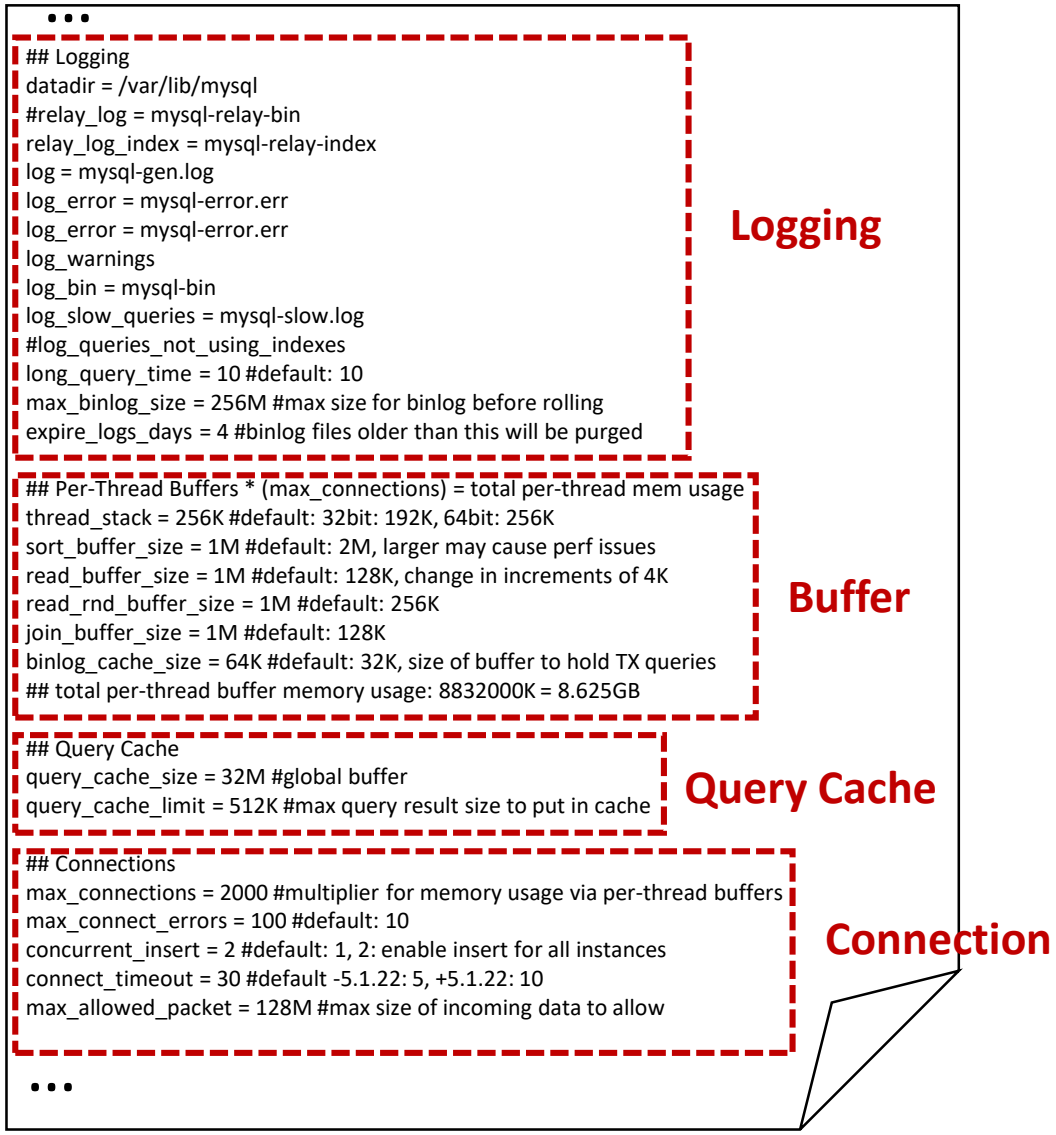
Automated Reasoning and Detection of Specious Configuration in Large Systems with Symbolic Execution

Yigong Hu, Gongqi Huang, Peng Huang
Johns Hopkins University

OSDI 2020

Setting Configuration Is Difficult



Setting Configuration Is Difficult

...

```
## Logging
datadir = /var/lib/mysql
#relay_log = mysql-relay-bin
relay_log_index = mysql-relay-index
log = mysql-gen.log
log_error = mysql-error.err
log_error = mysql-error.err
log_warnings
log_bin = mysql-bin
log_slow_queries = mysql-slow.log
#log_queries_not_using_indexes
long_query_time = 10 #default: 10
max_binlog_size = 256M #max size for binlog before rolling
expire_logs_days = 4 #binlog files older than this will be purged
```

Logging

```
## Per-Thread Buffers * (max_connections) = total per-thread mem usage
thread_stack = 256K #default: 32bit: 192K, 64bit: 256K
sort_buffer_size = 1M #default: 2M, larger may cause perf issues
read_buffer_size = 1M #default: 128K, change in increments of 4K
read_rnd_buffer_size = 1M #default: 256K
join_buffer_size = 1M #default: 128K
binlog_cache_size = 64K #default: 32K, size of buffer to hold TX queries
## total per-thread buffer memory usage: 8832000K = 8.625GB
```

Buffer

```
## Query Cache
query_cache_size = 32M #global buffer
query_cache_limit = 512K #max query result size to put in cache
```

Query Cache

```
## Connections
max_connections = 2000 #multiplier for memory usage via per-thread buffers
max_connect_errors = 100 #default: 10
concurrent_insert = 2 #default: 1, 2: enable insert for all instances
connect_timeout = 30 #default -5.1.22: 5, +5.1.22: 10
max_allowed_packet = 128M #max size of incoming data to allow
```

Connection

...

Misconfiguration ≠ Invalid Configuration

- **Misconfiguration detection (PeerPressure[OSDI'04], Pcheck[OSDI'16])**
  - Invalid setting
  - Introduced by average users

Misconfiguration ≠ Invalid Configuration

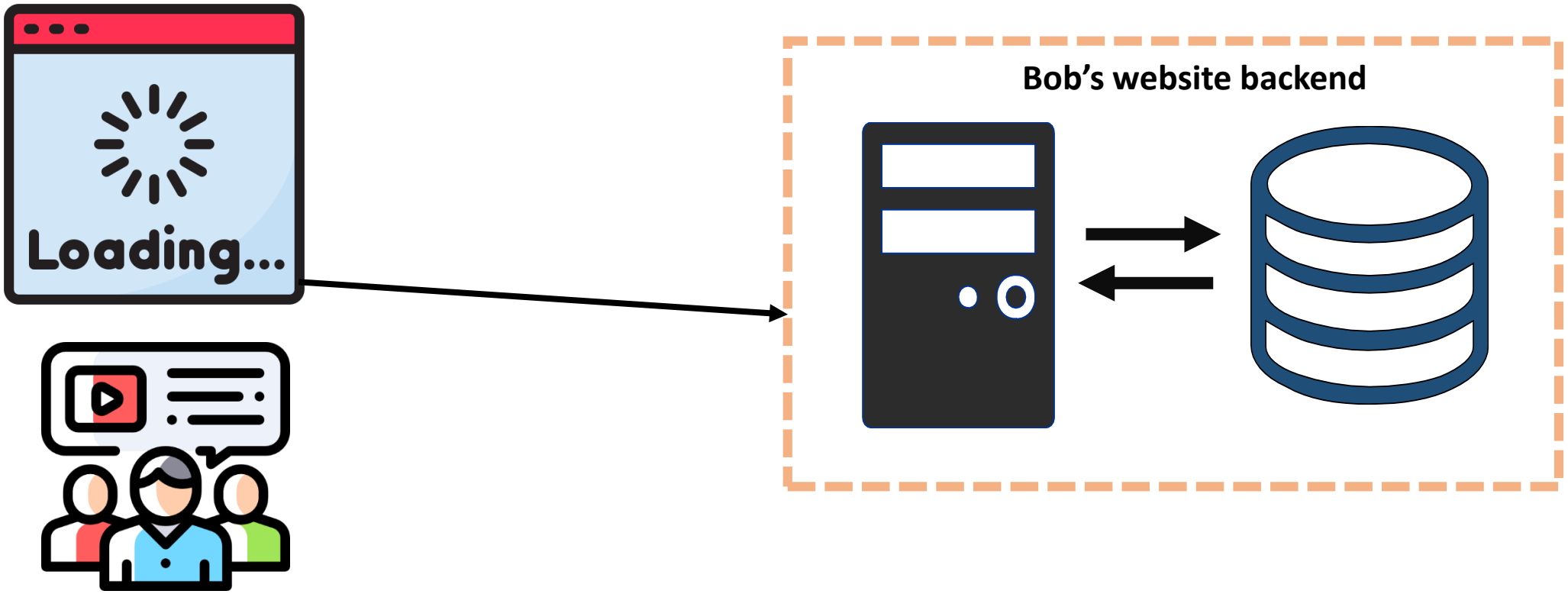
- **Misconfiguration detection (PeerPressure[OSDI'04], Pcheck[OSDI'16])**
  - Invalid setting
  - Introduced by average users

- **Many misconfiguration are valid setting**
  - 46.3% ~61.9% of misconfigurations have perfectly legal parameters*
  - The effect are hard to predict even for experts
  - Cause severe **performance** issue in production

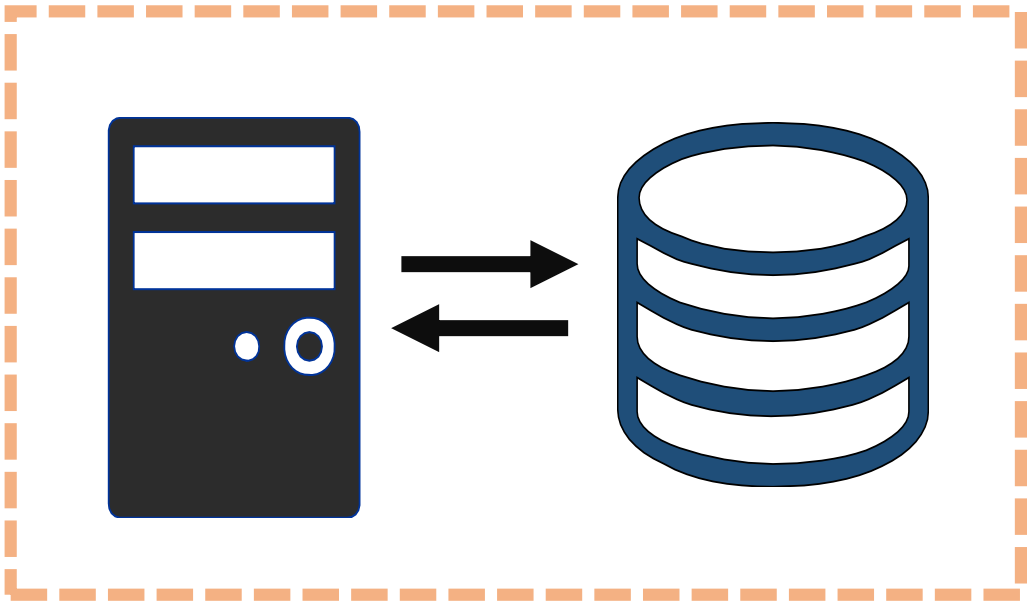
For simplicity, we call them specious configuration

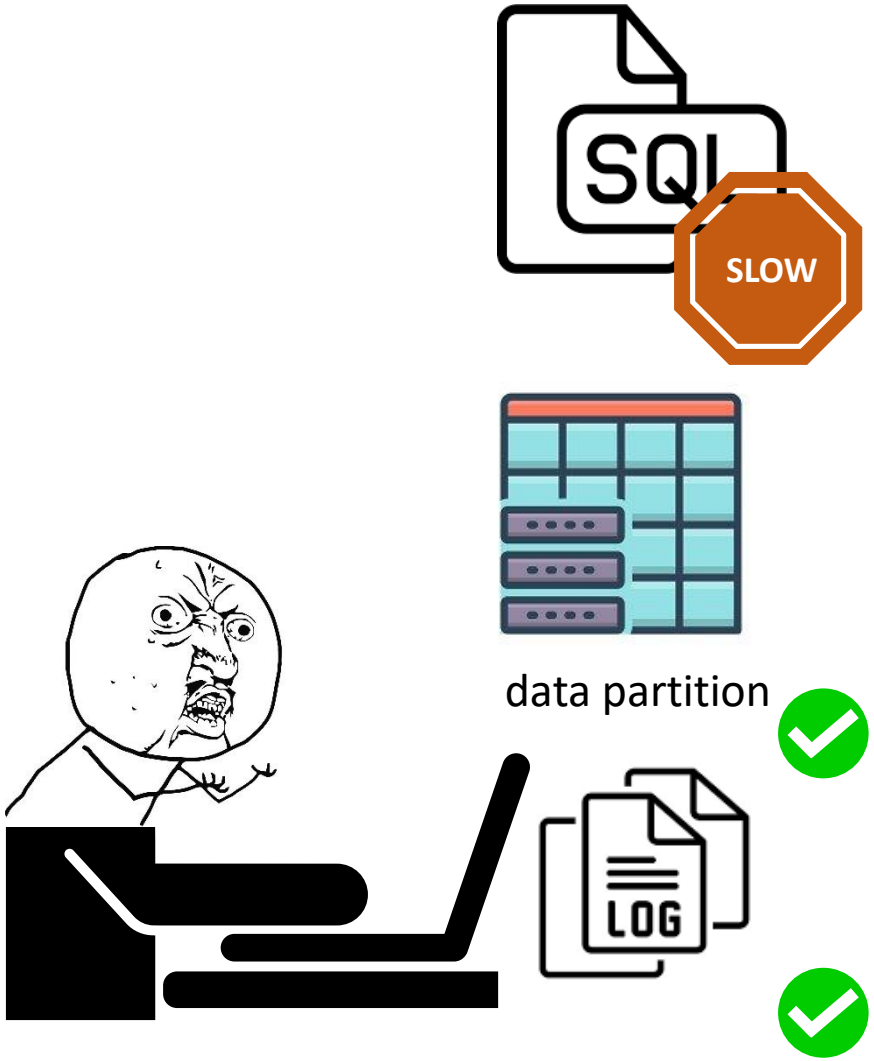
*: An Empirical Study on Configuration Errors in Commercial and Open Source Systems. SOSP'11

An Example Specious-Configuration Incident

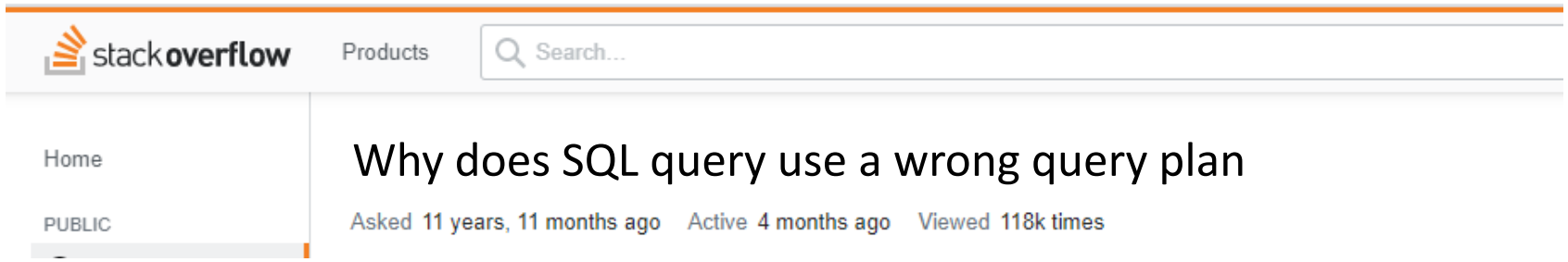


An Example Specious-Configuration Incident



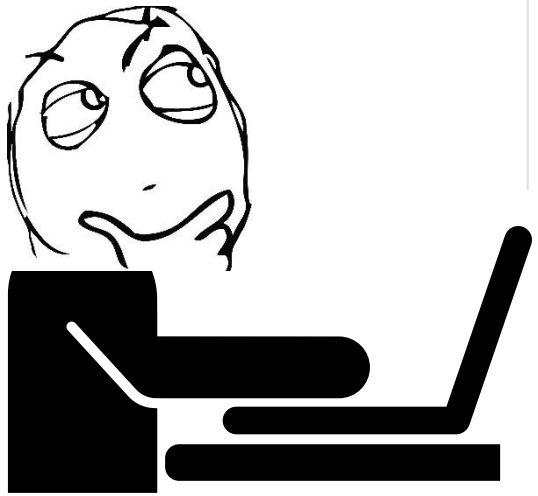


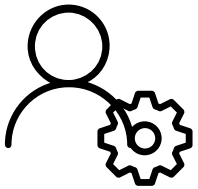
An Example Specious-Configuration Incident



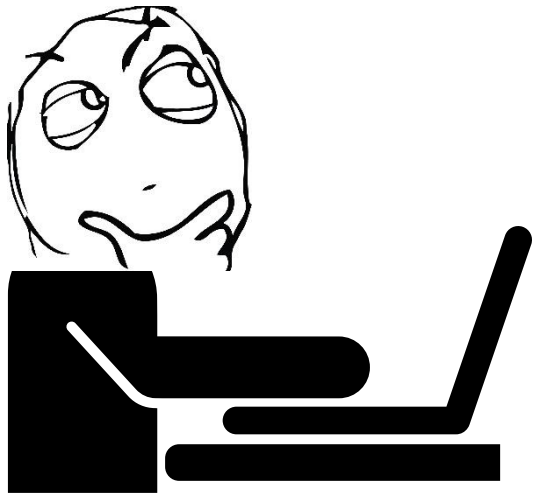
105

This problem typically happens when the estimated cost of an index scan is too high and doesn't correctly reflect reality.



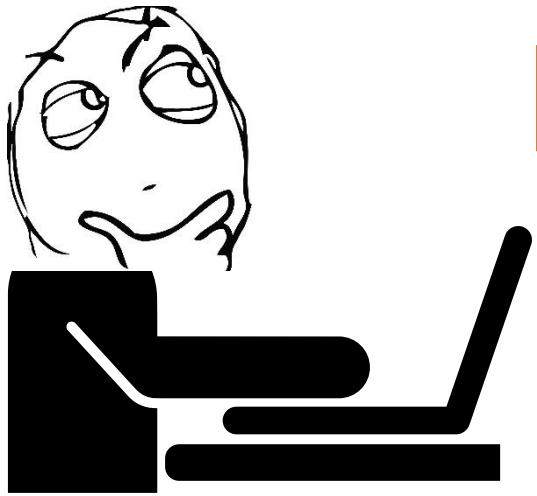


An Example Specious-Configuration Incident



```
…
#QUERY TUNING
enable_bitmapscan = on
enable_hashagg = on
enable_hashjoin = on
enable_indexscan = on
enable_indexonlyscan = on
enable_material = on
enable_mergejoin = on
enable_nestloop = on
enable_parallel_append = on
enable_seqscan = on
enable_sort = on

# - Planner Cost Constants
seq_page_cost = 1.0                     # measured on an arbitrary scale
random_page_cost = 1.0                  # same scale as above
cpu_tuple_cost = 0.01                   # same scale as above
cpu_index_tuple_cost = 0.005            # same scale as above
cpu_operator_cost = 0.0025              # same scale as above
parallel_tuple_cost = 0.1               # same scale as above
parallel_setup_cost = 1000.0            # same scale as above
jit_above_cost = 100000                 # perform JIT compilation
jit_inline_above_cost = 500000          # inline small functions
jit_optimize_above_cost = 500000        # use expensive JIT optimizations
min_parallel_table_scan_size = 8MB
min_parallel_index_scan_size = 512kB
effective_cache_size = 4GB
 …
```

An Example Specious-Configuration Incident



```
…
#QUERY TUNING
enable_bitmapscan = on
enable_hashagg = on
enable_hashjoin = on
enable_indexscan = on
enable_indexonlyscan = on
enable_material = on
enable_mergejoin = on
enable_nestloop = on
enable_parallel_append = on
enable_seqscan = on
enable_sort = on

# - Planner Cost Constants
seq_page_cost = 1.0                     # measured on an arbitrary scale
random_page_cost = 1.0                  # same scale as above
cpu_tuple_cost = 0.01                   # same scale as above
cpu_index_tuple_cost = 0.005            # same scale as above
cpu_operator_cost = 0.0025              # same scale as above
parallel_tuple_cost = 0.1               # same scale as above
parallel_setup_cost = 1000.0            # same scale as above
jit_above_cost = 100000                 # perform JIT compilation
jit_inline_above_cost = 500000          # inline small functions
jit_optimize_above_cost = 500000        # use expensive JIT optimizations
min_parallel_table_scan_size = 8MB
min_parallel_index_scan_size = 512kB
effective_cache_size = 4GB
…
```

Specious Configuration Is Prevalent

COMPANY ANNOUNCEMENTS

Today's outage for several Google services

Ben Treynor
VP, Engineering

Published Jan 24, 2014

Earlier today, most Google users who use logged-in services like Gmail, Google+, Calendar and Documents found they were unable to access those services for approximately 25 minutes. For about 10 percent of users, the problem persisted for as much as 30 minutes longer. Whether the effect was brief or lasted the better part of an

More Details on Today's Outage

September 24, 2010 at 8:29 AM

Early today Facebook was down or unreachable for many of you for approximately 2.5 hours. This is the worst outage we've had in over four years, and we wanted to first of all apologize for it. We also wanted to provide much more technical detail on what happened and share

Summary of the December 24, 2012 Amazon ELB Service Event in the

We would like to share more details with our customers about the event that occurred with the Amazon Elastic Load Balancing Service
While the service disruption only affected applications using the ELB service (and only a fraction of the ELB load balancers were affecte
impact for a prolonged period of time.

Google apologizes for service outage, reveals a 10% drop in YouTube views

A simple misconfiguration caused the outage

By Rounak Jain June 5, 2019, 1:13 p.m.

A "Server Misconfiguration" Was Behind the Facebook Outage

AlphaAtlas · Mar 15, 2019 · cloud facebook service uptime

Ma

Joi
Me

Thread

Lorin Hochstein E_TOO_SPOOKY
@lhochstein

And if config changes in cloud infrastructure systems contributing to incidents is your thing, we also have one from Google this week (@SREWeekly is the gift that keeps on giving):
status.cloud.google.com/incident/zall/...

11:34 AM · Oct 5, 2020 · Twitter Web App

What Is Missing From Current Tool?

- **Black-box testing is experimental**
  - Limited code coverage
  - Tailored to testing environment, specific configuration and input

- **Administrators have more questions:**
  - What happens if I change this setting from X to Y?
  - How would this setting perform with 100 nodes?
  - If my workload changes to mostly read-only, is this setting acceptable?
  - I plan to upgrade from HDD to SSD, should I update the config?
  - …

To tackle specious configuration, we need an analytical approach to systematically reason about the performance effect of configuration

Our Solution: Violet

S1: Explore performance effect with symbolic execution

- Make configuration and input as one type of symbolic input
- Symbolic explore the system *code* path with symbolic config & input
- Derive performance impact model for each configuration

S2: Given concrete input, parameters, env info

- Answer admins' questions
- Violet checker detects specious configuration based on the impact model

Outline

- Motivation
- Specious Configuration Code Patterns
- Violet Overview
- Evaluation

Code Pattern 1: Costly Operation

```
int write_row() {
   if (autocommit) {
      …
      trx_commit_complete();
   } else {
      trx_mark_sql_stat_end();
   }
}

ulint trx_commit_complete() {
   if (flush_at_trx_commit==1) {
      log_group_write_buf();
      fil_flush();
   } else if (flush_at_trx_commit==2) {
      log_group_write_buf();
   } else {
   /* do nothing */
   }
}
```

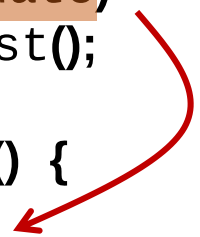
- Some expensive operations is executed in one branch

Code Pattern 2: Additional Synchronization

```
void mysql_parse(THD *thd) {
  if (send_result_to_client(thd) <= 0) {
    mysql_execute_command(thd);
  }
}
int mysql_execute_command(THD *thd) {
  case SQLCOM_SELECT:
    open_and_lock_tables(thd, all_tables);
    break;
  case SQLCOM_LOCK_TABLES:
    lock_tables_open_and_lock_tables(thd);
    if (query_cache_wlock_invalidate)
      invalidate_query_block_list();
}
void invalidate_query_block_list() {
  free_query(list_root->block());
}
```

Code Pattern 2: Additional Synchronization

```
void mysql_parse(THD *thd) {
  if (send_result_to_client(thd) <= 0) {
    mysql_execute_command(thd);
  }
}
int mysql_execute_command(THD *thd) {
  case SQLCOM_SELECT:
    open_and_lock_tables(thd, all_tables);
    break;
  case SQLCOM_LOCK_TABLES:
    lock_tables_open_and_lock_tables(thd);
    if (query_cache_wlock_invalidate)
      invalidate_query_block_list();
}
void invalidate_query_block_list() {
  free_query(list_root->block());
}
```



free query cache

Code Pattern 2: Additional Synchronization

```
void mysql_parse(THD *thd) {
  if (send_result_to_client(thd) <= 0) {
    mysql_execute_command(thd);
  }
}
int mysql_execute_command(THD *thd) {
  case SQLCOM_SELECT:
    open_and_lock_tables(thd, all_tables);
    break;
  case SQLCOM_LOCK_TABLES:
    lock_tables_open_and_lock_tables(thd);
    if (query_cache_wlock_invalidate)
      invalidate_query_block_list();
}
void invalidate_query_block_list() {
  free_query(list_root->block());
}
```

free query cache

- Lead to additional table lock

Code Pattern 3: Slow Execution Flow

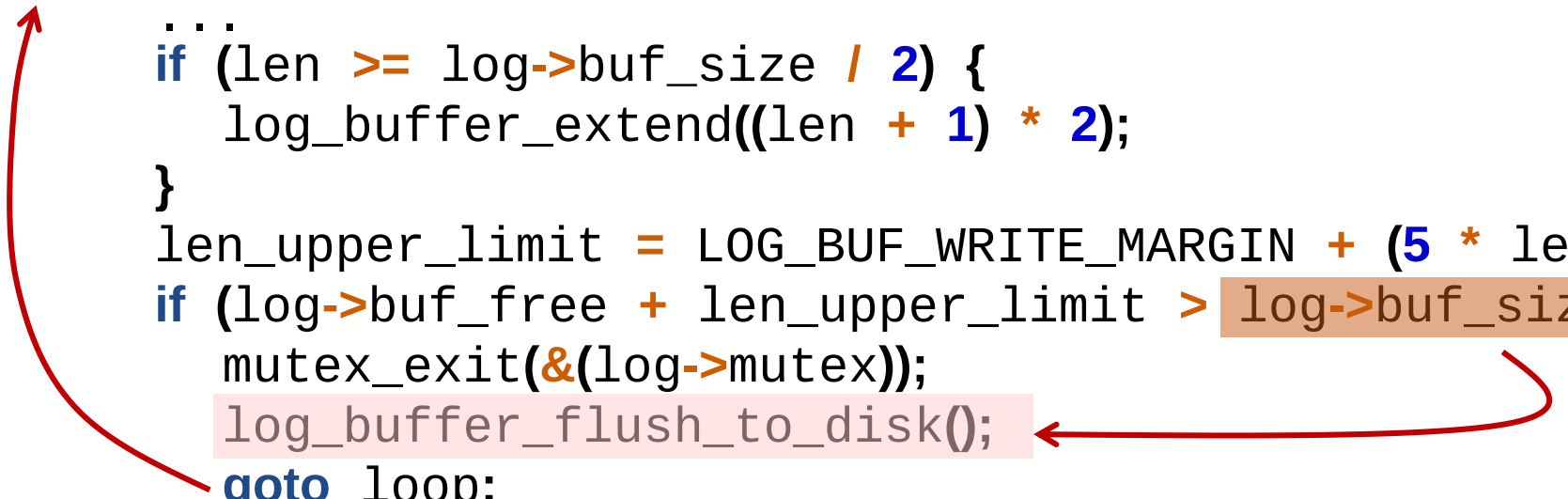
```
void mysql_parse(THD *thd) {
  if (send_result_to_client(thd) <= 0) {
    mysql_execute_command(thd);
  }
}
int mysql_execute_command(THD *thd) {
  case SQLCOM_SELECT:
    open_and_lock_tables(thd, all_tables);
    break;
  case SQLCOM_LOCK_TABLES:
    lock_tables_open_and_lock_tables(thd);
    if (query_cache_wlock_invalidate)
      invalidate_query_block_list();
}
void invalidate_query_block_list() {
  free_query(list_root->block());
}
```

free query cache

- Lead to slow execution flow

Code Pattern 4: Frequent Crossing Threshold

```
uint64_t log_reserve_and_open(uint len) {
  ...
loop:
    ...
    if (len >= log->buf_size / 2) {
      log_buffer_extend((len + 1) * 2);
    }
    len_upper_limit = LOG_BUF_WRITE_MARGIN + (5 * len) / 4;
    if (log->buf_free + len_upper_limit > log->buf_size) {
      mutex_exit(&(log->mutex));
      log_buffer_flush_to_disk();
      goto loop;
    }
}
```



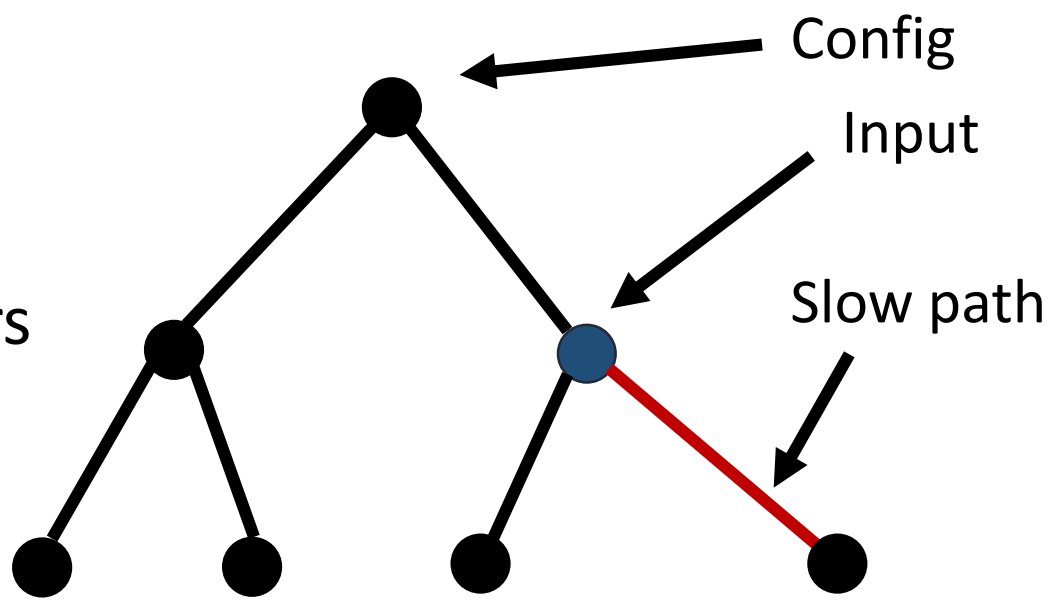
- Costly operation being frequently triggered the costly operation

Static Analysis?

- **The four patterns are high-level characterizations**
  - Mapping them to specific code requires a lot of domain knowledge
- **Patterns are incomplete**
  - Other patterns and many variants
- **Fundamental limitations**
  - Infeasible paths
  - Performance is hard to be estimated statically

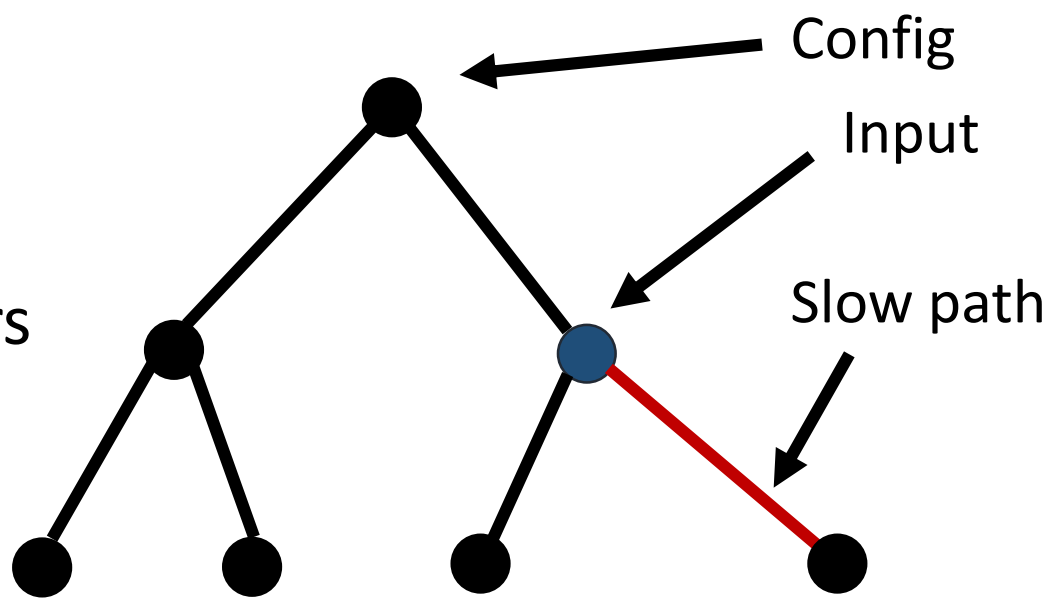
Parameter Affects Execution Flows

- **A general characteristic is…**
  - Different parameter causes different execution code path
  - Some path is extremely slower than others
  - Context-dependency



Parameter Affects Execution Flows

- **A general characteristic is…**
  - Different parameter causes different execution code path
  - Some path is extremely slower than others
  - Context-dependency



Detecting specious configuration = finding slow execution path + deducing triggering condition

Symbolic Execution

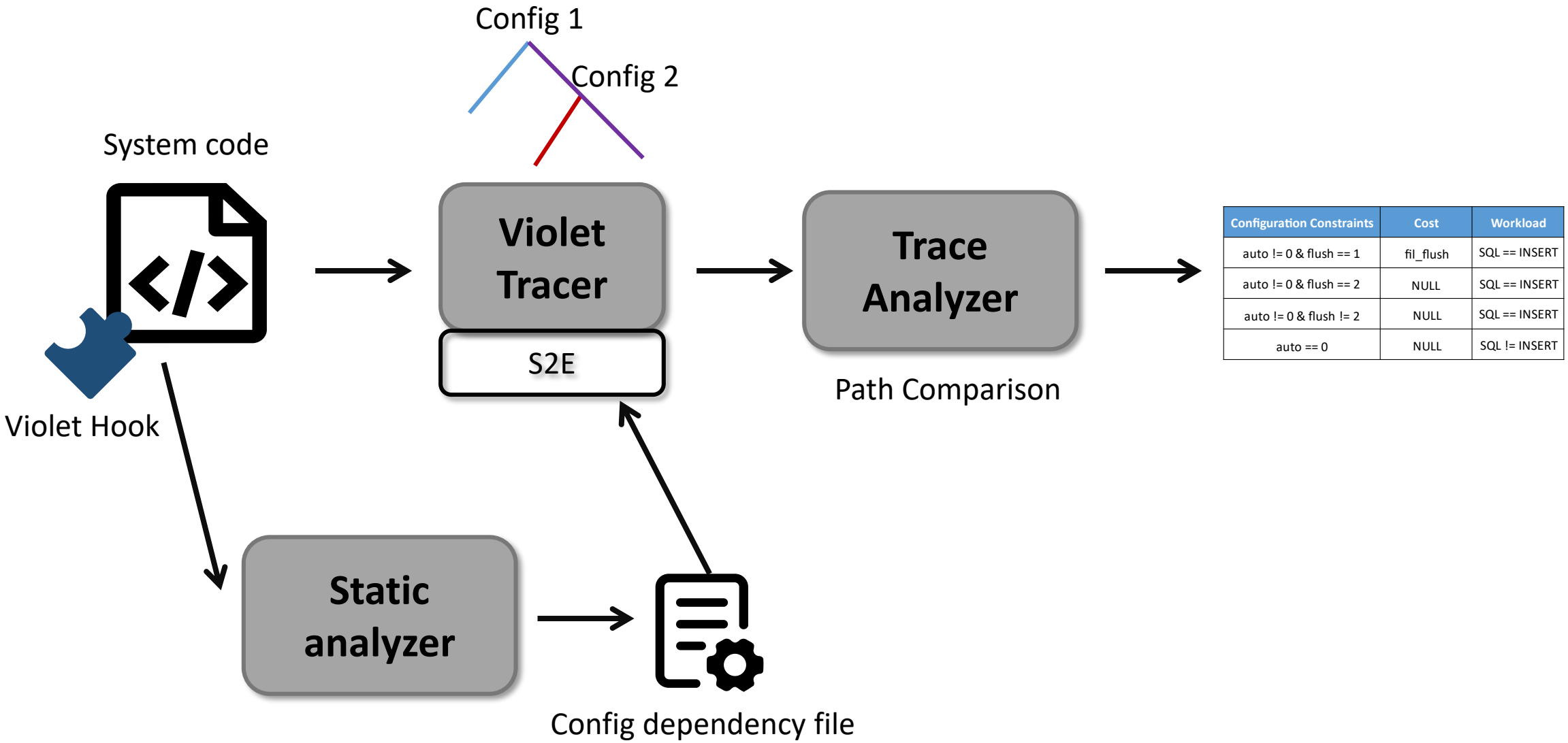
- **Violet uses symbolic execution to find many slow paths and deduce their triggering conditions**

- **Advantages**
  - Analyze system code without being limited by code patterns
  - Explored paths are feasible in native execution
  - Measure concrete performance from execution

Outline

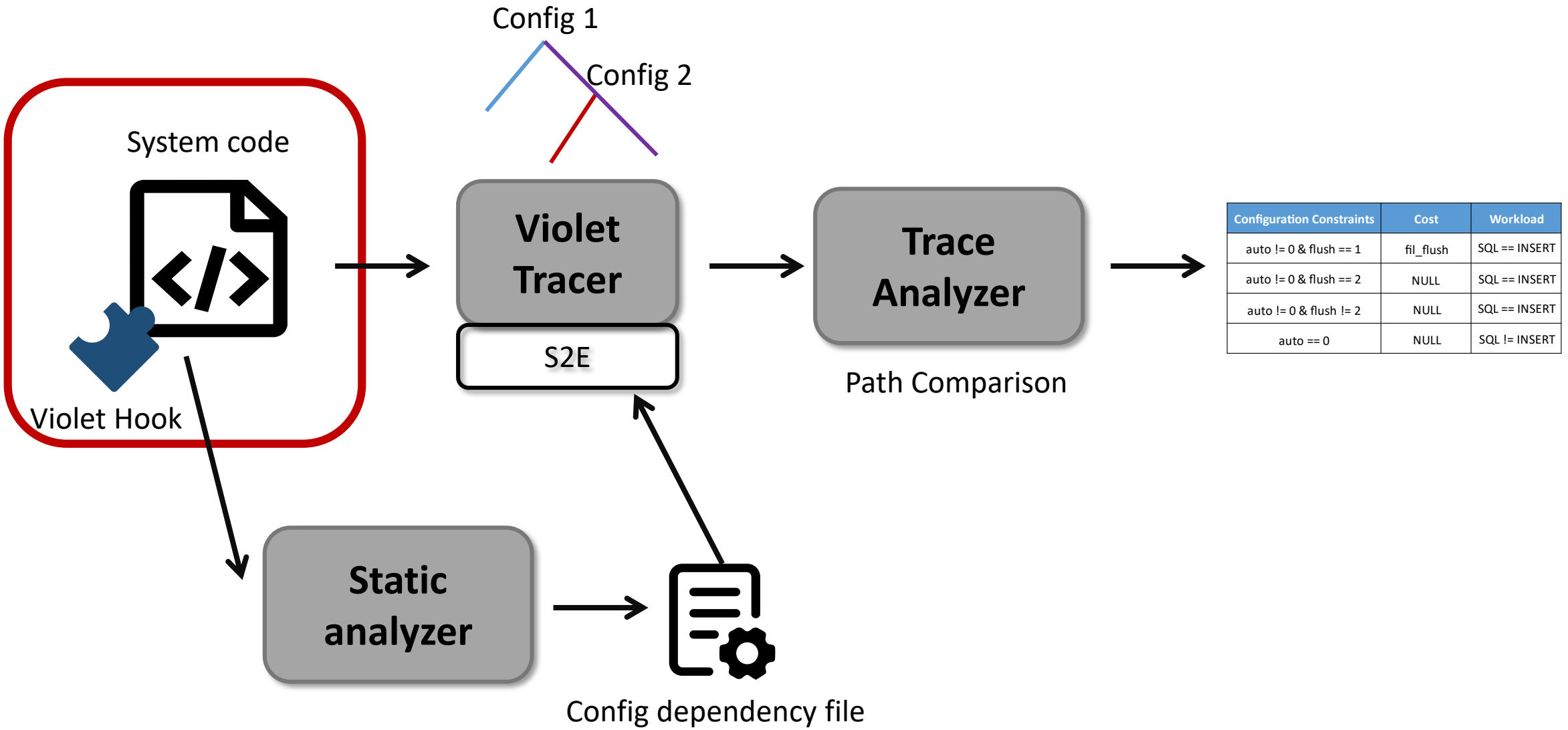
- Motivation
- Specious Configuration Code Patterns
- **Violet Overview**
- **Evaluation**

Violet Overview



Configuration Constraints	Cost	Workload
auto != 0 & flush == 1	fil_flush	SQL == INSERT
auto != 0 & flush == 2	NULL	SQL == INSERT
auto != 0 & flush != 2	NULL	SQL == INSERT
auto == 0	NULL	SQL != INSERT

Violet Overview



Configuration Constraints	Cost	Workload
auto != 0 & flush == 1	fil_flush	SQL == INSERT
auto != 0 & flush == 2	NULL	SQL == INSERT
auto != 0 & flush != 2	NULL	SQL == INSERT
auto == 0	NULL	SQL != INSERT

How to Make Configuration Symbolic

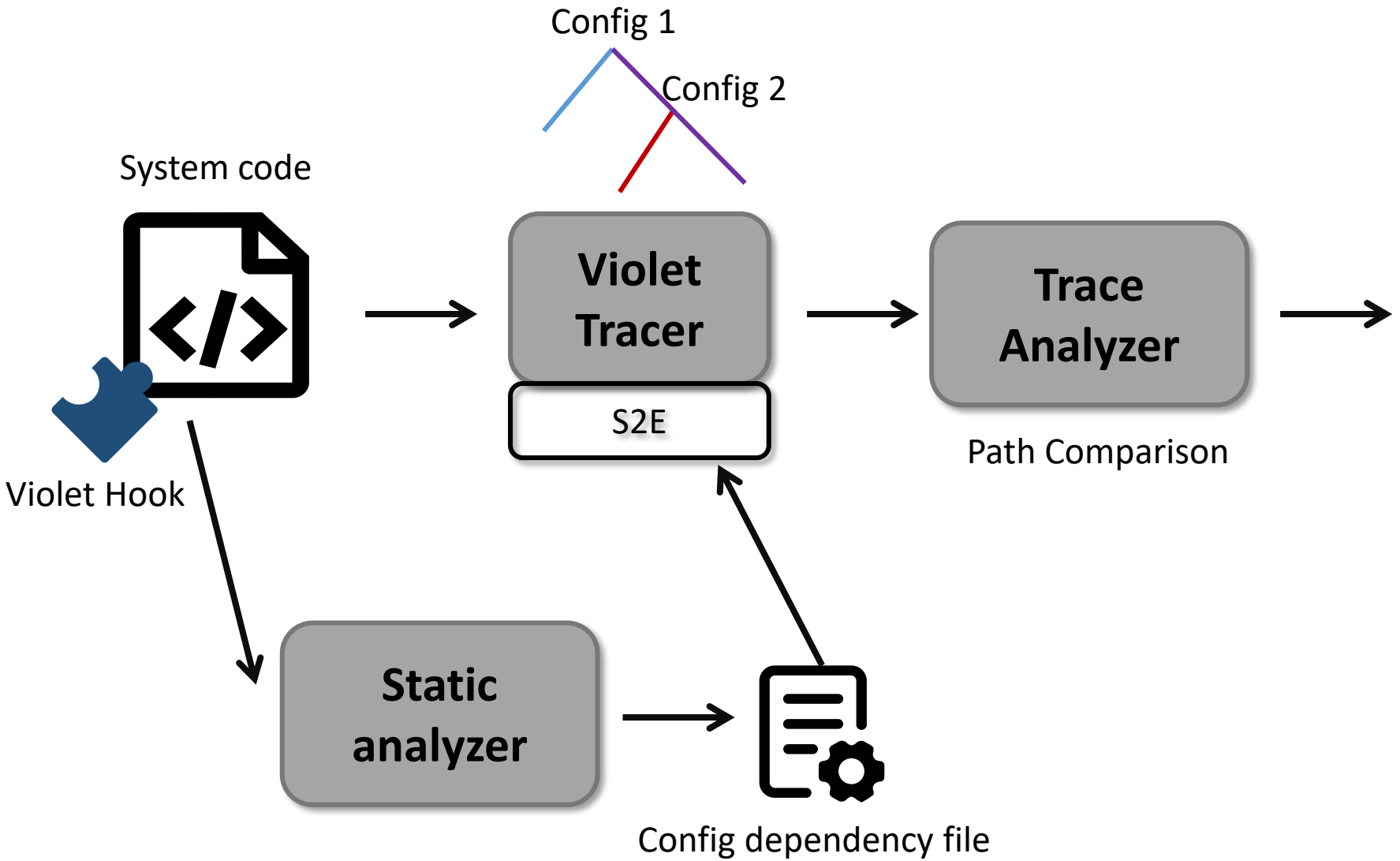
- **Making configuration file symbolic**
  - Path explosion due to the parser
- **Observation:**
  - System usually keeps a **dictionary** to map configuration to variable
  - But we also need variable type, range and default value to make it symbolic
- **Our approach:**
  - Insert a hook to enumerate config variables and make them symbolic

Hooking API

- Insert after parse function
- Iterate all the variable
- Implement make_symbolic for each variable type

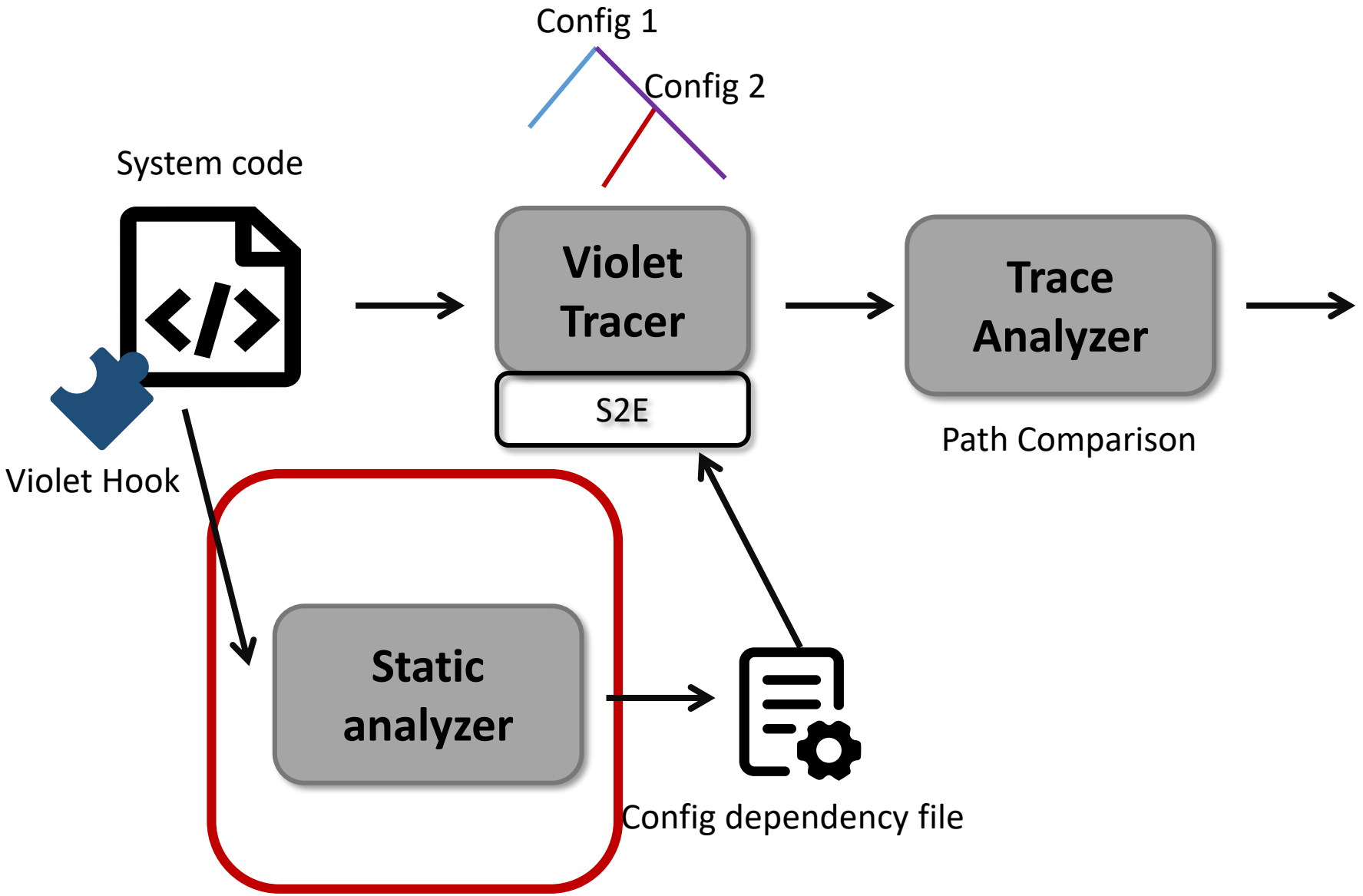
```
static int get_options(int *argc_ptr, char ***argv_ptr)
{
  my_init_dynamic_array(&all_options, sizeof(my_option));
  for (opt= my_long_options; opt < my_options_end; opt++) {
    insert_dynamic(&all_options, (uchar*) opt);
    …
  }
+ violet_make_mysql_options_symbolic();
  return 0;
}

+ void violet_make_mysql_options_symbolic()
+ {
+  for (sys_var *var=all_sys_vars.first; var; var= var->next)
+    if (is_config_in_targets(var->name.str))
+      var->make_symbolic();
+ }
```

Violet Overview



Configuration Constraints	Cost	Workload
auto != 0 & flush == 1	fil_flush	SQL == INSERT
auto != 0 & flush == 2	NULL	SQL == INSERT
auto != 0 & flush != 2	NULL	SQL == INSERT
auto == 0	NULL	SQL != INSERT

Violet Overview



Configuration Constraints	Cost	Workload
auto != 0 & flush == 1	fil_flush	SQL == INSERT
auto != 0 & flush == 2	NULL	SQL == INSERT
auto != 0 & flush != 2	NULL	SQL == INSERT
auto == 0	NULL	SQL != INSERT

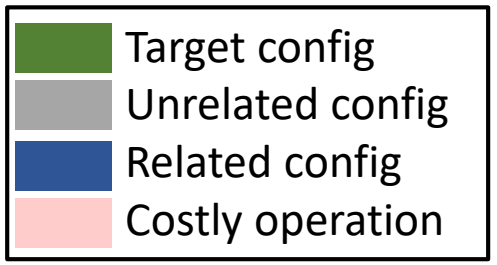
Which Configuration to Make Symbolic?

- **Making all configuration symbolic**
  - Too many configurations -> path explosion
  - Many paths waste time on irrelevant execution
  - A lot of path constraints are misleading

Making Irrelevant Configuration Symbolic

```
int write_row() {
  if (opt_c) {
    task1();
  } else {
    task2();
  }
  if (autocommit) {
    …
    trx_commit_complete();
  } else {
    trx_mark_sql_stat_end();
  }
}

ulint trx_commit_complete() {
  if (flush_at_trx_commit==1) {
    log_group_write_buf();
    fil_flush();
  } else if (flush_at_trx_commit==2) {
    log_group_write_buf();
  } else {
  /* do nothing */
  }
}
```

opt_c is irrelevant because it doesn't impact the autcommit

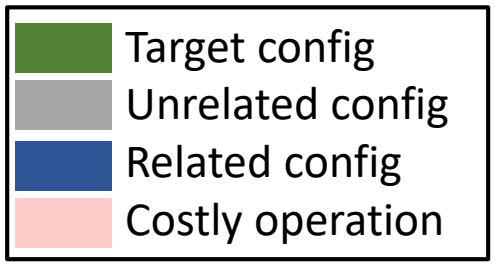


Making Irrelevant Configuration Symbolic

```
int write_row() {
  if (opt_c) {
    task1();
  } else {
    task2();
  }
  if (autocommit) {
    …
    trx_commit_complete();
  } else {
    trx_mark_sql_stat_end();
  }
}

ulint trx_commit_complete() {
  if (flush_at_trx_commit==1) {
    log_group_write_buf();
    fil_flush();
  } else if (flush_at_trx_commit==2) {
    log_group_write_buf();
  } else {
  /* do nothing */
  }
}
```

opt_c is irrelevant because it doesn't impact the autcommit

Wasting long time to reach target configuration



Making Irrelevant Configuration Symbolic

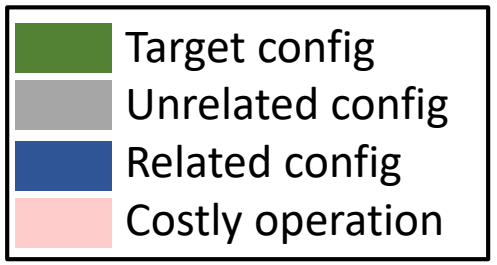
```
int write_row() {
  if (opt_c) {
    task1();
  } else {
    task2();
  }
  if (autocommit) {
    ...
    trx_commit_complete();
  } else {
    trx_mark_sql_stat_end();
  }
}

ulint trx_commit_complete() {
  if (flush_at_trx_commit==1) {
    log_group_write_buf();
    fil_flush();
  } else if (flush_at_trx_commit==2) {
    log_group_write_buf();
  } else {
  /* do nothing */
  }
}
```

opt_c is irrelevant because it doesn't impact the autcommit

Wasting long time to reach target configuration

misleading result

Constraints: autocommit!=0&flush==1&opt_c==1



Making Irrelevant Configuration Symbolic

```
int write_row() {
  if (opt_c) {
    task1();
  } else {
    task2();
  }
  if (autocommit) {
    ...
    trx_commit_complete();
  } else {
    trx_mark_sql_stat_end();
  }
}

ulint trx_commit_complete() {
  if (flush_at_trx_commit==1) {
    log_group_write_buf();
    fil_flush();
  } else if
    log_gr
  } else {
  /* do nothing */
  }
}
```

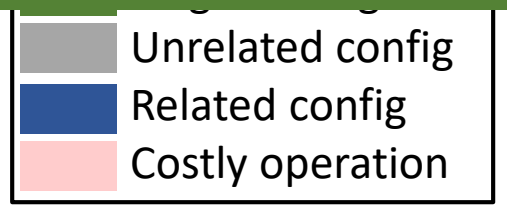
opt_c is irrelevant because it doesn't impact the autcommit

Wasting long time to reach target configuration

misleading result

Constraints: autocommit!=0&flush==1&opt_c==1

Only making *related* configuration symbolic



How to Find Related Configuration

- **A related config is in some execution flow of target config**

How to Find Related Configuration

- **A related config is in some execution flow of target config**
- **Control dependency**
  - X is control dependent on Y if X's execution depends on a test at Y

```
void main() {
  if (opty > 100)
    if (optx)
      init_x();
}
```

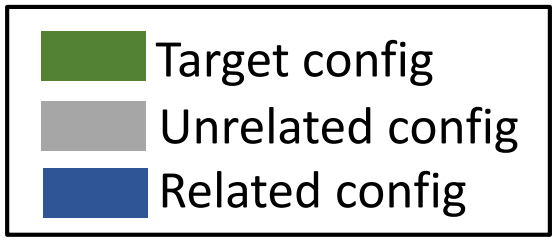
optx is control dependent on opty

optx,opty are related configurations

Relax Control Dependency

```
int write_row() {
 if (autocommit) {
     ...
     if (opt_c)
       trx_commit_complete();
  } else {
     trx_mark_sql_stat_end();
  }
}

ulint trx_commit_complete() {
  if (flush_at_trx_commit==1) {
     log_group_write_buf();
     fil_flush();
  } else if (flush_at_trx_commit==2) {
     log_group_write_buf();
  } else {
  /* do nothing */
  }
}
```

- *flush* is related to *autocommit*
- *flush* is not control dependent on autocommit because opt_c is between *autocommit* and *flush*

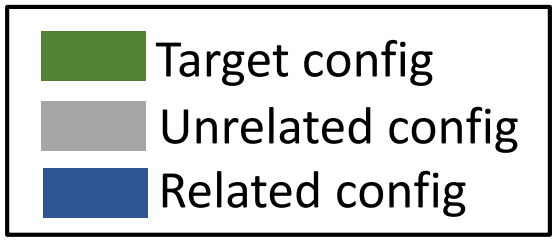


Relax Control Dependency

```
int write_row() {
 if (autocommit) {
     ...
     if (opt_c)
       trx_commit_complete();
  } else {
     trx_mark_sql_stat_end();
  }
}

ulint trx_commit_complete() {
  if (flush_at_trx_commit==1) {
     log_group_write_buf();
     fil_flush();
  } else if (flush_at_trx_commit==2) {
     log_group_write_buf();
  } else {
  /* do nothing */
  }
}
```

- *flush* is related to *autocommit*
- *flush* is not control dependent on autocommit because opt_c is between *autocommit* and *flush*

Relaxing the definition to X's execution depends on a test at Y and other parameters



Detecting Related Configuration

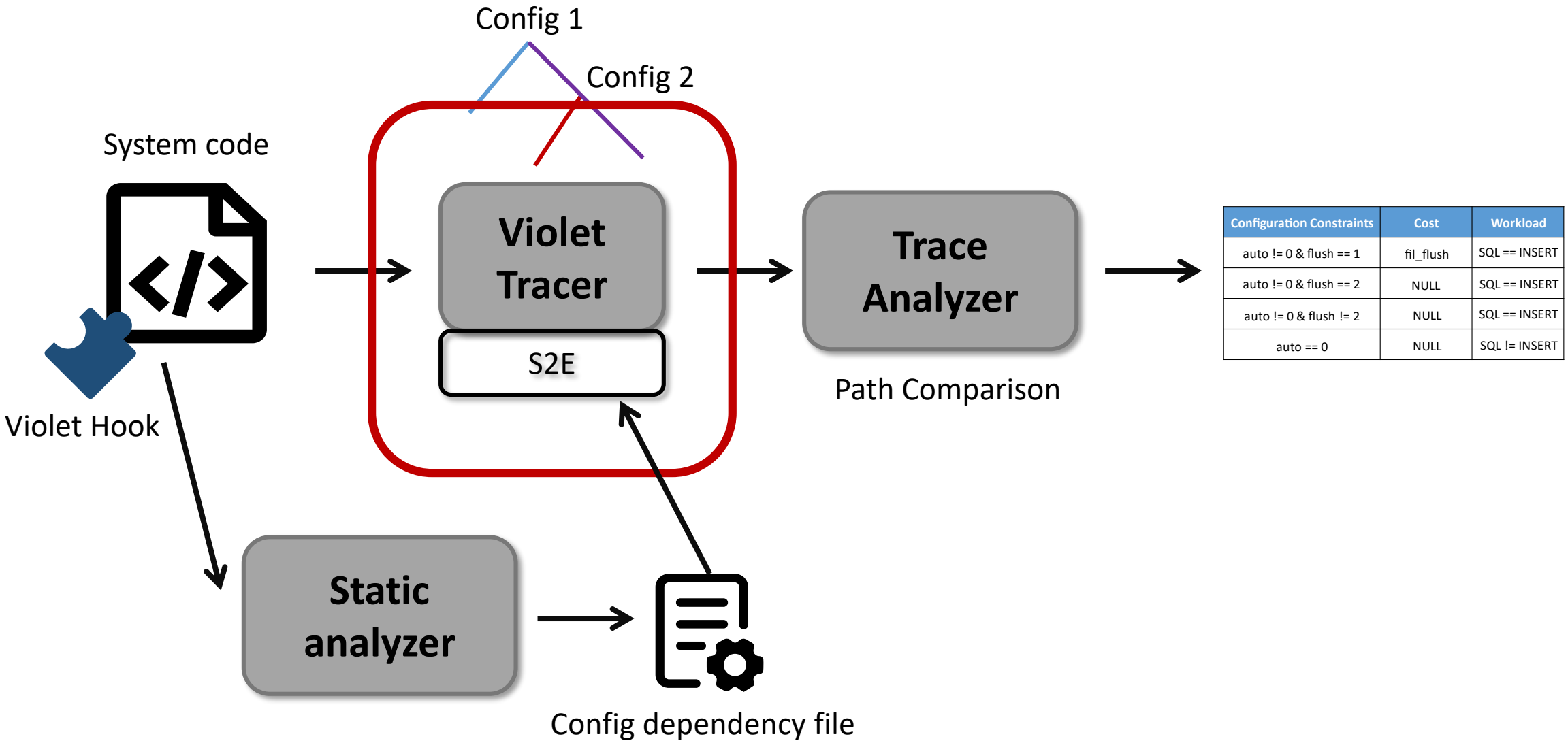
Algorithm 1: Compute related parameters

Func: `GetRelatedConfigs`
Input: $\mathcal{P}$: target program, $\mathcal{C}$: all parameter vars in $\mathcal{P}$
Output: $\mathcal{M}$: map from each parameter in $\mathcal{C}$ to the set of related parameters

```
M ← {}, es_map ← {}, ins_map ← {}
foreach p ∈ C do
    es ← GetEnablerConfig(p, P)
    es_map[p] ← es
    foreach q ∈ es do
        ins_map[q] ← ins_map[q] ∪ {p}
foreach p ∈ C do
    M[p] ← es_map[p] ∪ ins_map[p]
return M
```

- Find enabler parameter set
- Find influenced parameter set
- Union both parameter set as related parameter

Violet Overview



Configuration Constraints	Cost	Workload
auto != 0 & flush == 1	fil_flush	SQL == INSERT
auto != 0 & flush == 2	NULL	SQL == INSERT
auto != 0 & flush != 2	NULL	SQL == INSERT
auto == 0	NULL	SQL != INSERT

Lightweight Symbolic Tracer

- **Extensive profiling can incur too much overhead to the symbolic engine and cause inaccuracy of tracing result**

- **Principles of reducing tracing overhead**
  - Use Low-level signal if possible
  - Defer expensive computation to the end of each path
  - Avoid memory related operation

Trace Latency + Construct Call Chain

```
void f1() {
   …
   f2();
}
```

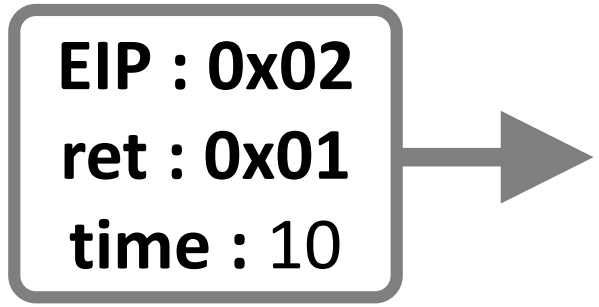
Call List

Return List

Trace Latency + Construct Call Chain

```
void f1() {
    ...
    f2();
}
```

call signal



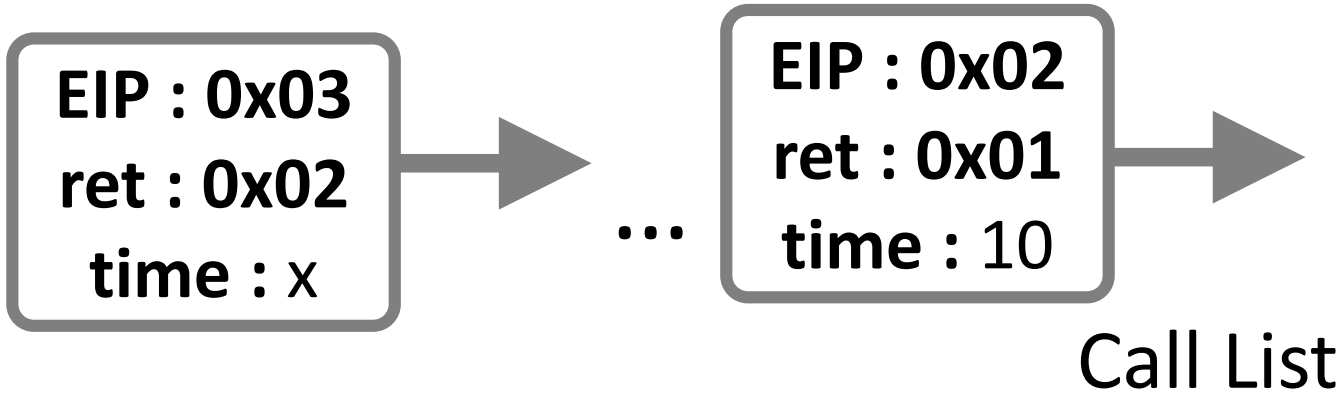
Call List

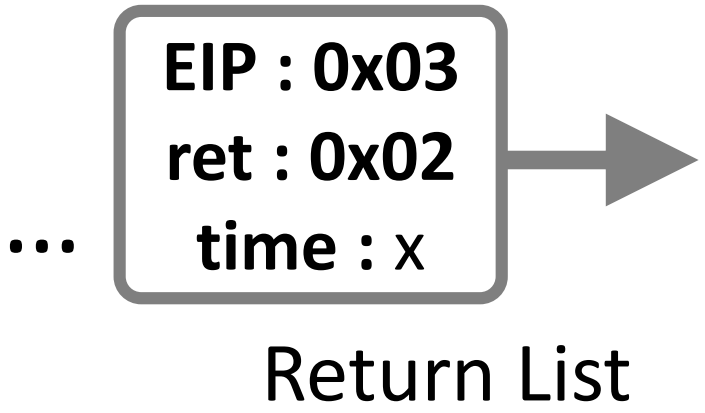
Return List

Trace Latency + Construct Call Chain

```
void f1() {
    ...
    f2();
}
```

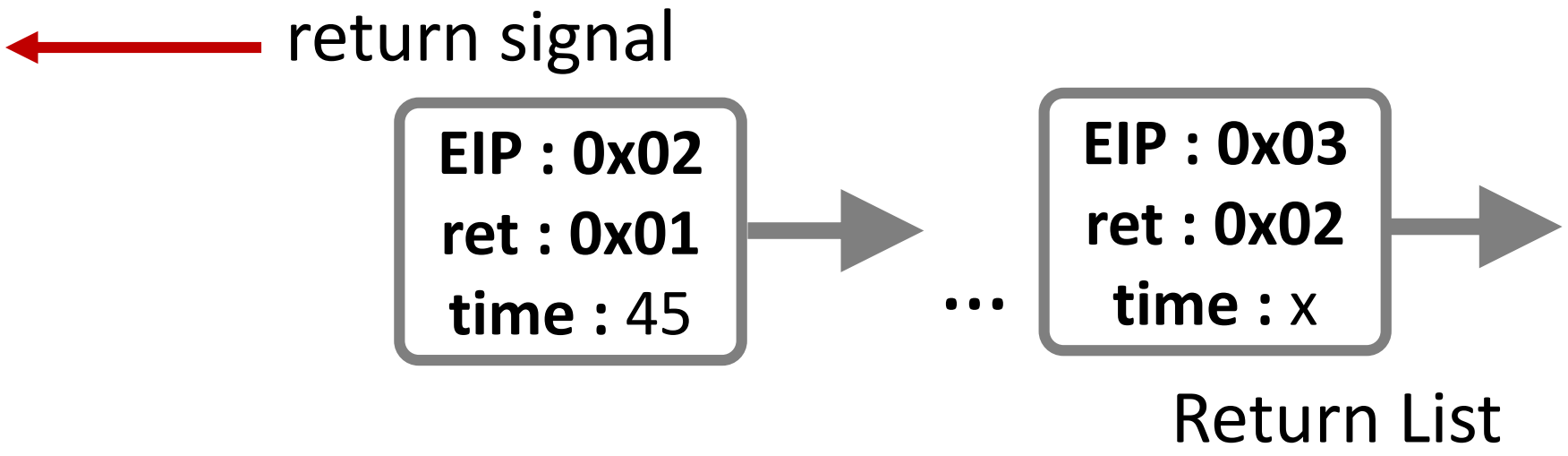
execution

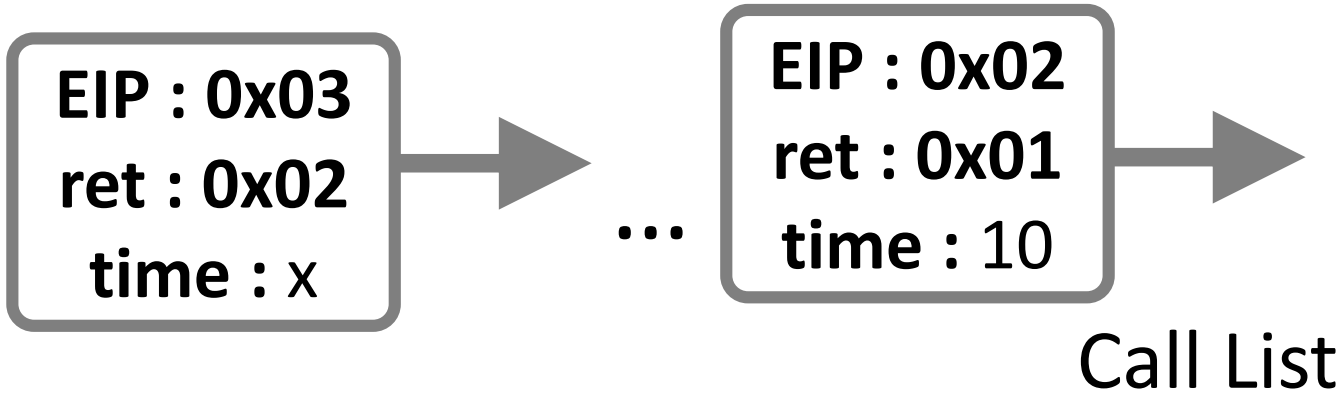




Trace Latency + Construct Call Chain

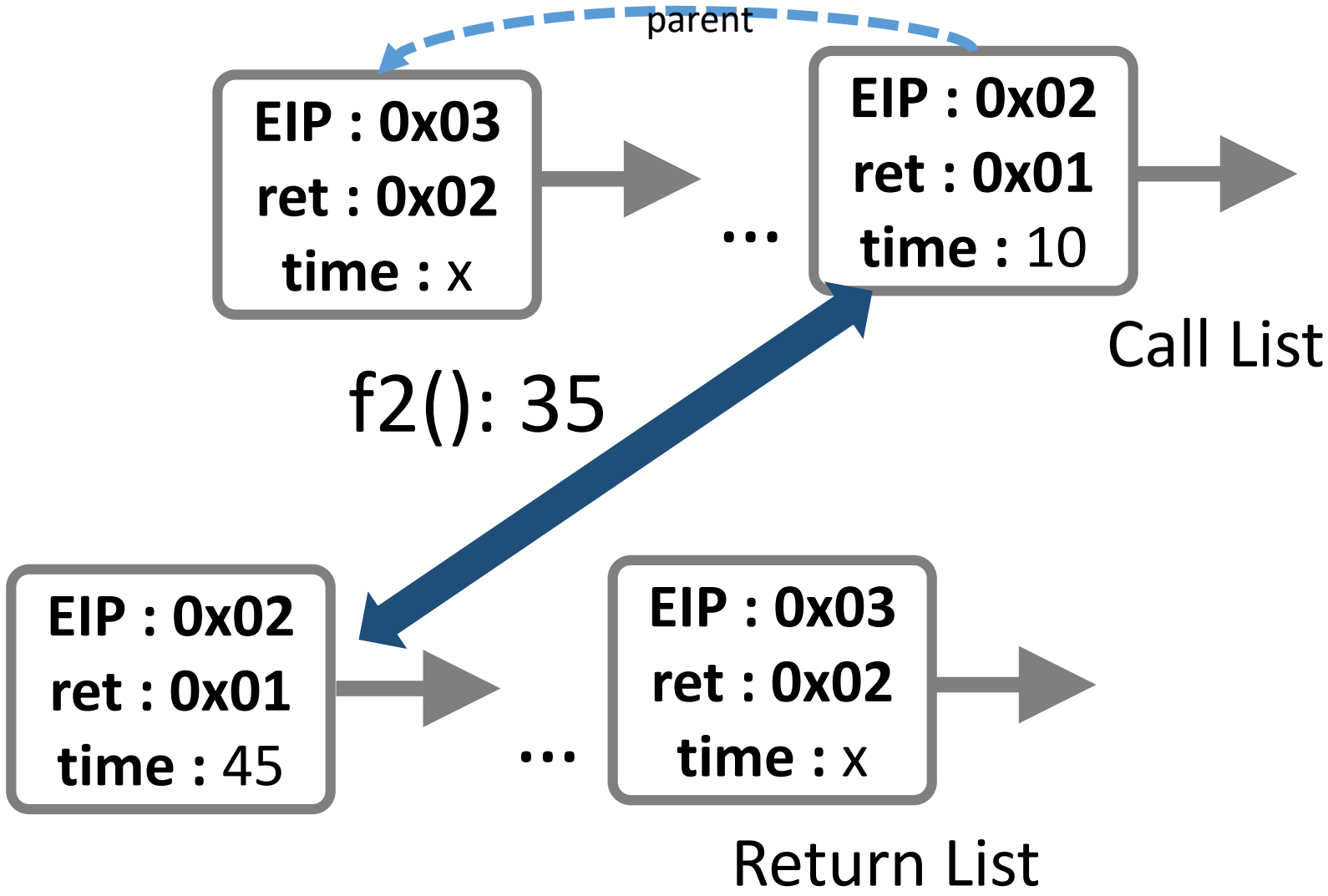
```
void f1() {
    ...
    f2();
}
```





Trace Latency + Construct Call Chain

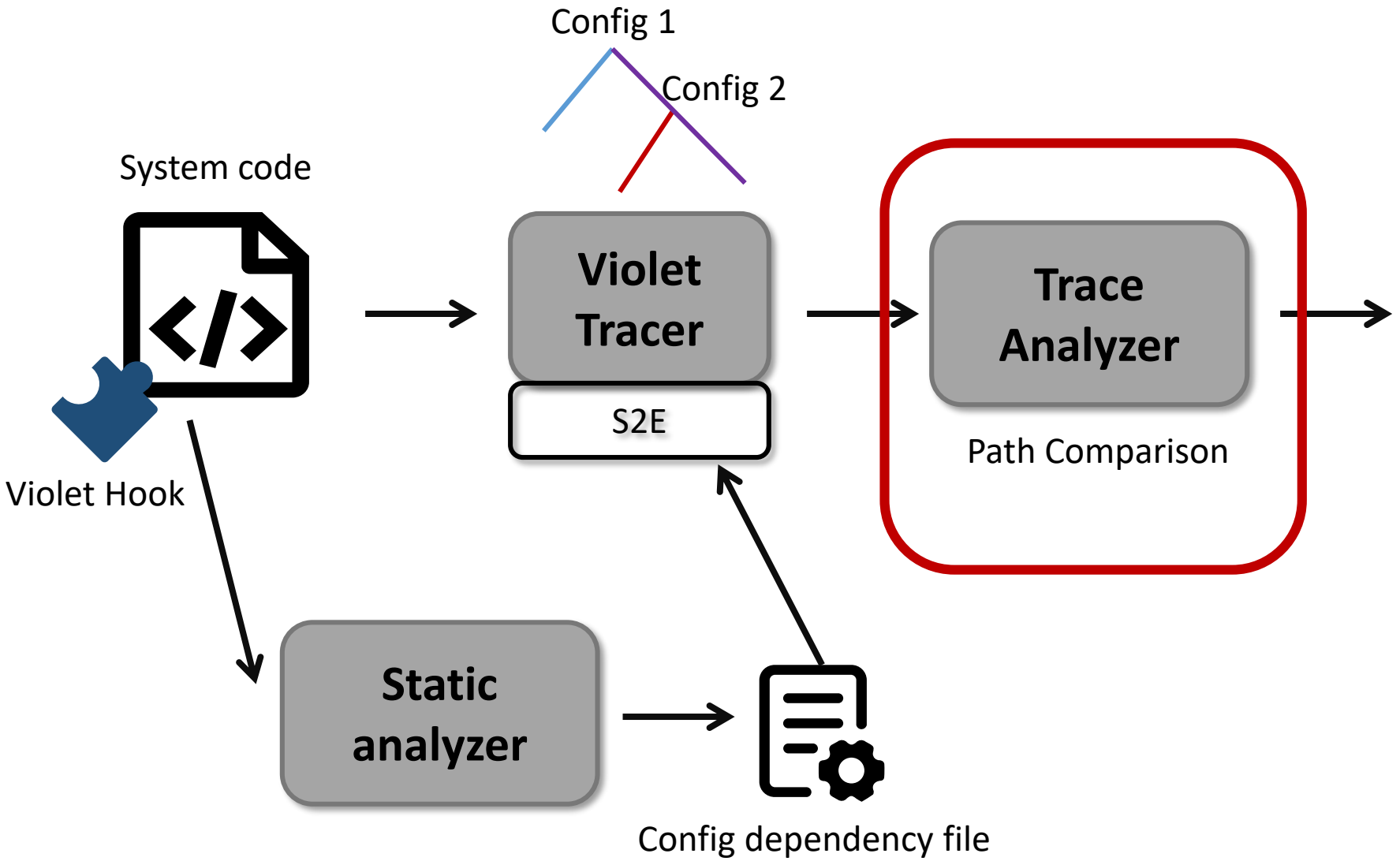
```
void f1() {
    ...
    f2();
}
```



Trace Logical Cost Metric

- **Besides latency and call stack, we also trace:**
  - The # of instructions, system calls, file I/O calls, I/O traffic and etc.
  - We call them logical cost metrics
- **Some specious configurations are not obvious in latency**
- **Logical metrics can capture subtle effect and are independent to the environment**

Violet Overview



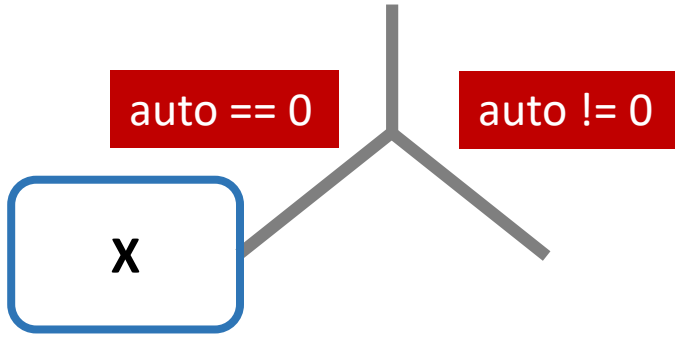
Configuration Constraints	Cost	Workload
auto != 0 & flush == 1	fil_flush	SQL == INSERT
auto != 0 & flush == 2	NULL	SQL == INSERT
auto != 0 & flush != 2	NULL	SQL == INSERT
auto == 0	NULL	SQL != INSERT

Generate Performance Impact Model

```
int write_row() {
   if (autocommit) {
      ...
      trx_commit_complete();
   } else {
      trx_mark_sql_stat_end();
   }
}

ulint trx_commit_complete() {
   if (flush_at_trx_commit==1) {
      log_group_write_buf();
      fil_flush();
   } else if (flush_at_trx_commit==2) {
      log_group_write_buf();
   } else {
   /* do nothing */
   }
}
```

Configuration

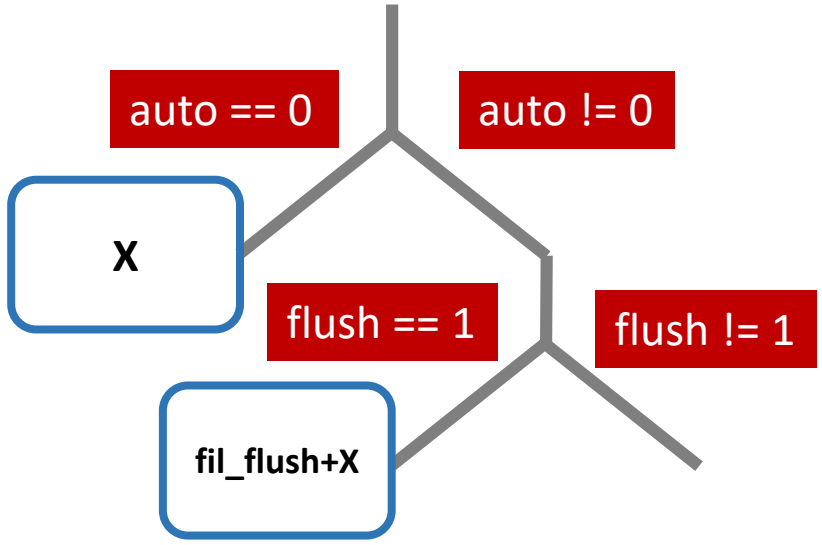
Costly operation



Constraints	Cost	Workload
auto == 0	X	SQL == ALL

Generate Performance Impact Model

```
int write_row() {
   if (autocommit) {
      …
      trx_commit_complete();
   } else {
      trx_mark_sql_stat_end();
   }
}

ulint trx_commit_complete() {
   if (flush_at_trx_commit==1) {
      log_group_write_buf();
      fil_flush();
   } else if (flush_at_trx_commit==2) {
      log_group_write_buf();
   } else {
   /* do nothing */
   }
}
```



Constraints	Cost	Workload
auto == 0	X	SQL == ALL
auto != 0 & flush == 1	fil_flush+X	SQL == INSERT

Configuration

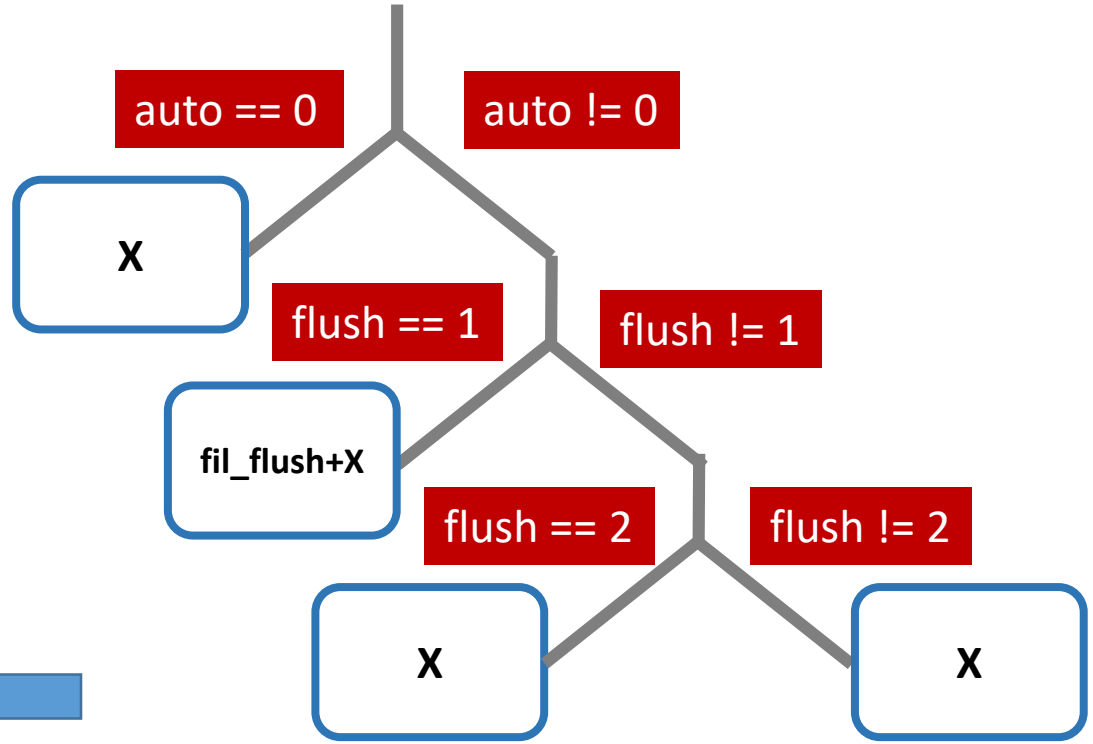
Costly operation

Generate Performance Impact Model

```
int write_row() {
   if (autocommit) {
      …
      trx_commit_complete();
   } else {
      trx_mark_sql_stat_end();
   }
}

ulint trx_commit_complete() {
   if (flush_at_trx_commit==1) {
      log_group_write_buf();
      fil_flush();
   } else if (flush_at_trx_commit==2) {
      log_group_write_buf();
   } else {
   /* do nothing */
   }
}
```

Configuration

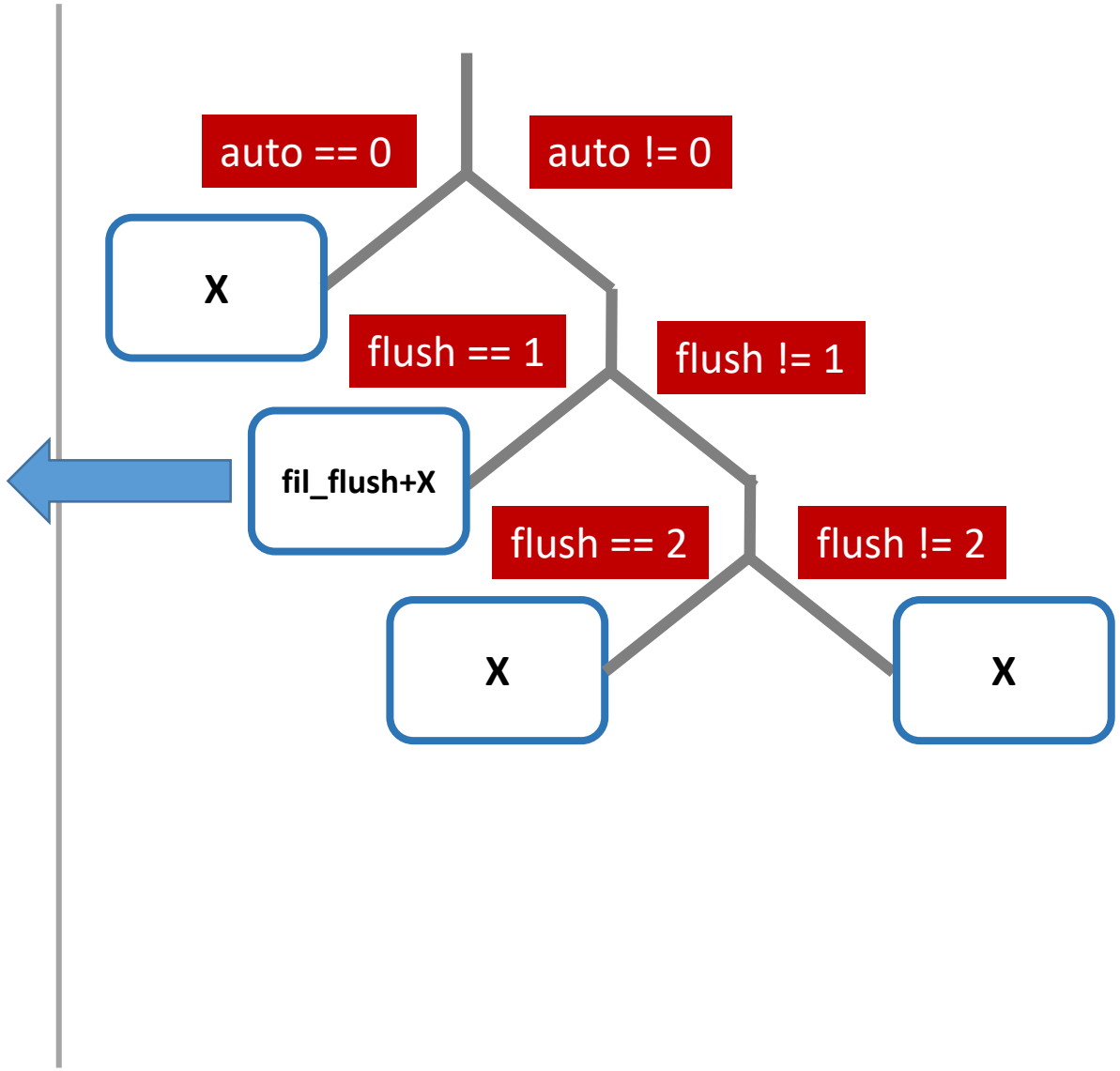
Costly operation



Constraints	Cost	Workload
auto == 0	X	SQL == ALL
auto != 0 & flush == 1	fil_flush+X	SQL == INSERT

Generate Performance Impact Model

Constraints	Cost	Workload
auto == 0	X	SQL == ALL
auto != 0 & flush == 1	fil_flush+X	SQL == INSERT
auto != 0 & flush == 2	X	SQL == INSERT
auto != 0 & flush != 2	X	SQL == INSERT



Performance Comparison

- **Compare the cost between each pair**

Constraints	Cost	Workload
auto == 0 & flush==1	X	SQL == ALL
auto == 0 & flush==2	X	SQL == ALL
auto==0 & flush!=2	X	SQL == ALL
auto!=0 & flush==1	fil_flush+X	SQL == INSERT
auto!=0 & flush==2	X	SQL == INSERT
auto!=0 & flush!=2	X	SQL == INSERT

Performance Comparison

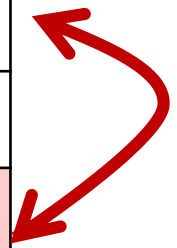
- **Compare the cost between each pair**

Constraints	Cost	Workload
auto == 0 & flush==1	X	SQL == ALL
auto == 0 & flush==2	X	SQL == ALL
auto==0 & flush!=2	X	SQL == ALL
auto!=0 & flush==1	fil_flush+X	SQL == INSERT
auto!=0 & flush==2	X	SQL == INSERT
auto!=0 & flush!=2	X	SQL == INSERT

Performance Comparison

- **Some path comparisons are not very meaningful**

Constraints	Cost	Workload
auto == 0 & flush==1	X	SQL == ALL
auto == 0 & flush==2	X	SQL == ALL
auto==0 & flush!=2	X	SQL == ALL
auto!=0 & flush==1	fil_flush+X	SQL == INSERT
auto!=0 & flush==2	X	SQL == INSERT
auto!=0 & flush!=2	X	SQL == INSERT



path 1: auto == 0 & flush == 2
path 2: auto != 0 & flush == 1

“Similar” Path First Comparison

- **The paths with the most “similar” constraint compare first**
  - If a constrain appears in both state, add one to similarity score
- **If two paths don’t have common constraint**
  - Don’t compare them

Implementation

- **Violet components are mostly written in C/C++**
  - Violet tracer is implemented as S2E plugins
  - Violet static analyzer is built on top of LLVM

- **S2E [ASPLOS ’11]**
  - Symbolic execution platform
  - Fast, in-vivo

Selective Symbolic Execution

- **Complex constraint and path explosion**
- **Selective symbolic execution**
  - Silently *concretize* variable before library call or syscall
  - Accurate but not complete
  - Relax rules to achieve good completeness

Outline

- Motivation
- Specious Configuration Code Patterns
- Violet Overview
- **Evaluation**

Evaluation Questions

- How effective is Violet in detecting specious configurations and unknow cases.
- How useful is Violet?
- What is the performance of Violet?

Experiment Setup

- **Evaluated systems**
  - MySQL, PostgreSQL, Apache, Squid
- **The manual effort to add hook is small**

Software	SLOC	# of config	Line of Hook
MySQL	1.2M	330	197
PostgreSQL	843K	294	165
Apache	199K	172	158
Squid	178K	327	96

17 Specious Configurations

Application	Configuration Name	Data Type	Detect
MySQL	autocommit	Boolean	√
MySQL	query_cache_wlock_invalidate	Boolean	√
MySQL	general_log	Boolean	√
MySQL	query_cache_type	Enumeration	√
MySQL	sync_binlog	Integer	√
MySQL	innodb_log_buffer_size	Integer	√
PostgreSQL	wal_sync_method	Enumeration	√
PostgreSQL	archive_mode	Enumeration	√
PostgreSQL	max_wal_size	Integer	√
PostgreSQL	checkpoint_completion_target	Float	√
PostgreSQL	bgwriter_lru_multiplier	Float	√
Apache	HostNamelookup	Enumberation	√
Apache	Deny/Domain	Enum/String	√
Apache	MaxKeepAliveRequests	Integer	×
Apache	KeepAliveTineOut	Integer	×
Squid	Cache	String	√
Squid	Buffered_logs	Integer	√

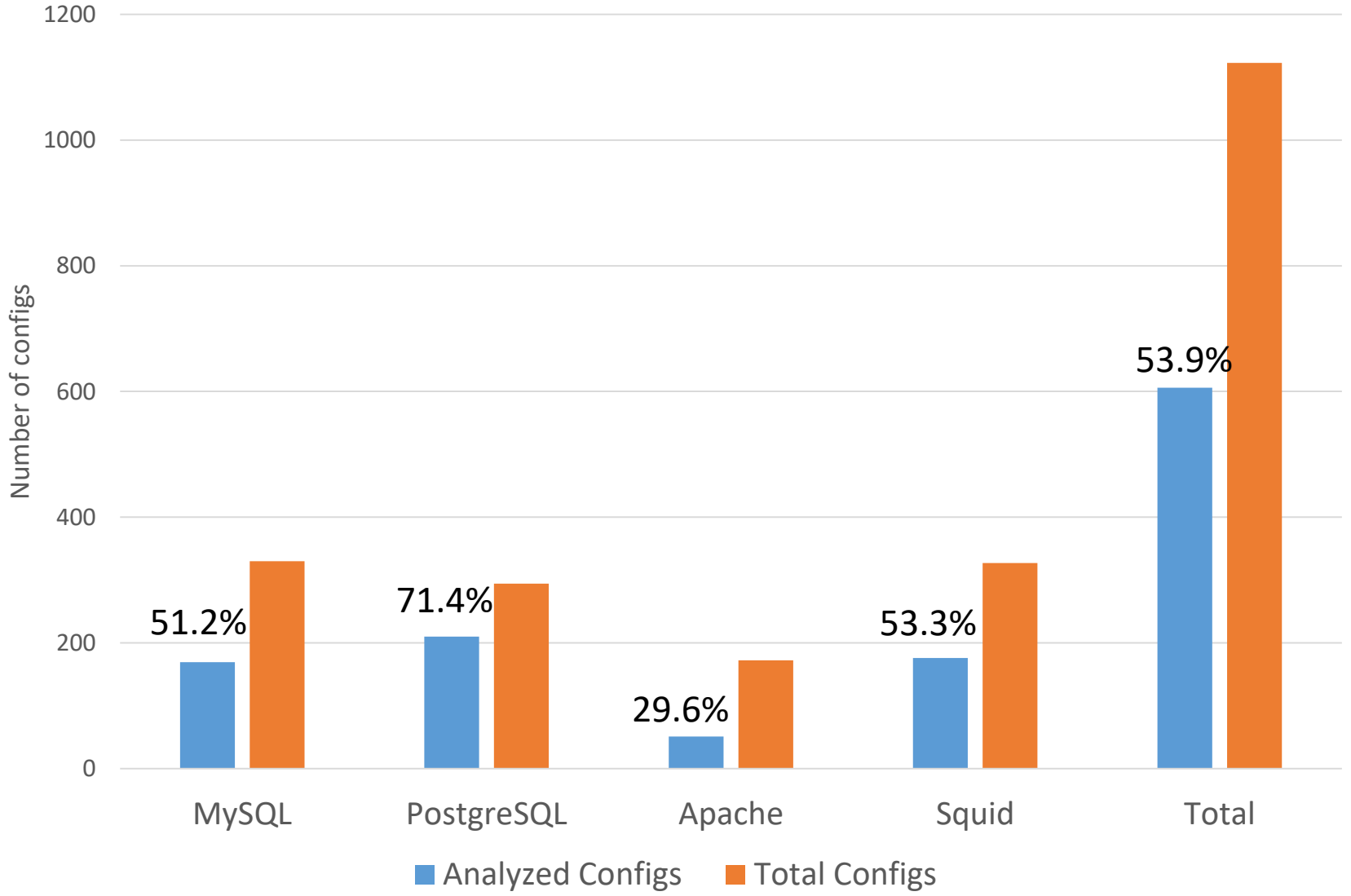
Discover New Specious Configuration

Specious configuration is 1) the setting whose default value causes performance regression; 2) some performance impact is not documented

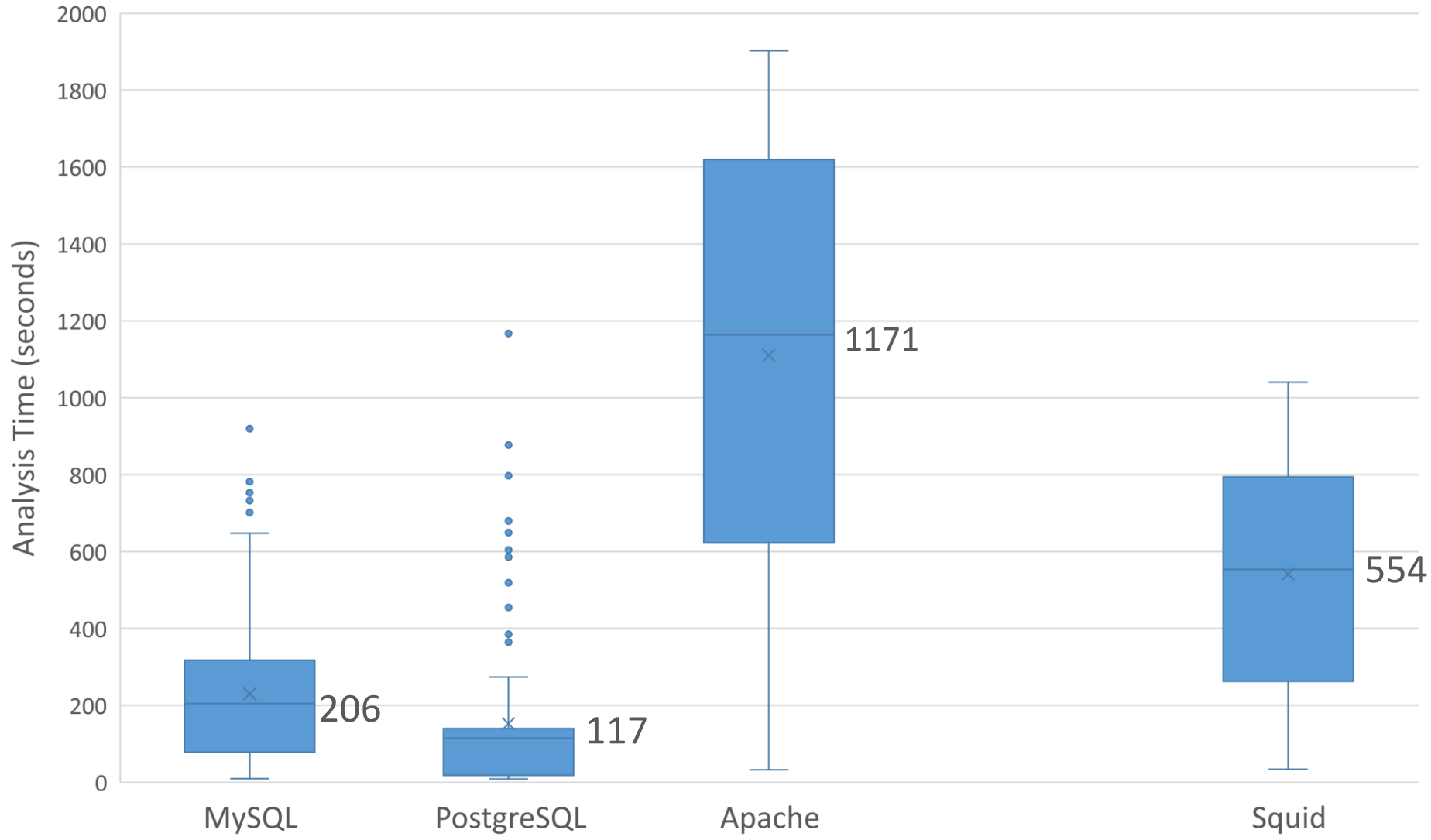
Application	Configuration Name	Performance Impact
MySQL	optimizer_search_depth	Default cause would cause bad performance for some join query
MySQL	concurrent_insert	Enable it would cause bad performance for read workload
PostgreSQL	vacuum_cost_delay	Default value is significantly worse than low values for write workload
PostgreSQL	archive_timeout	Small values cause performance penalties
PostgreSQL	random_page_cost	Value larger than 1.2 cause bad perf on SSD for join queries
PostgreSQL	log_statement	Setting mod cause bad perf for write workload when synchromous_commit is off
PostgreSQL	parallel_setup_cost	A higher value would avoid unnecessary parallelism
PostgreSQL	parallel_leader_participation	Enabling it can cause select join query to be slow
Squid	ipcache_size	The default value is relatively small and may cause performance reduction
Squid	cache_log	Enable cachelog with higher debug_option would cause extra I/O
Squid	store_objects_per_bucket	Decrease the setting would short the search time

8 new cases are confirmed by developers

Coverage Experiment for Violet

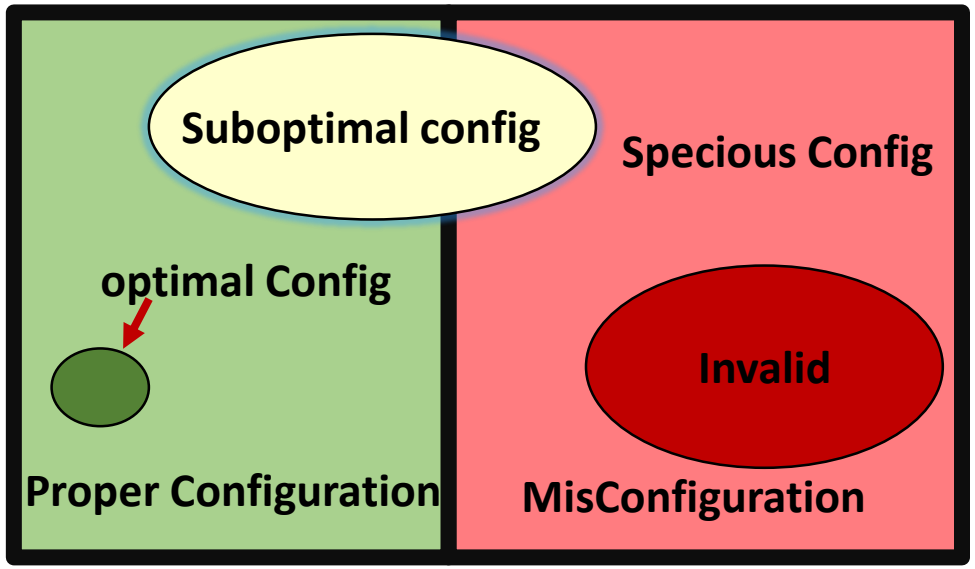


How Fast Is Violet



Related Work

- **Misconfiguration Detection**
  - Pcheck[OSDI'16], LearnConf[Eurosys'20], PeerPressure[OSDI'04], EnCore[ASPLOS'14]
- **Misconfiguration Diagnosis**
  - ConfAid[OSDI'10], X-ray[OSDI'12]
- **Performance Tuning**
  - Starfish [CIDR'11], Strider [LISA'03],SmartConf[ASPLOS'18]



Conclusion

1. Detecting specious configuration is a difficult task
2. Need to systematically reason about the performance effect of configuration from source code
3. Violet – an analytical approach to detect specious configuration in large system by symbolic execution
4. Detect 15 known specious configuration and 11 new cases

https://github.com/OrderLab/violet

Thank you!

Contact Information: hyigong1@jhu.edu