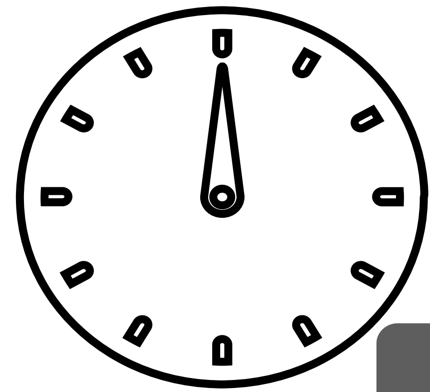
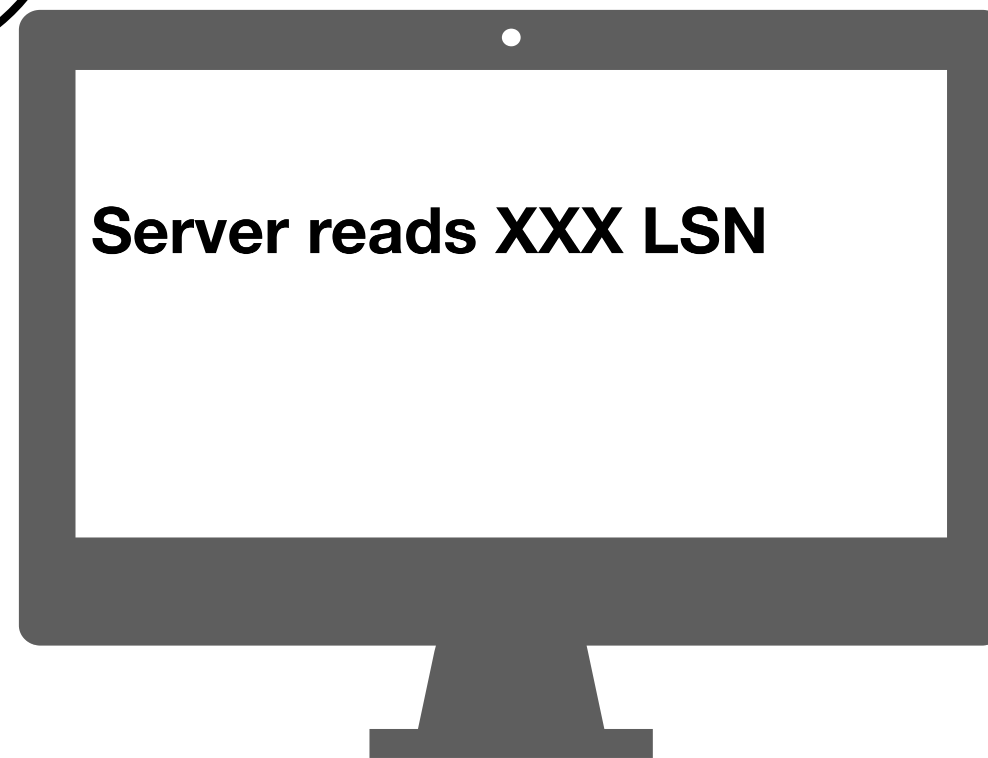
Effective Performance Issue Diagnosis with Value-Assisted Cost Profiling

Lingmei Weng, Columbia University
Yigong Hu, Johns Hopkins University
Peng Huang, University of Michigan
Jason Nieh, Columbia University
Junfeng Yang, Columbia University

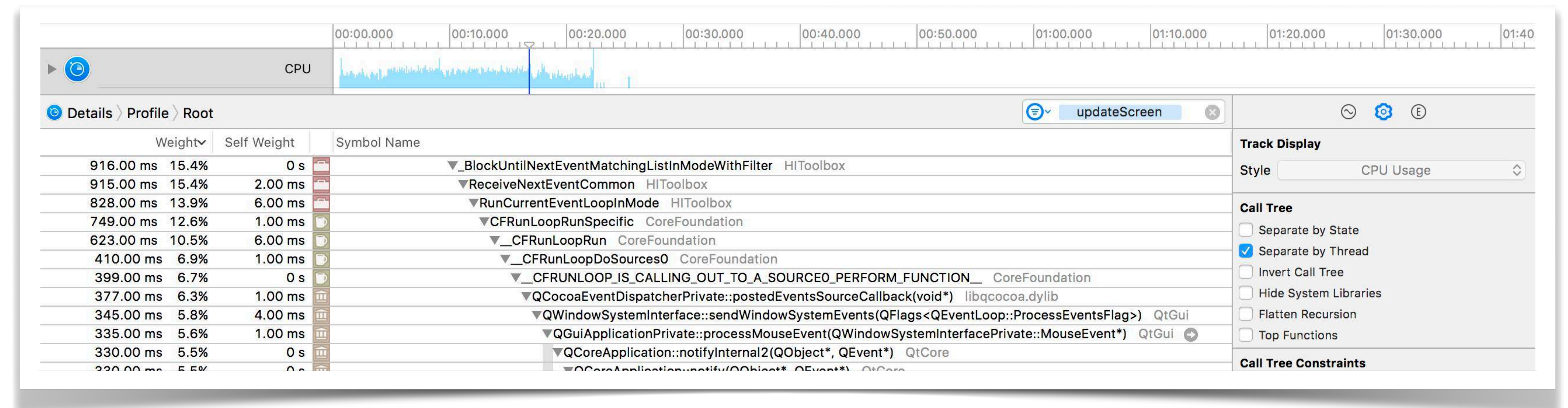
A Real World Performance Issue in MariaDB





In v10.3.22, MariaDB crash recovery takes a long time

Profilers are often recommended



Output from Existing Profilers

available_mem = 0

```
3388 bool recv_group_scan_log_recs(lsn_t ckpt_lsn, ...) {
3417   ulint available_mem = srv_page_size *
3418         (buf_pool_get_n_pages() -
3419         (recv_n_pool_free_frames * srv_buf_pool_ins));
3424   do {
                          ... ...
3431     recv_apply_hashed_log_recs(false);
3439     log.read_log_seg(&end_lsn, start_lsn + RSCAN_SIZE);
3440   } while (end_lsn != start_lsn &&
3441       !recv_scan_log_recs(available_mem, ...
```

function has more than 200 LOC

20+ branches

The diagnosis took 20+ days

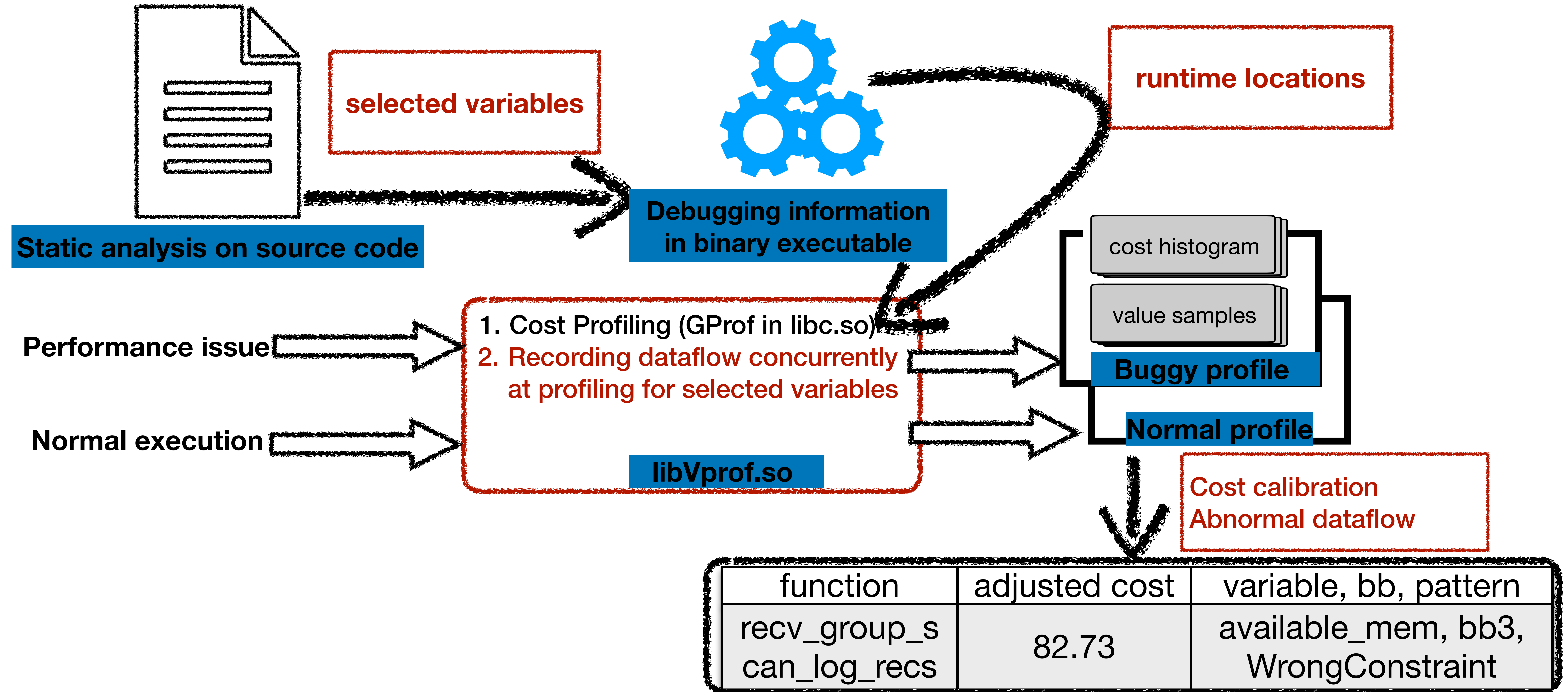
Problems

- Top ranked functions are not the culprit
- No info about buggy values that cause the issue

Key Insights

- ❏ **Function costs alone** are insufficient for performance diagnosis
- ❏ **Dataflow is necessary** to understand root causes of performance issues
  - ➡ a program variable's values over time
  - ➡ useful to calibrate raw costs and identify problematic code

vProf Workflow



function	adjusted cost	variable, bb, pattern
recv_group_scan_log_recs	82.73	available_mem, bb3, WrongConstraint

vProf Challenges

vProf needs to address **three challenges**:

- ✦ *Pre-profiling*: **select** variables to minimize the overhead
- ✦ *Profiling*: **record** value samples concurrently and efficiently at profiling signal handler
- ✦ *Post-profiling*: effectively **leverage** recorded samples for diagnosis

Select Variables

✦ Focus value recording in a component related to the performance issue

- e.g., **`storage/innobase/log`**

✦ Use static analysis to identify variables in code area that affects performance

- conditional expression => operands

```
s = b + 3*i;
if (i < a.min) goto Lerr
while (i < a.length) {
    i = i + 2;
    s = s + 6;
    goo(ptr, s, i);
}
```

- loop => induction variables

```
s = b + 3*i;
if (i < a.min) goto Lerr
while (i < a.length) {
    i = i + 2;
    s = s + 6;
    goo(ptr, s, i);
}
```

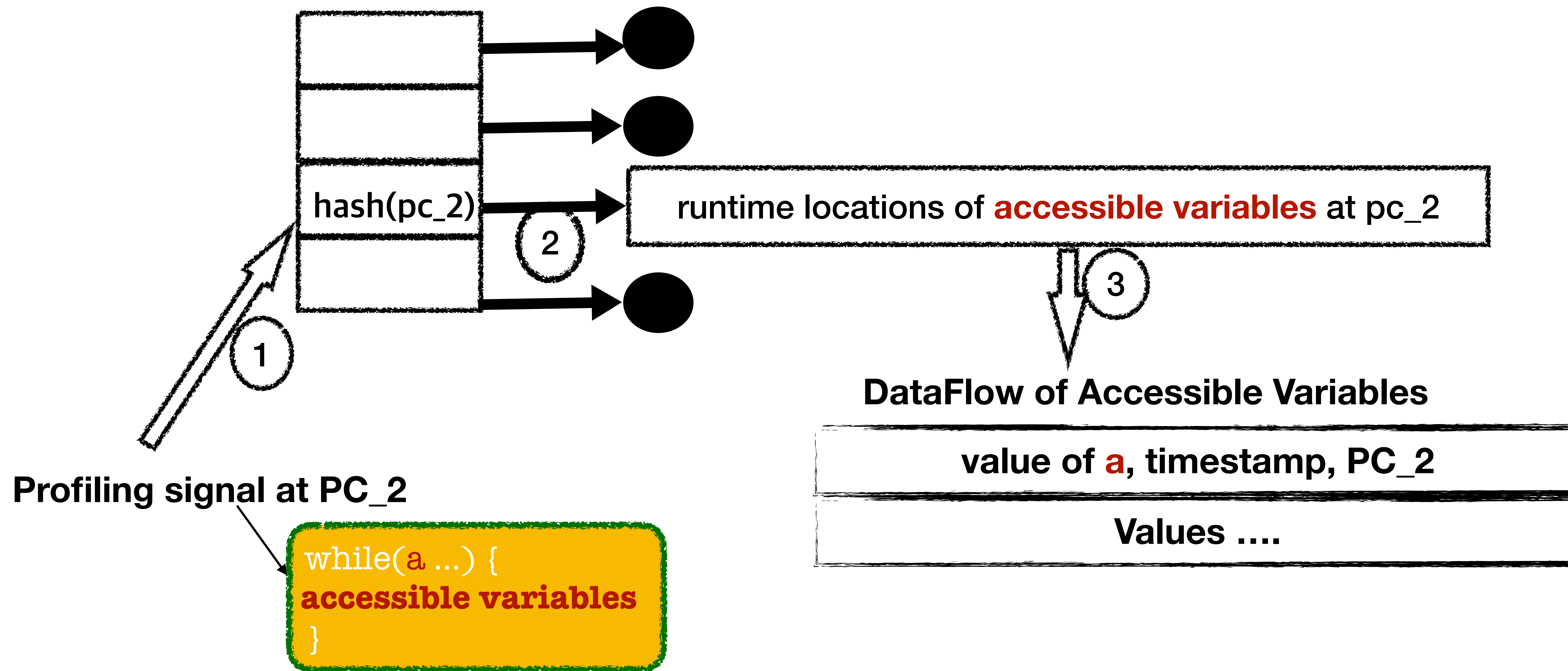
- function call => parameters

```
s = b + 3*i;
if (i < a.min) goto Lerr
while (i < a.length) {
    i = i + 2;
    s = s + 6;
    goo(ptr, s, i);
}
```

Access Selected Variables During Profiling

- Typical cost profiling is done by periodical sampling with signals
  - Profiling signals are delivered at different instruction addresses (PCs)
- **Problems**
  - **accessible variables** at different PCs changes
  - **runtime locations** for the same variable changes
- **Solution**
  - **Fast index** the **runtime locations** of **accessible variables** from arbitrary PCs

Efficient Recording of Value Samples

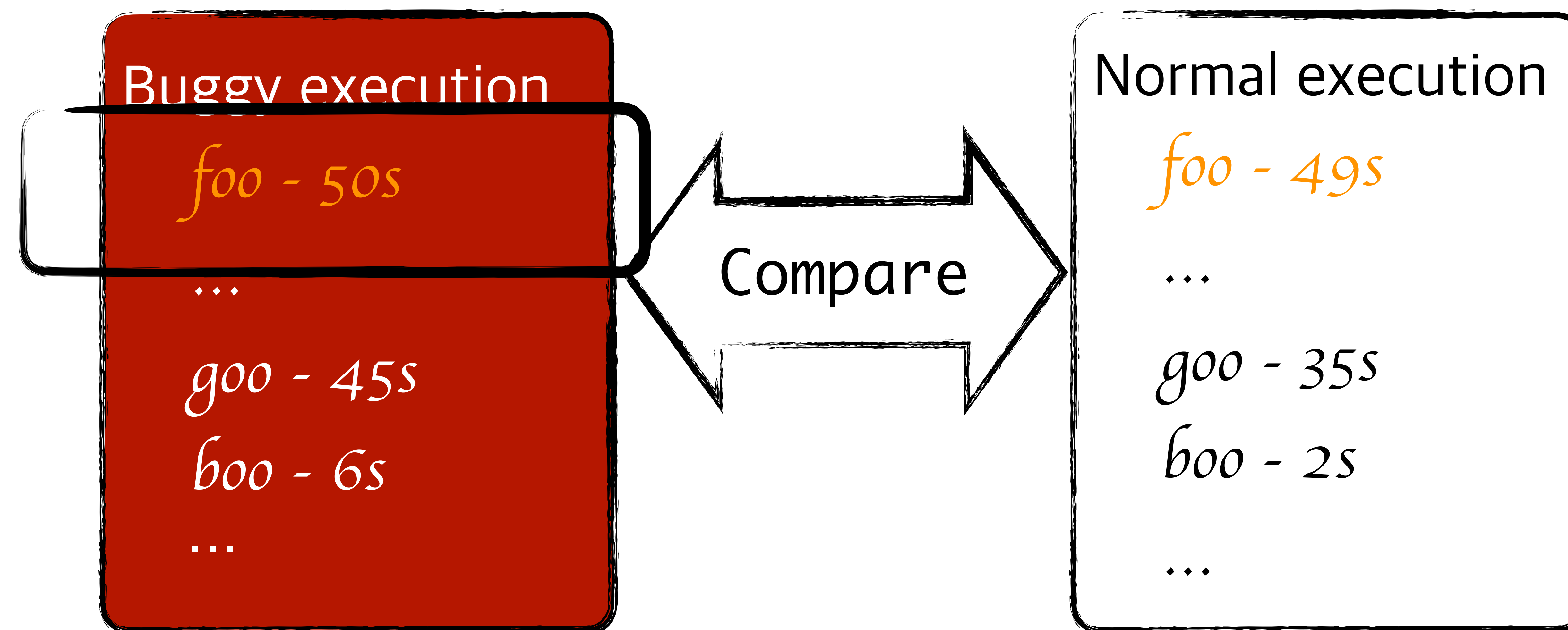


Offline Cost Calibration with Recorded Samples

- **Discount the cost of** inherent costly functions
  - We calculate two kinds of discounts
    - (1) ranking discount; (2) variable discount
    - (2) is critical
- **Boost the cost of under-estimated** functions
  - little time but cause execution of other costly functions

Discount Calculation Needs A Baseline

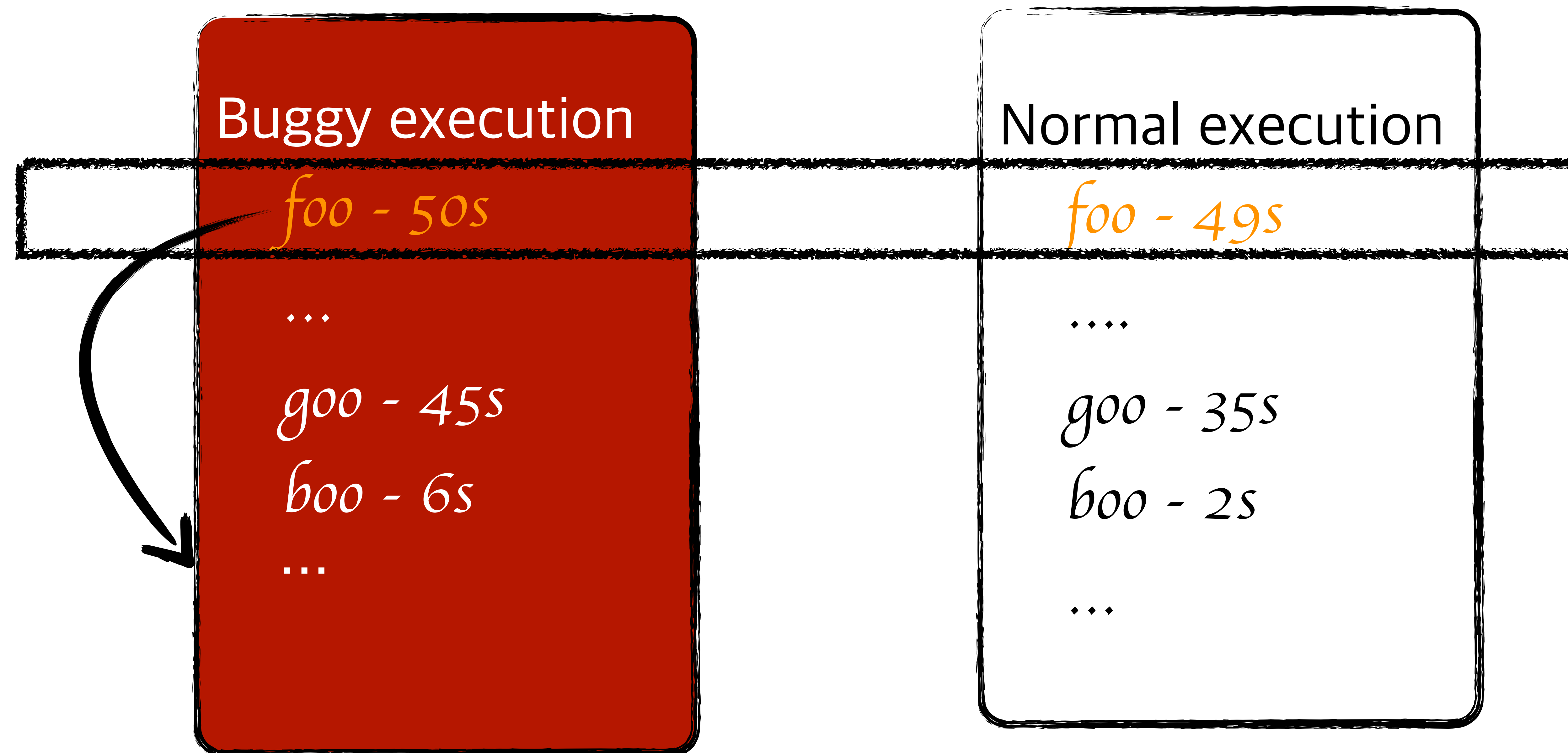
- ✦ Discount for inherent costly functions
  - Compare to normal execution to identify inherent costly



  - Baseline needs a similar use case, not necessarily identical

Ranking Discount

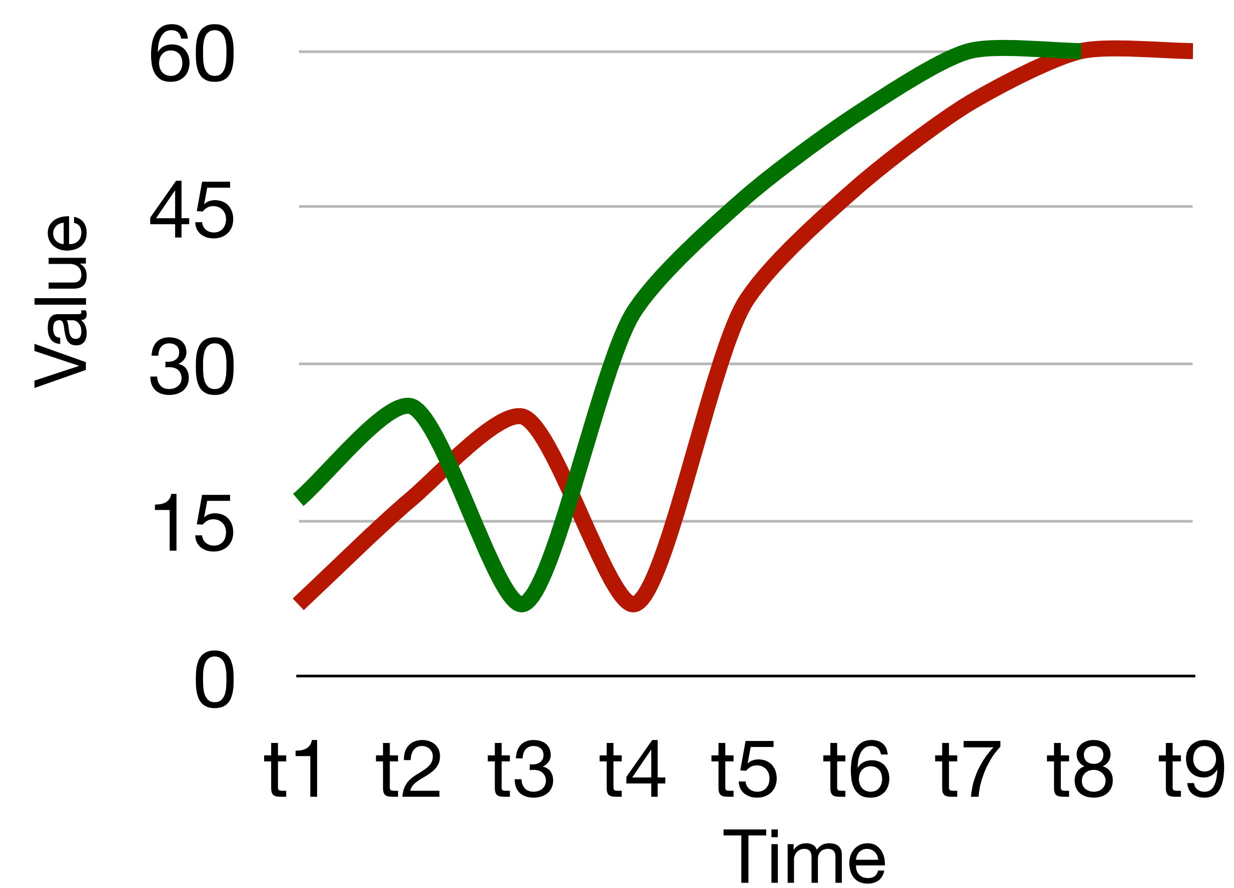
- Same rankings in buggy execution and normal execution

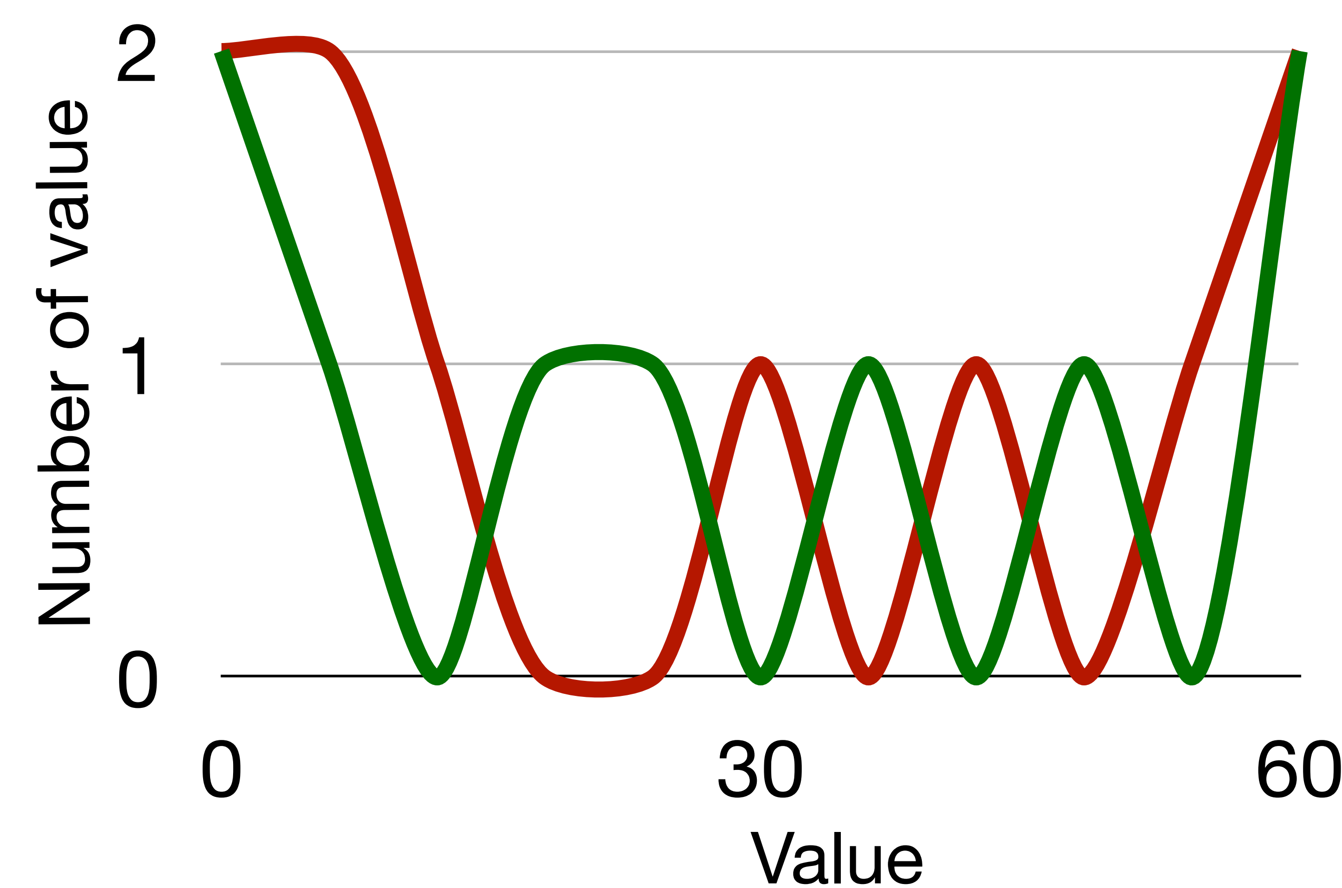


Variable Discount

✦ Similarity on distributions of values for variable *var* in function *goo*

— Normal execution — Buggy Execution





Buggy execution

goo - 45s

boo - 6s

...

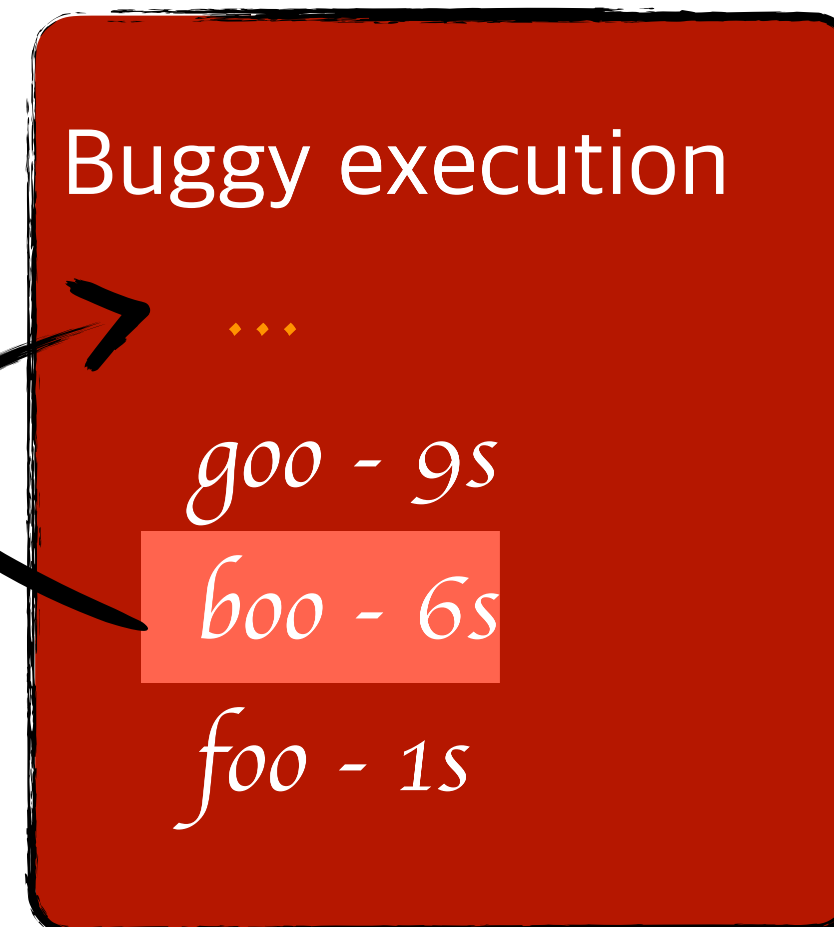
foo - 1s

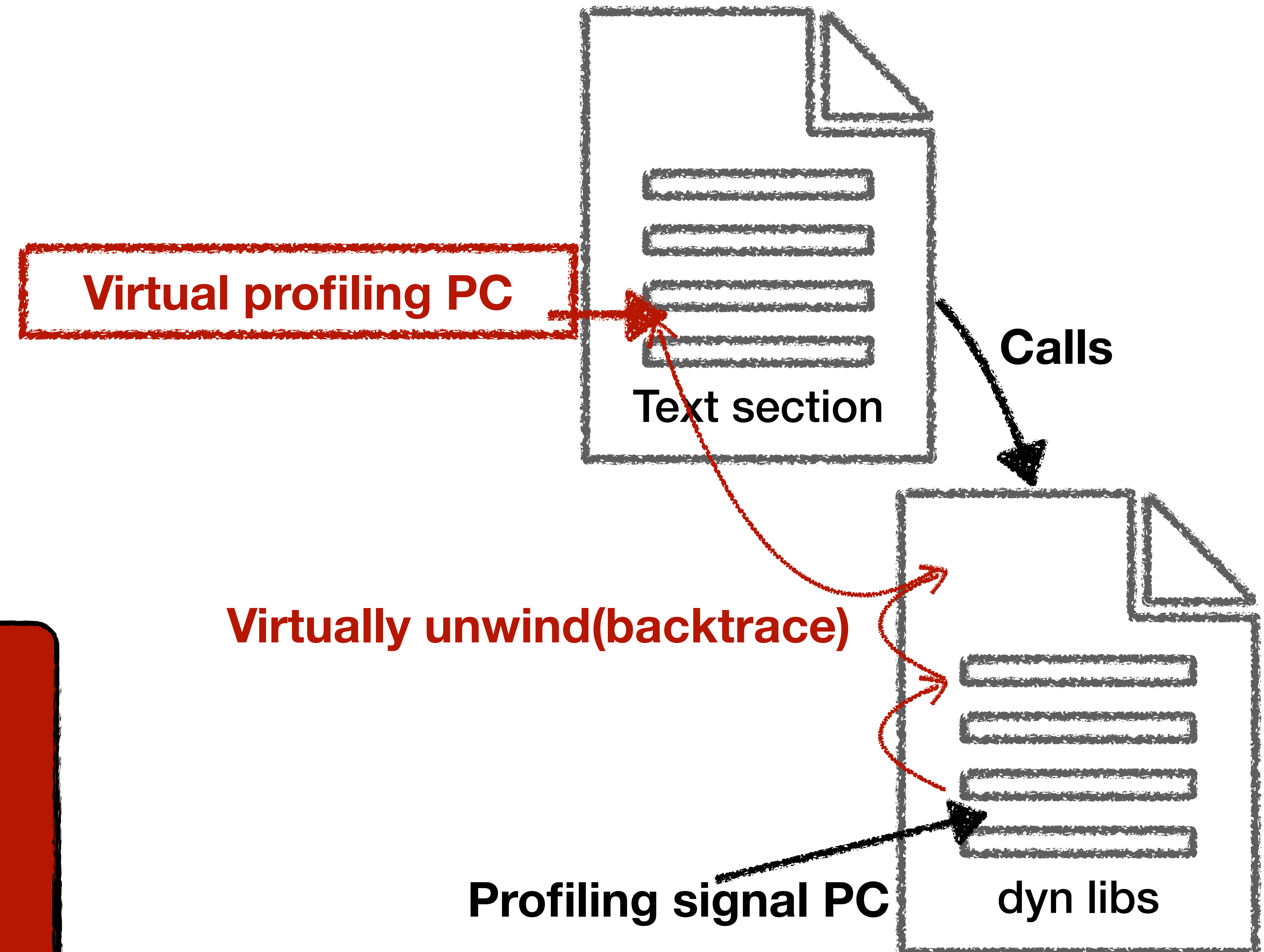
similar value distributions of var ➔ *adjusted_cost(goo) = (1 - discount) * profiling_cost(goo)*

Boost Under-Estimated Function Cost

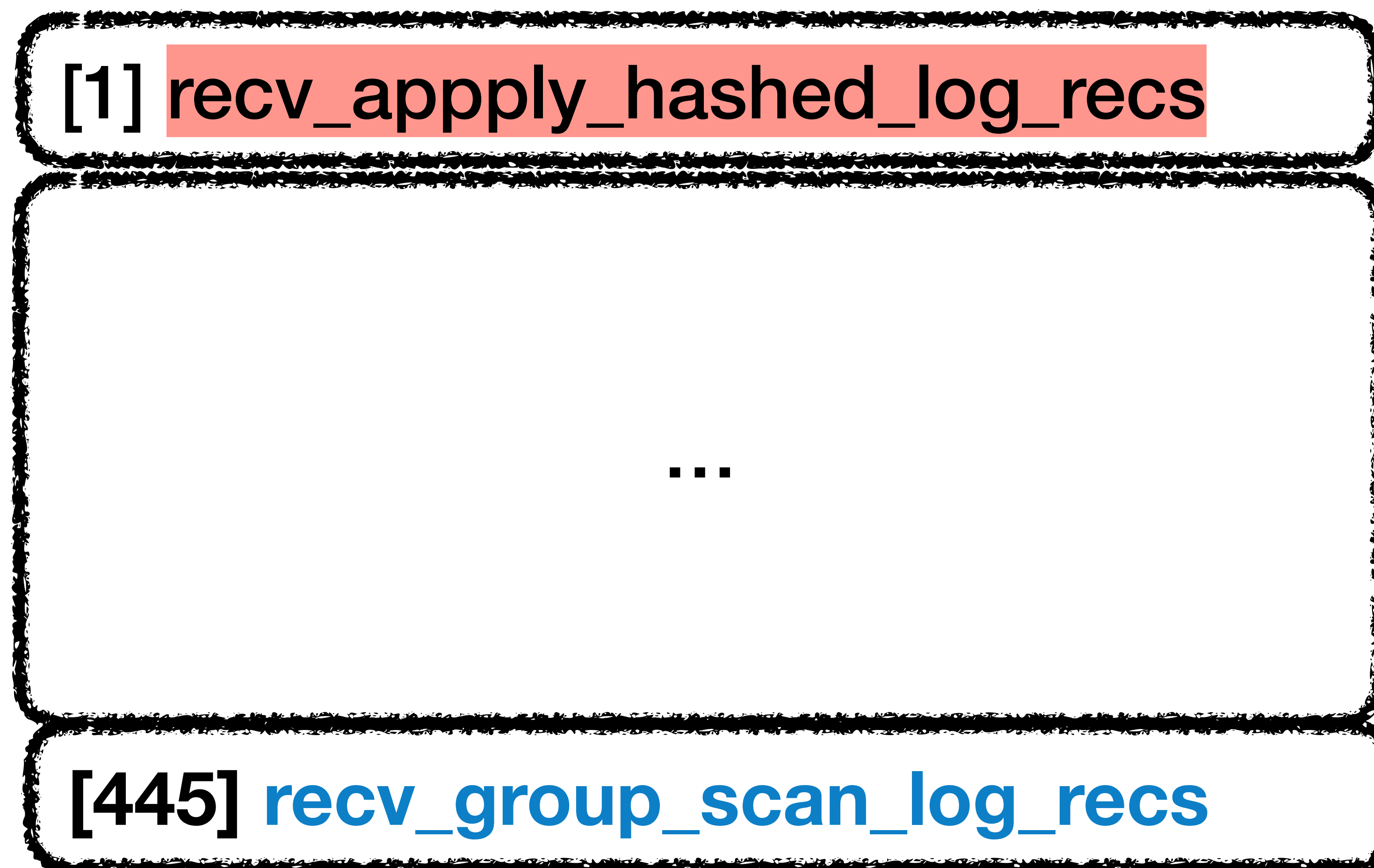
- ✦ Samples outside current program, eg. dynamic libraries, are omitted.
- ✦ Values of the variables accessible from callers are also missed
  - ✓ Virtually backtrace the call stack

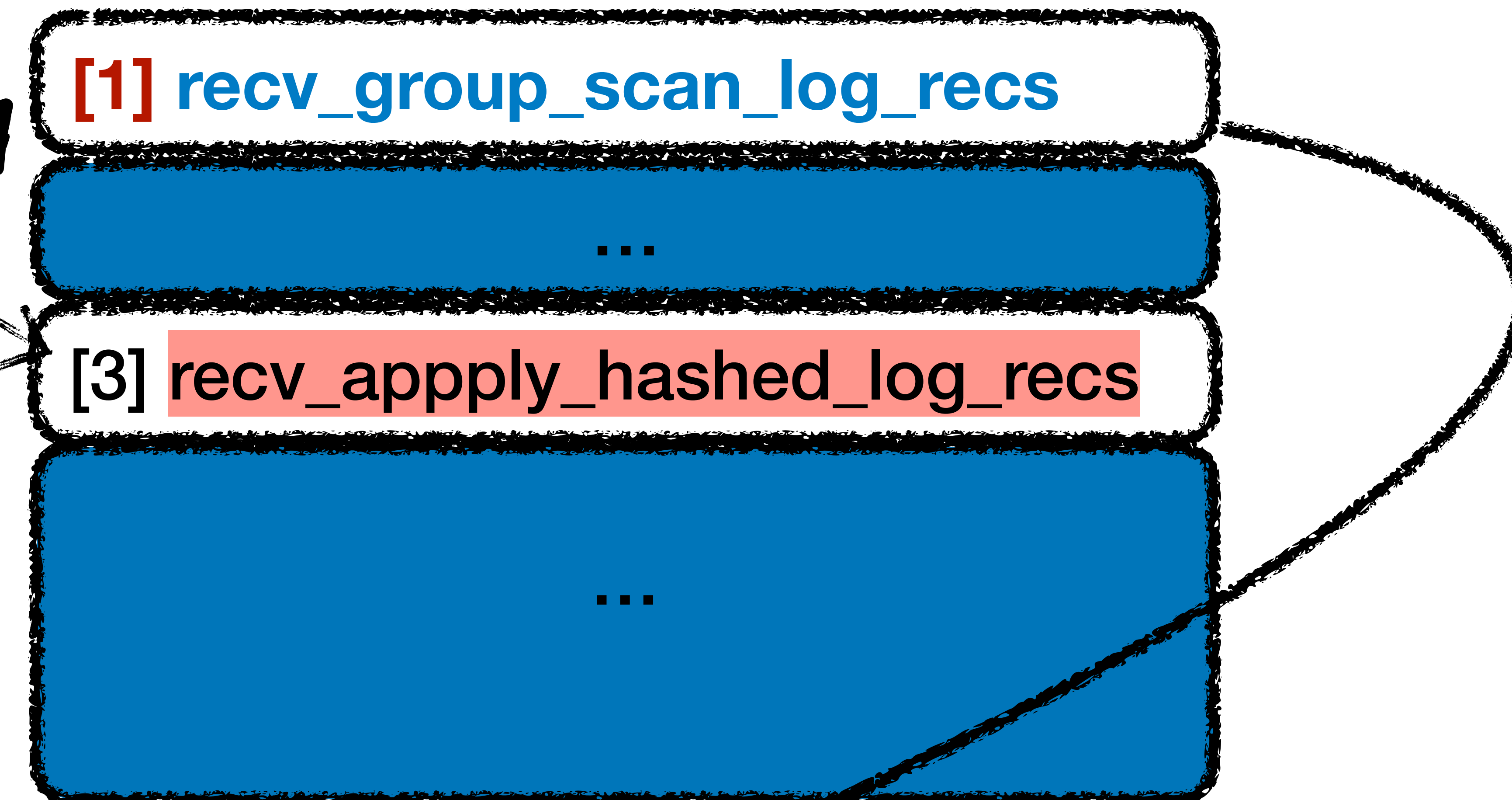
- **Costly library function;**
- **Abnormal value samples**





vProf Result for MariaDB Example






```
3388 bool recv_group_scan_log_recs(lsn_t ckpt_lsn, ...) {
3417   ulint available_mem = srv_page_size *
3418         (buf_pool_get_n_pages() -
3419         (recv_n_pool_free_frames * srv_buf_pool_ins));
3424   do {
3431      recv_apply_hashed_log_recs(false);
3439      log.read_log_seg(&end_lsn, start_lsn + RSCAN_SIZE);
3440    } while (end_lsn != start_lsn &&
3441       !recv_scan_log_recs(available_mem, ...
```

Additional Debugging Aid

- ✦ Besides cost calibration, vProf leverages the recorded value samples to provide further debugging aid
  - ✓ Identify the abnormal variables for a function
  - ✓ Locate the code regions where abnormal values are accessed
  - ✓ Infer potential performance bug patterns

Abnormal Value for MariaDB Example

Function cost ranking in vProf

[1]. recv_group_scan_log_recs

...

vProf debugging report

adjusted_cost: *87.73*

suspicious_variable: *available_mem*

(abnormal_value: *0*, location: *bb3*)...

bug_pattern: *WrongConstraint*

```
3388 bool recv_group_scan_log_recs(lsn_t ckpt_lsn, ...) {
3417   ulint available_mem = srv_page_size *
3418         (buf_pool_get_n_pages() -
3419         (recv_n_pool_free_frames * srv_buf_pool_ins));
3424   do {
3431     recv_apply_hashed_log_recs(false);
3439     log.read_log_seg(&end_lsn, start_lsn + RSCAN_SIZE);
3440    } while (end_lsn != start_lsn &&
3441        !recv_scan_log_recs(available_mem, ...
```

Evaluate vProf

✦ How effective vProf is?

✦ What is the advantages of vProf compared to other tools?

✦ Is vProf efficient enough to be practical?

Evaluation Settings:

- **Intel Core i5 and 48GB DRAM**
- **Apply vProf to real-world performance bugs via LD_PRELOAD**
- **No instrumentation to applications**

Real-World Performance Issues

- **All ground truth has already known in their bug reports**

ID	Apps	Bug Description
B1	MariaDB	Server crash recovery loops on the same log sequence number
B2	MariaDB	Performance drops when the size of dataset is larger than the size of
B3	MaraiDB	Deleting a table with CASCADE constraint is very slow
B4	MariaDB	Slow start-up even when .ibd file vali dation is off
B5	MariaDB	Checking the server status takes >10 seconds with 3M tables
B6	Apache httpd	Output filter endless loop so server process never terminates
B7	Apache httpd	Gracefully restart service with mmm-workers takes long time
B8	Apache httpd	Health check is executed more often than configured intervals
B9	Apache httpd	Slow startup/reload when many ghosts are configured
B10	Apache httpd	Workers take 60-100% CPU even though no client sent requests
B11	Redis	Cluster nodes command is costly in a large cluster
B12	Redis	BRPOP command becomes slow when a large number of clients
B13	Redis	ZREVRANGE command is 50% slower after upgrade
B14	PostgresSQL	EXPLAIN hangs for generating some query plans
B15	PostgresSQL	Vacuum process fails to prune all heap pages and endlessly retries

Effectiveness

ID	vProf
B1	1
B2	1
B3	1
B4	3
B5	4
B6	5
B7	3
B8	1
B9	2
B10	1
B11	1
B12	1
B13	2
B14	4
B15	3
Summary@top5	15/15

- vProf ranks root causes of all 15 issues within the top 5
- 7 of 15 have their root causes ranked at the top 1

Comparison with Other Tools

ID	vProf	gprof	Perf	Perf-pt	Coz	Statistical debugging
B1	1	454	32	32	NR	4
B2	1	5	2	2	NR	12
B3	1	2	3	6	1	30
B4	3	21	9	5	NR	18
B5	4	13	4	9	NR	566
B6	5	36	13	13	NR	NR
B7	3	182	1024	1024	Crash	7
B8	1	1	6	7	ChildProc	3
B9	2	11	28	28	NR	9
B10	1	4	16	16	ChildProc	161
B11	1	1	10	10	2	NR
B12	1	5	19	19	1	8
B13	2	16	13	13	9	NR
B14	4	NR	163	163	ChildProc	13
B15	3	14	56	56	ChildProc	18
@top5	**15/15**	**6/15**	**3/15**	**2/15**	**3/15**	**1/15**

✦ Other tools rank root causes within the top 5 for at most 6 cases

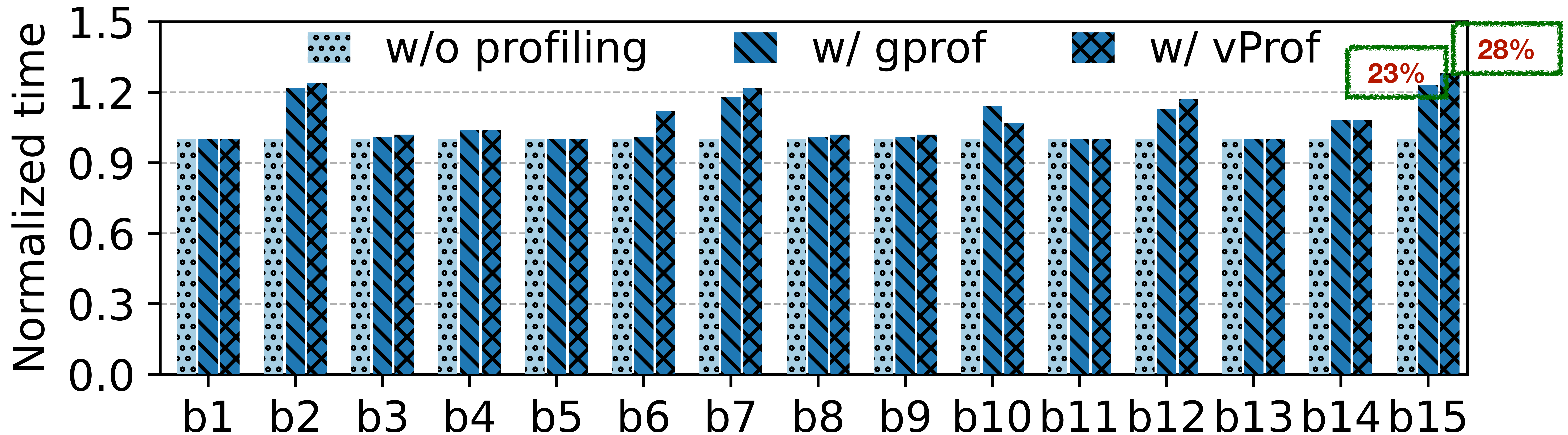
vProf is Effective in Diagnosing Unresolved Issues

ID	Bug Description	Date
Redis-10981	*lrange* command takes longer to finish when Redis is upgrade from version 6.2.7 to 7.0.3	07-14-2022
MDEV-16289	Query runs unexpectedly slow. The query **selects** records created within a given time period and **excludes** the records that are referenced by another table in a another given period	05-25-2018
MDEV-17878	Searching for the query execution plan for a SELECT query involving many joins takes forever for larger datasets, using 100% CPU	11-30-2018

- **All the above issues have both reporter and developer involving the debugging.**

vProf is Efficient: CPU Overhead

✦ The overhead gaps between gprof and vProf are mostly within 5%



vProf is Efficient: Memory Overhead

ID	#Vars	PCToVar(kB)	VariableArray(kB)	ValueSamples(kB)	Sum(kB)
B1	233	3862	430	21133	25425
B2	65	4143	29	153	4325
B3	399	4005	26	38563	42594
B4	852	3987	67	58	4112
B5	577	3575	22	8	3605
B6	501	673	287	2	962
B7	113	162	6	16	184
B8	169	260	127	43	430
B9	374	194	16	25	235
B10	164	642	186	13	841
B11	531	612	382	1216	2210
B12	623	591	44	1755	2390
B13	564	641	754	132	1527
B14	479	2037	1031	79	3147
B15	805	2297	927	3269	6493

Max: 42MB

Min: 184kB

Conclusions

- ✦ Missing dataflow in profiler makes performance diagnosis ineffective
- ✦ vProf integrates dataflow to re-rank functions and reveal root cause
- ✦ vProf successfully diagnosed all 15 resolved performance issues and three unresolved performance issues
- ✦ The overhead of value-assisted profiling is acceptable

https://github.com/wenglingmei/vprofAE