Static Profiling: Why should you try it?

Women in Compilers and Tools Meetup Series
30 June 2022

Angélica Moreira

Who am I?

- From **Belo Horizonte, MG - Brazil***

- Pursuing a PhD in Computer Science at **Federal University of Minas Gerais (UFMG)** in Brazil
  - Advised by dr. Fernando Magno Quintão Pereira (from UFMG)
  - Co-advised by dr. Guilherme Ottoni (from Meta)

- Working in compilers research area as a Graduate Student Researcher for almost 4 years (mostly **LLVM**)
  - Projects highlight:
    - Dead Code Elimination
    - Basic Block Reordering
    - Static Branch Predictor
    - MLIR-based Compiler for the MSCCL** project

* Yep, the city where Brazil was humiliated by Germany in the world cup, 7x1 :'(
** MSCCL stands for Microsoft Collective Communication Library (MSCCL) is a platform to execute custom collective communication algorithms for multiple accelerators supported by Microsoft Azure. https://github.com/microsoft/msccl

Basic Concepts

Prediction vs. Probability vs. Frequency

Prediction vs. Probability vs. Frequency

- For instance, in this code:

[1]

```
b1:      if (condition)
b2:           statement;
b3:      else statement;
```

  - A branch ***prediction*** :
    - "branch b1->b2 will be taken."

[1] Youfeng Wu and J. R. Larus, "Static branch frequency and program profile analysis," *Proceedings of MICRO-27. The 27th Annual IEEE/ACM International Symposium on Microarchitecture*, San Jose, CA, USA, 1994, pp. 1-11, doi: 10.1109/MICRO.1994.717399.

Prediction vs. Probability vs. Frequency

- For instance, in this code:

```
                                              [1]
b1:      if (condition)
b2:           statement;
b3:      else statement;
```

  - A branch ***probability*** :
    - "branch b1->b2 81% to be taken, while b1->b3 has 19% of being taken."

[1] Youfeng Wu and J. R. Larus, "Static branch frequency and program profile analysis," *Proceedings of MICRO-27. The 27th Annual IEEE/ACM International Symposium on Microarchitecture*, San Jose, CA, USA, 1994, pp. 1-11, doi: 10.1109/MICRO.1994.717399.

Prediction vs. Probability vs. Frequency

- For instance, in this code:

[1]

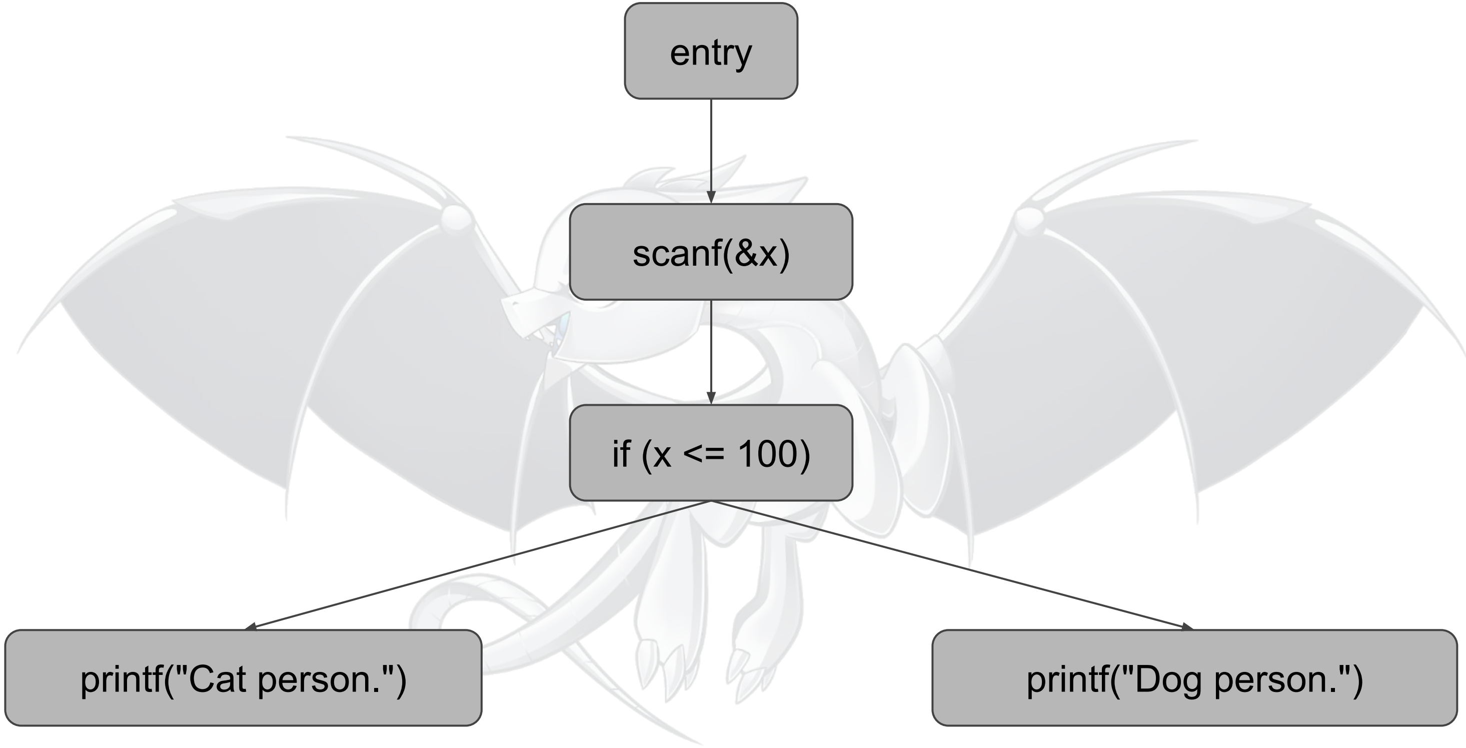
```
b1:      if (condition)
b2:           statement;
b3:      else statement;
```

  - A branch ***frequency*** :
    - "b1 executes 80 times, in 65 of which it branches to b2, and in 15 it branches to b3."

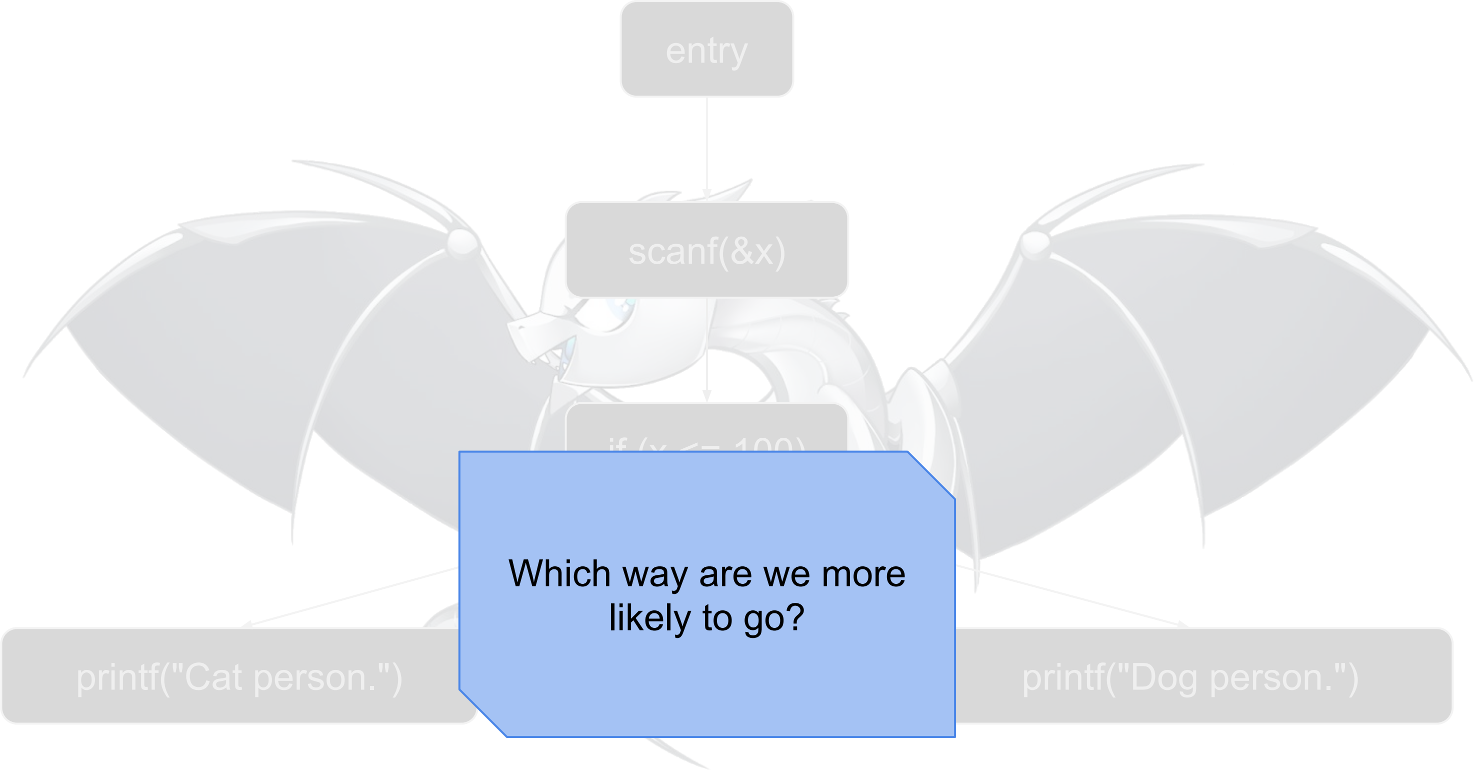
[1] Youfeng Wu and J. R. Larus, "Static branch frequency and program profile analysis," *Proceedings of MICRO-27. The 27th Annual IEEE/ACM International Symposium on Microarchitecture*, San Jose, CA, USA, 1994, pp. 1-11, doi: 10.1109/MICRO.1994.717399.

Software-Based Branch Prediction

Software-Based Branch Prediction

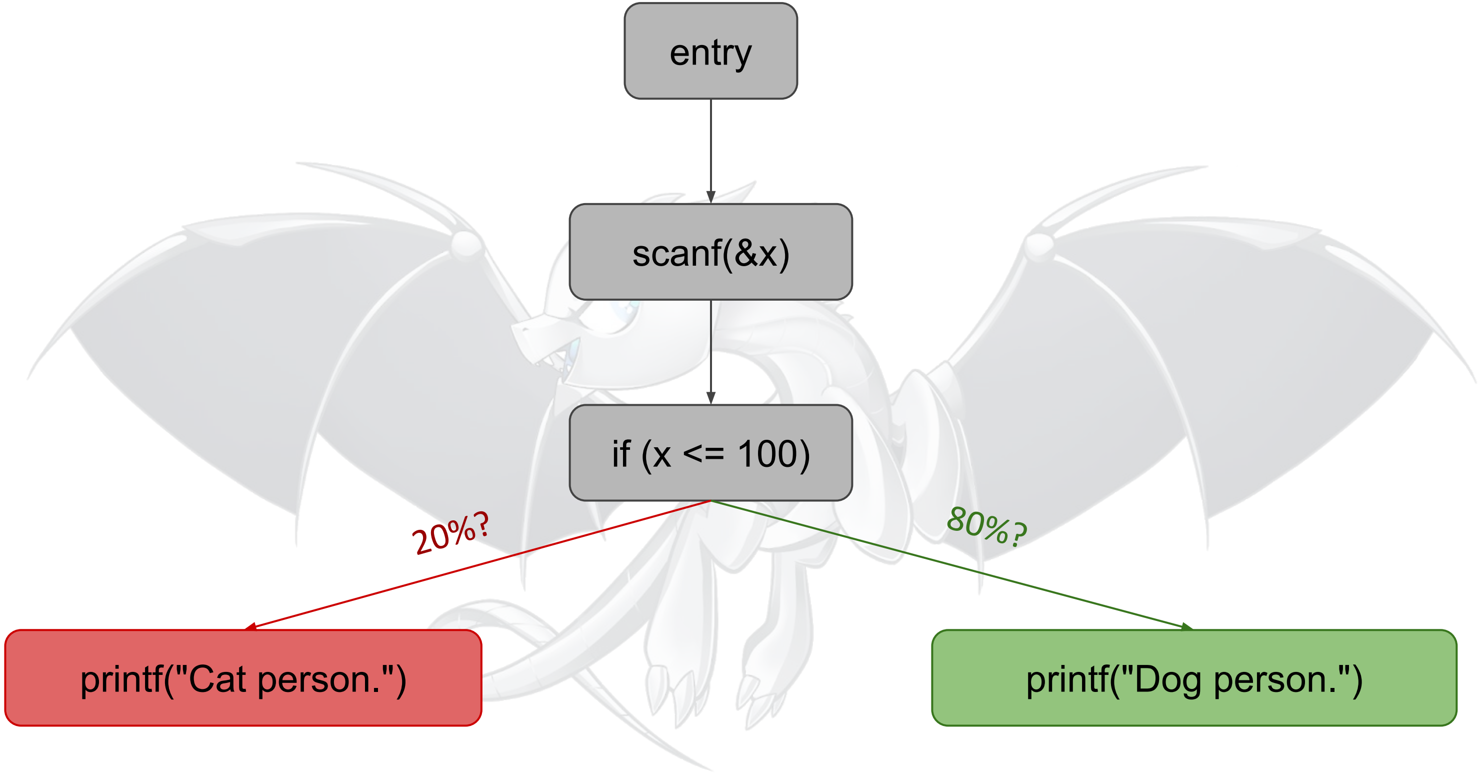


Software-Based Branch Prediction

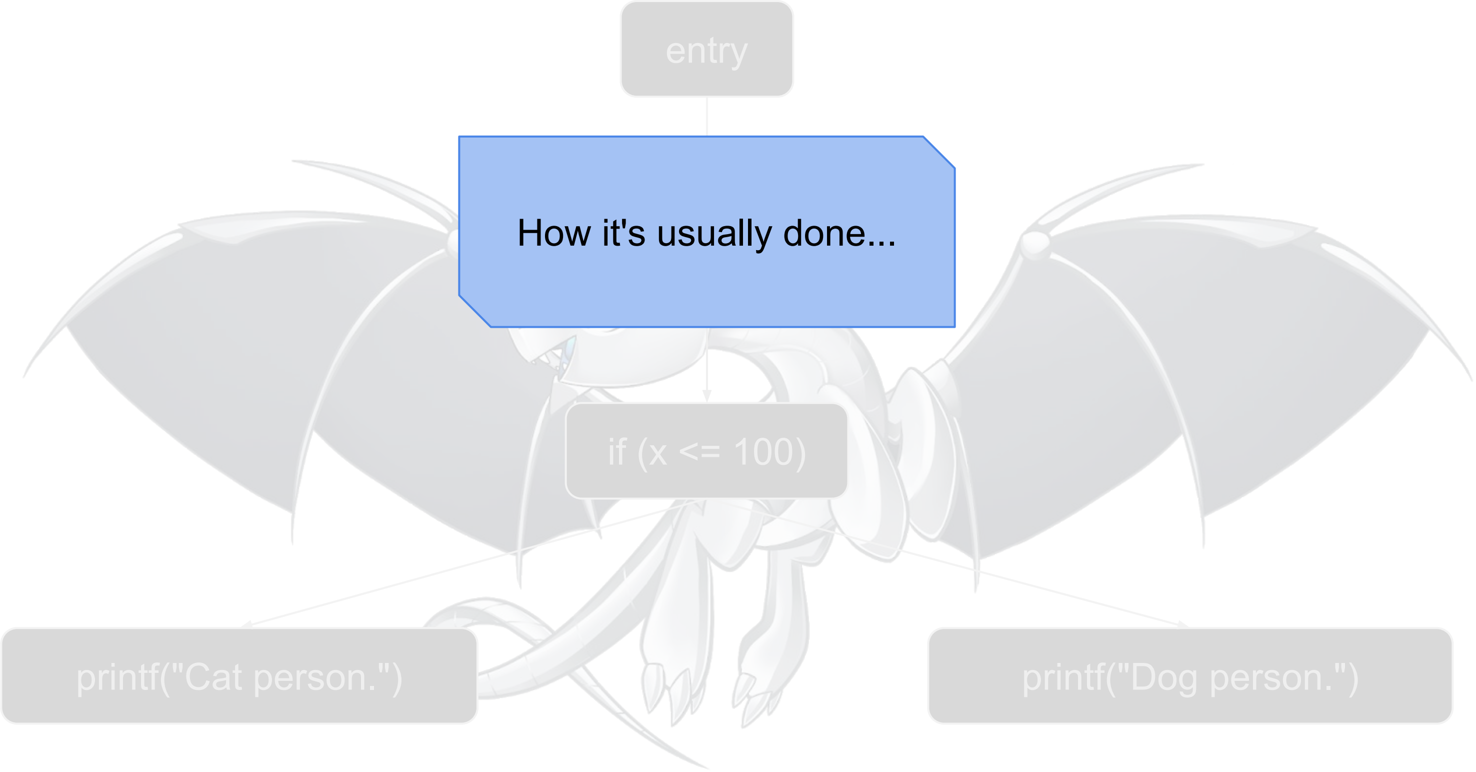


Which way are we more likely to go?

Software-Based Branch Prediction

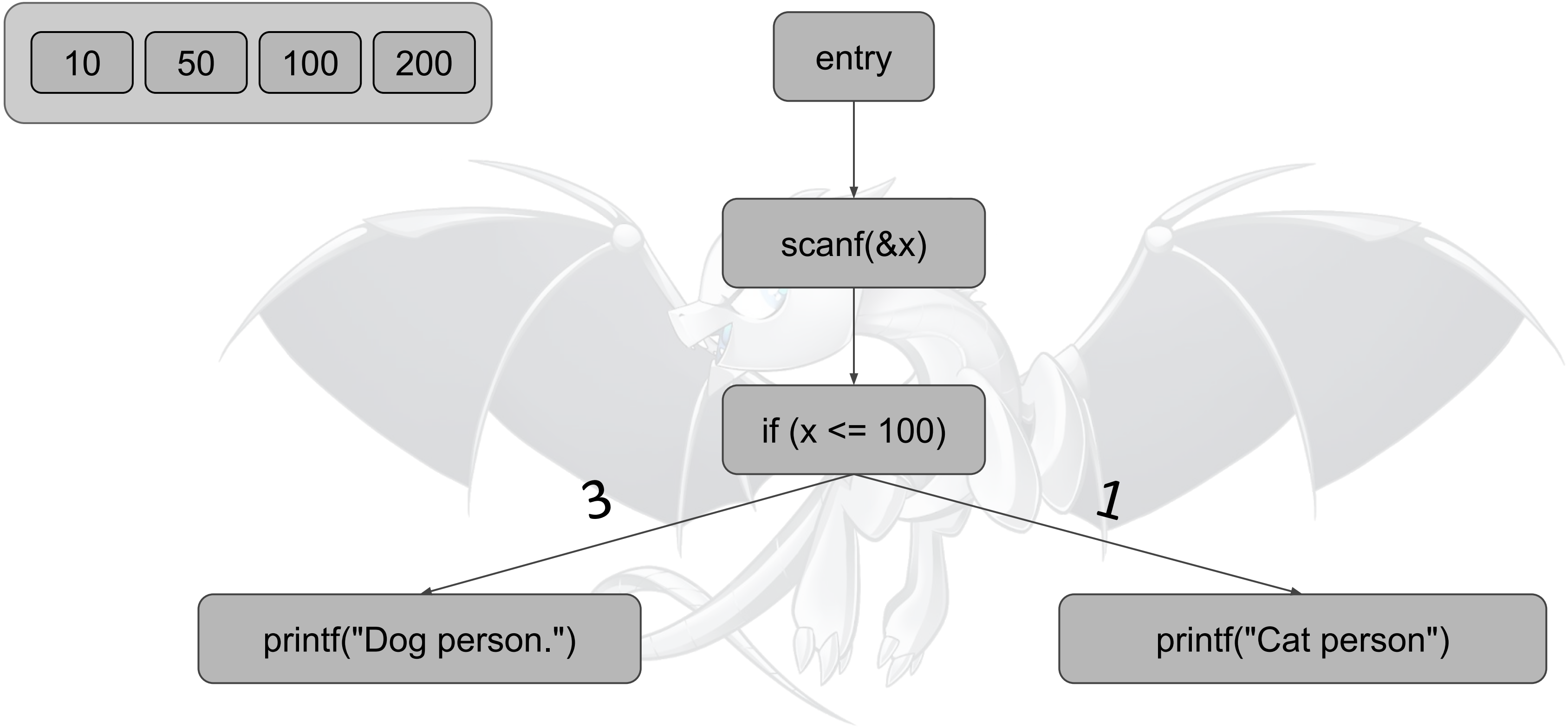


Software-Based Branch Prediction

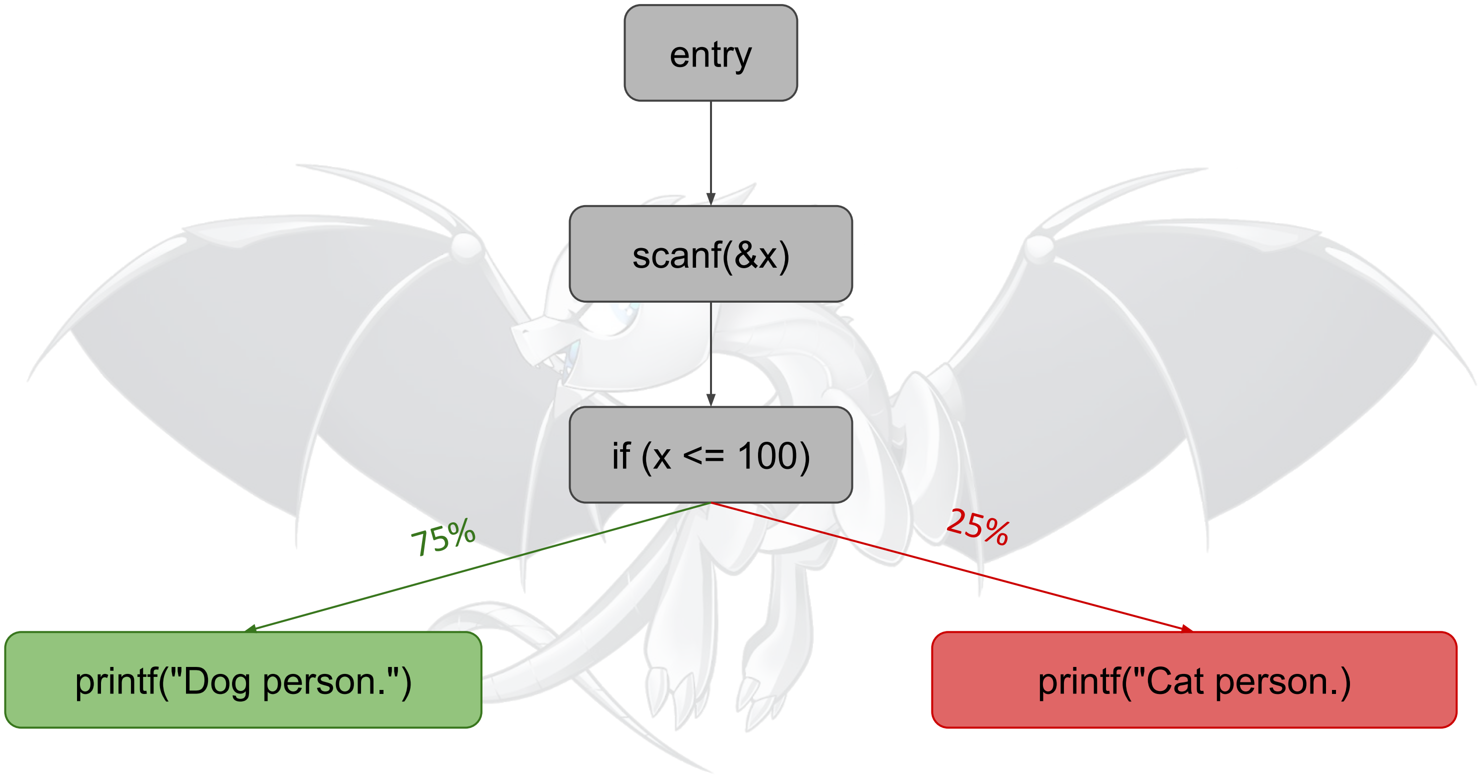


Dynamic Profiling

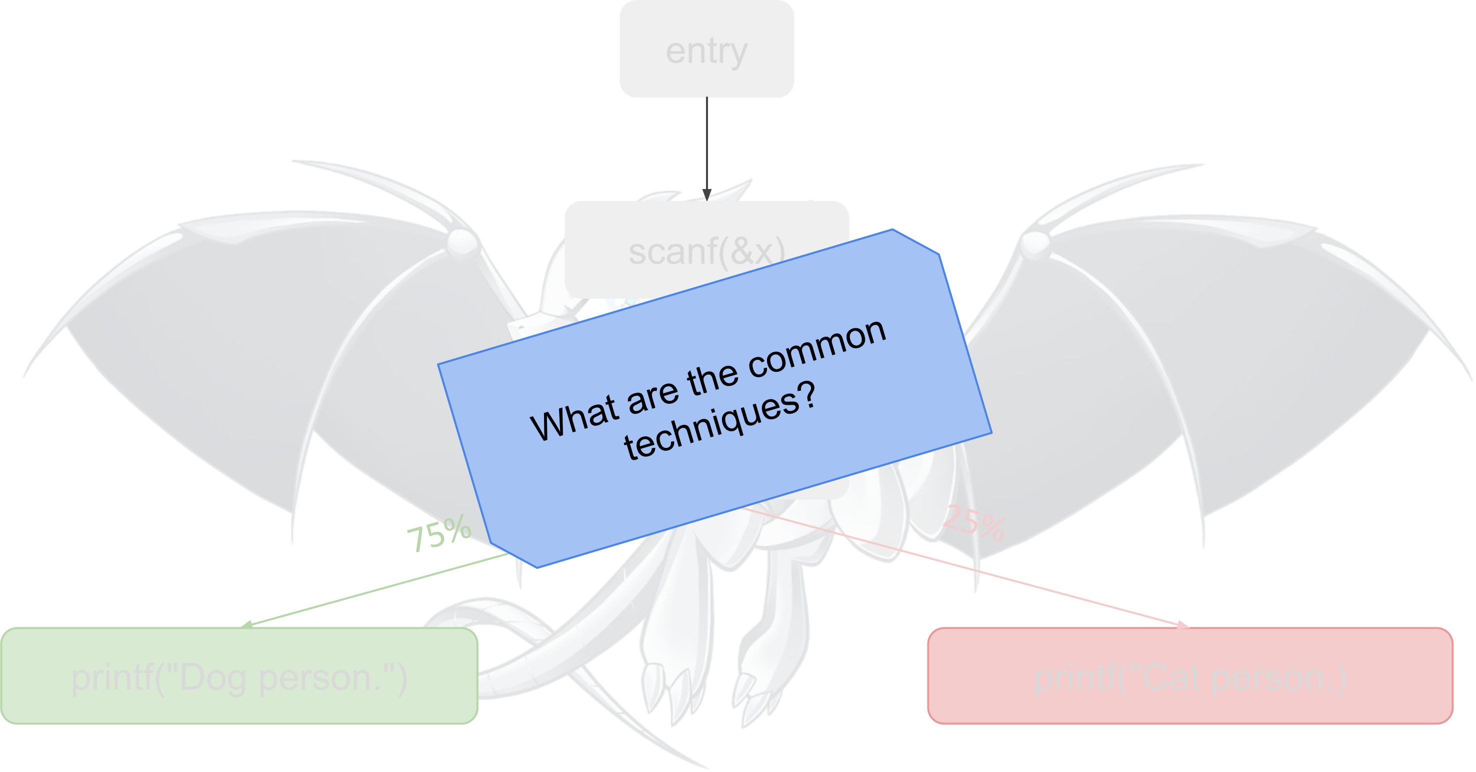
Inputs



Dynamic Profiling



Dynamic Profiling



What are the common techniques?

Dynamic Profiling

- Sampling-based approaches:
  - Runs the program
  - **Samples** instructions executed
  - Records sampled branch executions

Dynamic Profiling

- Sampling-based approaches:
  - Runs the program
  - **Samples** instructions executed
  - Records sampled branch executions

```
Samples: 1K of event 'cycles:u', Event count (approx.): 672732155
  Overhead         Samples  Command  Shared Object       Symbol
-    24.07%            242  Puzzle   Puzzle              [.] 0x00000000000005cf
   + 18.51% 0x4005b6
   + 4.96% 0x4005ef
-    7.24%              73  Puzzle   Puzzle              [.] 0x00000000000005c8
   + 6.65% 0x4005b6
     0.59% 0x4005ef (predicted:83.3%)
-    7.13%              72  Puzzle   Puzzle              [.] 0x00000000000005ef
   + 5.62% 0x4005b6
   + 1.40% 0x4005ef (predicted:92.9%)
-    6.58%              66  Puzzle   Puzzle              [.] 0x00000000000005bb
   + 5.17% 0x4005b6
   + 1.31% 0x4005ef (predicted:92.3%)
```

Dynamic Profiling

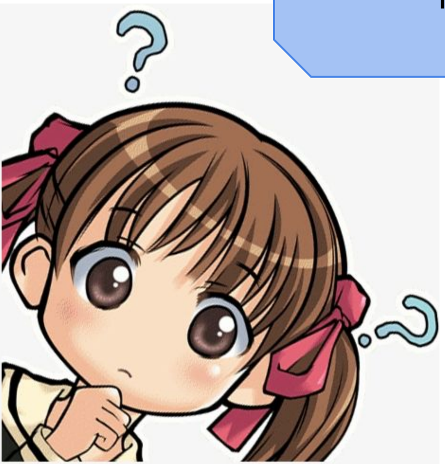
- Instrumentation based approaches:
  - Instrument every branch in the program with a counter
  - Increment counter whenever branch executes
  - Much higher CPU and memory overhead
  - Intrusive

Motivation

Speaking of intrusiveness...

Motivation

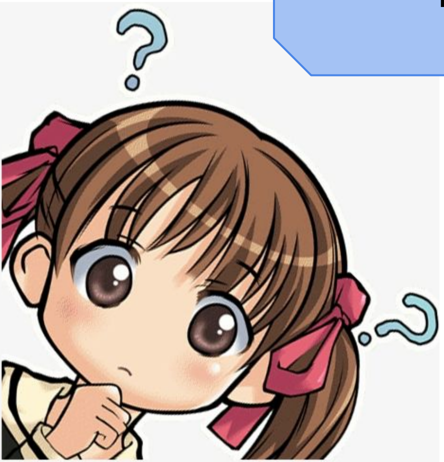
But why static branch prediction?



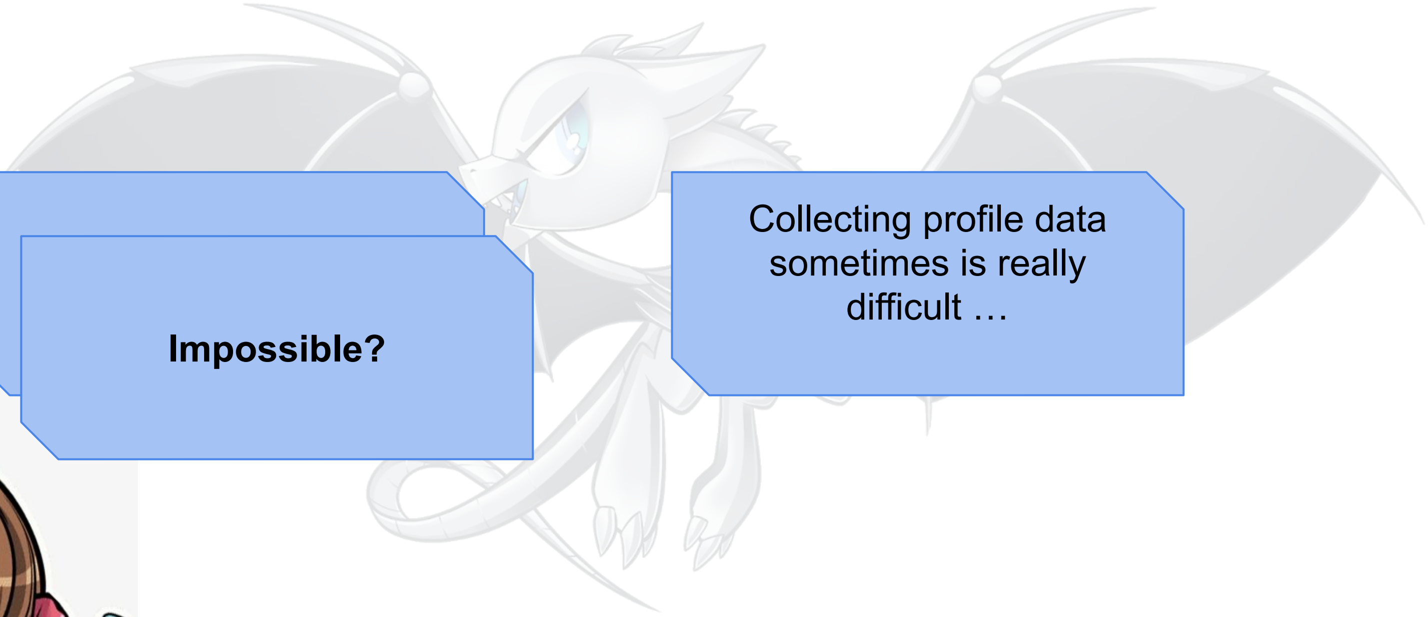
Motivation

But why static branch prediction?

Collecting profile data sometimes is really difficult …

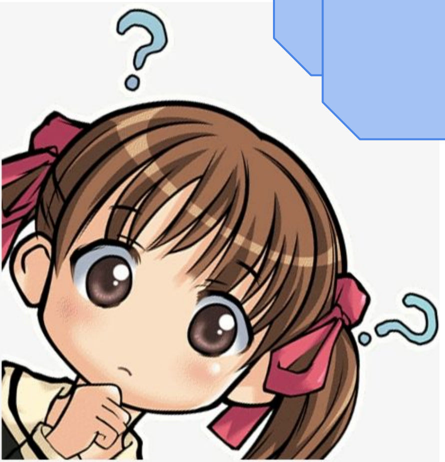


Motivation



Impossible?

Collecting profile data sometimes is really difficult …

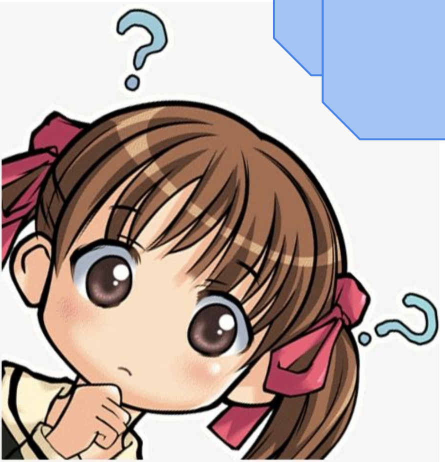


Motivation

Collecting profile data

Impossible?

Yes, think about mobile apps.



Static Profiling

- Look only at the **code**

Static Profiling

- Look only at the **code** (no execution!)

Static Profiling

- Look only at the **code** (no execution!)

Static Profiling

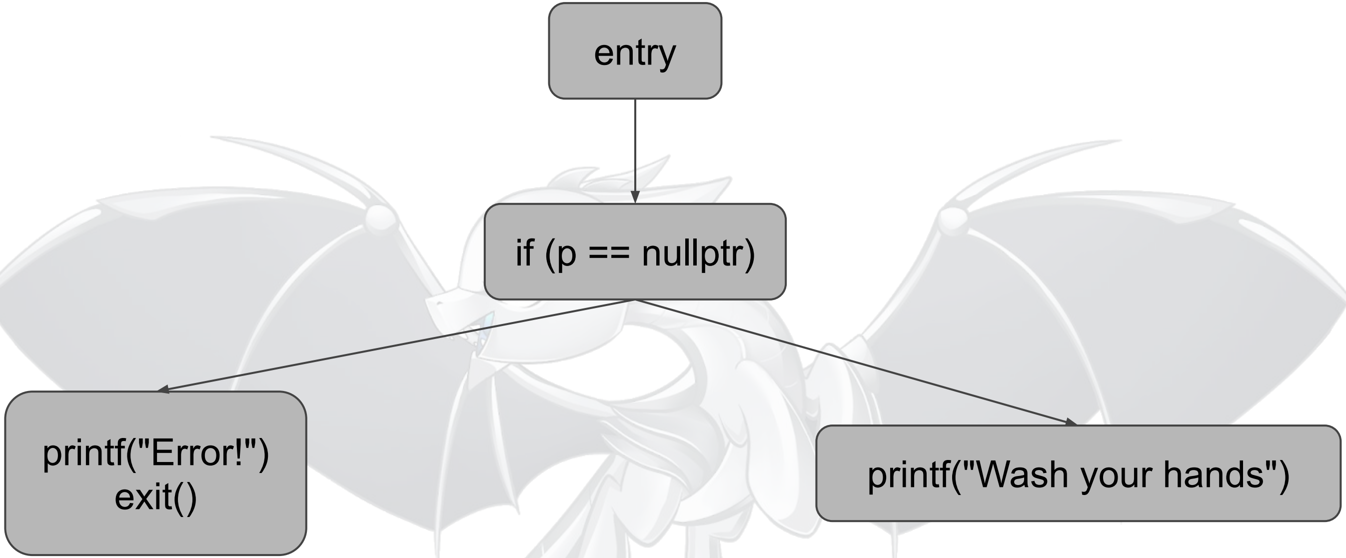
- Look only at the **code** (no execution!)
- Try to infer branch likelihood

Static Branch Prediction in the Wild

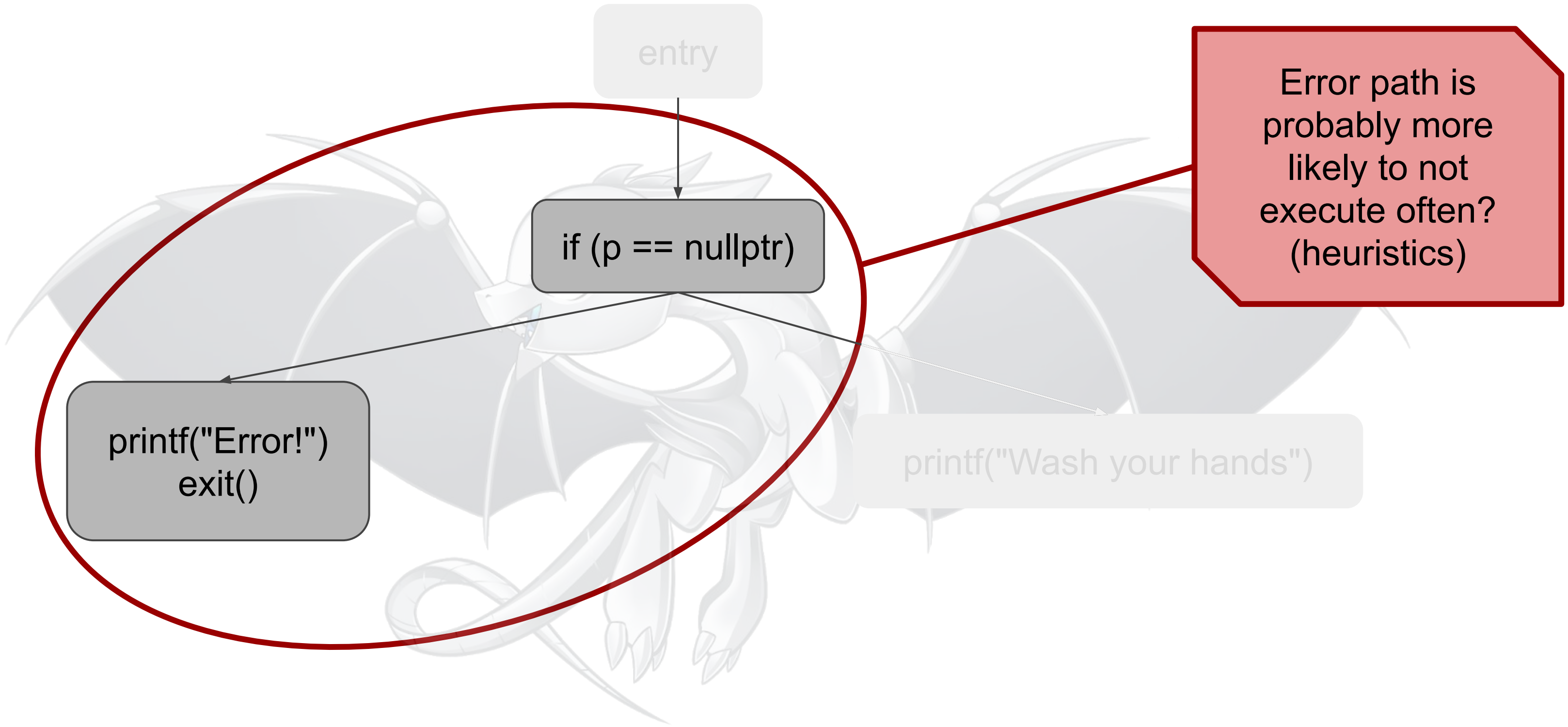
- Heuristic-based
- Machine Learning-based

Suggestion of paper to read:
Wenlei He, Julián Mestre, Sergey Pupyrev, Lei Wang, and Hongtao Yu. 2022. Profile inference revisited. Proc. ACM Program. Lang. 6, POPL, Article 52 (January 2022), 24 pages. https://doi.org/10.1145/3498714

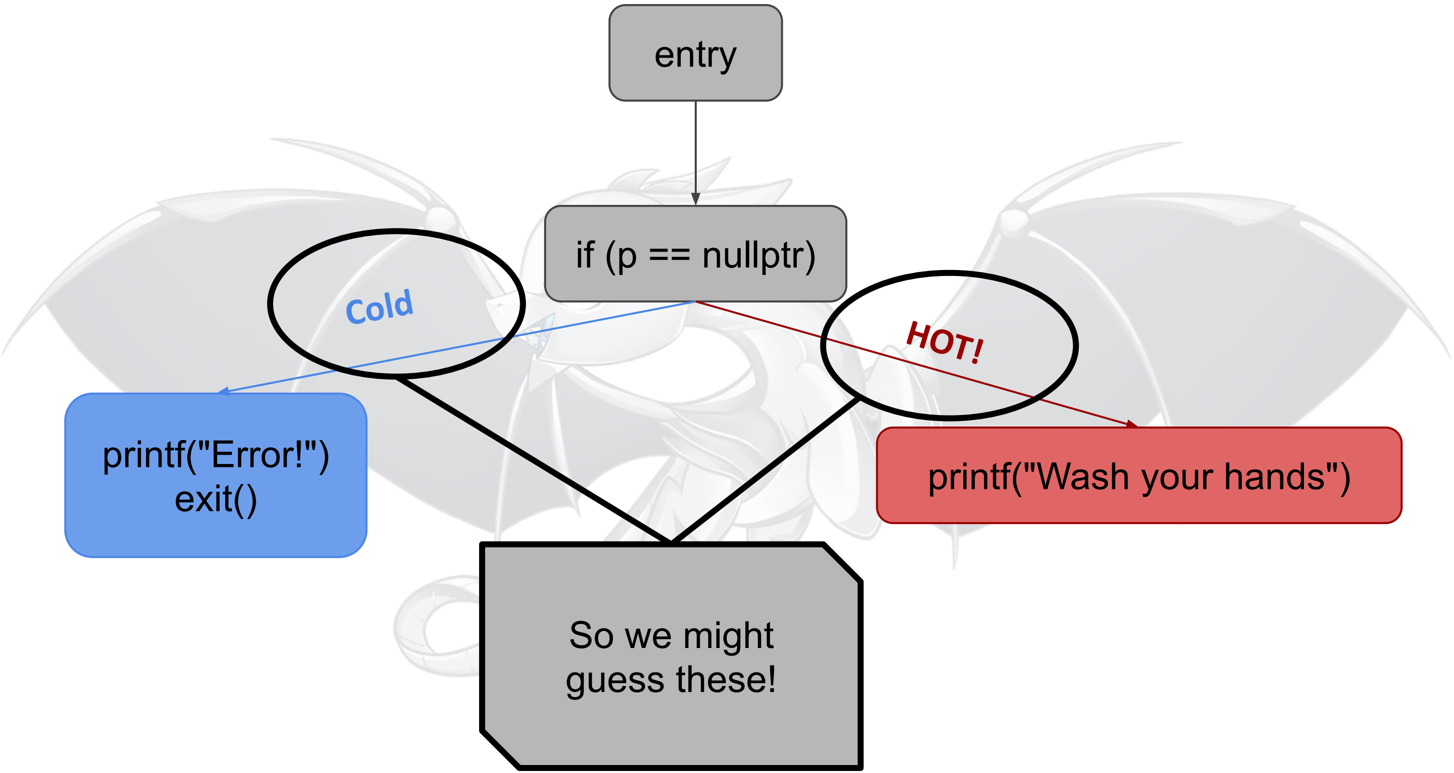
Heuristic-Based Static Profiling



Heuristic-Based Static Profiling



Heuristic-Based Static Profiling



LLVM's Static Branch Prediction

- Based solely on heuristics
  - Paper from Ball & Larus[1]
- Implemented in the BranchProbabilityInfo analysis pass

[1] Ball, Thomas, and James R. Larus. "**Branch prediction for free**." ACM SIGPLAN Notices 28.6 (1993): 300-313.

Heuristic-Based Static Profiling

- Very ad-hoc solution
- Relies on compiler developers themselves coming up with clever heuristics

Machine Learning-Based Static Profiling

- Collect corpus of programs, and static features that describe them
- Record their branch execution behaviour
- Train ML model based on features + branch data
- Create static profiles for unknown programs based on trained knowledge!

Some work in this area

To read click here

Evidence-based Static Branch Prediction using Machine Learning

Brad Calder,* Dirk Grunwald, Michael Jones, Donald Lindsay,
James Martin, Michael Mozer, and Benjamin Zorn
Department of Computer Science
Campus Box 430
University of Colorado
Boulder, CO 80309–0430 USA

September 19, 1996

Abstract

Correctly predicting the direction that branches will take is increasingly important in today's wide-issue computer architectures. The name program-based branch prediction is given to static branch prediction techniques that base their prediction on a program's structure. In this paper, we investigate a new approach to program-based branch prediction that uses a body of existing programs to predict the branch behavior in a new program. We call this approach to program-based branch prediction evidence-based static prediction, or ESP. The main idea of ESP is that the behavior of a corpus of programs can be used to infer the behavior of new programs. In this paper, we use neural networks and decision trees to map static features associated with each branch to a prediction that the branch will be taken. ESP shows significant advantages over other prediction mechanisms. Specifically, it is a program-based technique, it is effective across a range of programming languages and programming styles, and it does not rely on the use of expert-defined heuristics.

In this paper, we describe the application of ESP to the problem of static branch prediction and compare our results to existing program-based branch predictors. We also investigate the applicability of ESP across computer architectures, programming languages, compilers, and run-time systems. We provide results showing how sensitive ESP is to the number and type of static features and programs included in the ESP training sets, and compare the efficacy of static branch prediction for subroutine libraries. Averaging over a body of 43 C and Fortran programs, ESP branch prediction results in a miss rate of 20%, as compared with the 25% miss rate obtained using the best existing program-based heuristics.

To read click here

Profile Guided Optimization without Profiles: A Machine Learning Approach

Nadav Rotem
Meta, Inc.

Chris Cummins
Meta AI

January 5, 2022

Abstract

Profile guided optimization is an effective technique for improving the optimization ability of compilers based on dynamic behavior, but collecting profile data is expensive, cumbersome, and requires regular updating to remain fresh.

We present a novel statistical approach to inferring branch probabilities that improves the performance of programs that are compiled without profile guided optimizations. We perform offline training using information that is collected from a large corpus of binaries that have

```
// Calculate Edge Weights using "Pointer Heuristics".
// Predict a comparison between two pointer or pointer
// and NULL will fail.
bool BranchProbabilityInfo::
    calcPointerHeuristics(const BasicBlock *BB) {
  const BranchInst *BI = dyn_cast<BranchInst>(..);

  Value *Cond = BI->getCondition();
  ICmpInst *CI = dyn_cast<ICmpInst>(Cond);
  if (!CI || !CI->isEquality())
    return false;

 // p != 0   ->   isProb = true
 // p == 0   ->   isProb = false
 // p != q   ->   isProb = true
 // p == q   ->   isProb = false;
 bool prob = CI->getPredicate() == ICmpInst::ICMP_NE;
```

To read click here

VESPA: Static Profiling for Binary Optimization

ANGÉLICA APARECIDA MOREIRA, UFMG, Brazil
GUILHERME OTTONI, Facebook, Inc., USA
FERNANDO MAGNO QUINTÃO PEREIRA, UFMG, Brazil

Over the past few years, there has been a surge in the popularity of binary optimizers such as BOLT, Propeller, Janus and HALO. These tools use dynamic profiling information to make optimization decisions. Although effective, gathering runtime data presents developers with inconveniences such as unrepresentative inputs, the need to accommodate software modifications, and longer build times. In this paper, we revisit the static profiling technique proposed by Calder *et al.* in the late 90's, and investigate its application to drive binary optimizations, in the context of the BOLT binary optimizer, as a replacement for dynamic profiling. A few core modifications to Calder *et al.'s* original proposal, consisting of new program features and a new regression model, are sufficient to enable some of the gains obtained through runtime profiling. An evaluation of BOLT powered by our static profiler on four large benchmarks (clang, GCC, MySQL and PostgreSQL) yields binaries that are 5.47% faster than the executables produced by clang -O3.

The goal of this Work

- Optimize binaries using static profile inferred by a machine learning model

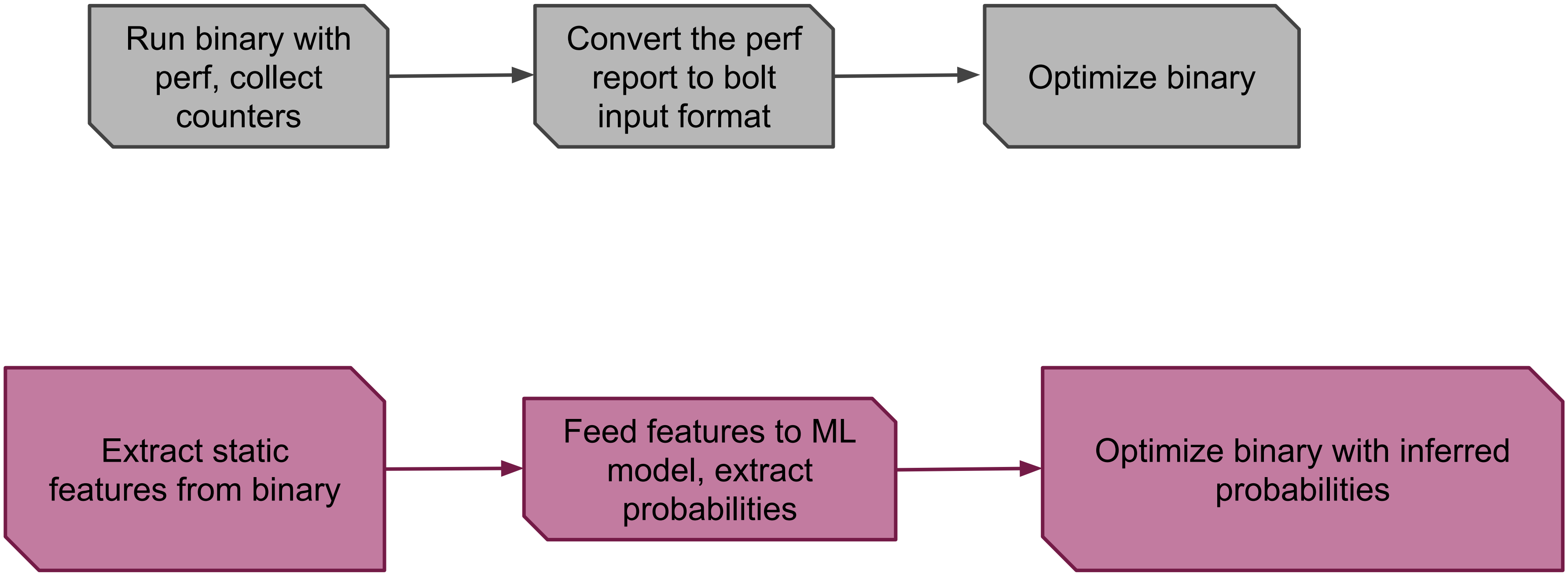
- **V**intage **ESP A**mended (VESPA)
  - Extension of Calder's work: **E**vidence-Based **S**tatic Branch **P**rediction (ESP) [1]

[1] Brad Calder, Dirk Grunwald, Michael Jones, Donald Lindsay, James Martin, Michael Mozer, and Benjamin Zorn. 1997. Evidence-based static branch prediction using machine learning. ACM Trans. Program. Lang. Syst. 19, 1 (Jan. 1997), 188–222. DOI:https://doi.org/10.1145/239912.239923

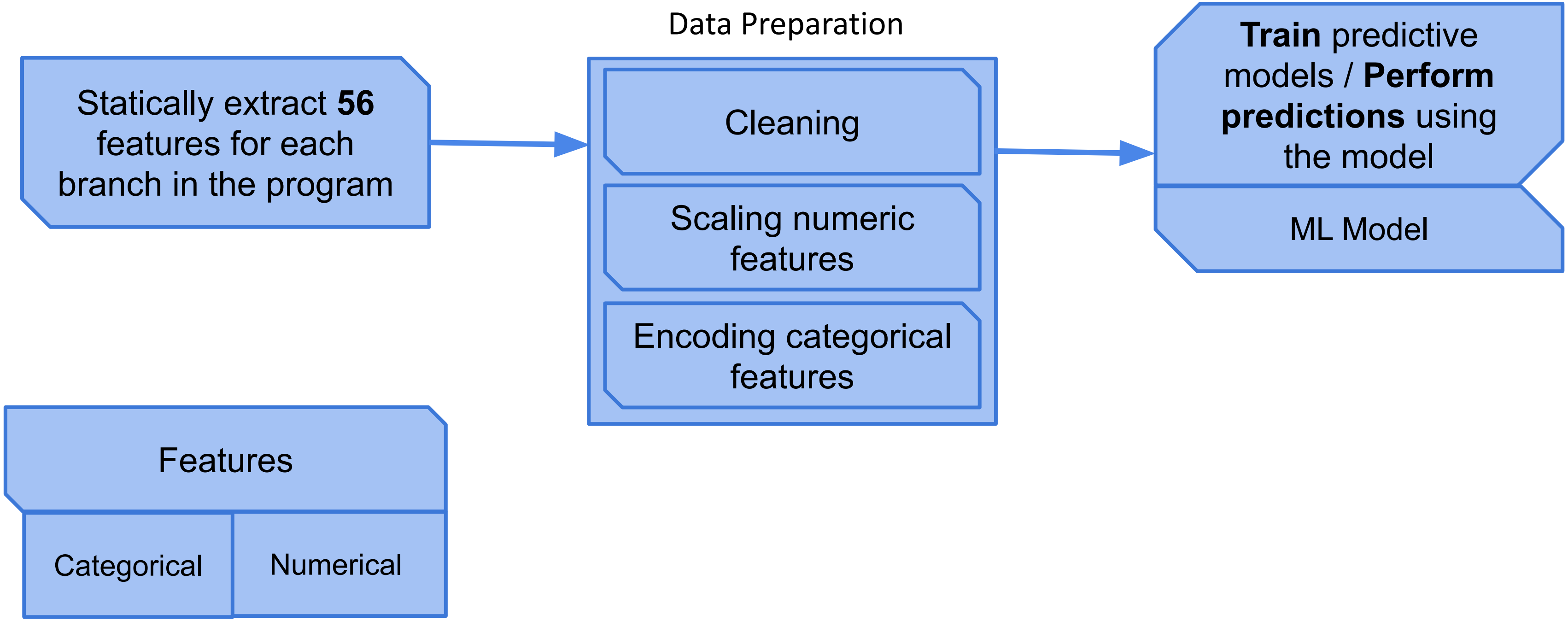
Calder vs VESPA

- Collect corpus of programs, and static features that describe them
- Record their branch execution behaviour
- Train ML model based on features + branch data
- Create static profiles for unknown programs based on trained knowledge!

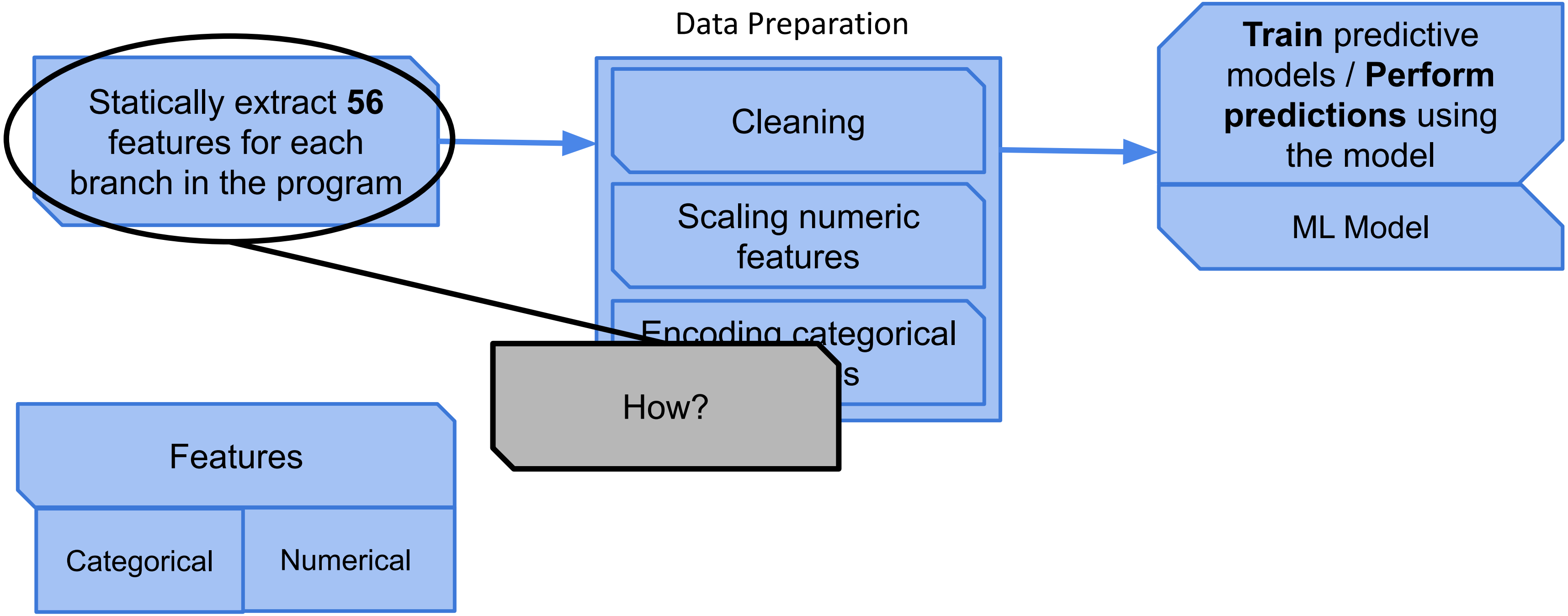
Infrastructure overview - Static BOLT usage!



ML Pipeline Overview



ML Pipeline Overview

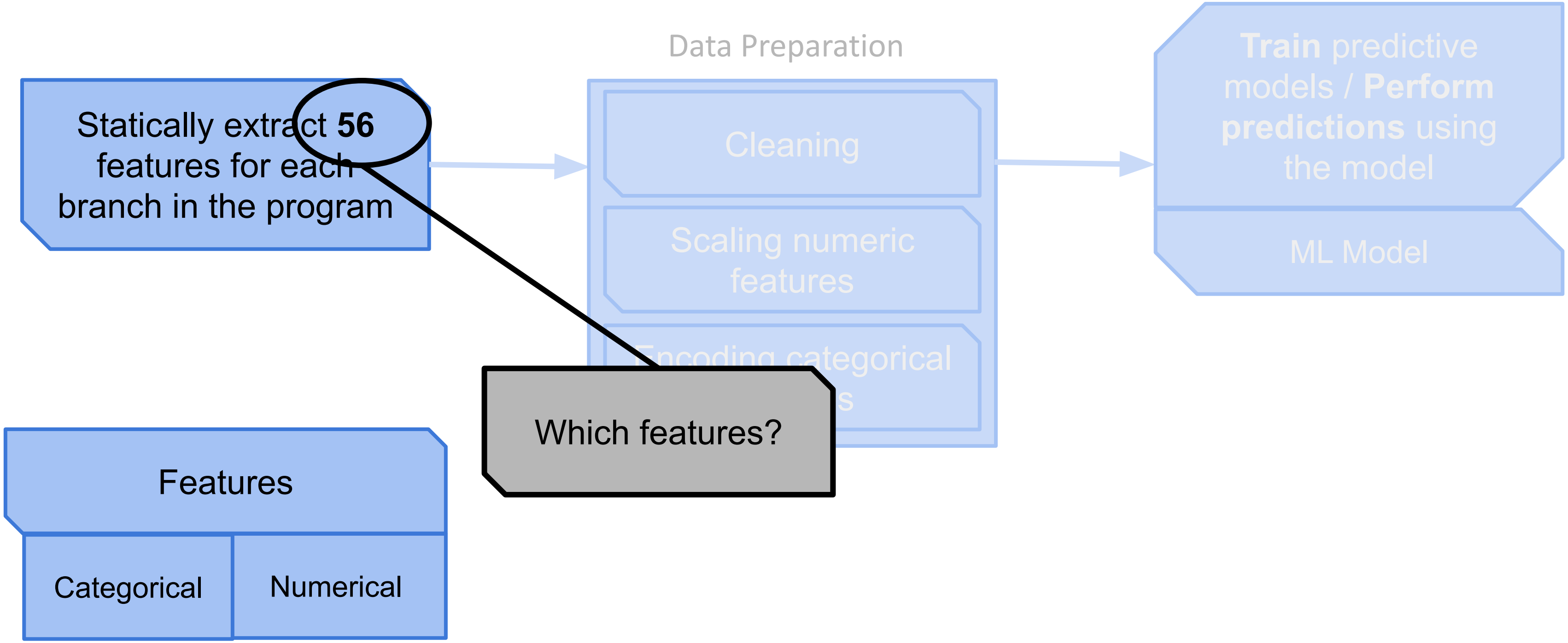


Feature Miner

- Implemented the FeatureMiner pass in BOLT
- Runs after binary disassembling/CFG construction
- Analyzes the CFG to collect static features proposed by Calder et al.[1], as well as others devised by us for each branch.

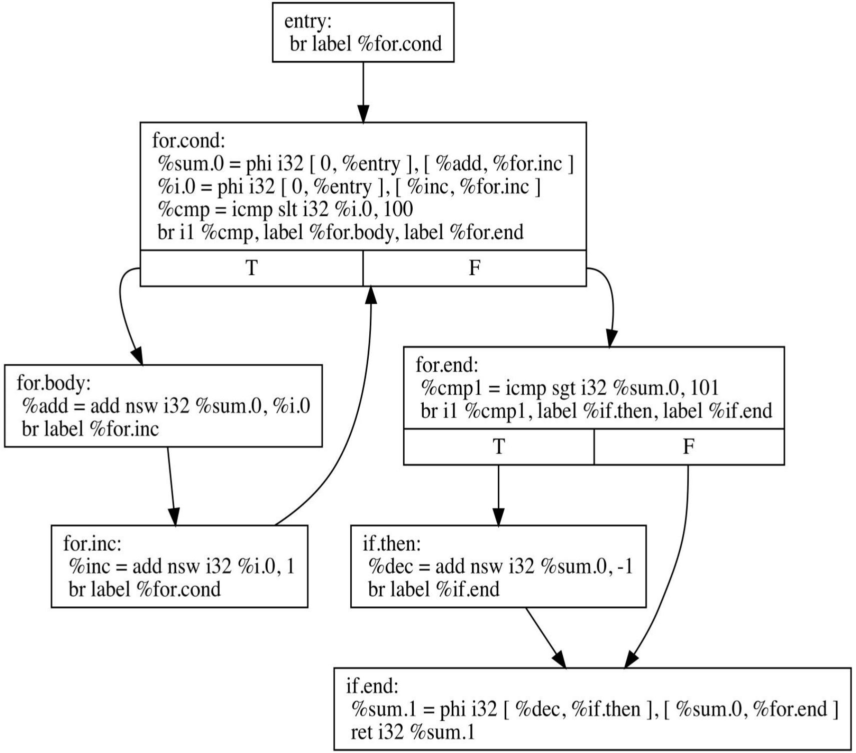
[1] Brad Calder, Dirk Grunwald, Michael Jones, Donald Lindsay, James Martin, Michael Mozer, and Benjamin Zorn. 1997. Evidence-based static branch prediction using machine learning. ACM Trans. Program. Lang. Syst. 19, 1 (Jan. 1997), 188–222. DOI:https://doi.org/10.1145/239912.239923

ML Pipeline Overview



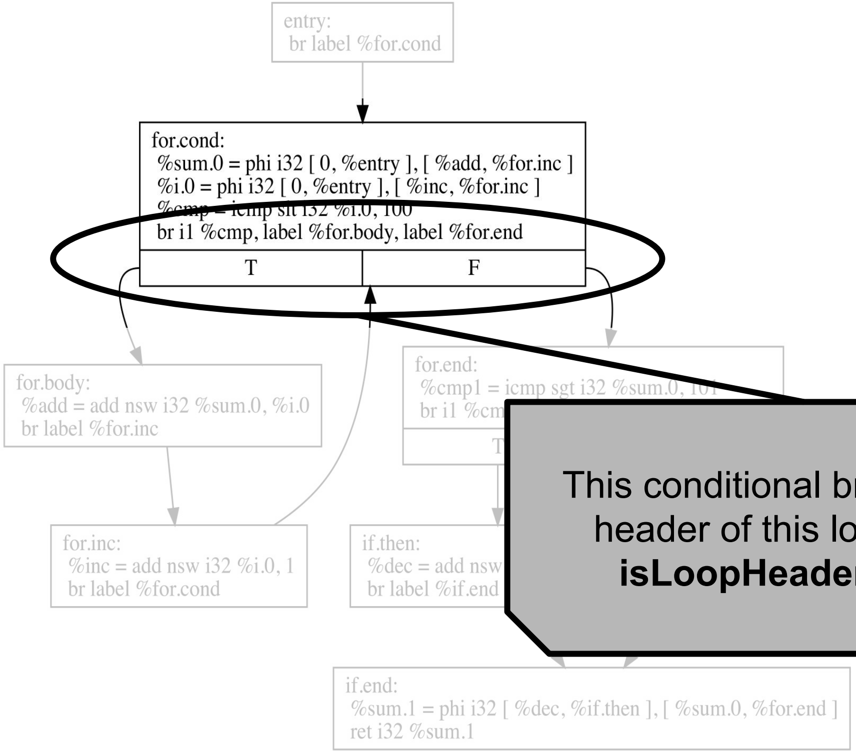
An Example

isLoopHeader



An Example

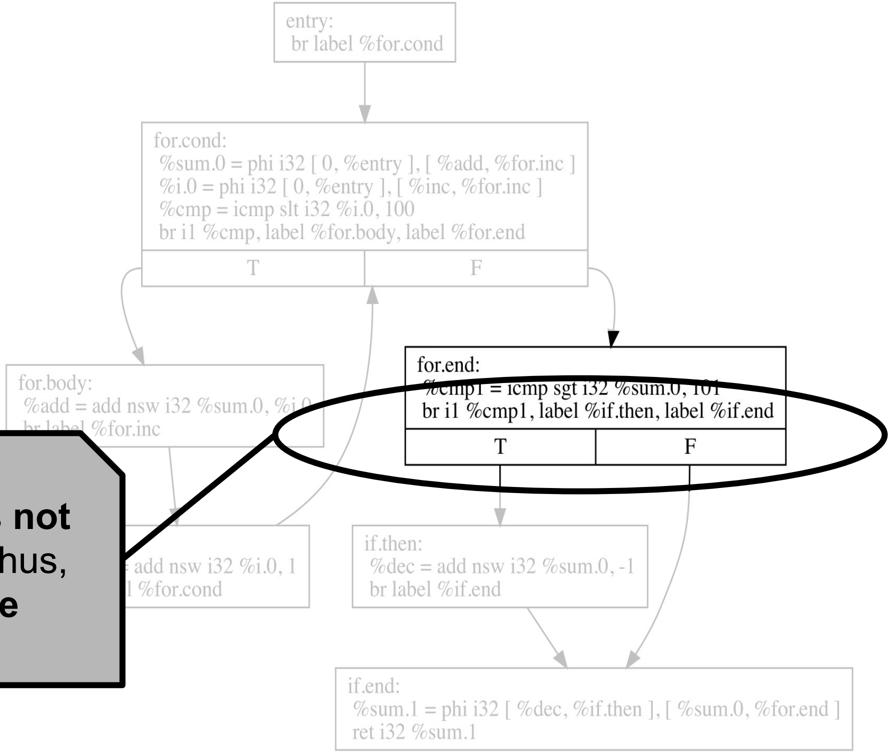
isLoopHeader



This conditional branch is the header of this loop. Thus, **isLoopHeader = True**

An Example

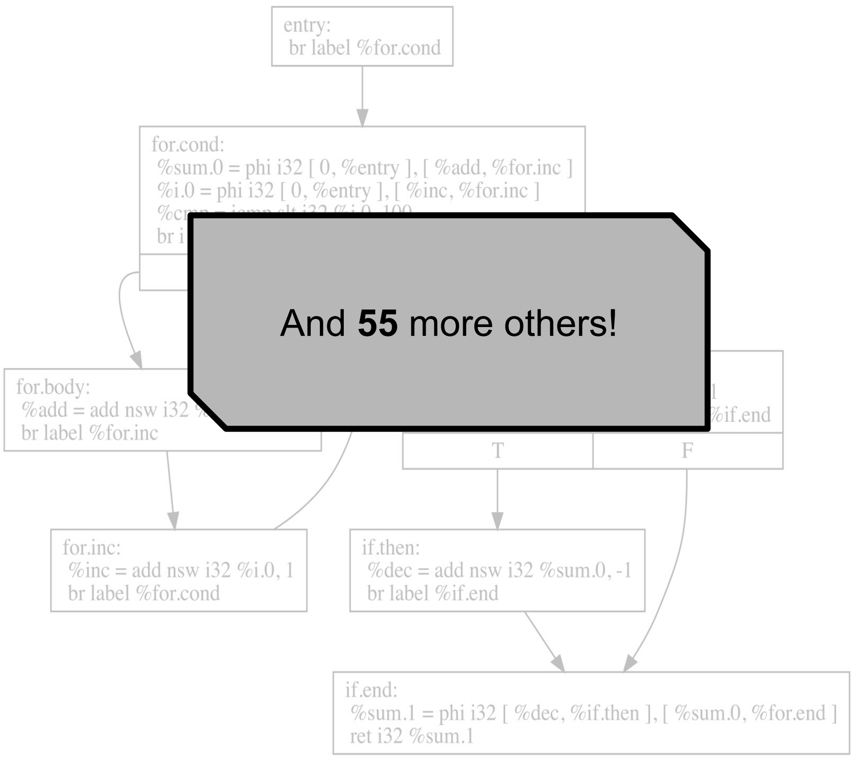
isLoopHeader



This conditional branch is **not** the header of any loop. Thus, **isLoopHeader = False**

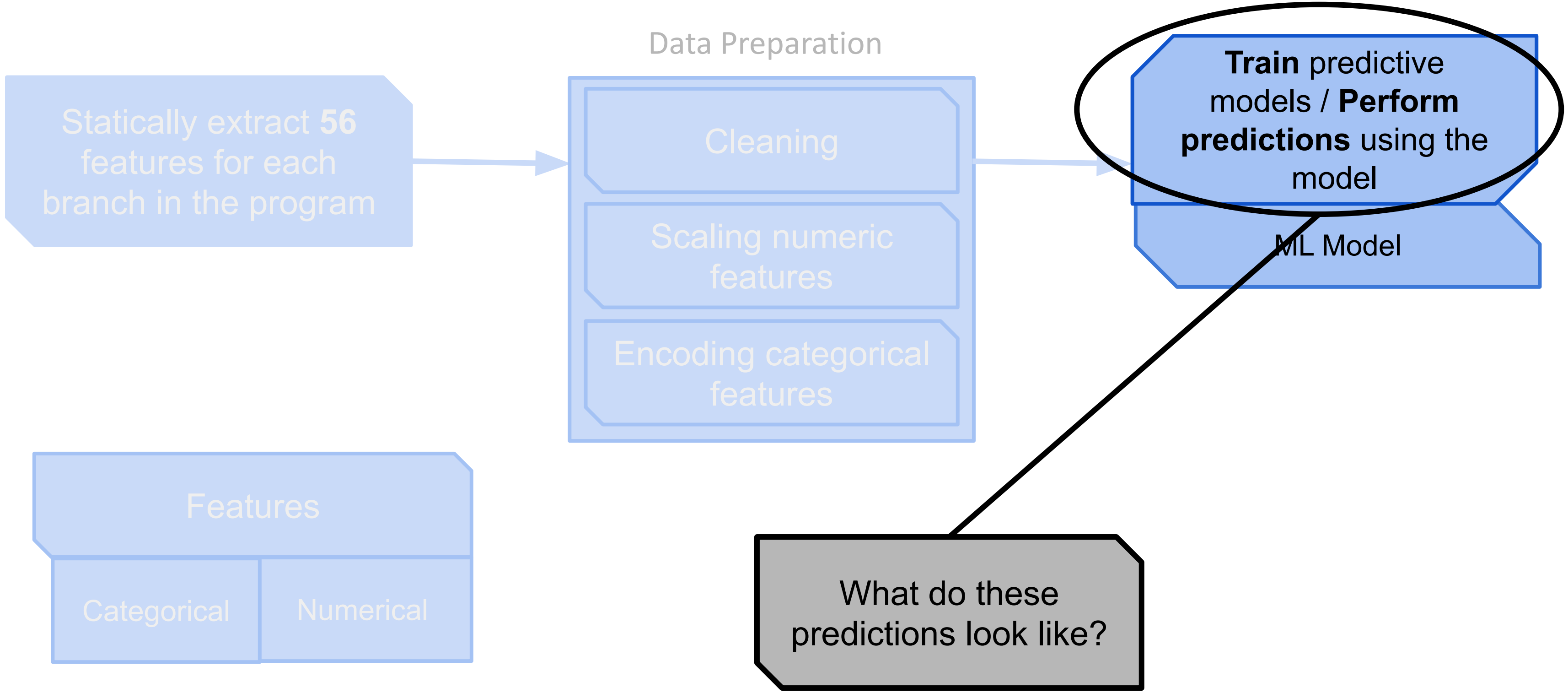
An Example

isLoopHeader

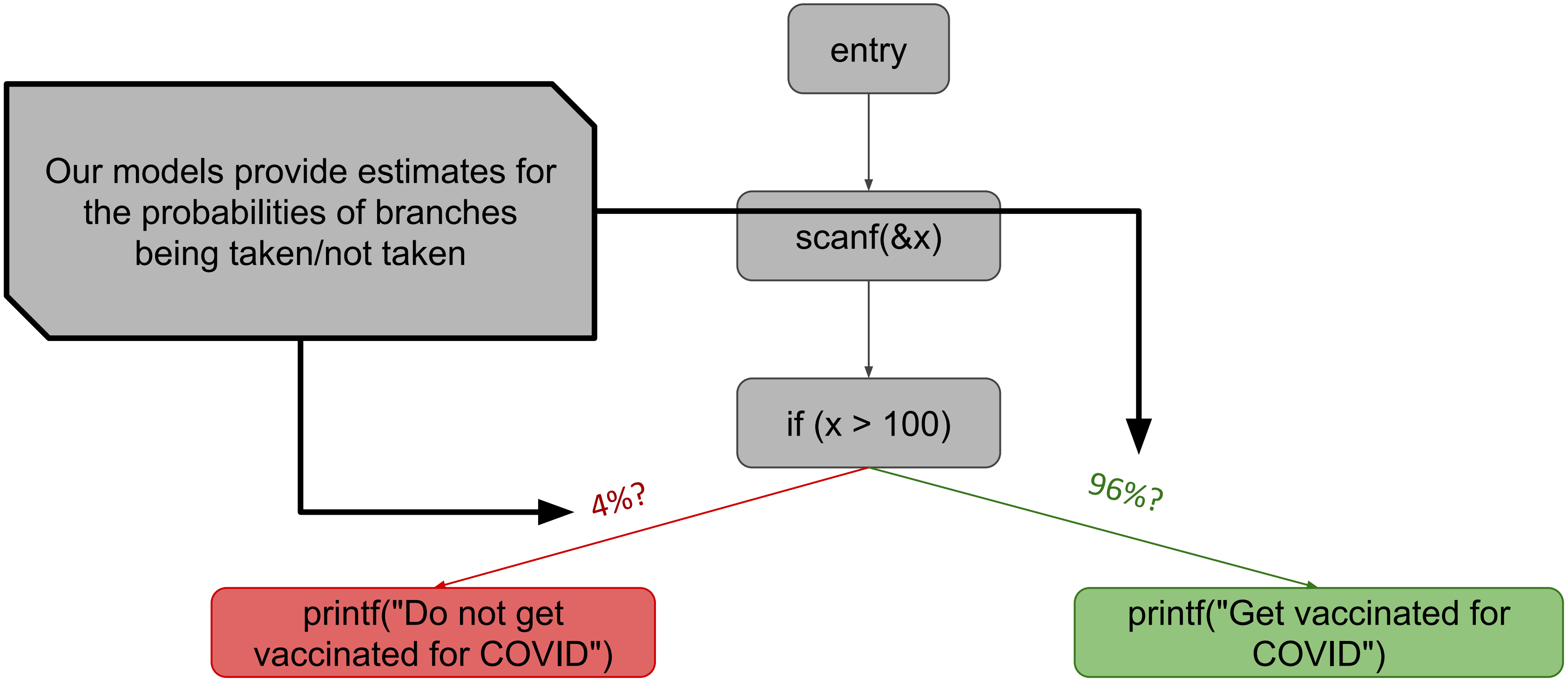


And **55** more others!

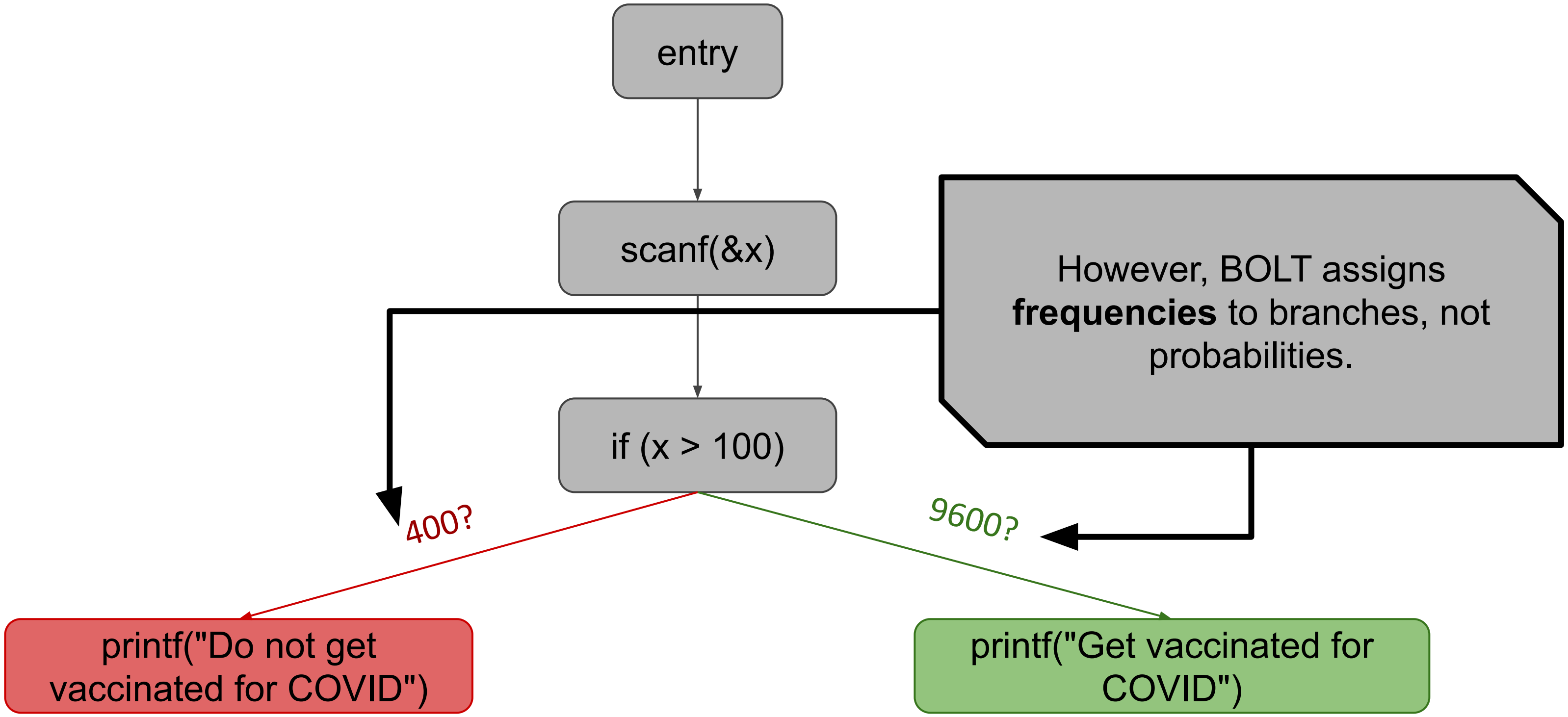
ML Pipeline Overview



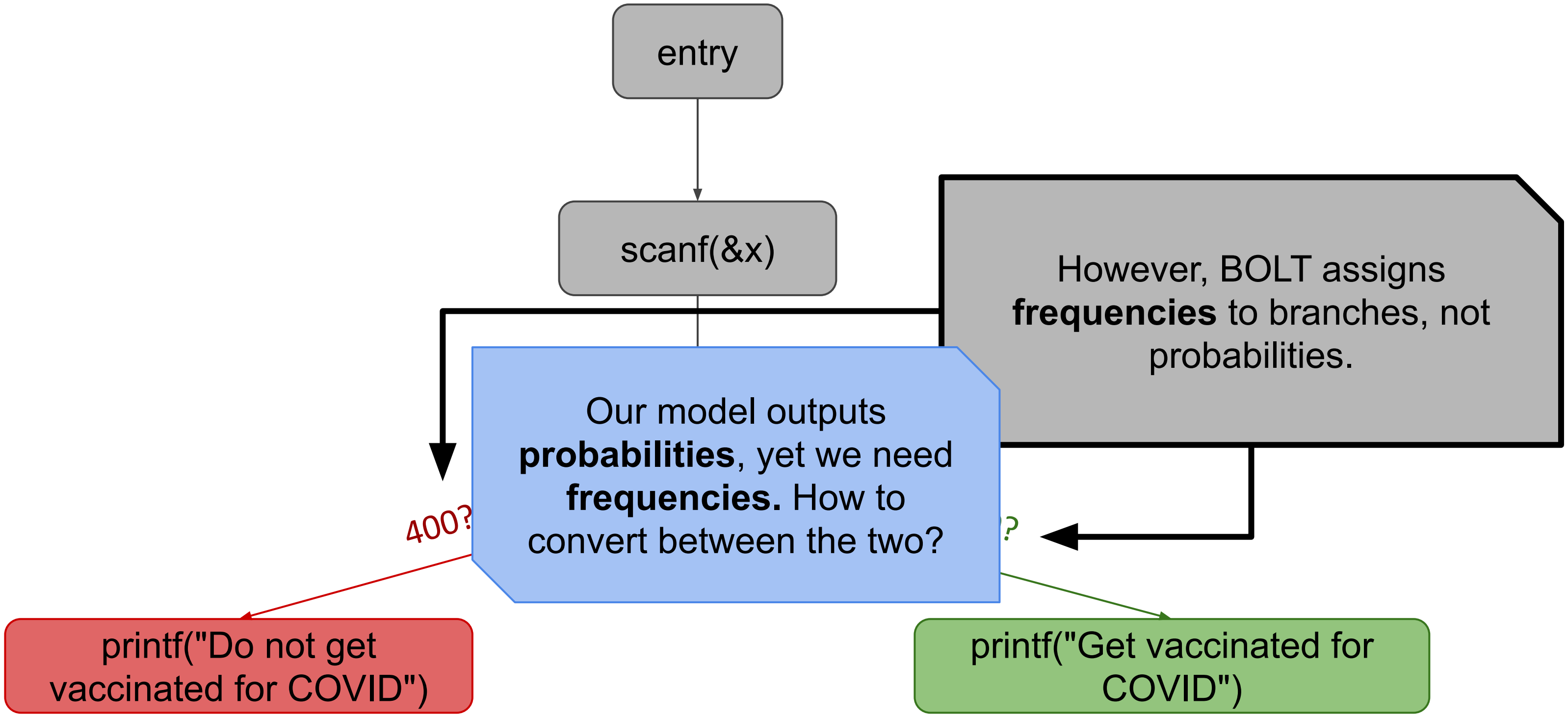
Prediction Output



Prediction Output



Prediction Output



Getting Frequencies out of Probabilities

- Technique proposed by Wu & Larus[1]
- Calculate basic block and Control Flow Graph (CFG) edge frequencies intra-procedurally (within functions)
- Propagate probabilities starting from the entry block, according to these equations:

$$bfreq(b_i) = 1 \qquad \text{(if entry block)}$$

$$bfreq(b_i) = \sum_{b_p \in pred(b_i)} freq(b_p \rightarrow b_i) \qquad \text{(otherwise)}$$

$$freq(b_i \rightarrow b_j) = bfreq(b_i) prob(b_i \rightarrow b_j)$$

[1] Wu, Youfeng, and James R. Larus. "**Static branch frequency and program profile analysis**." Proceedings of the 27th annual international symposium on Microarchitecture. 1994.

Getting Frequencies out of Probabilities

- Technique proposed by Wu & Larus[1]
- Calculate basic block and Control Flow Graph (CFG) edge frequencies intra-procedurally (within functions)
- Propagate probabilities starting from the entry ... ations:

$$bfreq(b_i) =$$

$$bfreq(b_i) = \sum_{b_p \in pred(b_i)}$$

$$freq(b_i \rightarrow b_j) = bfreq(b_i$$

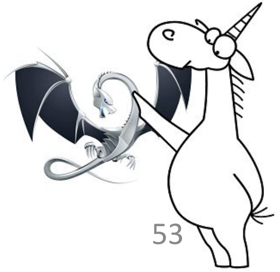
This technique estimates execution frequency (not absolute counts) with static program analysis!!!!

[1] Wu, Youfeng, and James R. Larus. **"Static branch frequency and program profile analysis."** Proceedings of the 27th annual international symposium on Microarchitecture. 1994.

Caveats to Static Inference

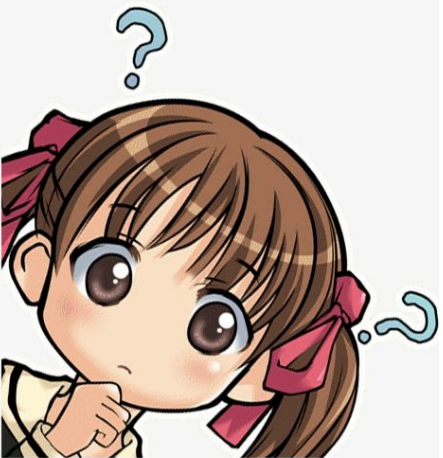
- Indirect branches cannot have their targets inferred statically
  - Adds imprecision to **intra-procedural** inference!
- Similarly, indirect procedure calls (virtual method invocations, function pointer calls, etc.) also cannot be resolved statically
  - Adds imprecision to **inter-procedural** inference!

Experiments



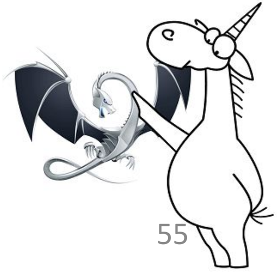
Hypothesis

"BOLT using **static profile data** can still provide some of the gains as using **real profile data**."



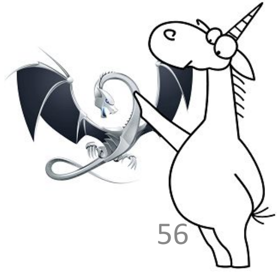
Setup

- Trained models on a dataset with 243 programs:
  - Corpus of 2,093,873 two-way conditional branched but only 513,316 associated with branch predictions
- All binaries used in training and as baseline were compiled using Clang 12 with -O3
- 80% of the branches used for training and 20% for test and validation

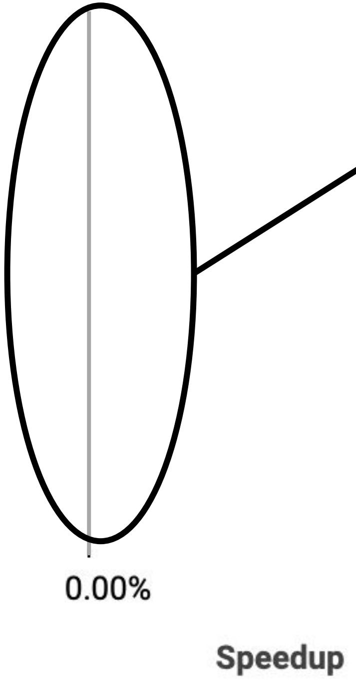


VESPA in Practice

RQ2: What are the performance gains of our approach when compared to the baseline compiler at its highest optimization level, and to a binary optimized with dynamic profiling information?

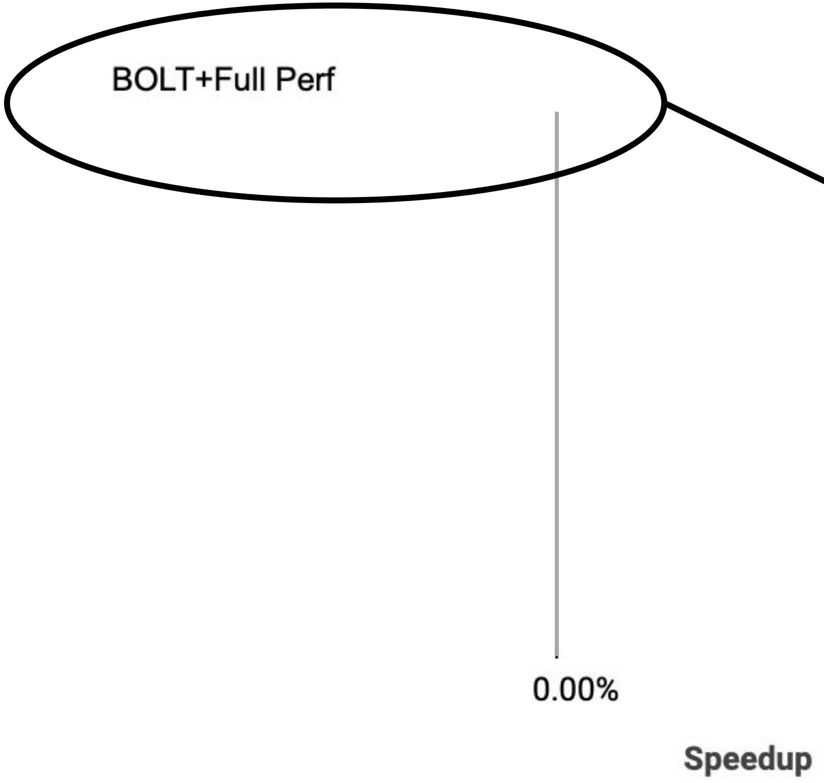


Experiments



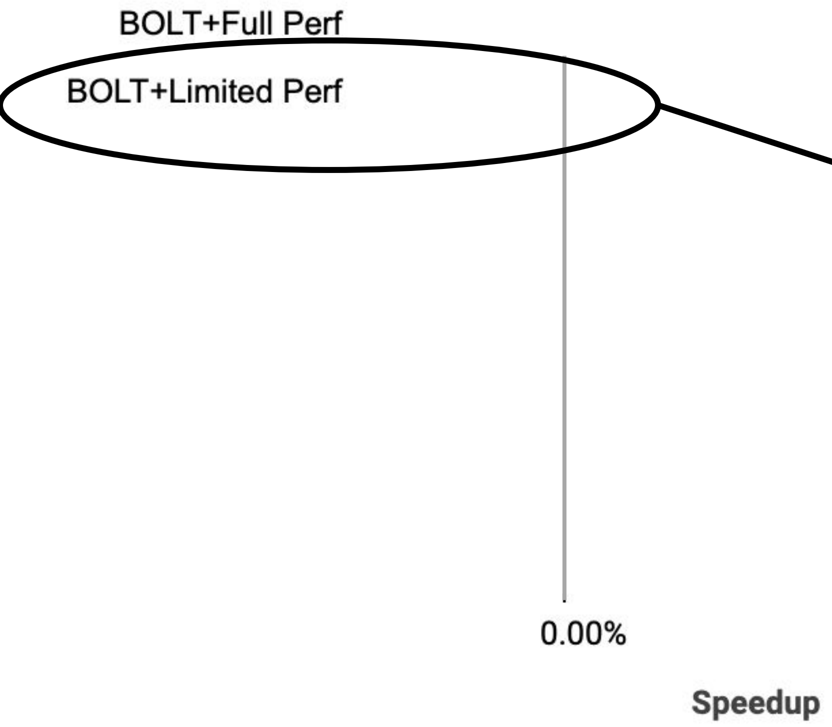
Our baseline is the original binary, optimized with -O3

Experiments



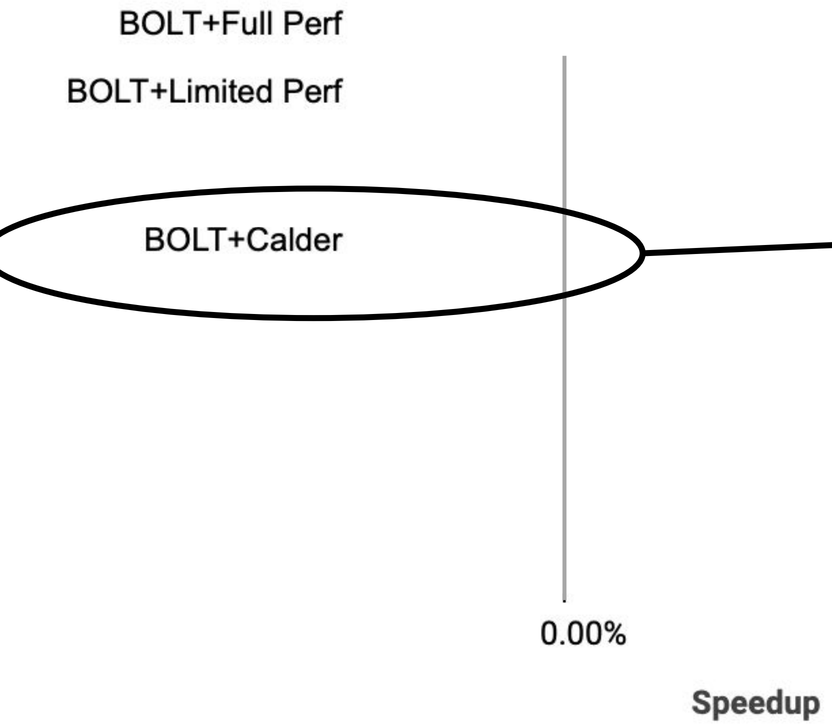
BOLT+Full Perf indicates a binary optimized using BOLT's traditional usage: with a dynamic execution profile collected with **perf**

Experiments



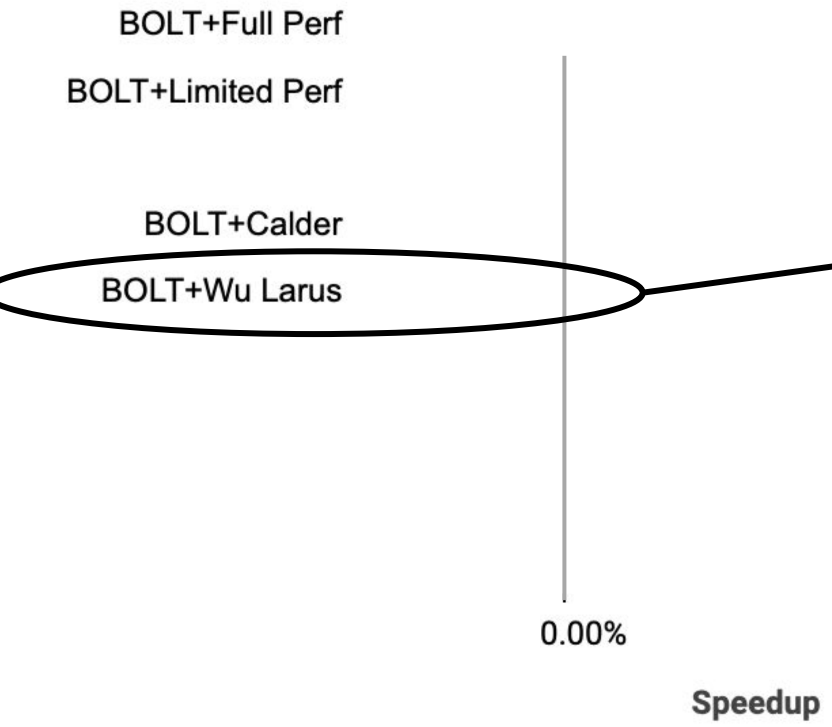
BOLT+Limited Perf indicates a binary optimized using BOLT's traditional usage: with a stripped dynamic execution profile collected with **perf.**

Experiments

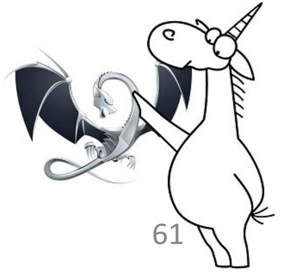


BOLT+Calder indicates a binary built using BOLT with a static profile inferred using the heuristics due to Calder.

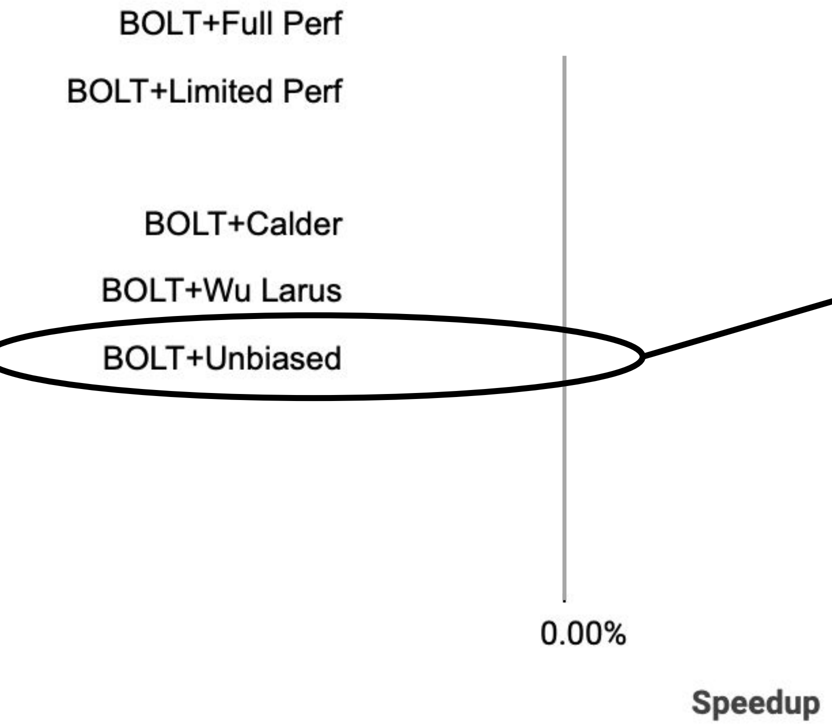
Experiments



BOLT+Wu Larus indicates a binary built using BOLT with a static profile inferred using the heuristics due to Wu et al.

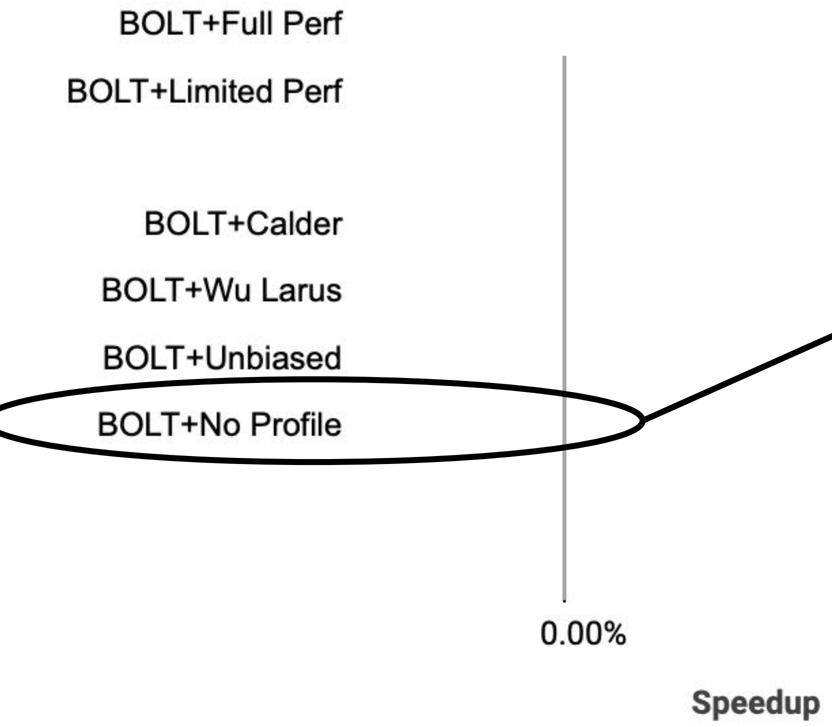


Experiments

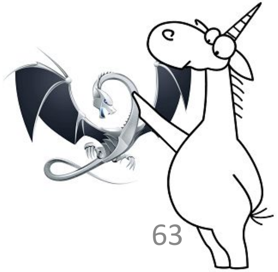


BOLT+Unbiased indicates a binary built using BOLT and assuming every two-way branch has a 50/50 probability split.

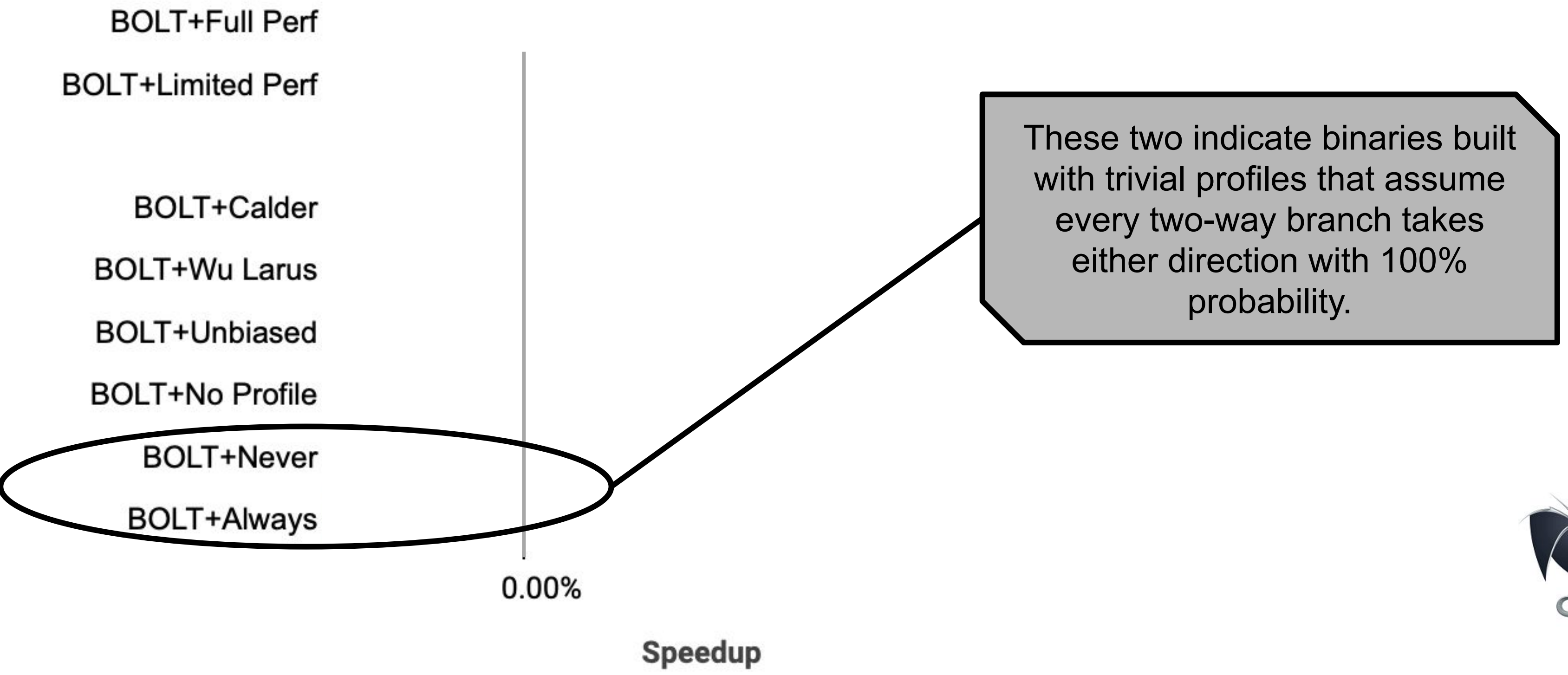
Experiments



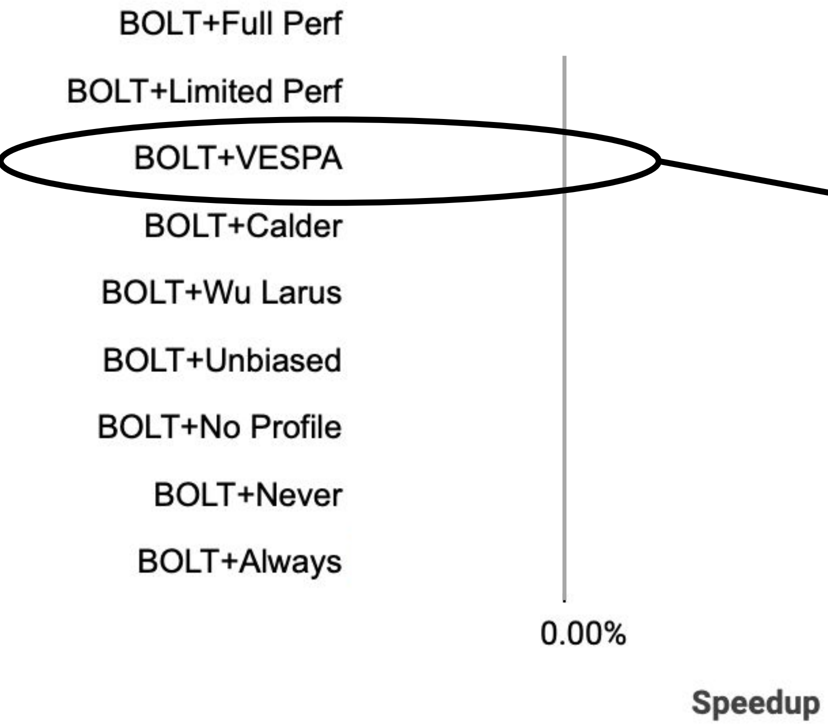
BOLT+No Profile is a binary built with BOLT fed with no profile information at all.



Experiments

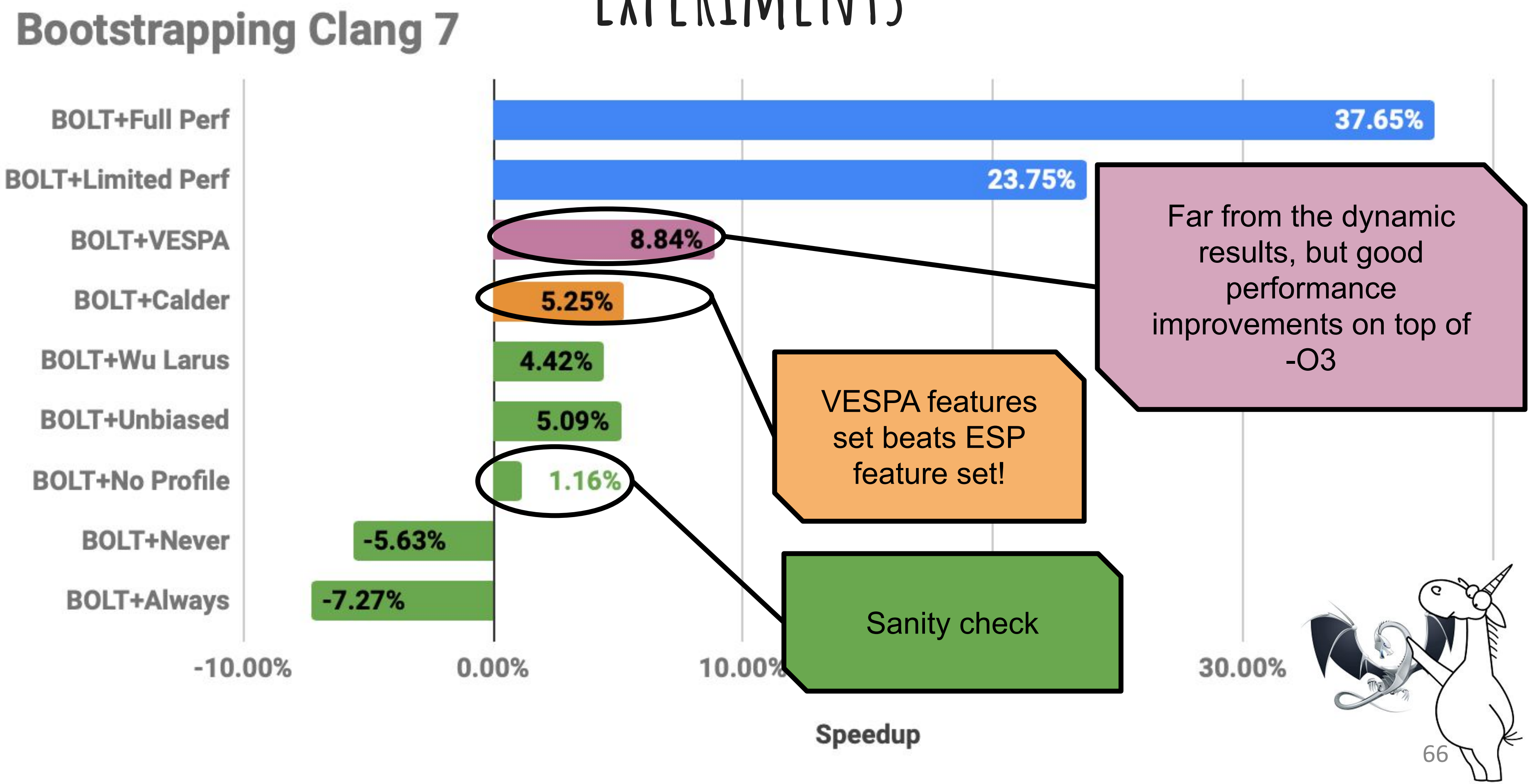


Experiments

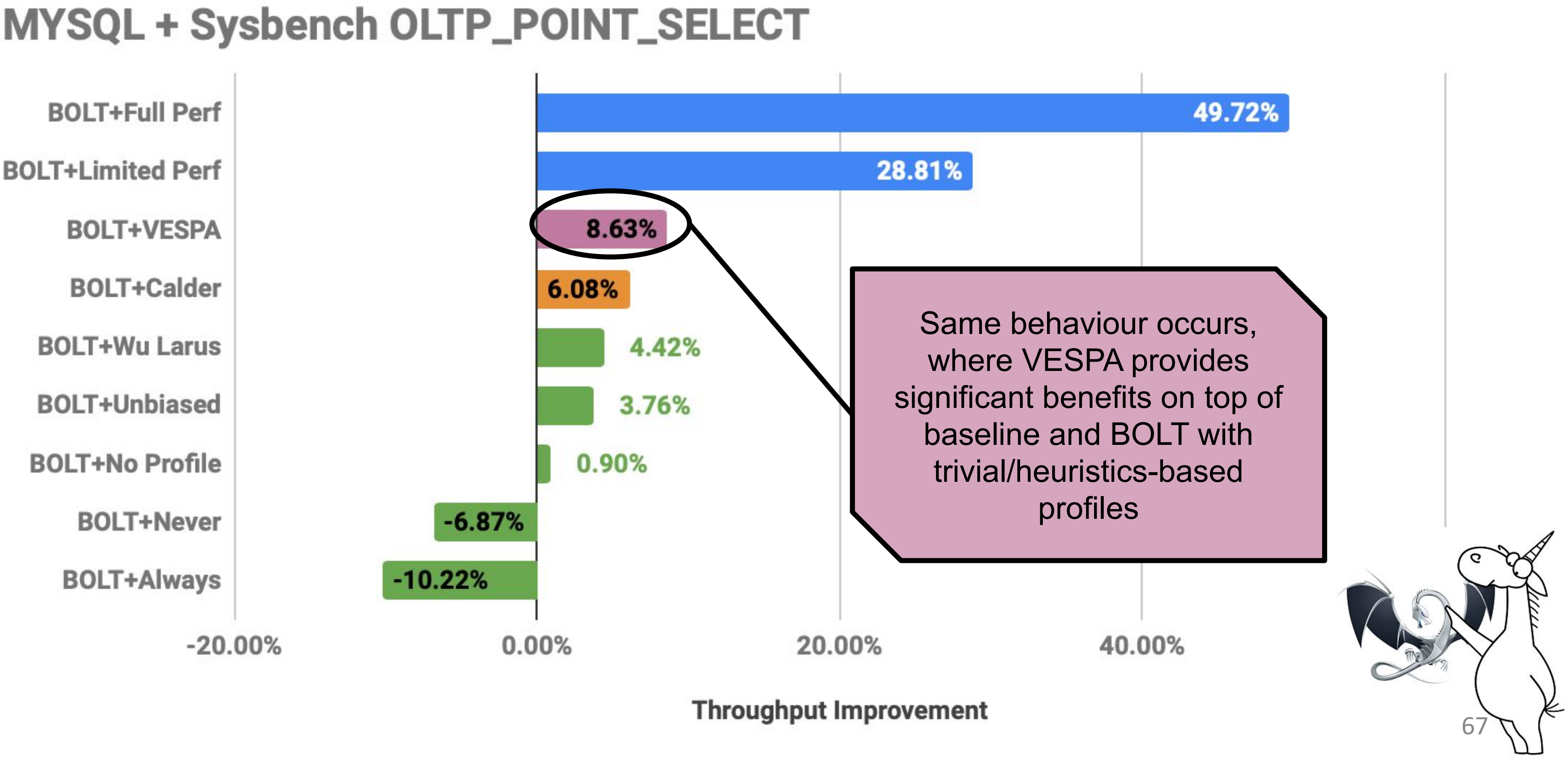


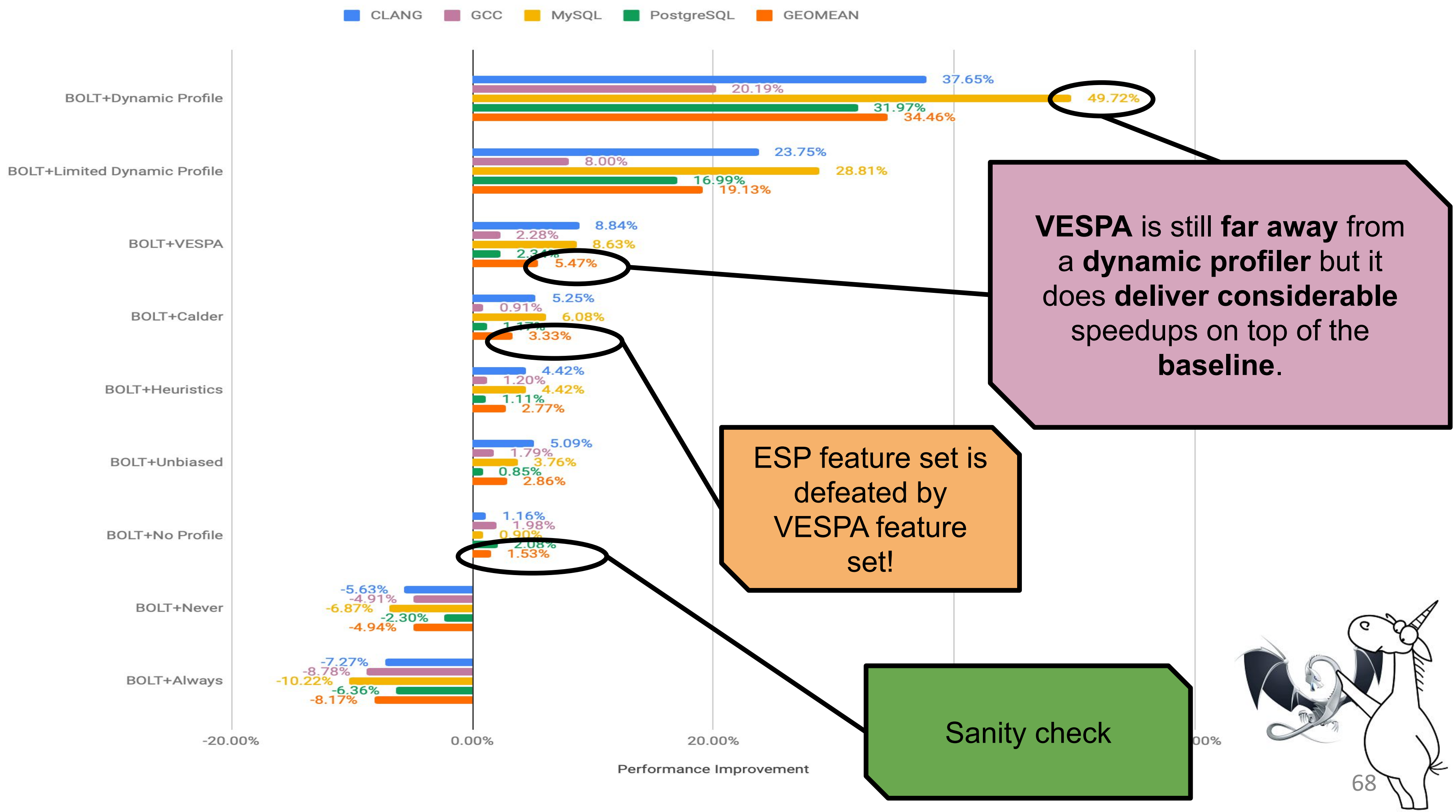
And finally, BOLT+VESPA is a binary built using BOLT with our technique for static profile inference.

Experiments



Experiments





CLANG
GCC
MySQL
PostgreSQL
GEOMEAN
BOLT+Dynamic Profile
37.65%
20.19%
49.72%
31.97%
34.46%
BOLT+Limited Dynamic Profile
23.75%
8.00%
28.81%
16.99%
19.13%
BOLT+VESPA
8.84%
2.28%
8.63%
5.47%
BOLT+Calder
5.25%
0.91%
6.08%
3.33%
BOLT+Heuristics
4.42%
1.20%
4.42%
1.11%
2.77%
BOLT+Unbiased
5.09%
1.79%
3.76%
0.85%
2.86%
BOLT+No Profile
1.16%
1.98%
0.90%
2.08%
1.53%
BOLT+Never
-5.63%
-4.91%
-6.87%
-2.30%
-4.94%
BOLT+Always
-7.27%
-8.78%
-10.22%
-6.36%
-8.17%
-20.00%
0.00%
20.00%
Performance Improvement
VESPA is still **far away** from a **dynamic profiler** but it does **deliver considerable** speedups on top of the **baseline**.
ESP feature set is defeated by VESPA feature set!
Sanity check

Compilers Laboratory

Department of Computer Science
Universidade Federal de Minas Gerais
Federal University of Minas Gerais, Brazil

angelica.moreira@dcc.ufmg.br

Thanks to my sponsors:

FACEBOOK Meta

Microsoft

What did you learn today?

- Ways to do software branch prediction;
- Types of static branch predictors;
- How we could do static profiling;
- How we can use ML as a tool to help in the task of generating static profile;
- How we can get branch and block frequencies when you only have probabilities.