

Static, Lightweight Includes Resolution for PHP

Mark Hills, Paul Klint, and Jurgen J. Vinju

29th IEEE/ACM International Conference on Automated Software Engineering
September 17-19, 2014

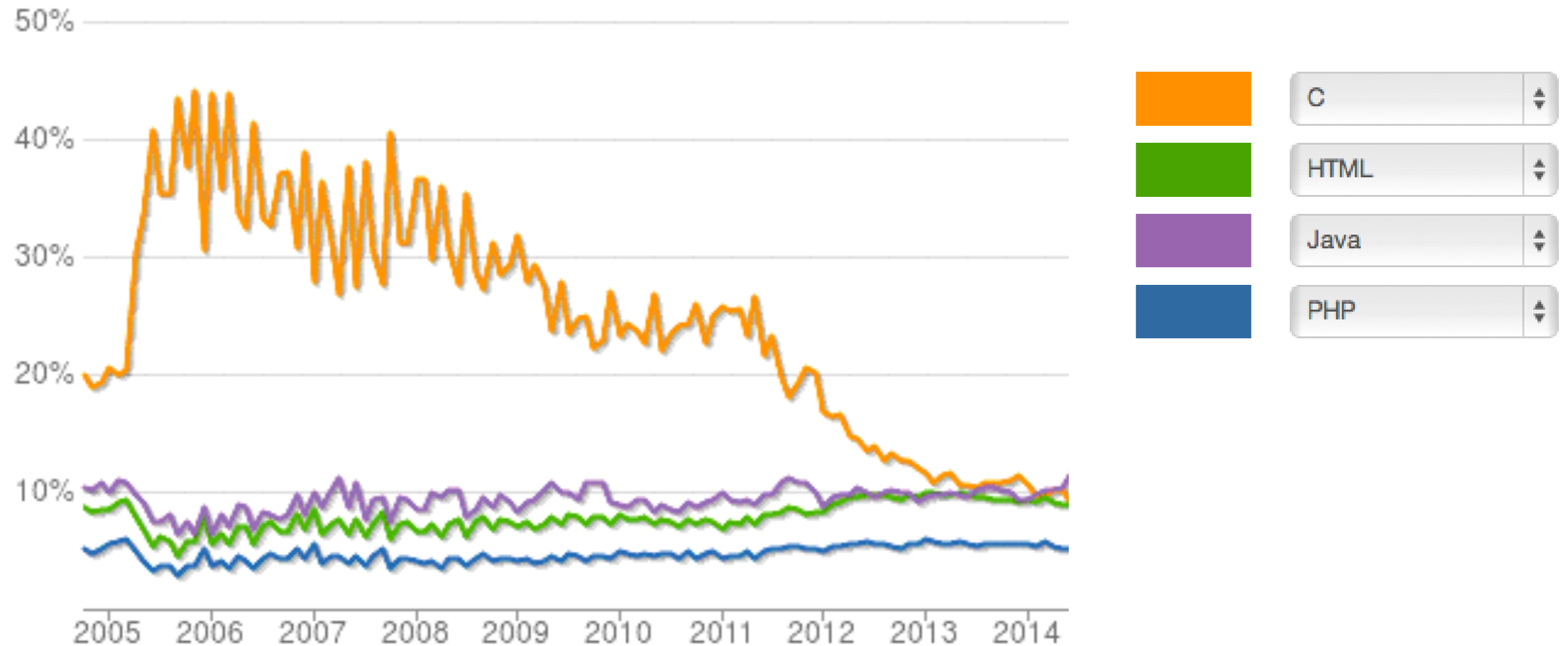
Västerås, Sweden



Motivating stats on PHP

- #7 on TIOBE Programming Community Index
- 4th most popular language on GitHub by repositories created
- Used by 82.2% of all websites whose server-side language can be determined
- Some figures show up to 20% of new sites run WordPress
- Big projects: MediaWiki 1.22.0 has more than 1 million lines of PHP

Open Source Commits by Language (Ohloh.net)



<http://www.ohloh.net/languages/compare?measure=commits&percent=true>

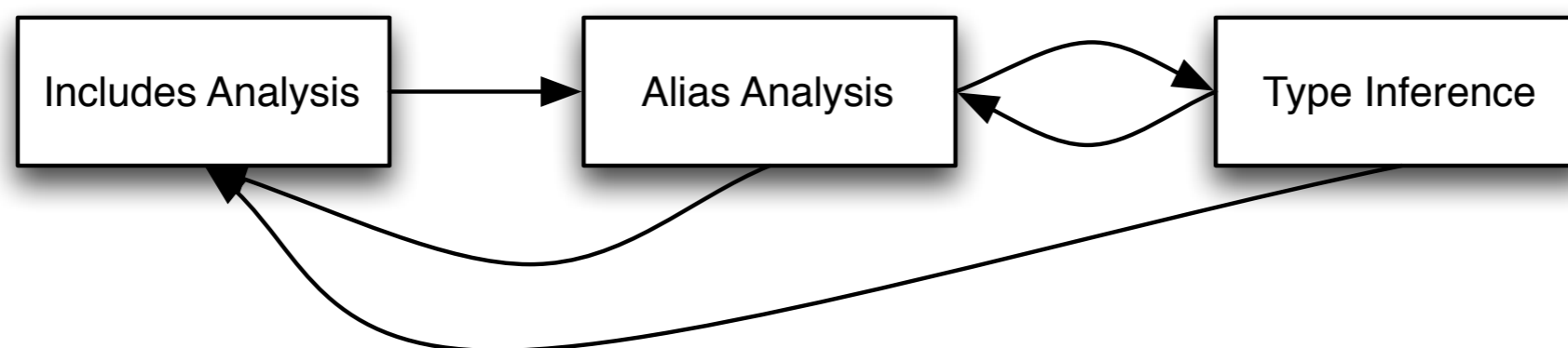
An Empirical Study of PHP Feature Usage (ISSTA 2013)

- Research questions:
 - How do people actually use PHP?
 - What assumptions can we make about code and still have precise analysis in practice?
- One finding: include expressions have a high impact on creating precise program analysis algorithms, and are a common feature



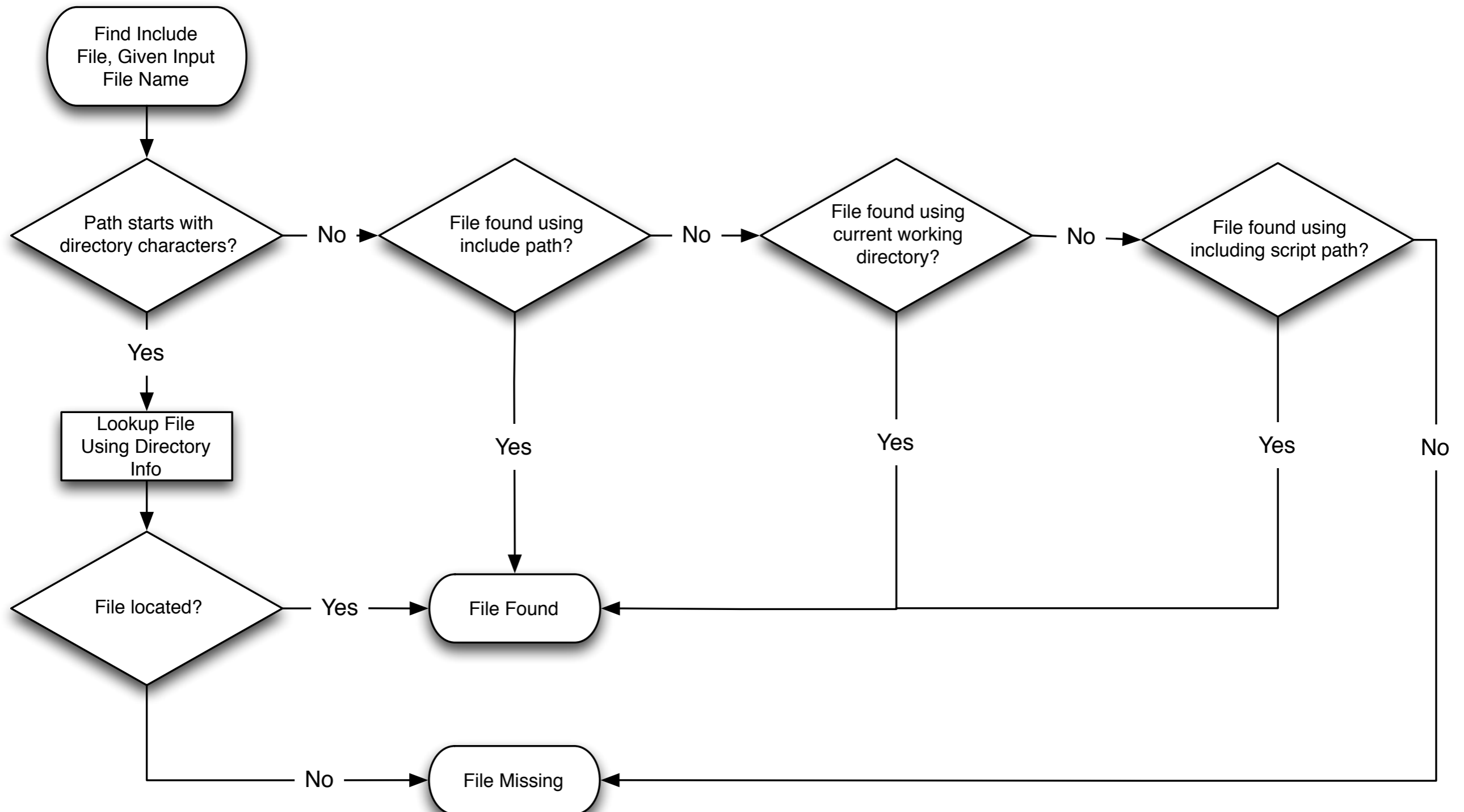
Research Questions

- Can we devise precise, lightweight static analysis algorithms for resolving PHP include expressions?
- Can we provide support that is fast enough to realistically integrate with IDEs?
- How far can we get without applying heavier-weight analysis, with assumption that these results can be refined in the future?





The (non-trivial) PHP File Inclusion Model



What are the challenges?



- Include expression may include concatenation, constants, function calls, or even arbitrary code
- Location to load file *from* may not be obvious:
 - Is it on the include path?
 - Is it based on the current working directory?
 - Is it based on the script directory?
 - Are the first two changed at runtime?

Statically resolving PHP includes: FLRES and PGRES

- FLRES: **F**ile-**L**evel Includes **RES**olution
- PGRES: **P**ro**G**ram-Level Includes **RES**olution
- Why two?
 - PGRES can take advantage of context information unavailable to FLRES
 - FLRES tuned to provide fast resolution

FLRES Building Blocks



- We may have no information on the base path
- We can take advantage of unique constants
- We can simulate some PHP expressions
- We can match the constant part of the path at the end of the given file name (if present)

Building block 1: Base paths for includes



template.php

...

```
require './headers.php'
```

...

Building block 1: Base paths for includes



template.php

```
...  
require './headers.php'  
...
```

headers.php

```
...  
...  
...
```

Building block 1: Base paths for includes



template.php

```
...  
require './headers.php'  
...
```



headers.php

```
...  
...  
...
```

Building block 1: Base paths for includes



Directory /

main.php

```
...  
require 'd/template.php'  
...
```

Directory d

template.php

```
...  
require './headers.php'  
...
```

headers.php

```
...  
...  
...
```

headers.php

```
...  
...  
...
```

Building block 1: Base paths for includes



Directory /

```
main.php  
...  
require 'd/template.php'  
...
```

Directory d

```
template.php  
...  
require './headers.php'  
...
```

```
headers.php  
...  
...  
...
```

```
headers.php  
...  
...  
...
```

Building block 1: Base paths for includes



Directory /

Directory d

main.php

```
...  
require 'd/template.php'  
...
```

template.php

```
...  
require './headers.php'  
...
```

headers.php

```
...  
...  
...
```

headers.php

```
...  
...  
...
```

Building block 1: Base paths for includes



Directory /

```
main.php
...
require 'd/template.php'
...
```

Directory d

```
template.php
...
require './headers.php'
...
```

```
headers.php
...
...
...
```

```
headers.php
...
...
...
```


Building block 1: Base paths for includes



- If we have a literal path starting with ‘/’, we can use this — rules say it must be looked up from web root
 - Note: this is very uncommon, forces install location
- Otherwise, path can’t tell us where to start looking for the file

Building block 2: Unique constants



- If a constant is always defined with the same value, we allow the algorithm to use it

wp-mail.php

```
...  
...Use Of WPINC...  
...
```

wp-load.php

```
...  
define( 'WPINC', 'wp-includes' );  
...
```

wp-settings.php

```
...  
define( 'WPINC', 'wp-includes' );  
...
```

Building block 2: Unique constants



- If a constant is always defined with the same value, we allow the algorithm to use it

wp-mail.php

```
...  
... 'wp-includes' ...  
...
```

wp-load.php

```
...  
define( 'WPINC', 'wp-includes' );  
...
```

wp-settings.php

```
...  
define( 'WPINC', 'wp-includes' );  
...
```

Building block 2: Unique constants



- If a constant is always defined with the same value, we allow the algorithm to use it
- Is this sound?
 - See discussion in paper
 - Working assumption: we know all declared constants
 - Short answer: no if constant is undefined but used anyway or is one we are unaware of, otherwise yes

Building block 3: PHP expression simulation



From `wp-comments-post.php`:

```
require( dirname(__FILE__) . '/wp-load.php' );
```

Building block 3: PHP expression simulation



From wp-comments-post.php:

```
require( dirname( __FILE__ ) . '/wp-load.php' );
```

Building block 3: PHP expression simulation



From `wp-comments-post.php`:

```
require( dirname( '/webroot/wp-comments-post.php' ) . '/wp-load.php' );
```

Building block 3: PHP expression simulation



From `wp-comments-post.php`:

```
require( dirname( '/webroot/wp-comments-post.php' ) . '/wp-load.php' );
```


Building block 3: PHP expression simulation



From `wp-comments-post.php`:

```
require( '/webroot' . '/wp-load.php' );
```

Building block 3: PHP expression simulation



From `wp-comments-post.php`:

```
require( '/webroot' . 'wp-load.php' );
```

Building block 3: PHP expression simulation



From `wp-comments-post.php`:

```
require( '/webroot/wp-load.php' );
```

Building block 3: PHP expression simulation



- Magic constants evaluated
- Functions and string operations simulated on constant strings
- This is a fixpoint computation — it can generate new string constants that allow further reduction

Building block 4: Path matching



Input Expression:
`require("$maintenanceDir/Maintenance.php");`

Generate RegExp

Generated RegExp:
`\S*Maintenance[.]php`

Match Available
Files

List of System Files:

...
/includes/ImageFunctions.php
/maintenance/Maintenance.php
/skins/Vector.php
...

Matched Files:
/maintenance/Maintenance.php

PGRES Building Blocks



- We now have information on the base path
- We can take advantage of non-unique constants
- We need to be aware of PHP functions that can change the include path or current working directory at runtime

Building block 1: We can use the base path




Directory /

Directory d

```
main.php
...
require 'd/template.php'
...
```

```
template.php
...
require './headers.php'
...
```

```
headers.php
...
...
...
```

```
hea...php
...
...
...

```

Building block 2: Unique constants



- If a constant could have multiple values, we can use it if all included definitions are the same

wp-mail.php

```
...  
...Use Of WPINC...  
...
```

wp-load.php

```
...  
define( 'WPINC', 'wp-includes' );  
...
```

wp-settings.php

```
...  
define( 'WPINC', 'includes' );  
...
```


Building block 2: Unique constants



- If a constant could have multiple values, we can use it if all included definitions are the same

```
wp-mail.php  
...  
...Use Of WPINC...  
...
```



```
wp-load.php  
...  
define( 'WPINC', 'wp-includes' );  
...
```

```
wp-settings.php  
...  
define( 'WPINC', 'includes' );  
...
```

Building block 2: Unique constants



- If a constant could have multiple values, we can use it if all included definitions are the same

```
wp-mail.php  
...  
... 'wp-includes' ...  
...
```



```
wp-load.php  
...  
define( 'WPINC', 'wp-includes' );  
...
```

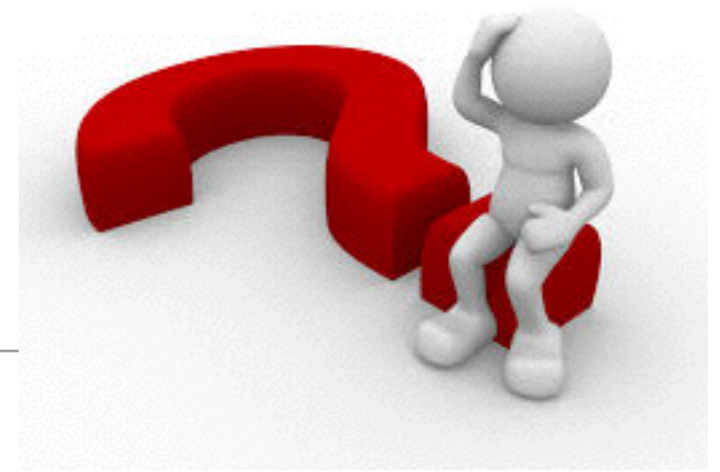
```
wp-settings.php  
...  
define( 'WPINC', 'includes' );  
...
```

Building block 3: functions can impact lookups



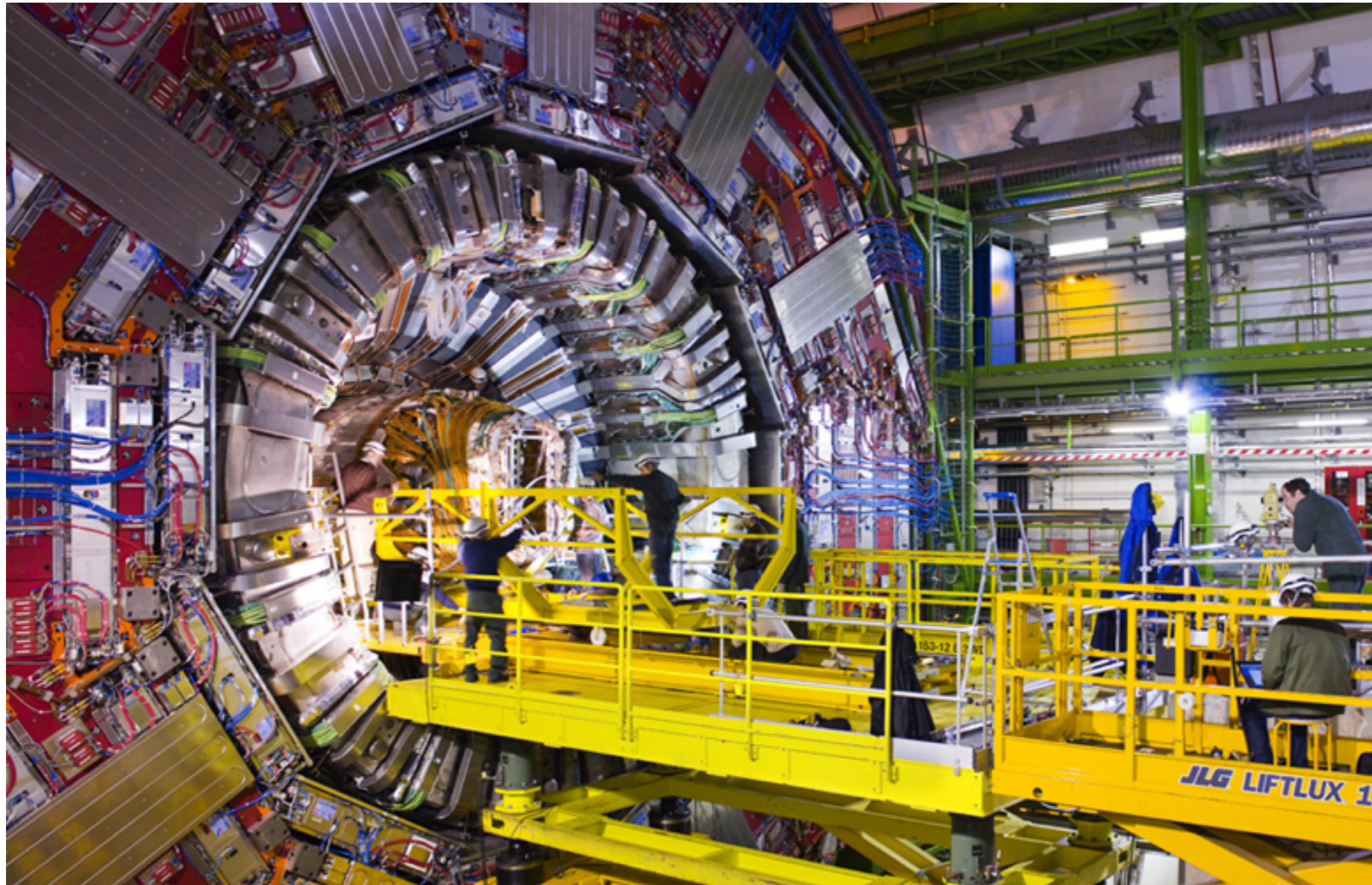
- PHP include paths and working directories can be changed at runtime
 - `chdir` changes the current working directory
 - `set_include_path` sets the include path
 - `ini_set` can also set the include path
- Reachable uses of these cause us to ignore base path info, just like in FLRES

Any new soundness concerns?



- Inherits all soundness concerns from FLRES
- One new one: we assume functions that change include path and working directory not called in obfuscated ways (e.g., using eval)

Setting Up the Experiment: Tools & Methods



http://cache.boston.com/universal/site_graphics/blogs/bigpicture/lhc_08_01/lhc11.jpg

Building an open-source PHP corpus

- Same corpus as used in ISSTA 2013, updated versions, added Magento
- Systems selected based on Ohloh (now Black Duck) rankings
- Totals: 20 open-source PHP systems, 4.59 million lines of PHP code, 32,682 files



Evaluating FLRES: Technique



- Run FLRES over entire corpus
- Track execution time on each file
- Basic stats: how many includes have static or dynamic args?
- Includes stats: how many resolve to a unique file? to any file? to something in between?

Evaluating FLRES: Overall



| System | Includes | | | Results | | | | |
|--------|----------|--------|---------|---------|---------|-------|-------|---------|
| | Total | Static | Dynamic | Unique | Missing | Any | Other | Average |
| TOTAL | 28,219 | 18,560 | 9,659 | 24,259 | 243 | 1,329 | 2,388 | 86.46 |

- Almost 86% of all includes resolved to a unique file
- 4.71% of all includes still could reference any file
- Most files analyzed in 5 to 50 milliseconds, median just over 5 (but some outliers)

Evaluating FLRES: WordPress



| System | Includes | | | Results | | | | |
|-----------|----------|--------|---------|---------|---------|-----|-------|---------|
| | Total | Static | Dynamic | Unique | Missing | Any | Other | Average |
| WordPress | 656 | 3 | 653 | 609 | 10 | 28 | 9 | 5.78 |

- 609 of 656 resolve uniquely, 28 could be any file, 9 could be multiple files (on average, out of 6 files)

Evaluating FLRES: MediaWiki



| System | Includes | | | Results | | | | |
|-----------|----------|--------|---------|---------|---------|-----|-------|---------|
| | Total | Static | Dynamic | Unique | Missing | Any | Other | Average |
| MediaWiki | 514 | 43 | 471 | 480 | 7 | 25 | 2 | 10.50 |

- 480 of 514 resolve uniquely, 25 could be any, 2 could be any of (on average) 11 files

Evaluating FLRES: Moodle and phpBB



| System | Includes | | | Results | | | | |
|--------|----------|--------|---------|---------|---------|-----|-------|---------|
| | Total | Static | Dynamic | Unique | Missing | Any | Other | Average |
| Moodle | 8,619 | 3,438 | 5,181 | 6,798 | 114 | 237 | 1,470 | 138.27 |
| phpBB | 415 | 0 | 415 | 0 | 0 | 415 | 0 | 0.00 |

- Not everything is as good:
 - Moodle has a large number of “Other” includes with a high average
 - phpBB has nothing that can be resolved

Evaluating PGRES: Technique



- Evaluation requires more in-depth knowledge of system being evaluated
- Picked 408 programs from MediaWiki (137), WordPress (91), phpMyAdmin (90), osCommerce (88), CakePHP (2)
- Added threat: if these are not programs, any improvements shown by PGRES could be accidental

Evaluating PGRES: Results



- No improvements: MediaWiki, WordPress
- Other systems show at least some improvements
 - phpMyAdmin and CakePHP shows small reduction in candidate sets
 - osCommerce shows significant improvement: candidate sets with higher numbers shrink or disappear, unique matches increase significantly
- Execution time: median is 17.483s, average is 20.962s

Evaluating PGRES: Explaining the results



- MediaWiki and WordPress have unresolved includes for plugin support (plugins, extensions, skins, etc)
- osCommerce has file structure with repeated file names — use of base location necessary to properly resolve
- Better resolution of constants and file paths both contribute to improvements — but we need to gather precise figures on this from the analysis traces

Beyond FLRES and PGRES

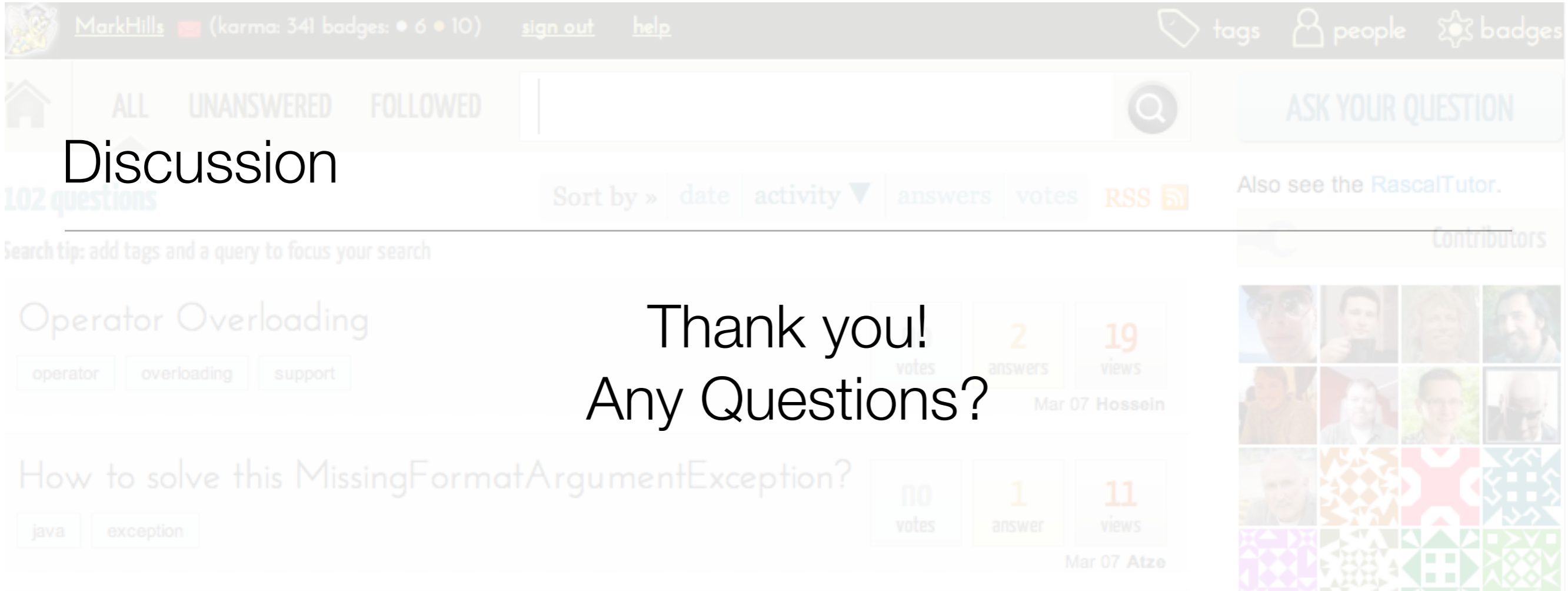


- Some systems make odd use of variables — we could do better in these cases, given a stronger analysis (although this would be slower as well)
- In many cases, we believe we cannot do better
 - Many unresolved includes support dynamic features, like plugins
 - It may be possible to resolve these *in a specific environment*, but not in general
 - Using pipeline approach shown earlier may be most fruitful approach

Wrapping Up



- Dynamic includes make static analysis of PHP code much harder
- Building on our earlier results from ISSTA 2013, we created two static analyses to resolve includes
 - FLRES provides a fast, file-level analysis that is very effective
 - PGRES provides a program level analysis that is more precise
- FLRES and PGRES can yield precise results in many cases on real PHP code



- Rascal: <http://www.rascal-mpl.org>
- PHP AiR: <https://github.com/cwi-swat/php-analysis>
- SWAT: <http://www.cwi.nl/sen1>
- Me: <http://www.cs.ecu.edu/hillsma>

Threats to validity

- Results could be very corpus-specific
- Large, well-known open-source PHP systems may not be representative of typical PHP code
- Some systems may include parts of other systems, could skew results by measuring same thing multiple times
- Answers: diversity of systems mitigates first two points, while the third is actually representative of real systems



PHP Analysis in Rascal (PHP AiR)

- Big picture: develop a framework for PHP source code analysis
- Domains:
 - Program analysis (static/dynamic)
 - Software metrics
 - Empirical software engineering
 - Developer tool support

