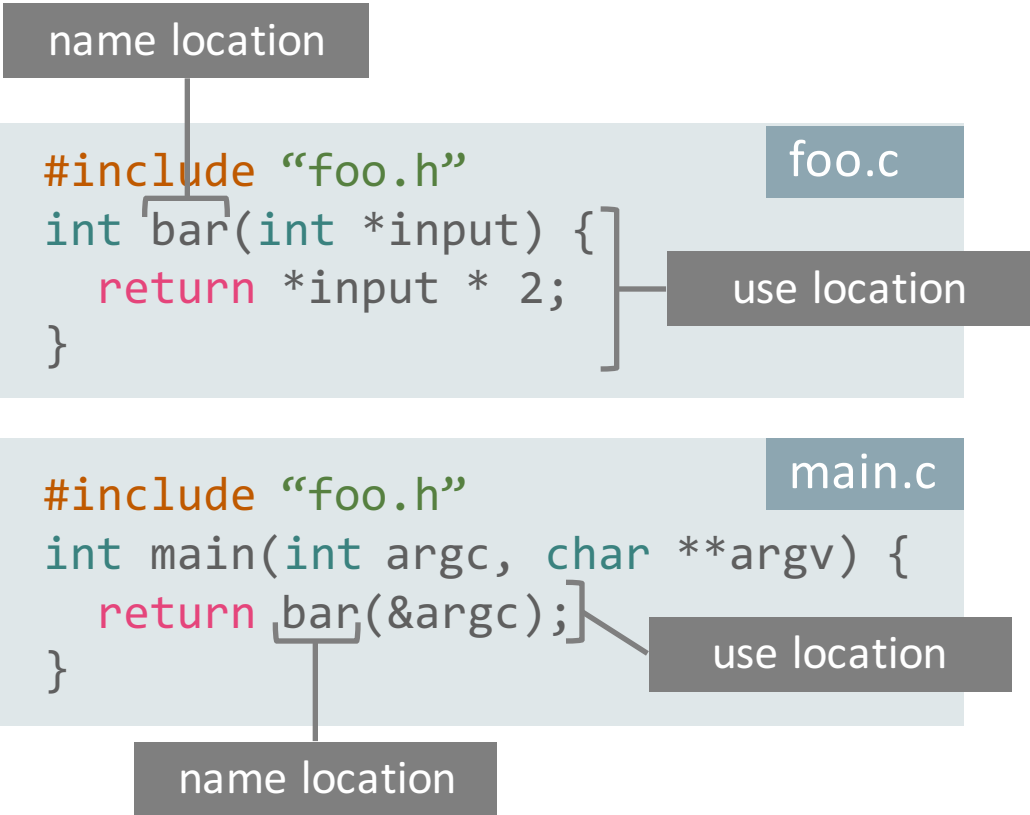
<http://www.oracle.com/technetwork/oracle-labs/datasets/downloads/index.html>

The screenshot shows the Oracle Technology Network (OTN) website. The top navigation bar includes the Oracle logo, a search bar, and links for Account, Sign Out, Help, Country, Communities, I am a..., and I want to... Below this is a secondary navigation bar with links for Products, Solutions, Downloads, Store, Support, Training, and Partners. The main content area is titled "Oracle Technology Network > Oracle Labs > Datasets > Downloads". On the left, there is a sidebar menu with options: Parallel Graph Analytics, Programming Languages and Runtimes, Souffle, and Datasets. The main content area has two tabs: "Overview" and "Downloads". The "Downloads" tab is active, showing the "Oracle Labs Downloads - Datasets" section. This section contains a paragraph explaining that Oracle makes various large data sets available for free, derived from open code sources. It also mentions that the samples are released under the FreeBSD license. Below this is a table with three columns: Name, Description, and Download. The table lists the "Frappe FreeBSD graph data dataset" with a detailed description of the dataset's contents and a download link for "freebsd-10.x-graphs-otn.zip".

Name	Description	Download
Frappe FreeBSD graph data dataset	This dataset includes the dependency graphs of the kernel code of three versions of the FreeBSD operating system: 10.1, 10.2, and 10.3. Each graph has around 2 million nodes and 10 million edges and is provided in the Oracle Big Data Spatial and Graph flat file format. Nodes in the graph correspond to entities in the source code (e.g. functions, structs, macros, variables, parameters, files, directories and executables) and edges to the references and containment relationships between them (e.g. calls, reads, writes, expands, compiled from, and contains). Both are labelled with type information and properties that provide their location in the source code, qualifiers, and more. See the documentation included in the distribution for details.	freebsd-10.x-graphs-otn.zip

ORACLE®

Graph model example



Graph model example

```
#define BAR(A) bar(A)
int bar(int);
```

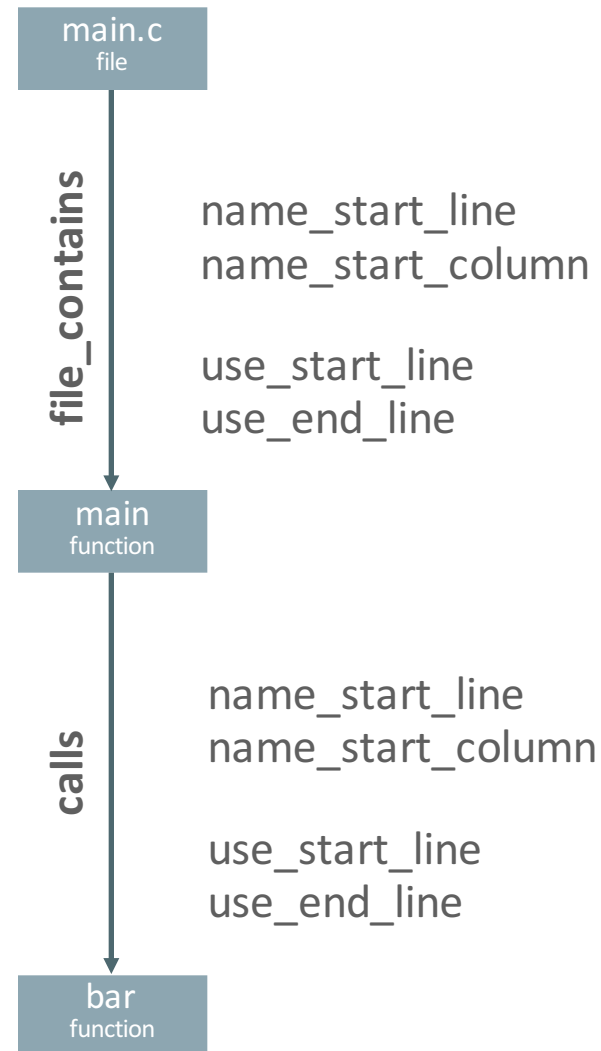
foo.h

```
#include "foo.h"
int bar(int *input) {
    return *input * 2;
}
```

foo.c

```
#include "foo.h"
int main(int argc, char **argv) {
    return BAR(&argc);
}
```

main.c



Graph model example

```
#define BAR(A) bar(A)
int bar(int);
```

foo.h

name location

```
#include "foo.h"
int bar(int *input) {
    return *input * 2;
}
```

foo.c

```
#include "foo.h"
int main(int argc, char **argv) {
    return BAR(&argc);
}
```

main.c

use location

