Performance Annotations for Complex Software Systems

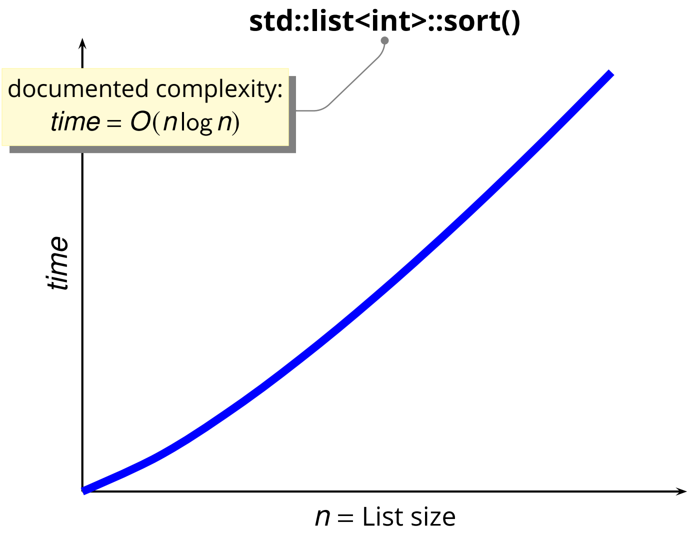
Daniele Rogora* Antonio Carzaniga* Amer Diwan$ Matthias Hauswirth*
Robert Soulé†

*USI, Switzerland †Yale University, USA $Google, USA

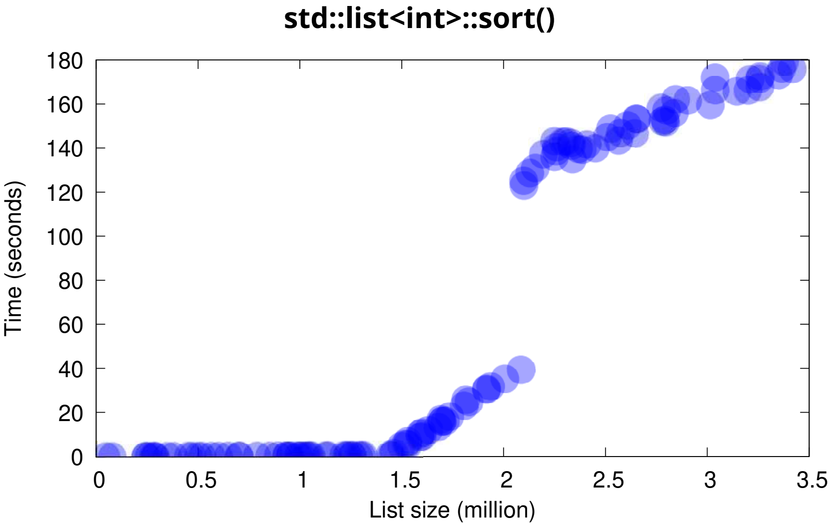
EuroSys'20

Performance Analisys is Complex!

std::list<int>::sort()


std::list<int>::sort()
documented complexity:
$time = O(n \log n)$
time
n = List size

Real Performance



Performance Analysis with Traditional Profilers

std::list<int>::sort()

```
                     Call graph (explanation follows)


granularity: each sample hit covers 2 byte(s) for 0.65% of 1.54 seconds

index % time    self  children    called     name
                                                 <spontaneous>
[1]     98.7    0.01    1.51                 main [1]
                0.04    1.17       1/1           std::__cxx11::list<int, std::allocator<int> >::sort() [2]
                0.01    0.17 1176575/1176575     int std::uniform_int_distribution<int>::operator()<std::linear_congruential_engine<unsigned long, 16807ul, 0ul, 2147483647ul> >(std::linear_congruential_engine<unsigned long, 16807ul, 0ul, 2147483647ul>&) [8]
                0.00    0.12 1176574/1176574     std::__cxx11::list<int, std::allocator<int> >::push_back(int&&) [11]
                0.00    0.00       2/2           std::__cxx11::list<int, std::allocator<int> >::clear() [63]
                0.00    0.00       1/66          std::__cxx11::list<int, std::allocator<int> >::~list() [21]
                0.00    0.00       1/66          std::__cxx11::list<int, std::allocator<int> >::list() [39]
                0.00    0.00       2/2           std::uniform_int_distribution<int>::uniform_int_distribution(int, int) [65]
                0.00    0.00       1/1           std::operator|(std::_Ios_Openmode, std::_Ios_Openmode) [109]
                0.00    0.00       1/1           std::linear_congruential_engine<unsigned long, 16807ul, 0ul, 2147483647ul>::linear_congruential_engine(unsigned long) [102]
                0.00    0.00       1/1           std::numeric_limits<int>::min() [101]
                0.00    0.00       1/1           std::numeric_limits<int>::max() [100]
                0.00    0.00       1/2           std::linear_congruential_engine<unsigned long, 16807ul, 0ul, 2147483647ul>::seed(unsigned long) [96]
                0.00    0.00       1/1           std::common_type<std::chrono::duration<long, std::ratio<1l, 1000000000l> >, std::chrono::duration<long, std::ratio<1l, 1000000000l> > >::type std::chrono::operator-<std::chrono::_V2::system_clock, std::chrono::duration<long, std::ratio<1l, 1000000000l> >, std::chrono::duration<long, std::ratio<1l, 1000000000l> > >(std::chrono::time_point<std::chrono::_V2::system_clock, std::chrono::duration<long, std::ratio<1l, 1000000000l> > > const&, std::chrono::time_point<std::chrono::_V2::system_clock, std::chrono::duration<long, std::ratio<1l, 1000000000l> > > const&) [107]
                0.00    0.00       1/1           std::enable_if<std::chrono::__is_duration<std::chrono::duration<long, std::ratio<1l, 1000000l> > >::value, std::chrono::duration<long, std::ratio<1l, 1000000l> > >::type std::chrono::duration_cast<std::chrono::duration<long, std::ratio<1l, 1000000l> >, long, std::ratio<1l, 1000000000l> >(std::chrono::duration<long, std::ratio<1l, 1000000000l> > const&) [103]
                0.00    0.00       1/1           std::chrono::duration<long, std::ratio<1l, 1000000l> >::count() const [99]
-----------------------------------------------
                0.04    1.17       1/1           main [1]
[2]     78.4    0.04    1.17       1         std::__cxx11::list<int, std::allocator<int> >::sort() [2]
                0.00    0.74 1176579/1176579     std::__cxx11::list<int, std::allocator<int> >::merge(std::__cxx11::list<int, std::allocator<int> >&) [3]
                0.01    0.19 1176574/1176574     std::__cxx11::list<int, std::allocator<int> >::splice(std::_List_const_iterator<int>, std::__cxx11::list<int, std::allocator<int> >&, std::_List_const_iterator<int>) [6]
                0.01    0.07 2353134/2353134     std::__cxx11::list<int, std::allocator<int> >::swap(std::__cxx11::list<int, std::allocator<int> >&) [18]
                0.00    0.07      65/66          std::__cxx11::list<int, std::allocator<int> >::~list() [21]
                0.03    0.01 2353148/4706306     std::__cxx11::list<int, std::allocator<int> >::begin() [22]
                0.03    0.00 3529686/3529686     std::__cxx11::list<int, std::allocator<int> >::empty() const [34]
                0.00    0.02      65/66          std::__cxx11::list<int, std::allocator<int> >::list() [39]
                0.01    0.00 2353148/3529722     std::_List_const_iterator<int>::_List_const_iterator(std::_List_iterator<int> const&) [47]
-----------------------------------------------
                0.00    0.74 1176579/1176579     std::__cxx11::list<int, std::allocator<int> >::sort() [2]
[3]     48.0    0.00    0.74 1176579         std::__cxx11::list<int, std::allocator<int> >::merge(std::__cxx11::list<int, std::allocator<int> >&) [3]
                0.26    0.48 1176579/1176579     std::__cxx11::list<int, std::allocator<int> >::merge(std::__cxx11::list<int, std::allocator<int> >&&) [4]
                0.00    0.00 1176579/2353153     std::remove_reference<std::__cxx11::list<int, std::allocator<int> >&>::type&& std::move<std::__cxx11::list<int, std::allocator<int> >&>(std::__cxx11::list<int, std::allocator<int> >&) [61]
-----------------------------------------------
                0.26    0.48 1176579/1176579     std::__cxx11::list<int, std::allocator<int> >::merge(std::__cxx11::list<int, std::allocator<int> >&) [3]
[4]     47.8    0.26    0.48 1176579         std::__cxx11::list<int, std::allocator<int> >::merge(std::__cxx11::list<int, std::allocator<int> >&&) [4]
                0.07    0.13 45669018/45669018     std::_List_iterator<int>::operator*() const [5]
                0.10    0.00 48610882/48610882     std::_List_iterator<int>::operator!=(std::_List_iterator<int> const&) const [14]
                0.04    0.00 22834509/24011083     std::_List_iterator<int>::operator++() [27]
                0.03    0.01 2353158/3529732     std::__cxx11::list<int, std::allocator<int> >::end() [24]
                0.03    0.01 2353158/4706306     std::__cxx11::list<int, std::allocator<int> >::begin() [22]
                0.02    0.00 11537904/12714478     std::__cxx11::list<int, std::allocator<int> >::_M_transfer(std::_List_iterator<int>, std::_List_iterator<int>, std::_List_iterator<int>) [40]
```

Real, expected behavior as a function of input/state features

Performance Analysis with *Performance Annotations*

actual behavior
concrete metrics

Real, expected behavior as a function of input/state features

Performance Analysis with *Performance Annotations*

actual behavior
concrete metrics

Real, expected behavior as a function of input/state features

significant
statistics

Performance Analysis with *Performance Annotations*

actual behavior
concrete metrics

specific characterization
not merely an *aggregate* profile

Real, expected behavior as a function of input/state features

significant
statistics

Performance Analysis with *Performance Annotations*

actual behavior
concrete metrics

specific characterization
not merely an *aggregate* profile

Real, expected behavior as a function of input/state features

significant
statistics

For each module/function of interest:

$$metric_i = f_i(feature, \ldots)$$

Performance Analysis with *Performance Annotations*

actual behavior
concrete metrics

specific characterization
not merely an *aggregate* profile

Real, expected behavior as a function of input/state features

significant
statistics

For each module/function of interest:

$$metric_i = f_i(feature, \ldots)$$

run-time
memory allocation
lock-holding time
...

Performance Analysis with *Performance Annotations*

actual behavior
concrete metrics

specific characterization
not merely an *aggregate* profile

Real, expected behavior as a function of input/state features

significant
statistics

For each module/function of interest:

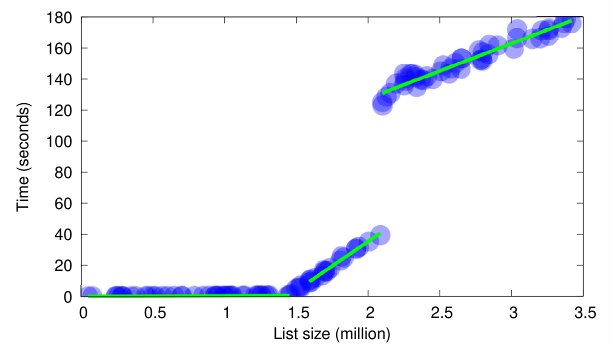
$$metric_i = f_i(feature, \ldots)$$

run-time
memory allocation
lock-holding time
...

input parameters, global variables,
...
even in nested, structured objects
identified automatically!

Performance Annotations

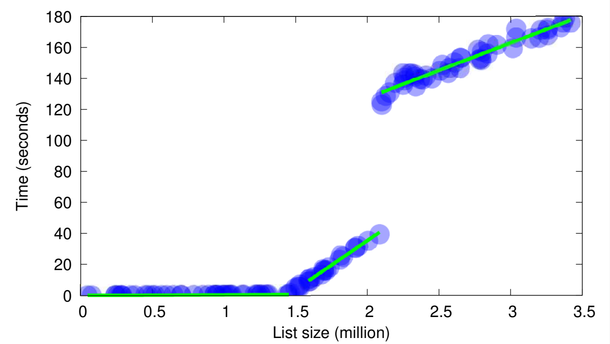
```
std::list<int>::sort.time(this) {
  uint s = *(this->_M_impl._M_node._M_storage._M_storage);

  [s > 49584 && s < 1450341]
  Norm(53350.31 - 2.10*s + 0.12*s*log(s), 12463.88);

  [s > 1589482 && s < 2085480]
  Norm(-90901042.29 + 63.11*s, 899547.29);

  [s > 2098759 && s < 3415880]
  Norm(56712024.50 + 35.38*s, 3379580.27);
}
```



function of interest

Performance Annotations

```
std::list<int>::sort.time(this) {
  uint s = *(this->_M_impl._M_node._M_storage._M_storage);

  [s > 49584 && s < 1450341]
  Norm(53350.31 - 2.10*s + 0.12*s*log(s), 12463.88);

  [s > 1589482 && s < 2085480]
  Norm(-90901042.29 + 63.11*s, 899547.29);

  [s > 2098759 && s < 3415880]
  Norm(56712024.50 + 35.38*s, 3379580.27);
}
```

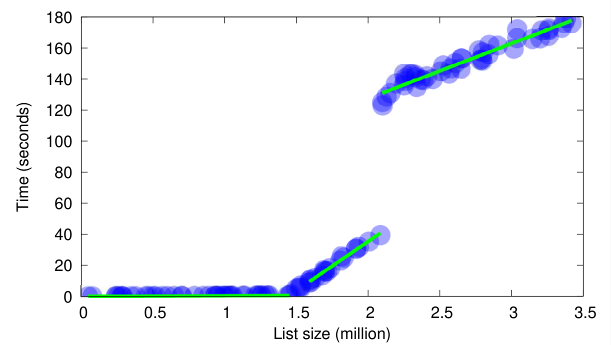


Performance Annotations

function of interest

metric

```
std::list<int>::sort.time(this) {
  uint s = *(this->_M_impl._M_node._M_storage._M_storage);

  [s > 49584 && s < 1450341]
  Norm(53350.31 - 2.10*s + 0.12*s*log(s), 12463.88);

  [s > 1589482 && s < 2085480]
  Norm(-90901042.29 + 63.11*s, 899547.29);

  [s > 2098759 && s < 3415880]
  Norm(56712024.50 + 35.38*s, 3379580.27);
}
```



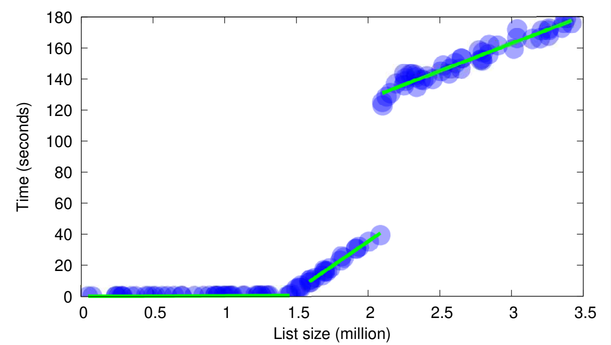
Performance Annotations

function of interest

metric

feature: s=list size

```
std::list<int>::sort.time(this) {
  uint s = *(this->_M_impl._M_node._M_storage._M_storage);

  [s > 49584 && s < 1450341]
  Norm(53350.31 - 2.10*s + 0.12*s*log(s), 12463.88);

  [s > 1589482 && s < 2085480]
  Norm(-90901042.29 + 63.11*s, 899547.29);

  [s > 2098759 && s < 3415880]
  Norm(56712024.50 + 35.38*s, 3379580.27);
}
```



Performance Annotations

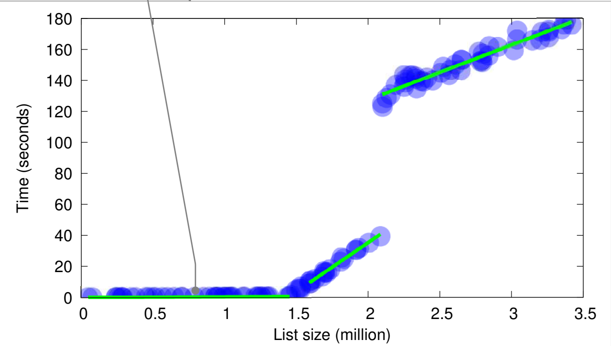
function of interest

metric

feature: s=list size

scope (1)

```
std::list<int>::sort.time(this) {
  uint s = *(this->_M_impl._M_node._M_storage._M_storage);

  [s > 49584 && s < 1450341]
  Norm(53350.31 - 2.10*s + 0.12*s*log(s), 12463.88);

  [s > 1589482 && s < 2085480]
  Norm(-90901042.29 + 63.11*s, 899547.29);

  [s > 2098759 && s < 3415880]
  Norm(56712024.50 + 35.38*s, 3379580.27);
}
```



Performance Annotations

function of interest

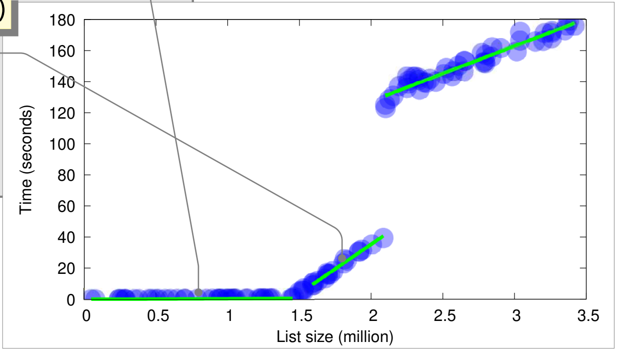
metric

feature: s=list size

scope (1)

scope (2)

```
std::list<int>::sort.time(this) {
  uint s = *(this->_M_impl._M_node._M_storage._M_storage);

  [s > 49584 && s < 1450341]
  Norm(53350.31 - 2.10*s + 0.12*s*log(s), 12463.88);

  [s > 1589482 && s < 2085480]
  Norm(-90901042.29 + 63.11*s, 899547.29);

  [s > 2098759 && s < 3415880]
  Norm(56712024.50 + 35.38*s, 3379580.27);
}
```



Performance Annotations

function of interest

metric

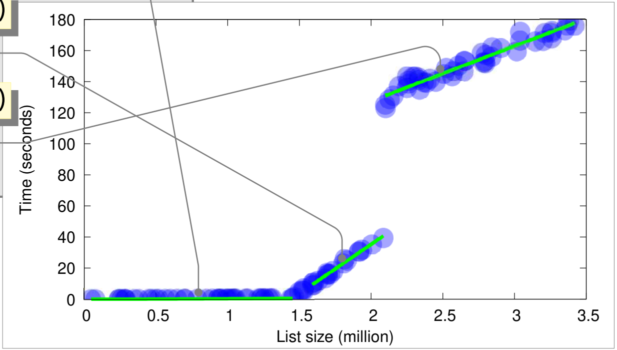
feature: s=list size

```
std::list<int>::sort.time(this) {
  uint s = *(this->_M_impl._M_node._M_storage._M_storage);

  [s > 49584 && s < 1450341]
  Norm(53350.31 - 2.10*s + 0.12*s*log(s), 12463.88);

  [s > 1589482 && s < 2085480]
  Norm(-90901042.29 + 63.11*s, 899547.29);

  [s > 2098759 && s < 3415880]
  Norm(56712024.50 + 35.38*s, 3379580.27);
}
```

scope (1)

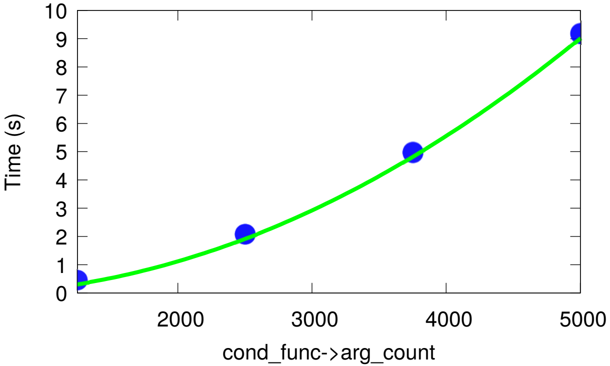
scope (2)

scope (3)



Automatic Feature Discovery

```
get_func_mm_tree(RANGE_OPT_PARAM *param,
                 Item *pred,
                 Item_func *cond_func,
                 Item *val,
                 bool inv);
```

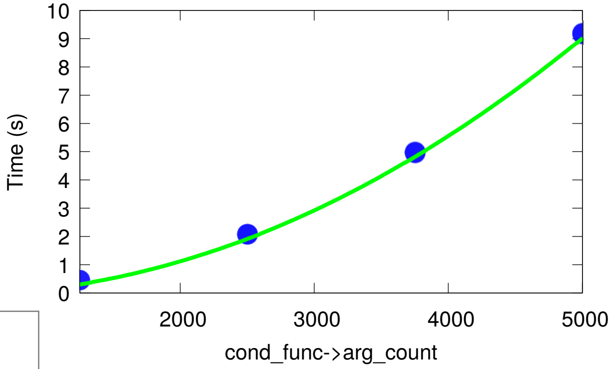



```
get_func_mm_tree.time(cond_func) {
  uint ac = cond_func->arg_count;
  Norm(156569 - 269.041*ac + 0.414447*ac^2, 15781.22);
}
```

Automatic Feature Discovery

```
get_func_mm_tree(RANGE_OPT_PARAM *param,
                 Item *pred,
                 Item_func *cond_func,
                 Item *val,
                 bool inv);
```

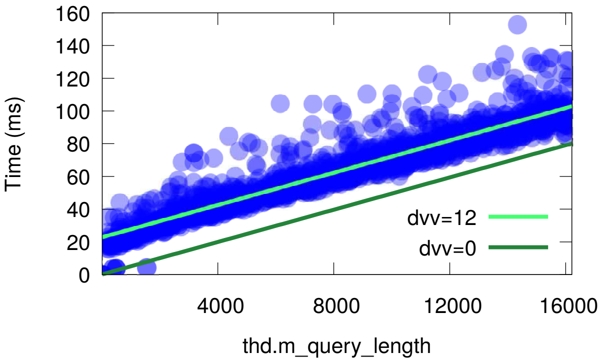
item_func.h alone is 3885 lines!




```
get_func_mm_tree.time(cond_func) {
  uint ac = cond_func->arg_count;
  Norm(156569 - 269.041*ac + 0.414447*ac^2, 15781.22);
}
```

Automatic Feature Discovery

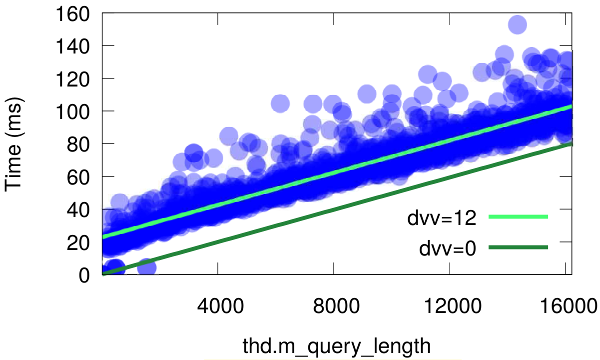
```
mysql_execute_command(THD *thd,
                      bool first_level);
```




```
mysql_execute_command.time(thd) {
  uint len = thd->m_query_string.len;
  uint dvv = thd->variables.dynamic_variable_version;
  Norm(168.65 + 4.94*len + 1886.87*dvv, 2489.04);
}
```

Automatic Feature Discovery

```
mysql_execute_command(THD *thd,
                      bool first_level);
```



struct traversal

unexpected feature!

```
mysql_execute_command.time(thd) {
  uint len = thd->m_query_string.len;
  uint dvv = thd->variables.dynamic_variable_version;
  Norm(168.65 + 4.94*len + 1886.87*dvv, 2489.04);
}
```

Uses of Performance Annotations

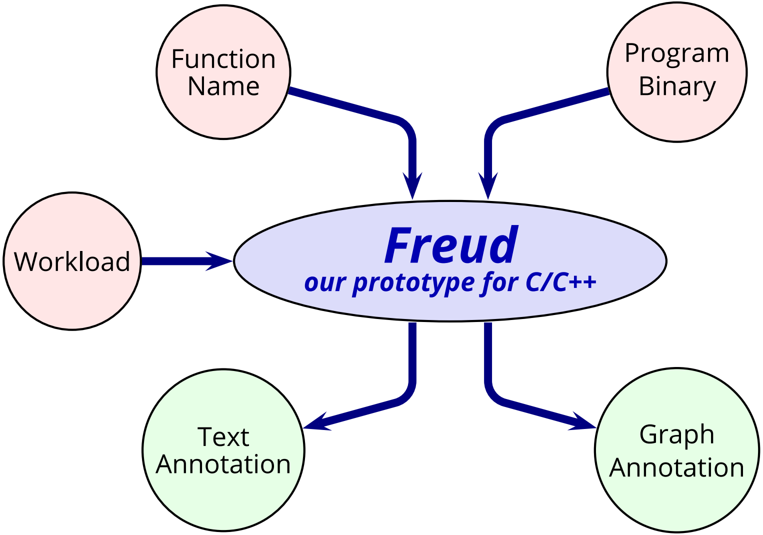
- Documentation
  - automatic creation
  - readable annotations and graphs for performance analyst
  - feature names as in the program

- Annotations as performance assertions
  - detecting performance anomalies and regressions

- Prediction
  - extrapolation to unobserved feature values
  - annotation composition: new code that uses annotated functions

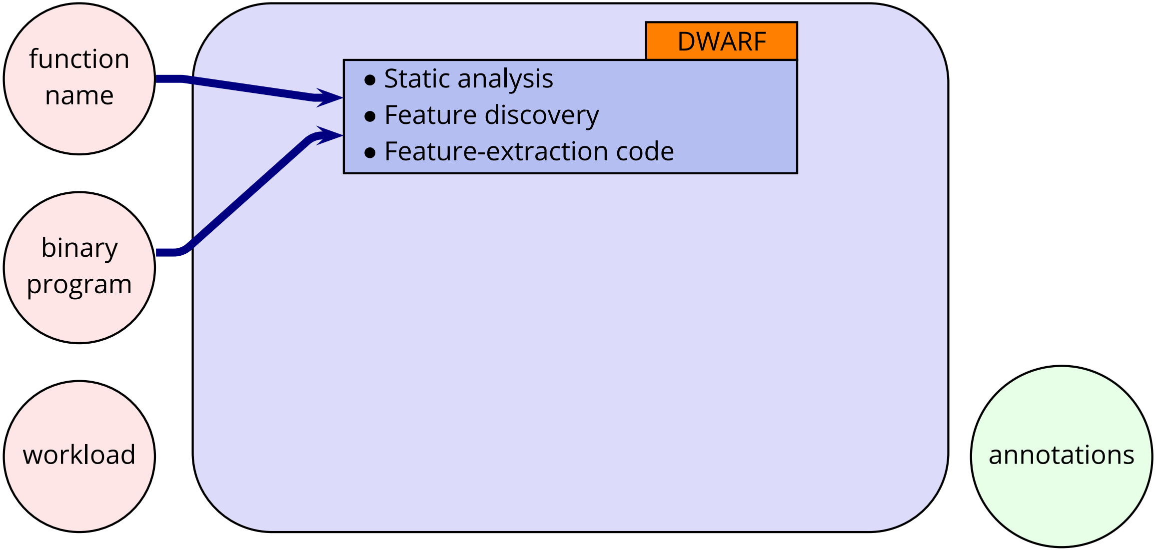
Freud

our prototype for C/C++

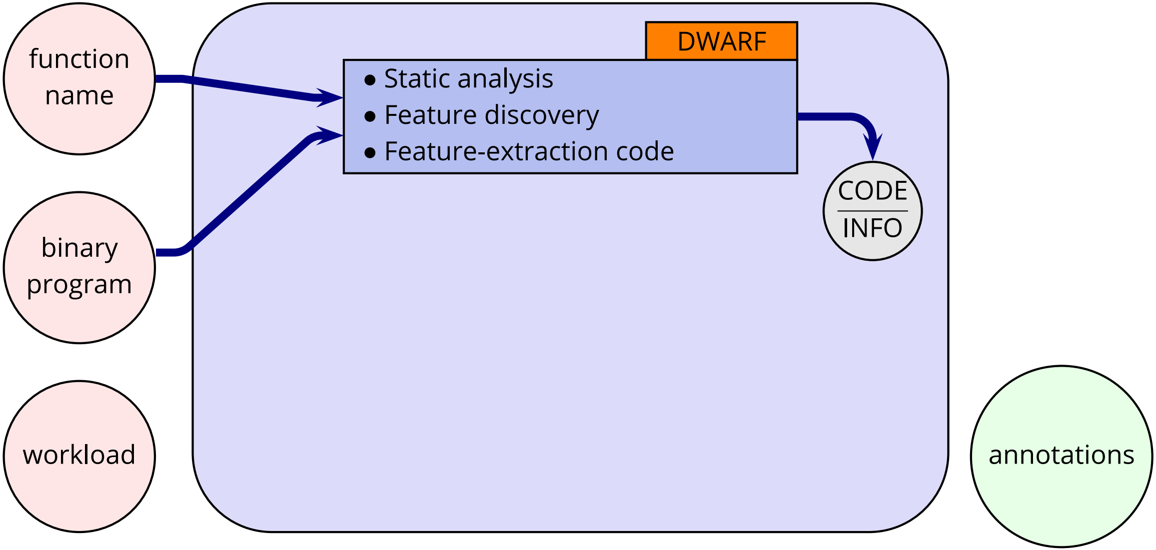

Function Name
Program Binary
Workload
Freud
our prototype for C/C++
Text Annotation
Graph Annotation


function name
binary program
workload
annotations

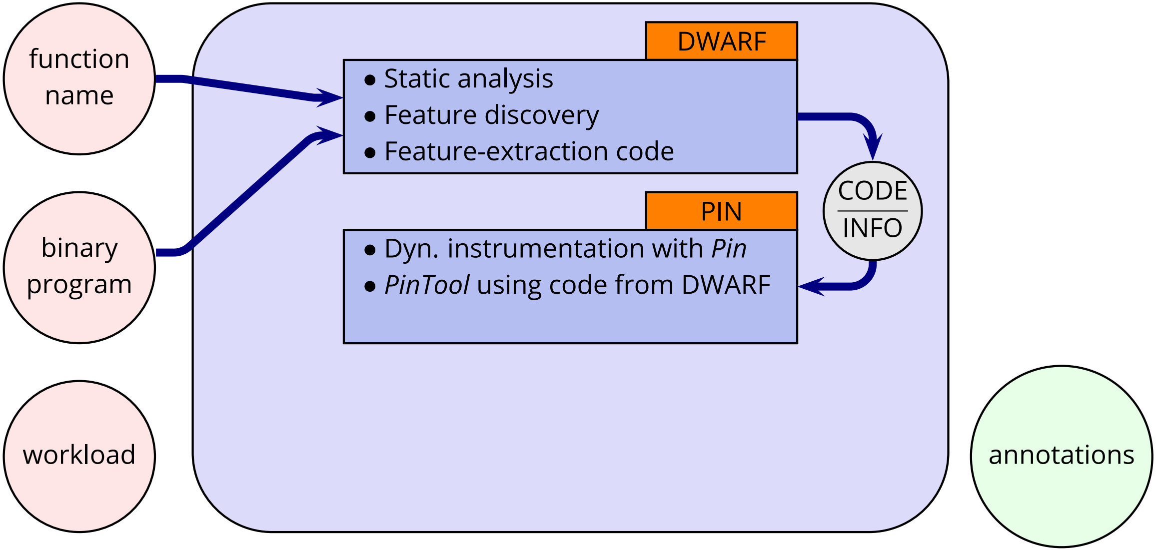
Freud

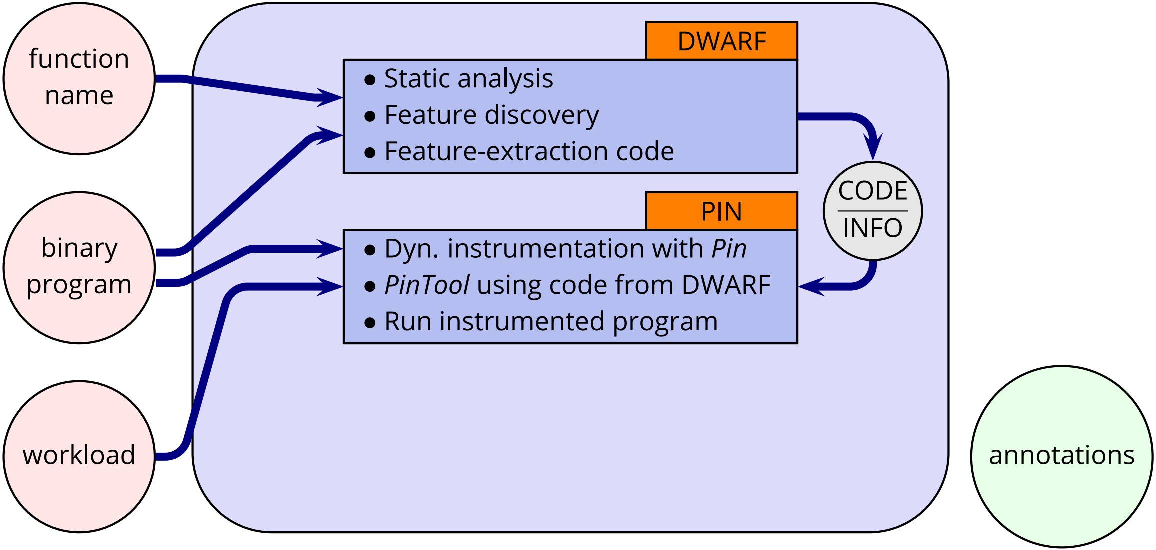

function name
binary program
workload
DWARF
• Static analysis
• Feature discovery
• Feature-extraction code
annotations

Freud

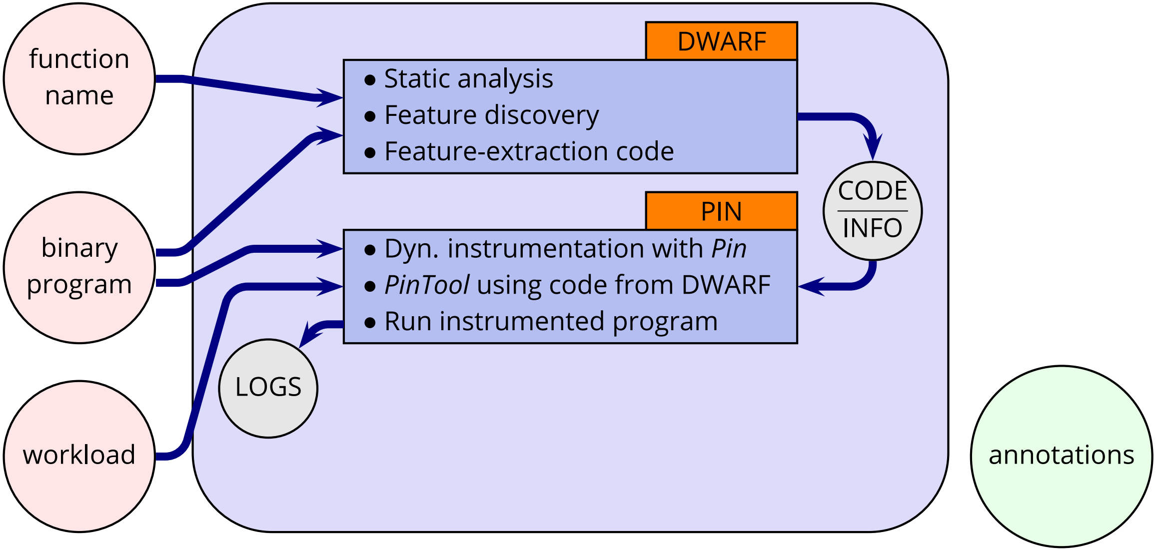

function name
binary program
workload
DWARF
• Static analysis
• Feature discovery
• Feature-extraction code
CODE
INFO
annotations

Freud

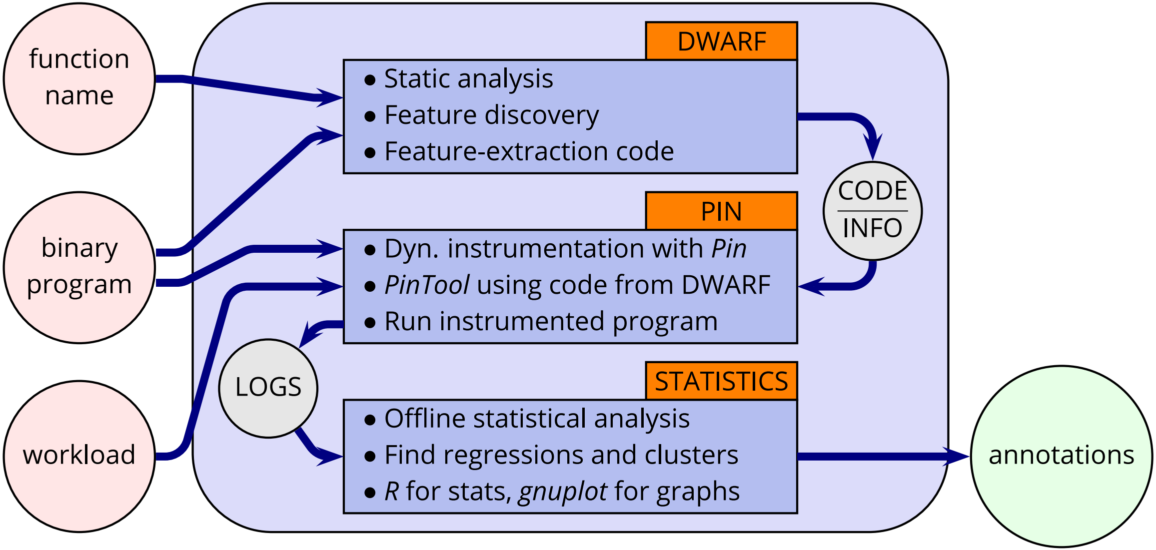

function name
binary program
workload
DWARF
• Static analysis
• Feature discovery
• Feature-extraction code
CODE
INFO
PIN
• Dyn. instrumentation with *Pin*
• *PinTool* using code from DWARF
annotations


function name
binary program
workload
DWARF
• Static analysis
• Feature discovery
• Feature-extraction code
CODE
INFO
PIN
• Dyn. instrumentation with *Pin*
• *PinTool* using code from DWARF
• Run instrumented program
annotations

Freud

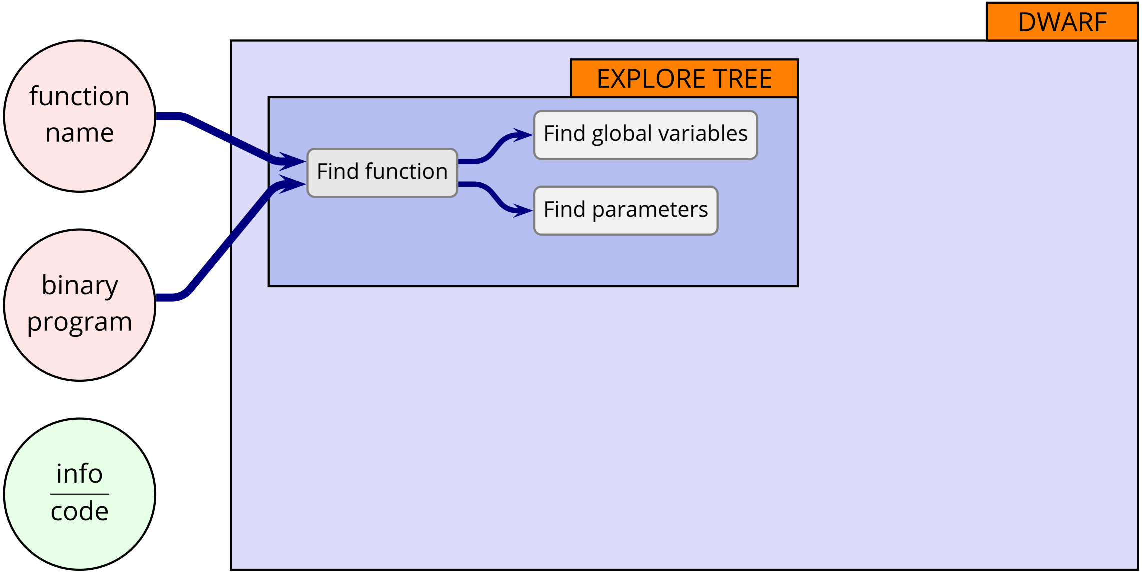

function name
binary program
workload
DWARF
• Static analysis
• Feature discovery
• Feature-extraction code
CODE
INFO
PIN
• Dyn. instrumentation with *Pin*
• *PinTool* using code from DWARF
• Run instrumented program
LOGS
annotations

Freud

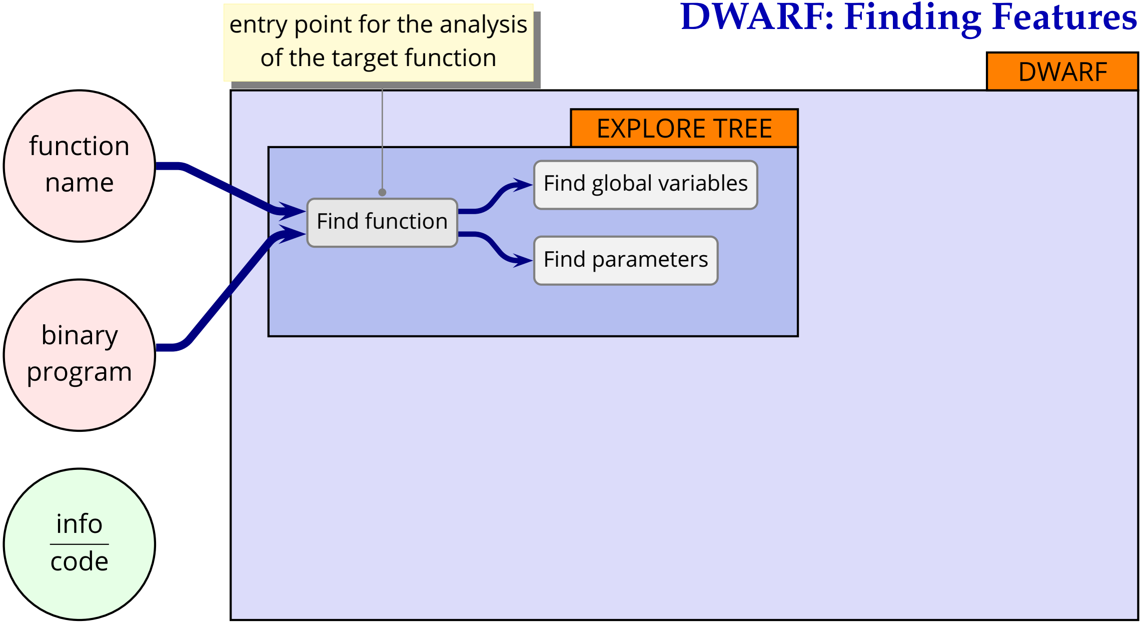

function name
binary program
workload
DWARF
• Static analysis
• Feature discovery
• Feature-extraction code
CODE
INFO
PIN
• Dyn. instrumentation with *Pin*
• *PinTool* using code from DWARF
• Run instrumented program
LOGS
STATISTICS
• Offline statistical analysis
• Find regressions and clusters
• *R* for stats, *gnuplot* for graphs
annotations

DWARF: Finding Features

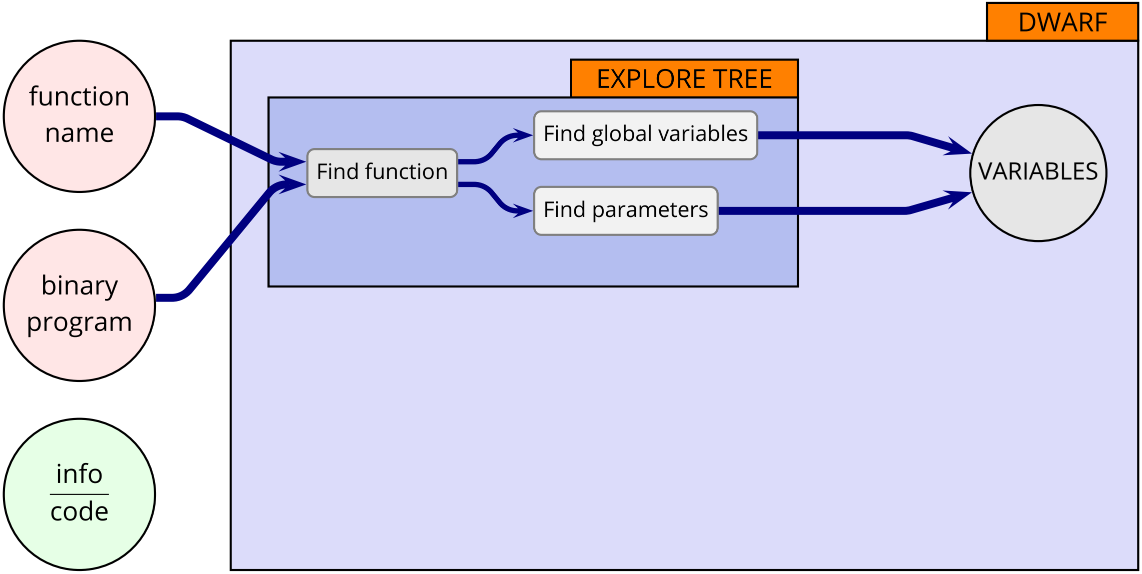
DWARF: Finding Features



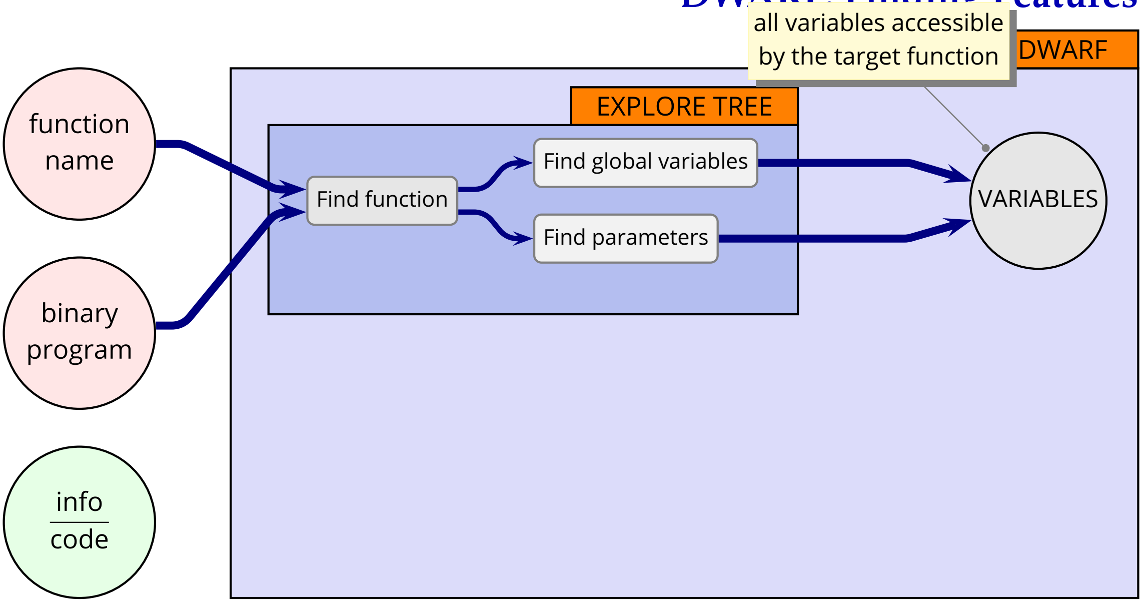
DWARF: Finding Features



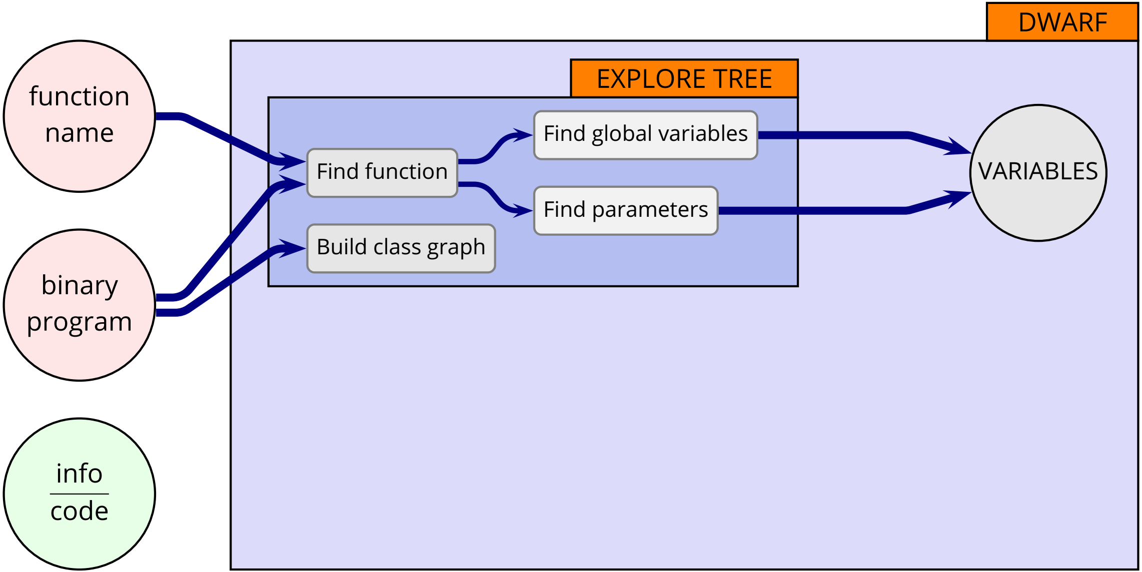
DWARF: Finding Features



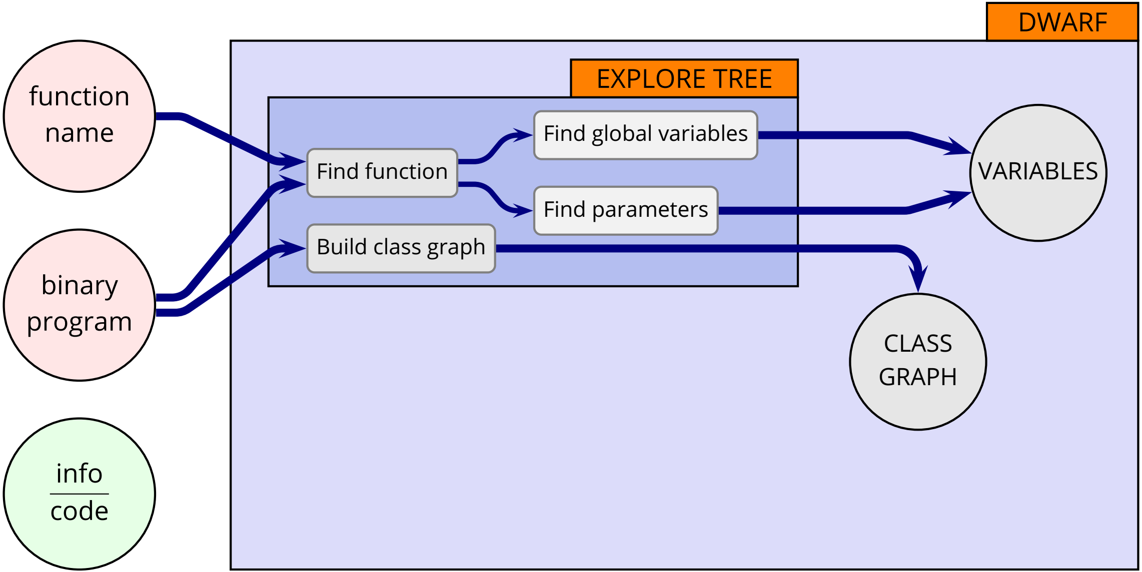
DWARF: Finding Features



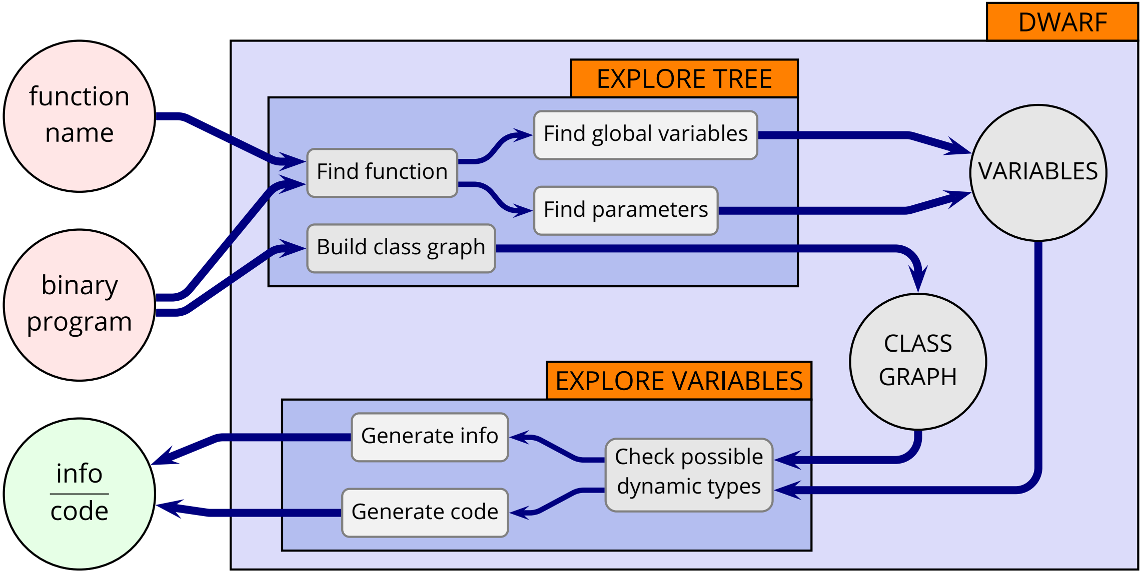
DWARF: Finding Features



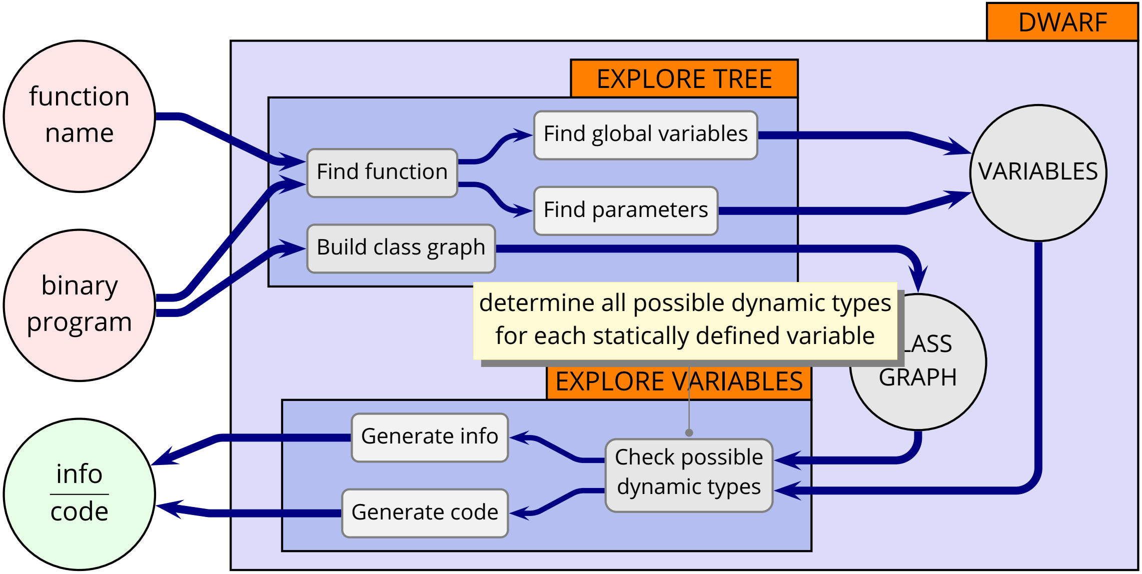
DWARF: Finding Features



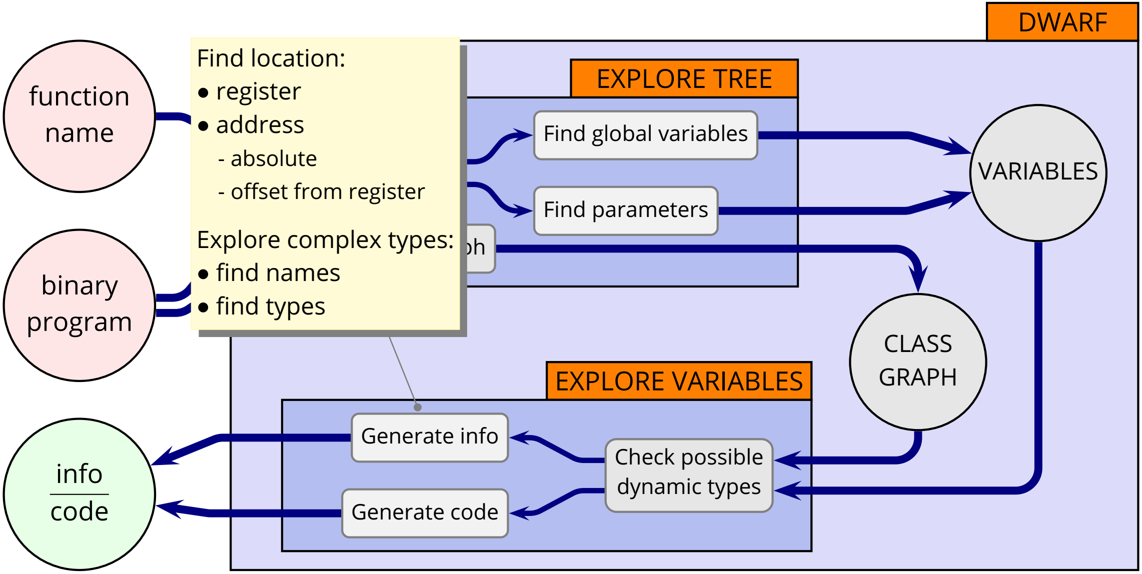
DWARF: Finding Features



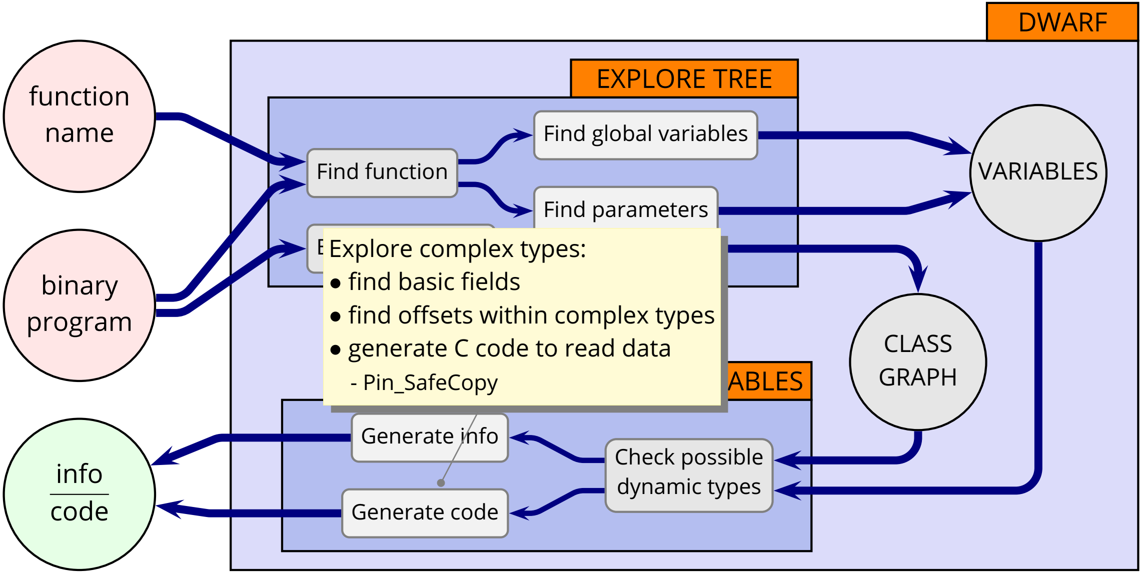
DWARF: Finding Features



DWARF: Finding Features



DWARF: Finding Features



Evaluation

Evaluation

- Does *Freud* Produce Correct Information?
  - set of basic functions using that use sleep to exhibit a known performance
- Does *Freud* help understanding performance?
  - real world experiments with complex Php and C++ software
- Does *Freud* find performance bugs?
  - real world experiments with performance bugs from the MySQL bugtracker

Does *Freud* Produce Correct Information?

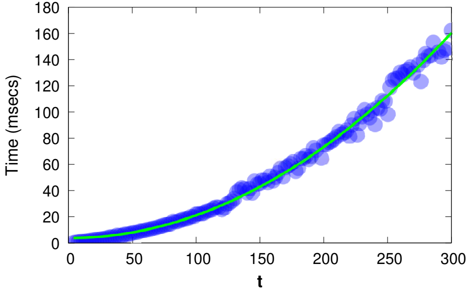
Quadratic

```
void __attribute__ ((noinline)) test_quad_int(int t) {
  for (int i = 0; i < t; i++) {
    usleep(t);
  }
}
```

Does *Freud* Produce Correct Information?

Quadratic

```
void __attribute__ ((noinline)) test_quad_int(int t) {
  for (int i = 0; i < t; i++) {
    usleep(t);
  }
}
```

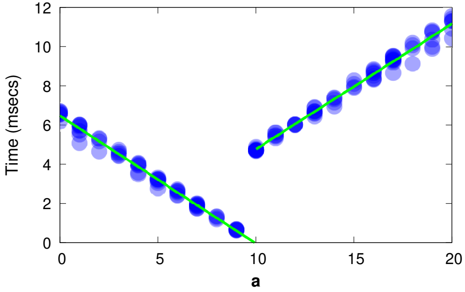



```
test_quad_int(t).time {
Norm(3657.73 + 1.74*t^2, 19.31);
}
```

Does *Freud* Produce Correct Information?

Branches

```
void __attribute__ ((noinline)) test_linear_branches_one_f(int a, int b, int c) {
  if (a < 10) {  for (int i = 0; i < 10 - a; i++)  {  usleep(400);  }  }
  else {
    usleep(4000);
    for (int i = 0; i < a - 10; i++)  usleep(400);
  }
}
```

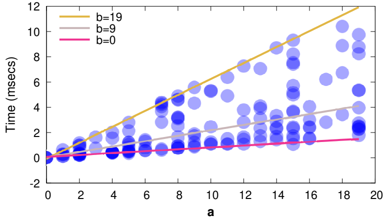


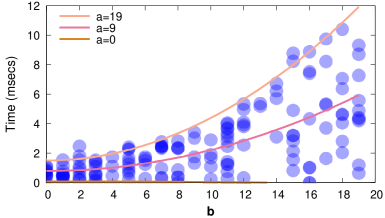

```
test_linear_branches_one_f(a).time {
[a <= 9]
Norm(6472.36 - 651.01*a, 46.55);
[a > 9]
Norm(-1613.27 + 638.57*a, 32.88);
}
```

Does *Freud* Produce Correct Information?

Interaction Terms

```
void __attribute__ ((noinline)) test_interaction_linear_quad(int a, int b) {
  for (int i = 0; i < a; i++)
    usleep(b*b);
}
```



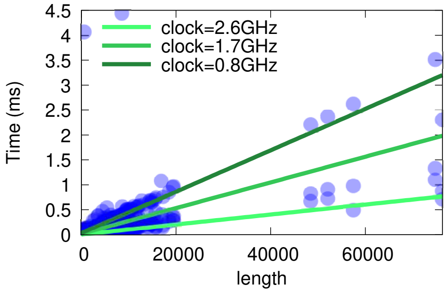



```
test_interaction(a,b).time {
Norm(69.51 + 75.26 * a - 0.39 * b^2
+ 1.54*a*b^2, 11.69);
}
```

Evaluation

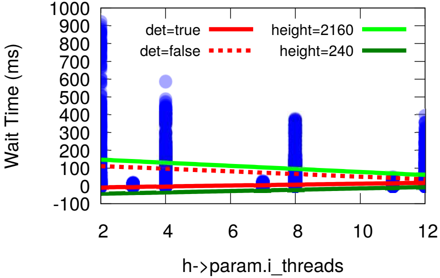
- Does *Freud* Produce Correct Information?
  - set of basic functions using that use sleep to exhibit a known performance
- Does *Freud* help understanding performance?
  - real world experiments with complex Php and C++ software
- Does *Freud* find performance bugs?
  - real world experiments with performance bugs from the MySQL bugtracker

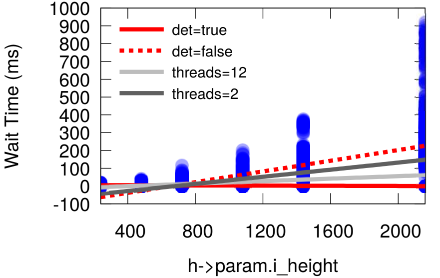
Does *Freud* Help Understanding?




```
ff_h2645_extract_rbsp.time(length, cpu_clock) {
uint l = length;
uint clock = cpu_clock;
Norm(43.32 + 0.055*l - 1.46e-05*clock
- 1.75e-08*l*clock, 4.56);
}
```

Does *Freud* Work with Complex Cases?






```
x264_8_encoder_encode.wait_time(h, pic_in) {
bool sliced = h->param.b_sliced_threads;
uint height = h->param.i_height;
uint threads = h->param.i_threads;
uint dequant = h->thread.dequant4_mf;
bool det = pic_in->param.b_deterministic;

[sliced]
Norm(-56362 + 189.17*height - 3221.21*threads
- 1378.66*dequant - 152.83*height*det
- 6.48*height*threads + 10044*threads*det, 1.05e+05 )

[!sliced]
0.55Norm(108.7, 188.65); 0.30Norm(7282, 51465.24);  ...
}
```

Evaluation

- Does *Freud* Produce Correct Information?
  - set of basic functions using that use sleep to exhibit a known performance
- Does *Freud* help understanding performance?
  - real world experiments with complex Php and C++ software
- Does *Freud* find performance bugs?
  - real world experiments with performance bugs from the MySQL bugtracker

Does *Freud* Find Performance Regressions?

Bug #92979	**MySQL 8.0 performance degradation on INSERT with foreign_key_checks=0**		
Submitted:	28 Oct 2018 13:51	**Modified:**	30 Oct 2018 8:38
Reporter:	Predrag Zivanovic	**Email Updates:**	Subscribe
Status:	Verified	**Impact on me:**	None Affects Me
Category:	MySQL Server: InnoDB storage engine	**Severity:**	S5 (Performance)
Version:	8.0.13 Communty Server	**OS:**	Any
Assigned to:		**CPU Architecture:**	x86
Tags:	dump, foreign keys		

View | Add Comment | Files | Developer | Edit Submission | View Progress Log | Contributions

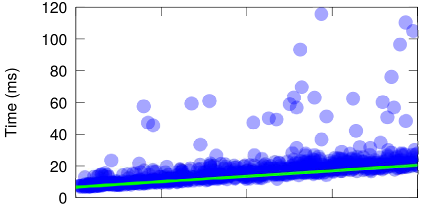
[28 Oct 2018 13:51] Predrag Zivanovic

Description:
There is significant performance degradation between MySQL 5.7 and MySQL 8.0 when importing SQL dump with foreign keys and with foreign_key_checks=0. It looks like MySQL 8.0 is checking foreign keys references even with foreign_key_checks=0, only without error message.

How to repeat:
Here is MySQL dump file attached. On new fresh installation of MySQL 5.7 it took 15 seconds to import ... on MySQL 8.0 it took more then 400 seconds. InnoDB storage engine, default settings in both cases.

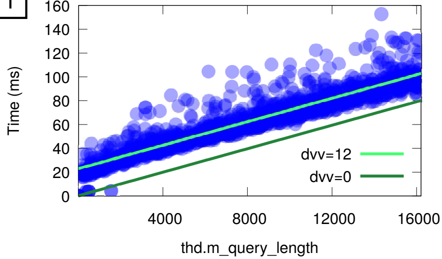
Does *Freud* Find Performance Regressions?

5.7.24




```
mysql_execute_command(thd).time{
uint len = thd->m_query_string.len;
Norm(6630.19 + 0.86*len, 15.78);
}
```

8.0.11




```
mysql_execute_command(thd).time{
uint len = thd->m_query_string.len;
uint dvv = thd->variables.dynamic_variable_version;
Norm(168.65 + 4.94*len + 1886.87*dvv, 2489.04);
}
```

Does *Freud* Help Finding Bugs?

Bug #94296	Poor Optimizer Performance with Composite Index, IN() function, and many Tuples		
Submitted:	12 Feb 2019 18:17	Modified:	13 Feb 2019 19:41
Reporter:	Daniel Jeffery	Email Updates:	Subscribe
Status:	Closed	Impact on me:	None Affects Me
Category:	MySQL Server: Optimizer	Severity:	S5 (Performance)
Version:	8.0.11	OS:	Ubuntu (Ubuntu 16.04.1 LTS)
Assigned to:		CPU Architecture:	x86 (x86_64)
Tags:	composite_index		

View | Add Comment | Files | Developer | Edit Submission | View Progress Log | Contributions

[12 Feb 2019 18:17] Daniel Jeffery

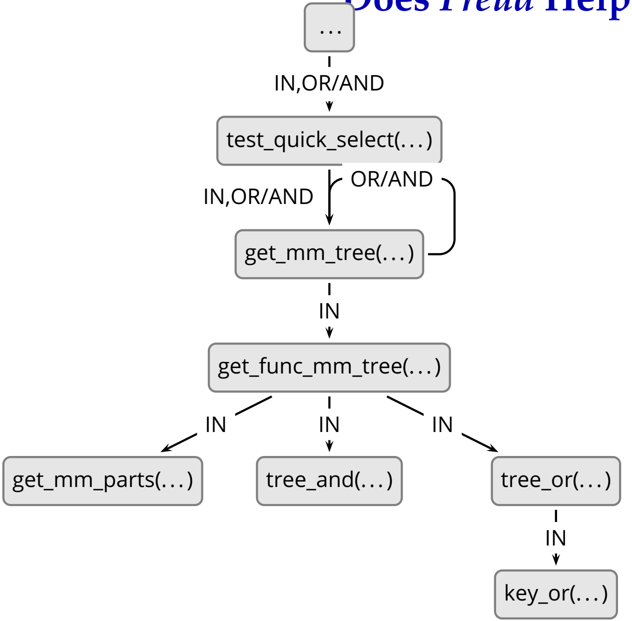
Description:
Query optimization takes a very long time for a SELECT query on a composite index with a large list of tuples. The performance degradation as the list of tuples grows seems to be geometric, compared to linear performance of an unindexed query or one using simple AND/OR clauses.

My expectation is that performance of the IN() function using an index would be similar, if not better, than alternatives, and that query optimization would not take more time than query execution.

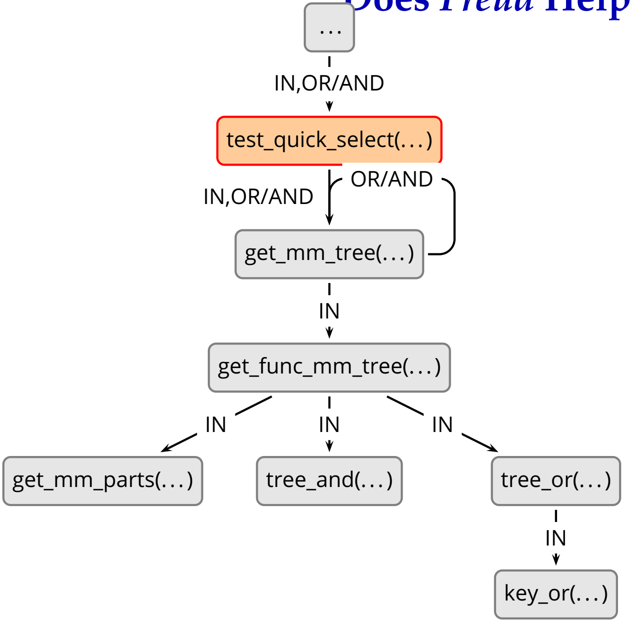
I believe this is an issue with the optimizer, as the use of the index even affects "EXPLAIN SELECT ..." queries.

How to repeat:

Does *Freud* Help Finding Bugs?

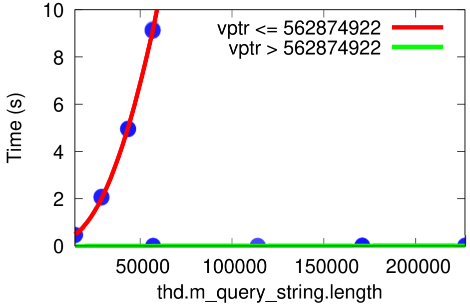


Does *Freud* Help Finding Bugs?



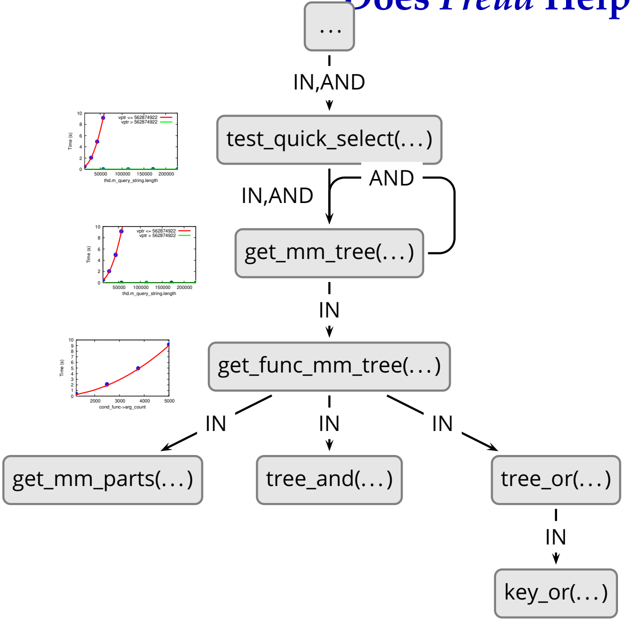
Does *Freud* Help Finding Bugs?

```
test_quick_select(THD *thd, Key_map keys_to_use, table_map prev_tables, ha_rows limit,
      bool force_quick_range, const enum_order interesting_order, const QEP_shared_owner *tab,
      Item *cond, Key_map *needed_reg, QUICK_SELECT_I **quick, bool ignore_table_scan);
```

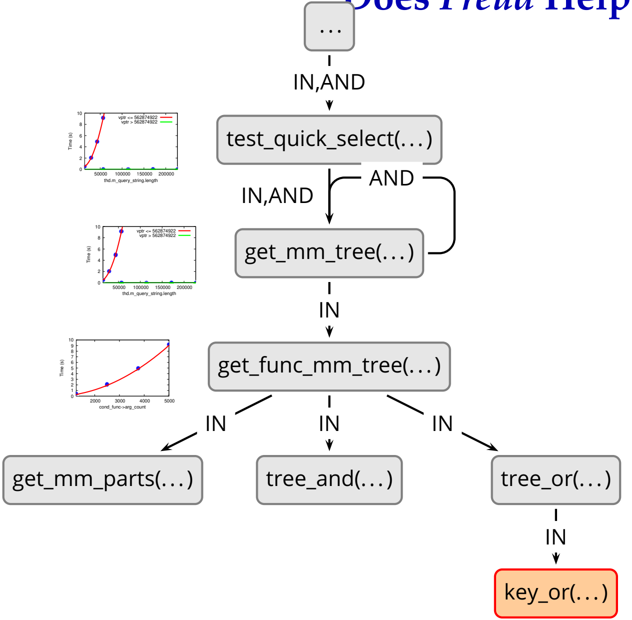



```
test_quick_select.time(thd, cond) {
uint len = thd->m_query_string.len;
uint vptr = cond->_vptr.Parse_tree_node_tmpl;
[vptr <= 562874922]
Norm(467533 - 50.21*len + 0.0036*len^2,282711.59);
[vptr > 562874922]
Norm(-53.603 + 0.057*len, 157.57);
}
```

Does *Freud* Help Finding Bugs?

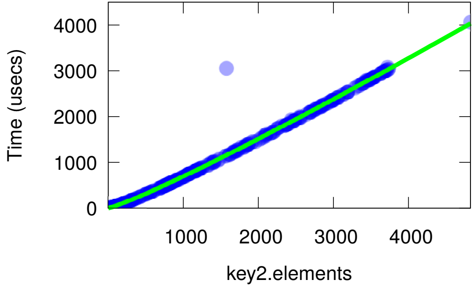


Does *Freud* Help Finding Bugs?



Does *Freud* Help Finding Bugs?

```
key_or(RANGE_OPT_PARAM *param, SEL_ROOT *key1, SEL_ROOT *key2);
```




```
key_or.time(key2) {
uint e = key2->elements;
Norm(-0.276 + 0.073*e + 0.062*e*log(e), 2.24);
}
```

Conclusion

- ***Performance Annotations***
  - probabilistic representation of expected performance
  - account for different modalities in the behavior

- ***Performance Annotations***
  - probabilistic representation of expected performance
  - account for different modalities in the behavior

- ***Freud***
  - automatically creates *performance annotations* for C/C++ programs
  - *https://github.com/usi-systems/freud*

Conclusion

- ***Performance Annotations***
  - probabilistic representation of expected performance
  - account for different modalities in the behavior
- ***Freud***
  - automatically creates *performance annotations* for C/C++ programs
  - *https://github.com/usi-systems/freud*
- We shown that performance annotations can be used in different real world cases
  - documentation
  - performance assertions
  - a tool to find performance bugs

Conclusion

- ***Performance Annotations***
  - probabilistic representation of expected performance
  - account for different modalities in the behavior
- ***Freud***
  - automatically creates *performance annotations* for C/C++ programs
  - *https://github.com/usi-systems/freud*
- We shown that performance annotations can be used in different real world cases
  - documentation
  - performance assertions
  - a tool to find performance bugs
- Future work
  - prediction
  - composition

Performance Annotations for Complex Software Systems

Daniele Rogora[*] Antonio Carzaniga[*] Amer Diwan[$] Matthias Hauswirth[*]
Robert Soulé[†]

[*]USI, Switzerland [†]Yale University, USA [$]Google, USA

EuroSys'20