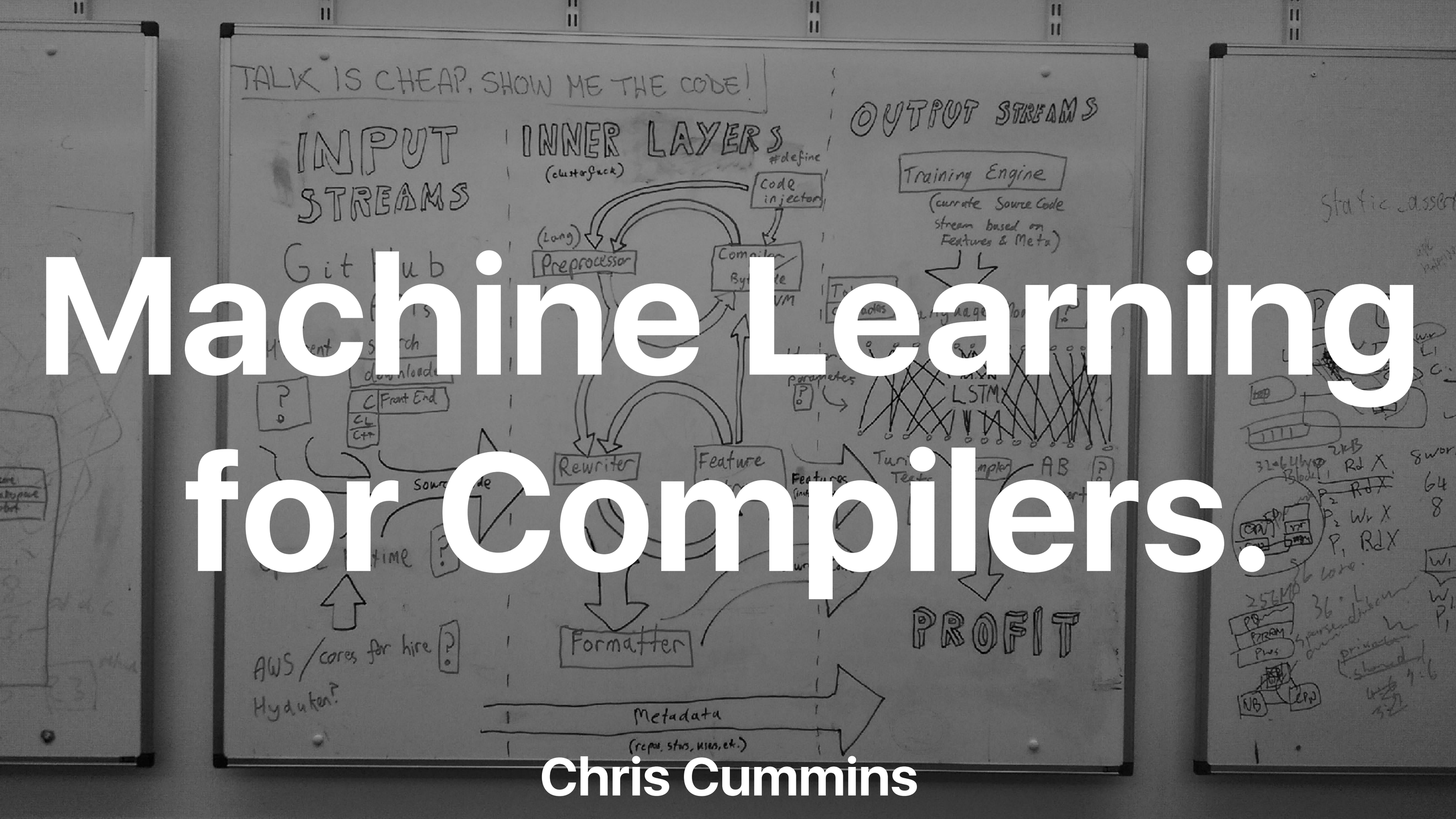
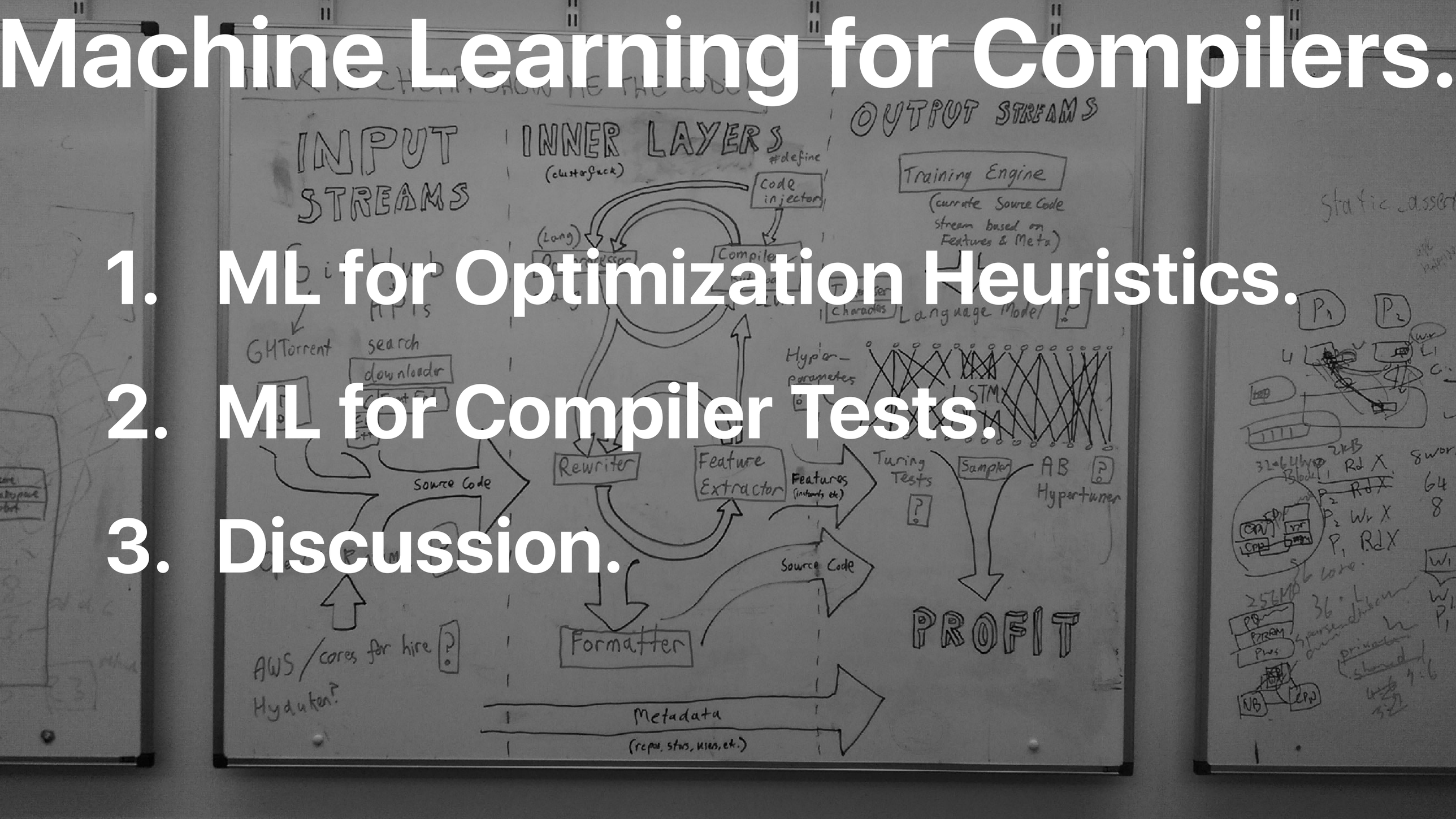


Machine Learning for Compilers.

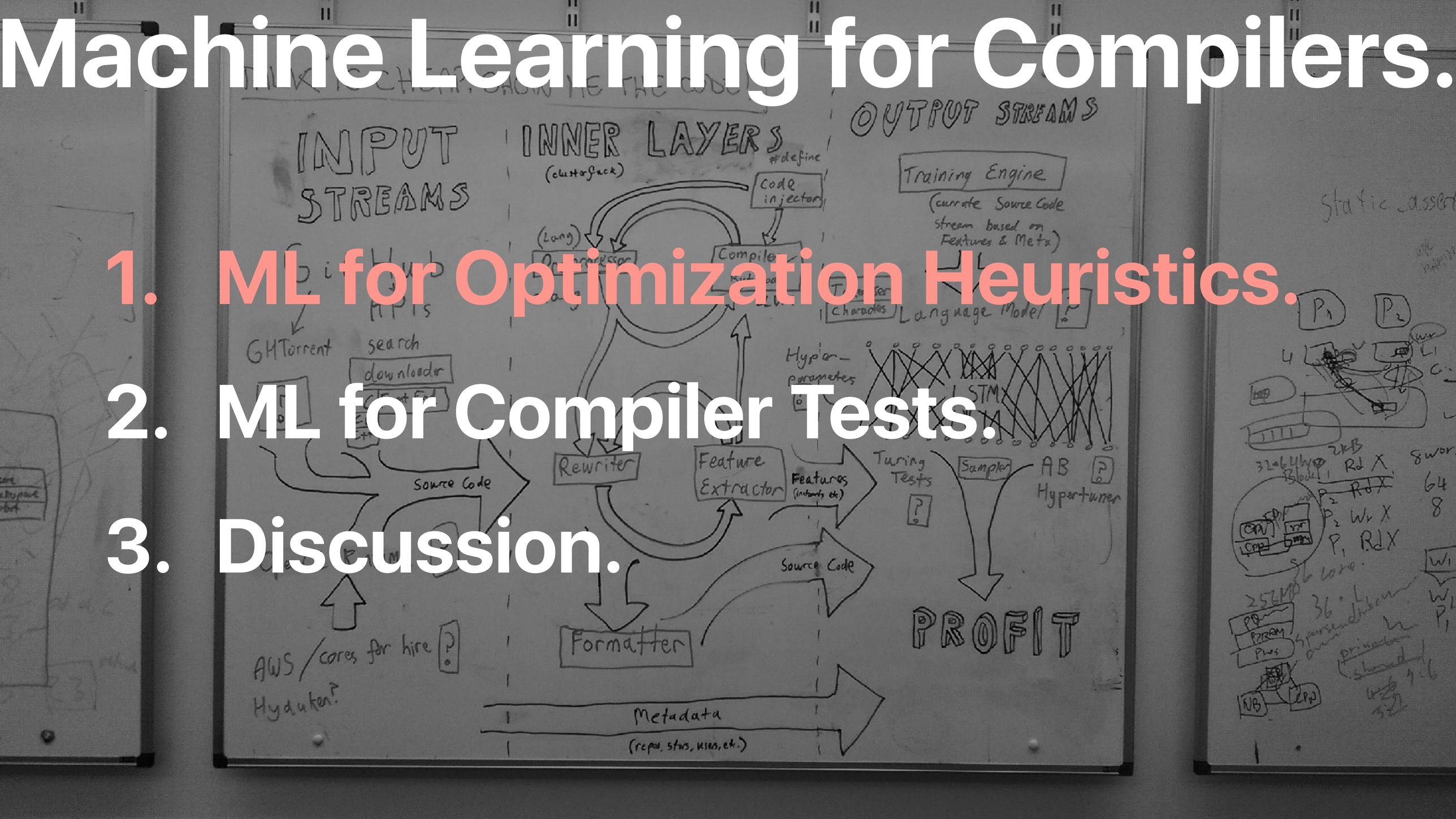
Chris Cummins

Machine Learning for Compilers.



1. ML for Optimization Heuristics.
2. ML for Compiler Tests.
3. Discussion.

Machine Learning for Compilers.



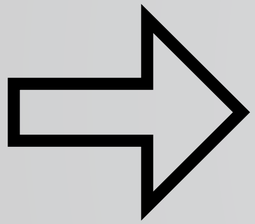
1. ML for Optimization Heuristics.
2. ML for Compiler Tests.
3. Discussion.

Problem:
Optimization heuristics are too coarse.

(aka. it doesn’t take much to do better than -O3)

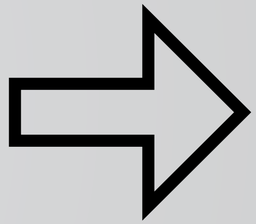
compilers are very complex

```
int main(
 int argc,
 char** arg)
 {...
```



hundreds,
thousands,
millions

of choices



```
_main:
   .cfi_start
proc
## BB#0:
   pushq %rbp
...
```

hand-coded heuristics
(out of date by time of release)

Hand-coded heuristics:

Analytical models difficult

Huge number of variables

Architectures change

Other bits of the compiler change

Execution is non-deterministic (OoO exec, caches)

Simple components are complex together

Better than -O3

```
#!/bin/sh

while true; do
    sort --random-sort < "cflags.txt" | head -n 20 | xargs gcc -O1 app.c
    time ./a.out
done
```

after 200 attempts, ~5% speedup

Iterative Compilation

Very simple idea

Don't get best heuristic analytically

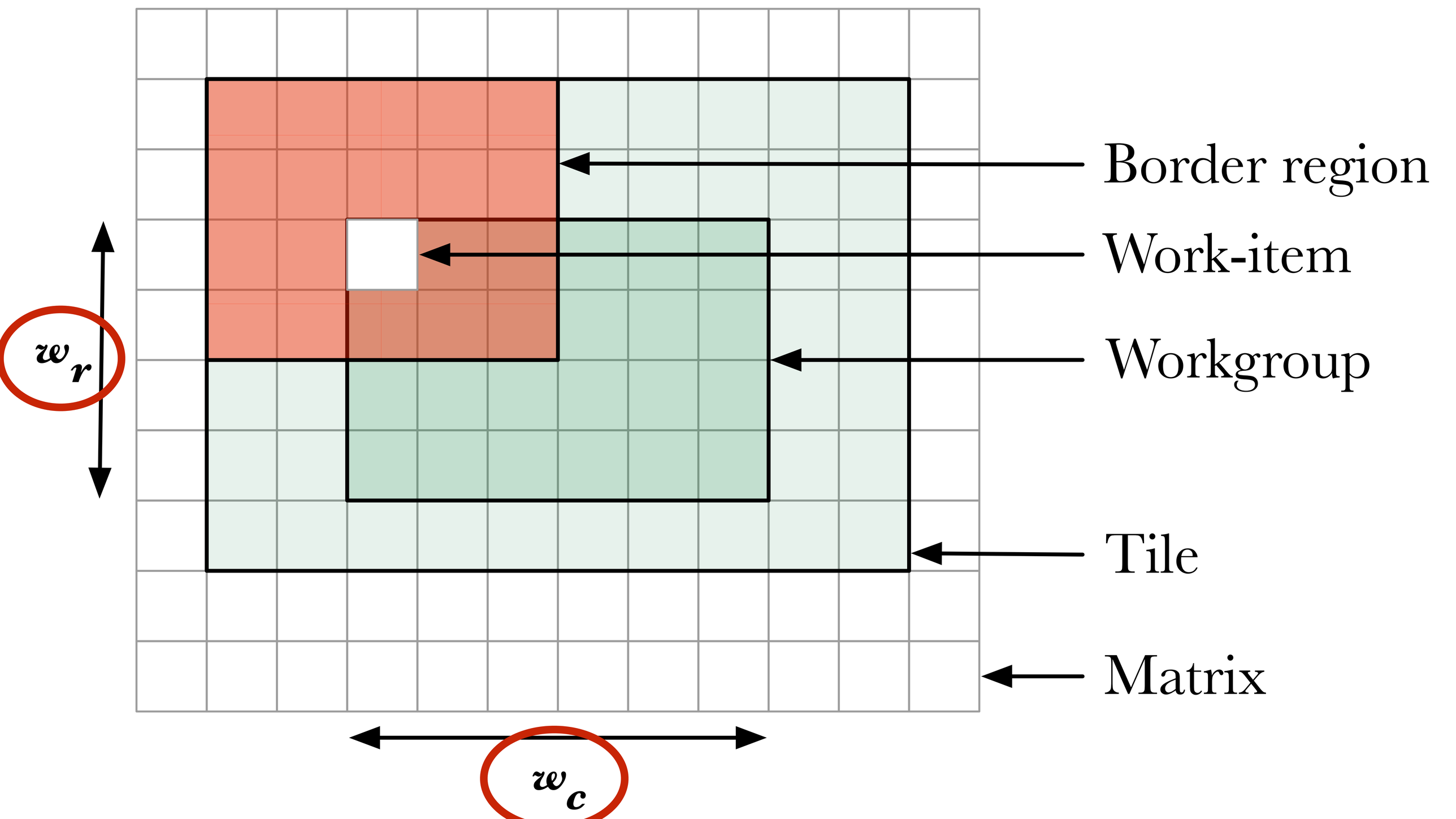
Try many versions of heuristic and choose best

Platform blind

Adapts to every change

Always based on evidence not belief

Example: Stencil Workgroup size

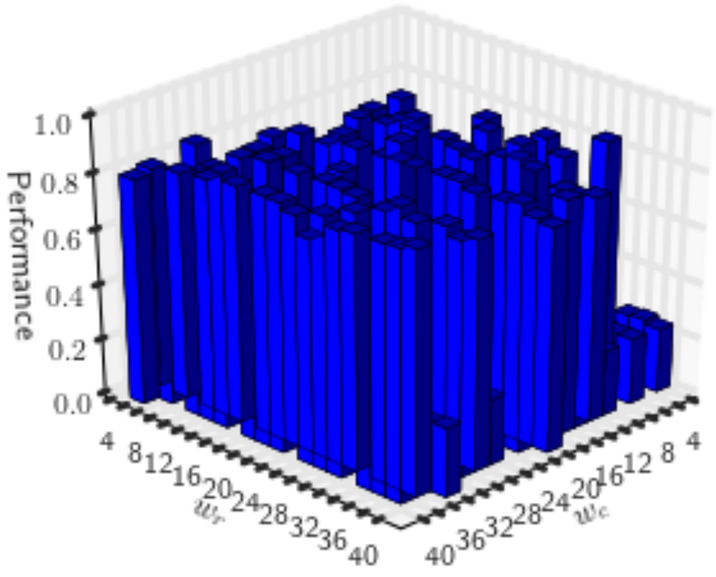

Border region
Work-item
Workgroup
Tile
Matrix
w_r
w_c

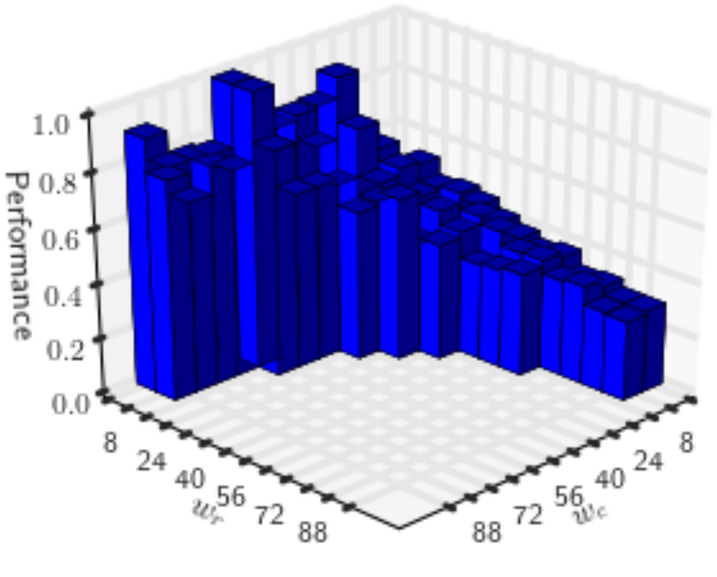
Workgroup size affects

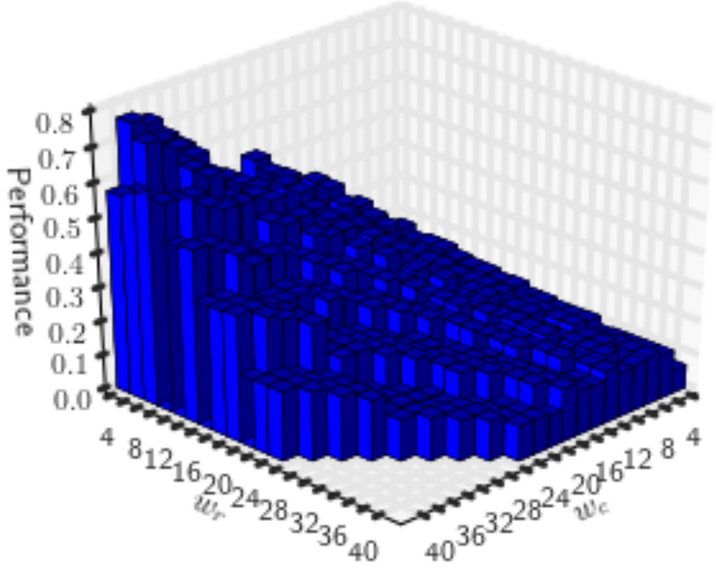
mapping to SIMD hardware.
device occupancy.
local memory utilisation.

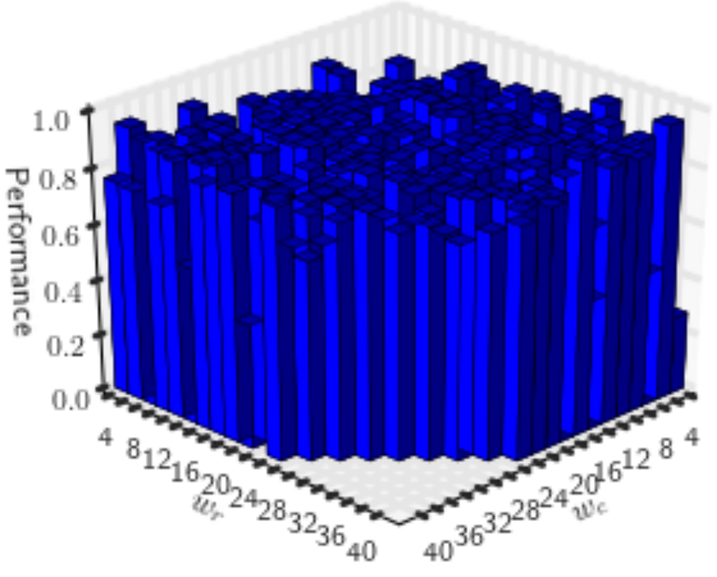
Choosing workgroup size depends on

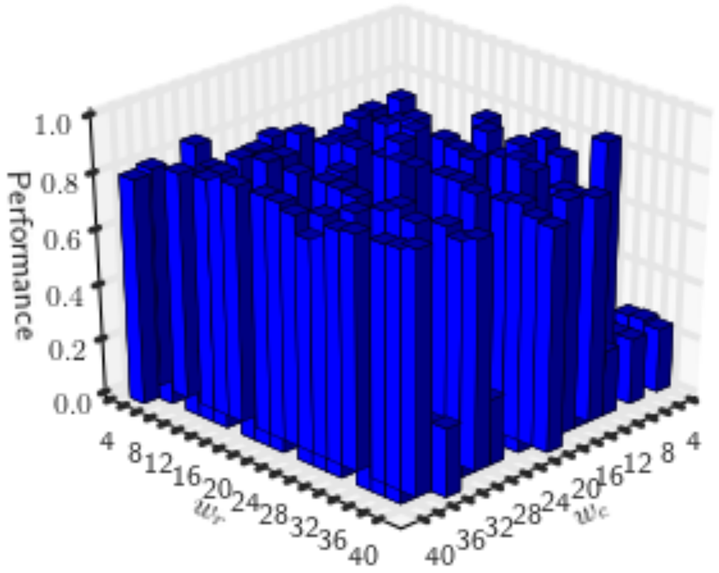
1. Device
2. Program
3. Dataset

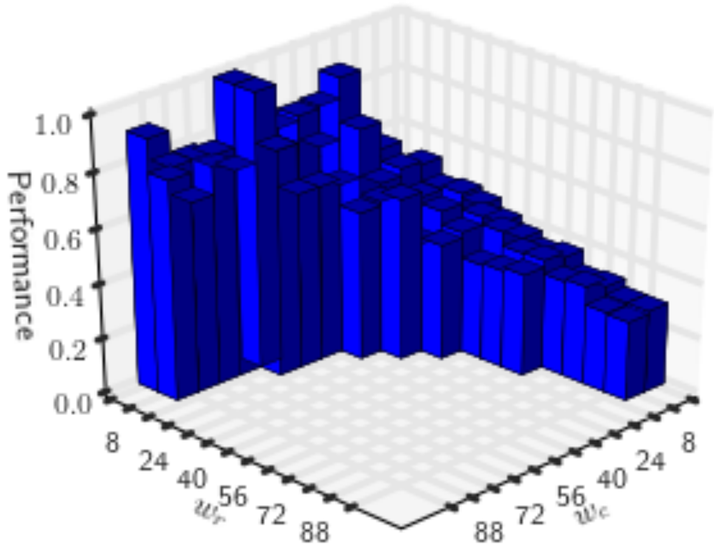

Performance
1.0
0.8
0.6
0.4
0.2
0.0
4 8 12 16 20 24 28 32 36 40
w_r
40 36 32 28 24 20 16 12 8 4
w_c


Performance
1.0
0.8
0.6
0.4
0.2
0.0
8 24 40 56 72 88
w_r
88 72 56 40 24 8
w_c

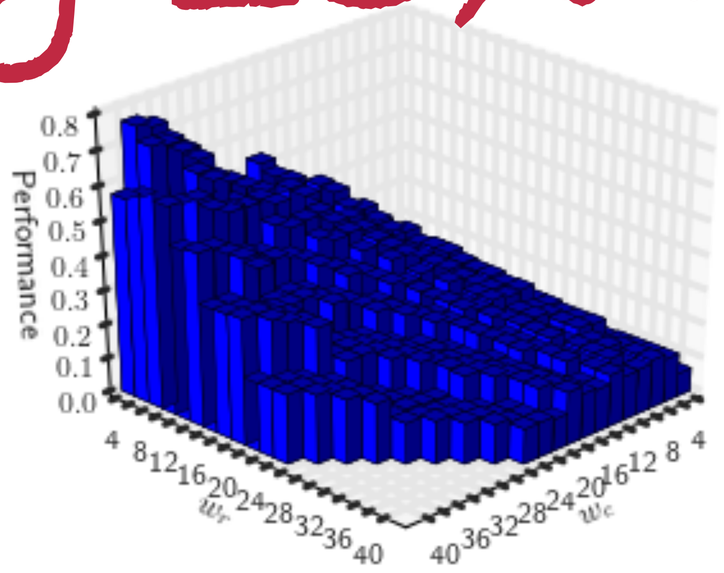

Performance
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
4 8 12 16 20 24 28 32 36 40
w_r
40 36 32 28 24 20 16 12 8 4
w_c

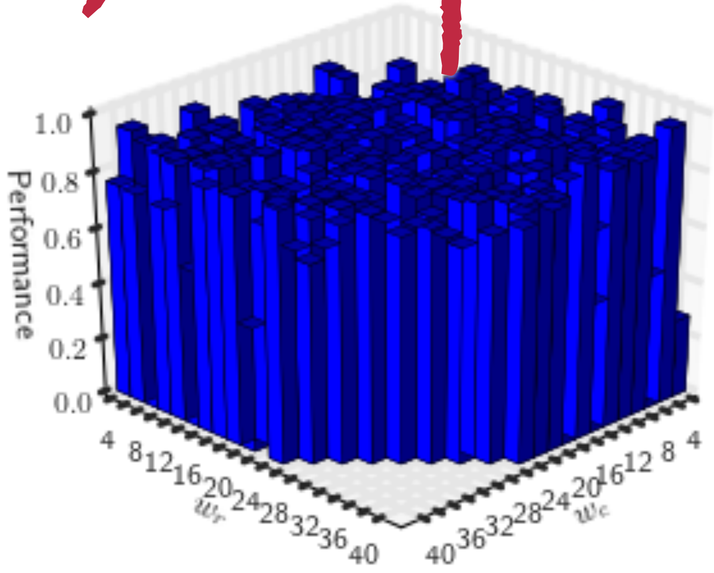

Performance
1.0
0.8
0.6
0.4
0.2
0.0
4 8 12 16 20 24 28 32 36 40
w_r
40 36 32 28 24 20 16 12 8 4
w_c

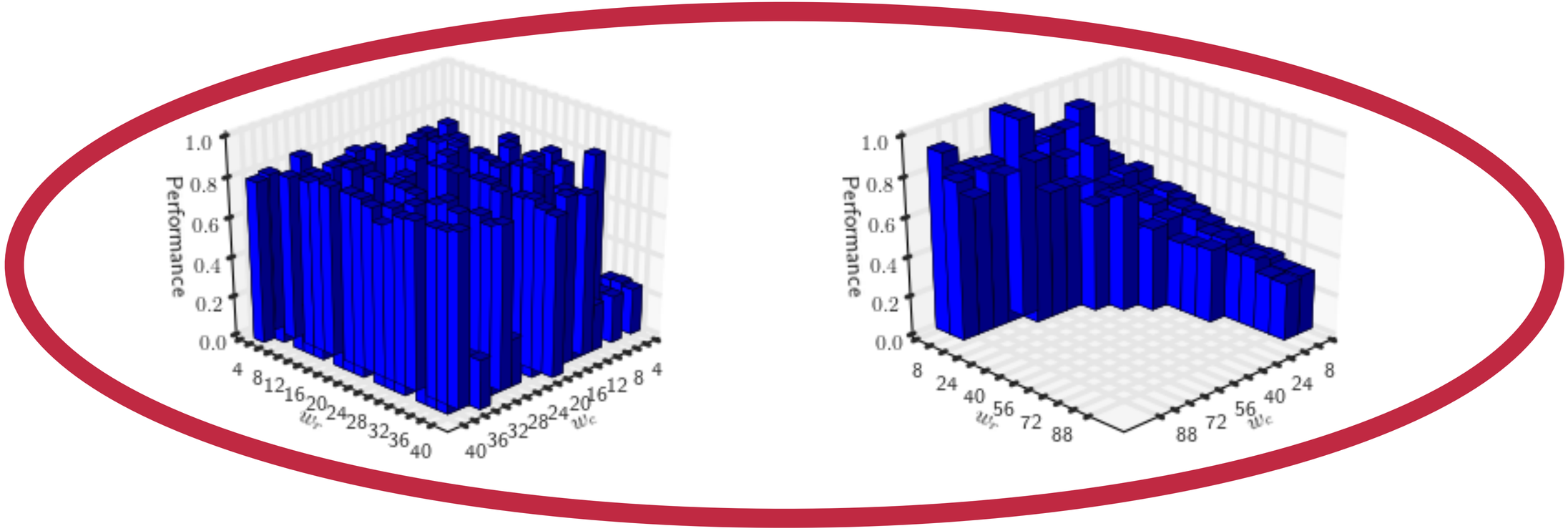




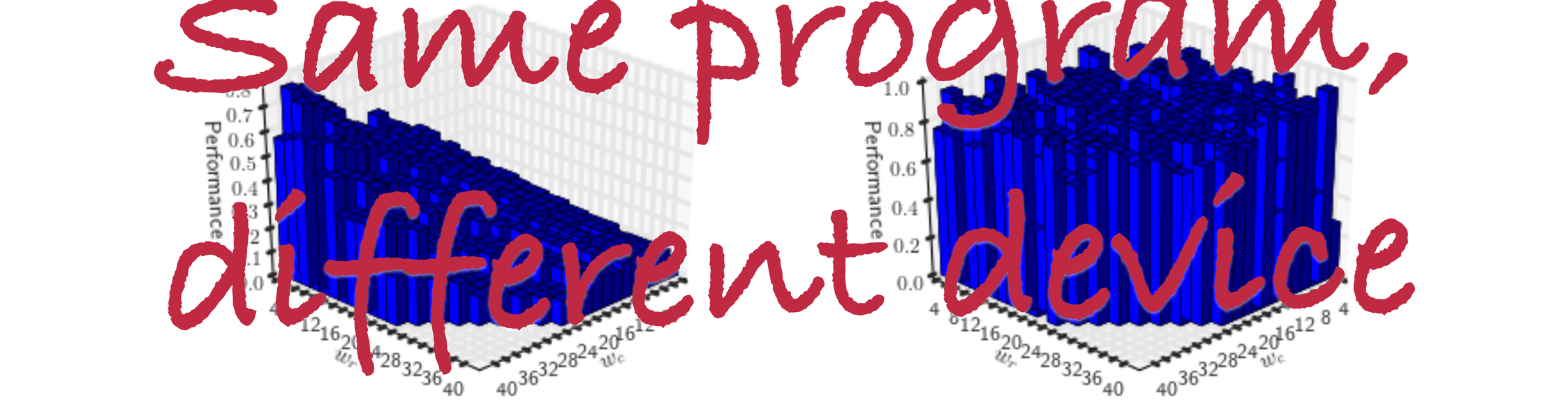
Avg 15x max speedup



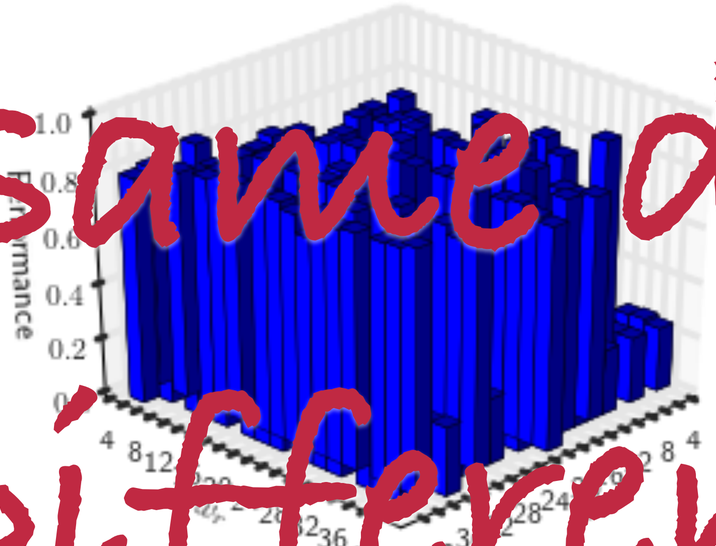


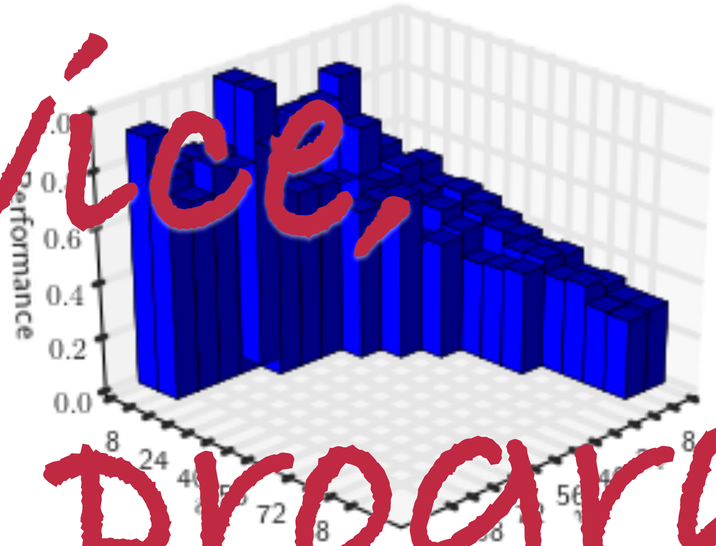


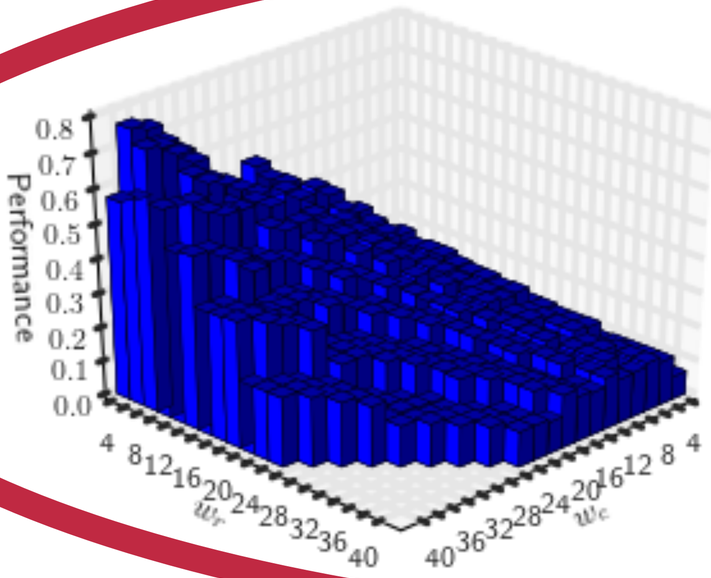
same program,
different device

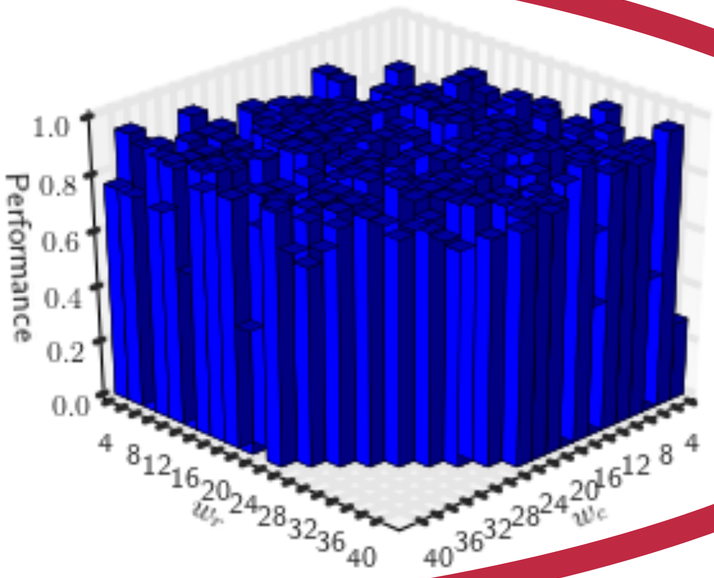


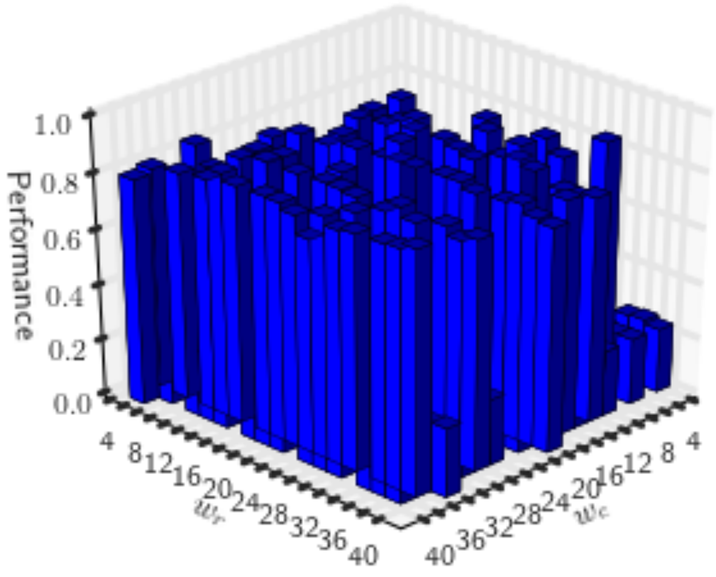
Same device, Different program

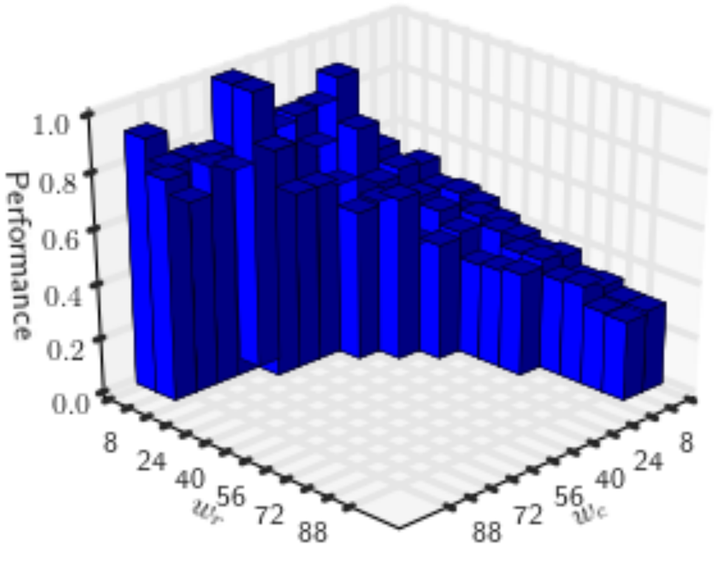


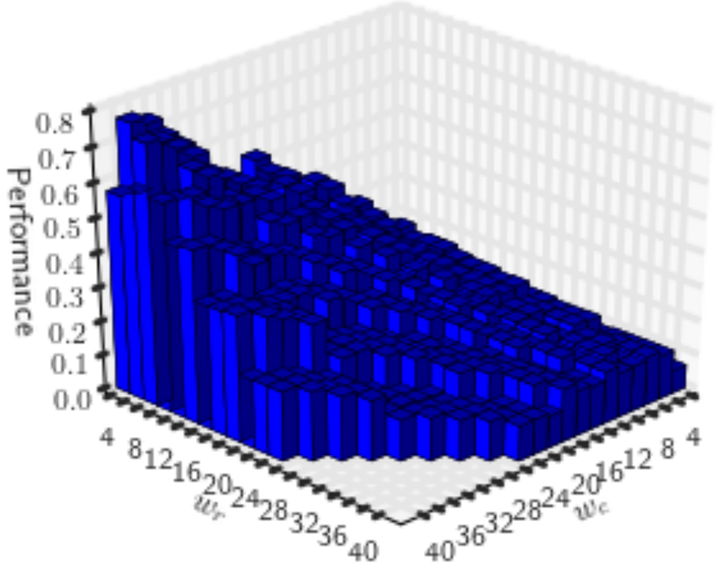


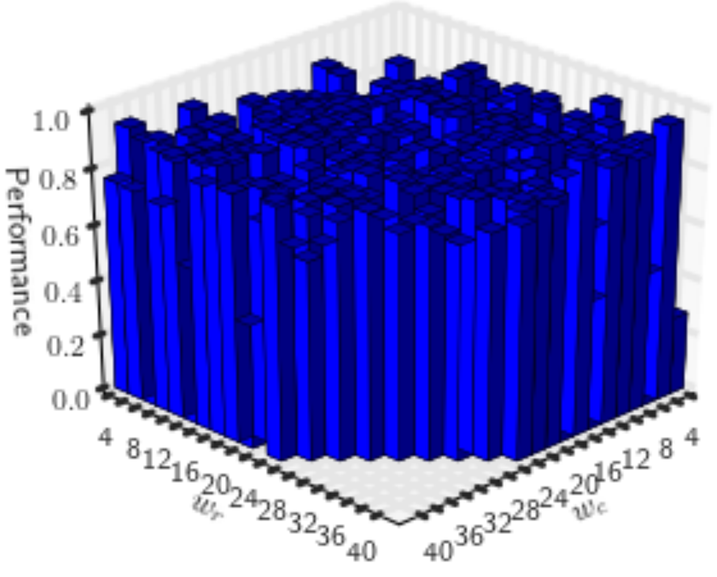





Performance
1.0
0.8
0.6
0.4
0.2
0.0
4 8 12 16 20 24 28 32 36 40
w_r
40 36 32 28 24 20 16 12 8 4
w_c

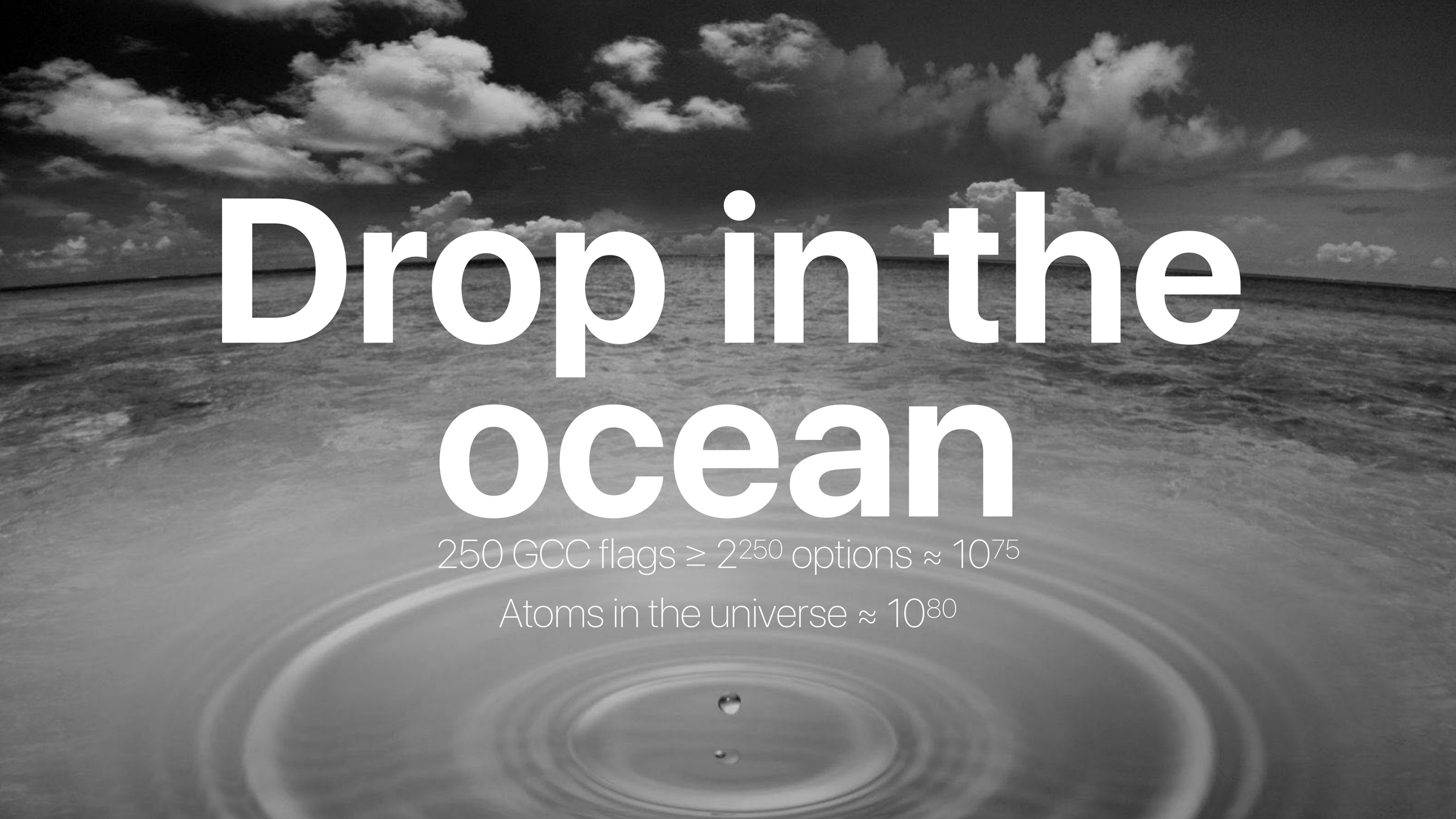

Performance
1.0
0.8
0.6
0.4
0.2
0.0
8 24 40 56 72 88
w_r
88 72 56 40 24 8
w_c


Performance
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
4 8 12 16 20 24 28 32 36 40
w_r
40 36 32 28 24 20 16 12 8 4
w_c


Performance
1.0
0.8
0.6
0.4
0.2
0.0
4 8 12 16 20 24 28 32 36 40
w_r
40 36 32 28 24 20 16 12 8 4
w_c

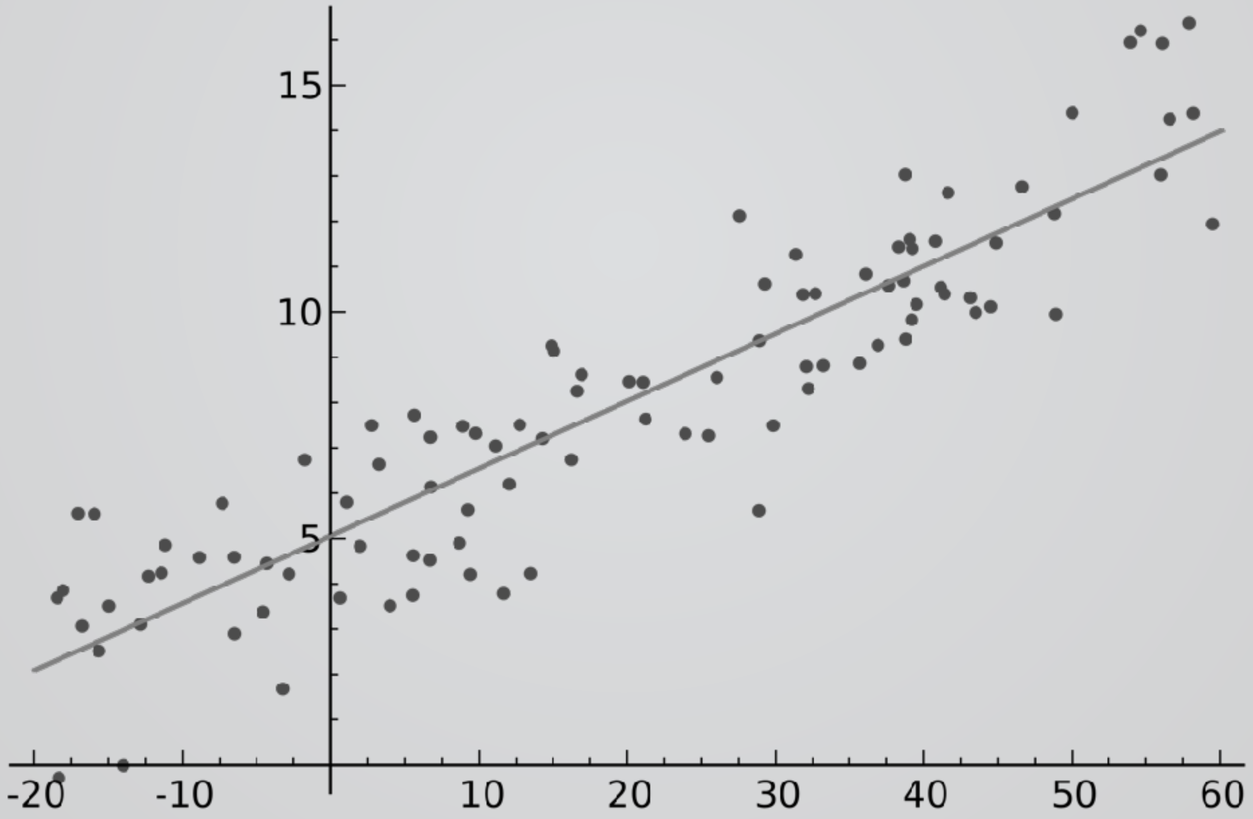
324 attempts sounds like a lot...

(324 x 10 x 30 s ≈ 27 hrs)


Drop in the ocean
250 GCC flags ≥ 2^{250} options ≈ 10^{75}
Atoms in the universe ≈ 10^{80}

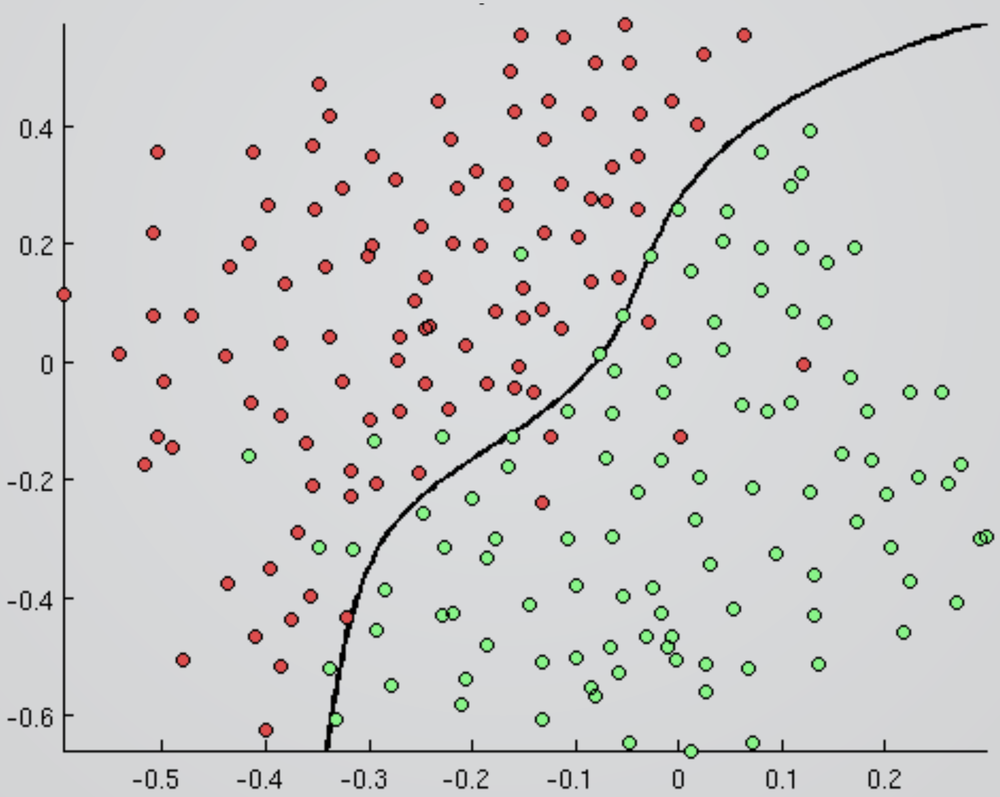
Machine Learning

Estimate y = f(x)



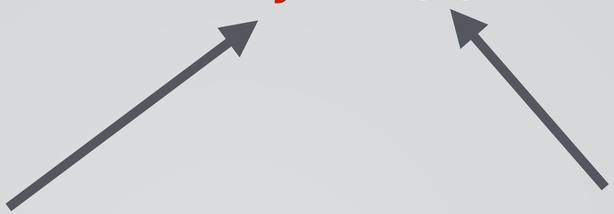
Machine Learning

Estimate y = f(x)



Machine Learning

Estimate y = f(x)



Optimisations

Cflags

Workgroup size

CPU or GPU

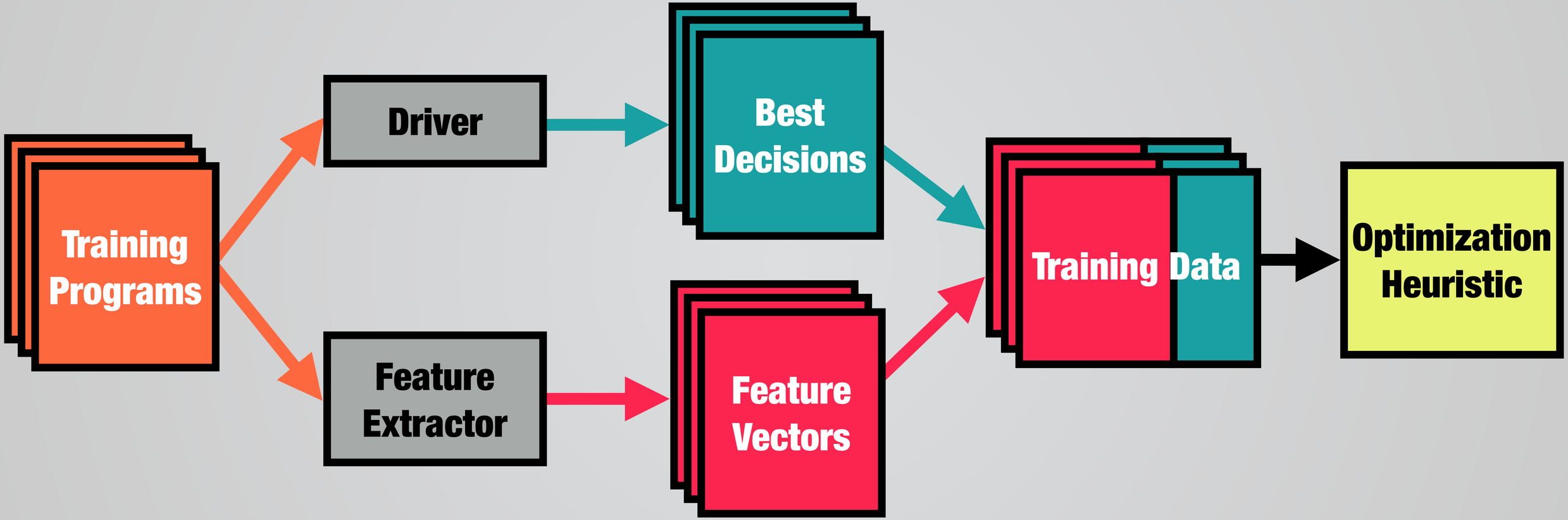
Features

instructions

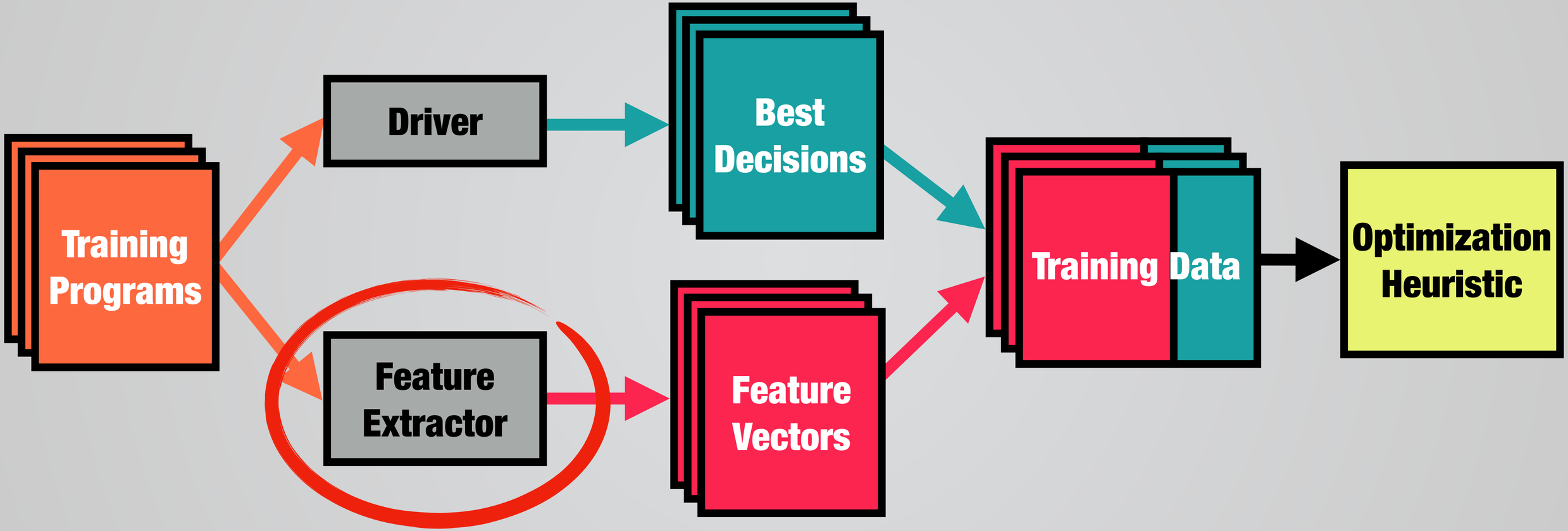
Arithmetic density

Dataset size

Learning an Optimization Heuristic



Learning an Optimization Heuristic

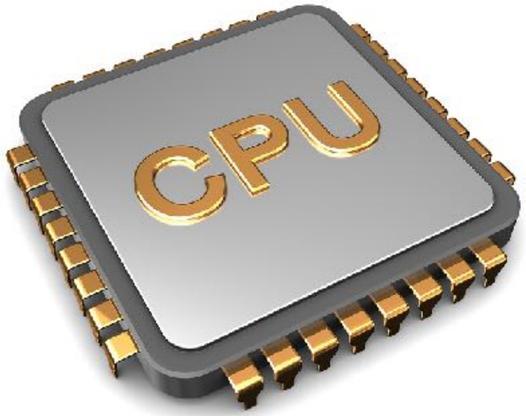


Features for Stencil Workgroup size

1. Device
2. Program
3. Dataset

Features for Stencil Workgroup size

1. Device
2. Program
3. Dataset



or

How many compute units?

How much memory?

Cache size?

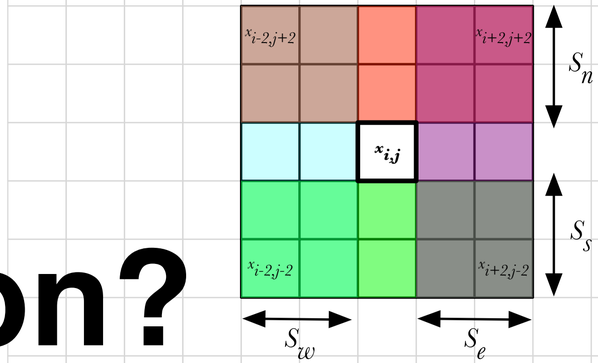
etc.

Features for Stencil Workgroup size

1. Device
2. Program
3. Dataset

Features for Stencil Workgroup size

1. Device
2. Program
3. Dataset



How big is border region?

What shape is it?

How many instructions?

What type of instructions?

etc.

Features for Stencil Workgroup size

1. Device
2. Program
3. Dataset

Features for Stencil Workgroup size

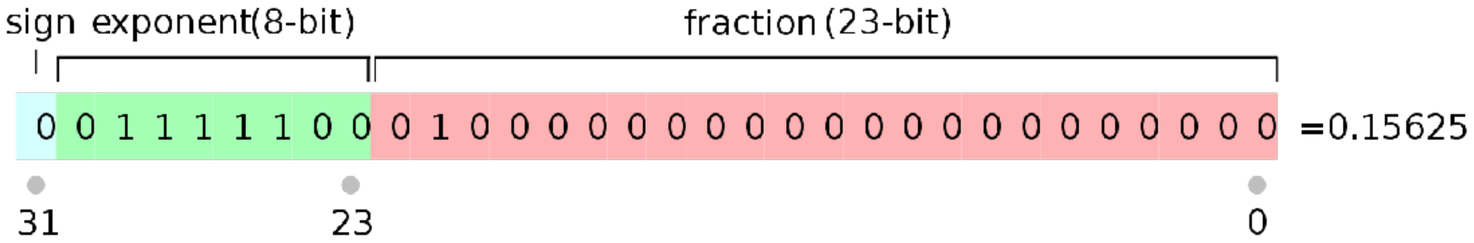
1. Device
2. Program
3. Dataset

$$\boldsymbol{A} = \begin{bmatrix} A_{11} & A_{12} & \cdots & A_{1n} \\ A_{21} & & & A_{2n} \\ \vdots & & & \vdots \\ A_{n1} & A_{n2} & \cdots & A_{nn} \end{bmatrix}$$

How big is the data?

What type is the input?

What type is the output?



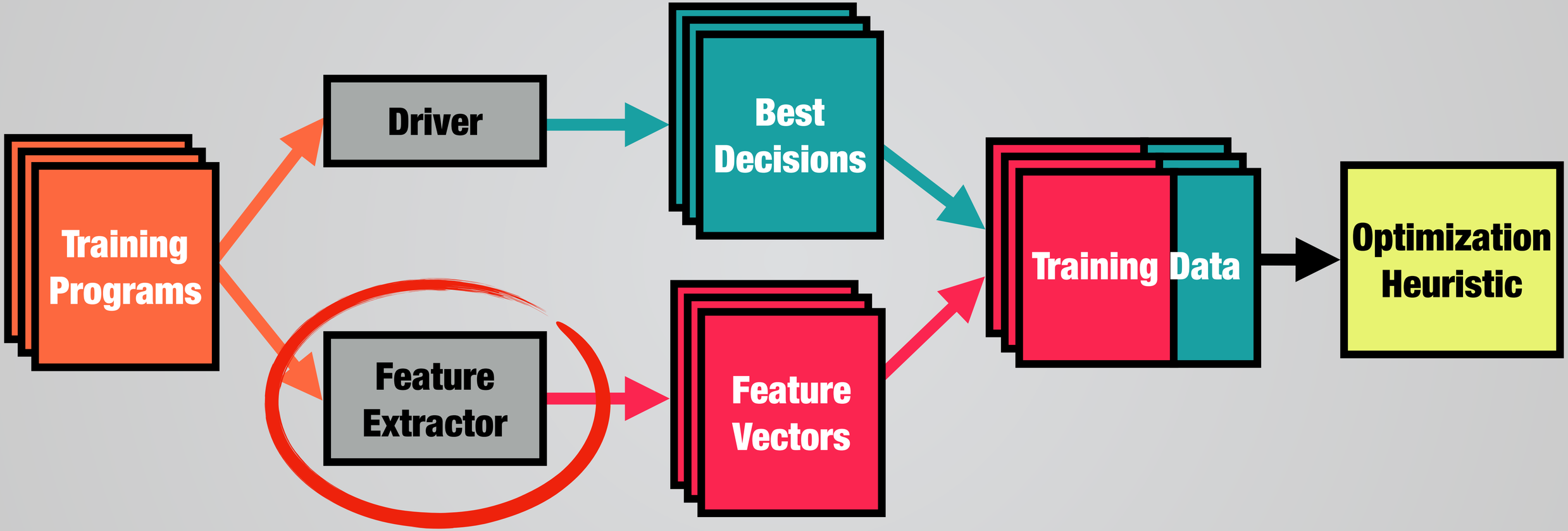
Features for Stencil Workgroup size

1. Device
2. Program
3. Dataset

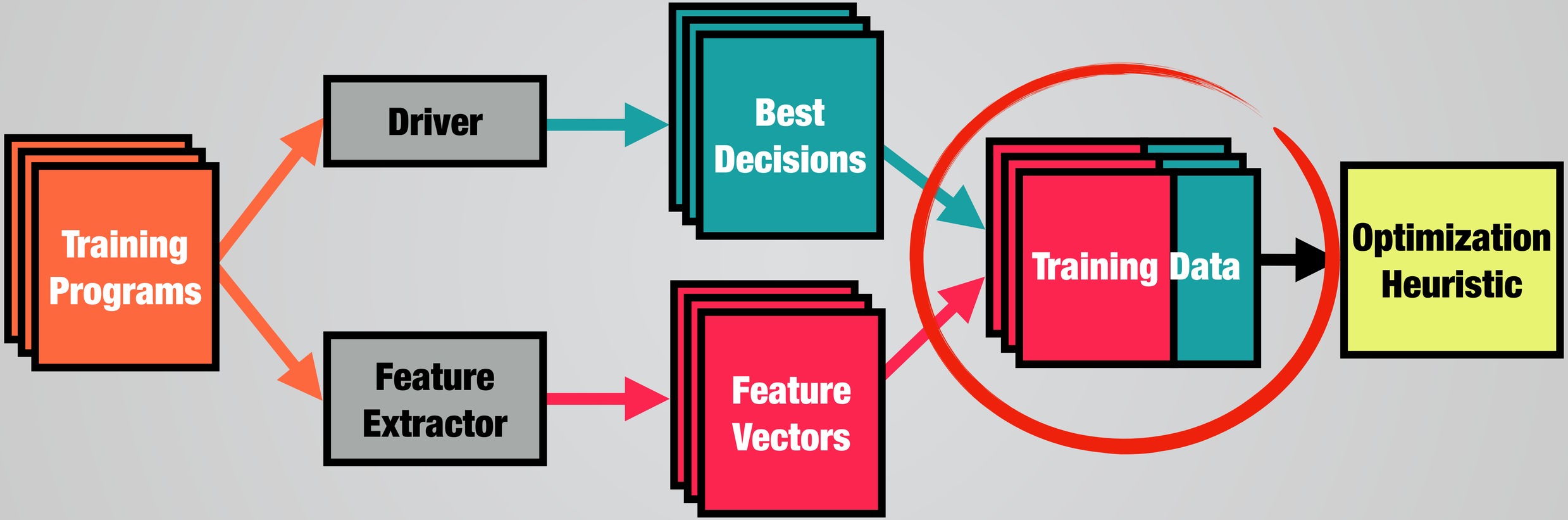
Features for Stencil Workgroup size

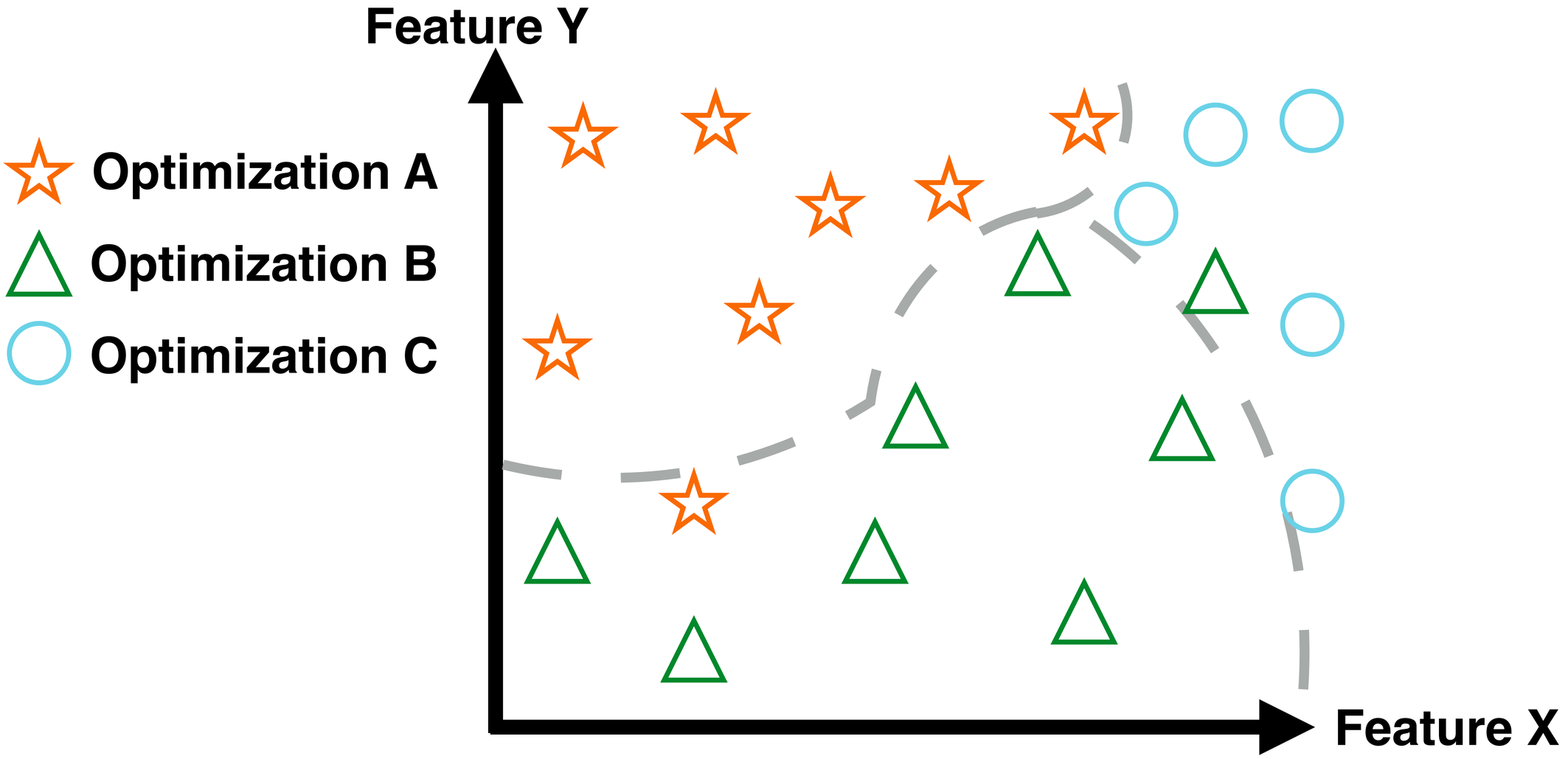
1. Device
2. Program
3. Dataset

Learning an Optimization Heuristic



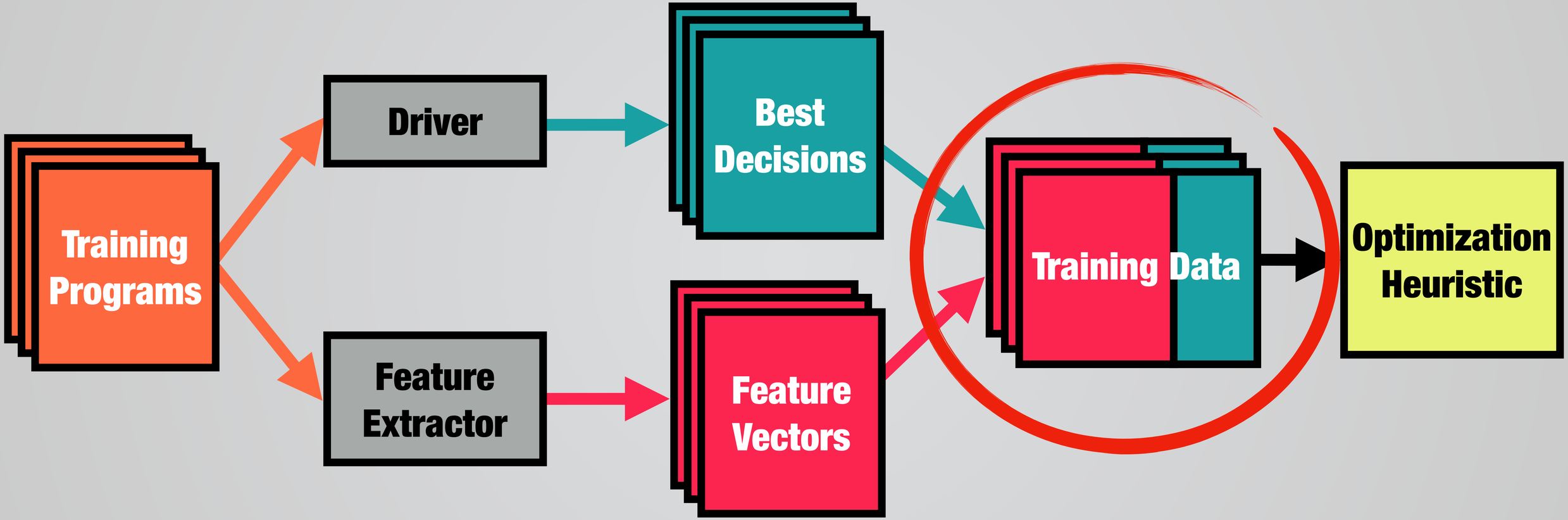
Learning an Optimization Heuristic



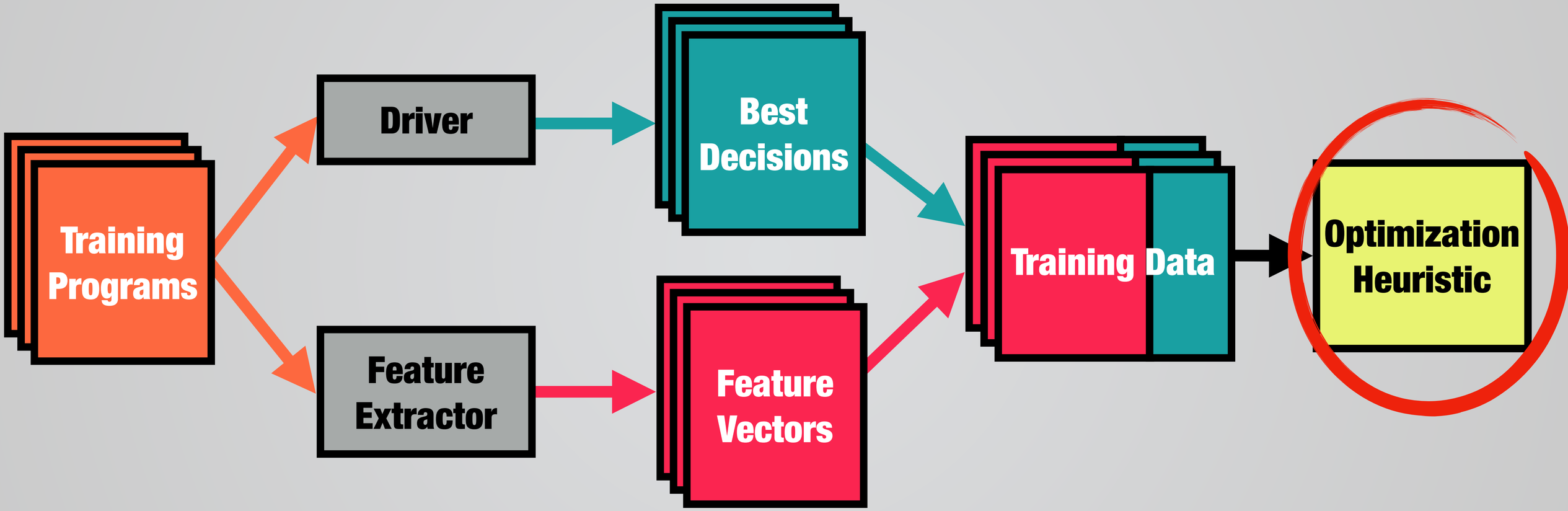


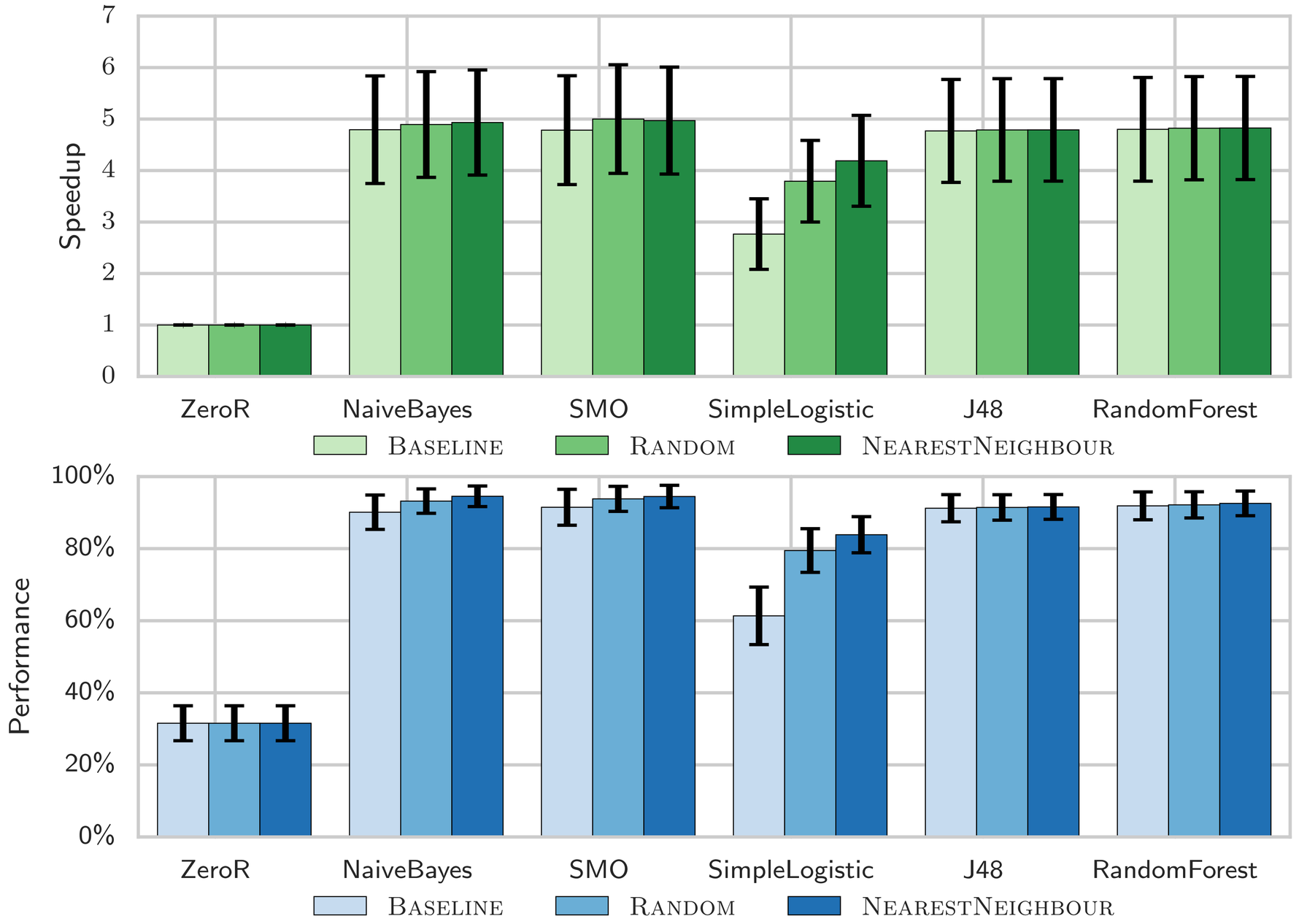
Learn a classification *(dotted line)*

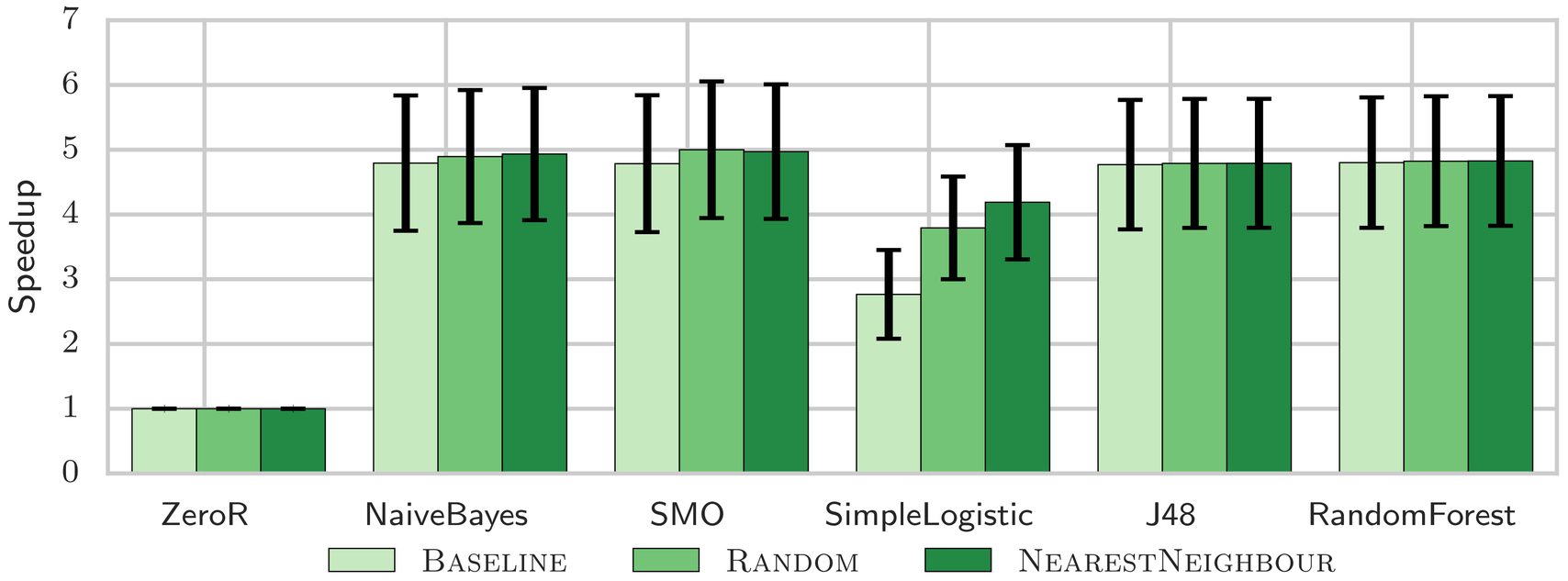
Learning an Optimization Heuristic

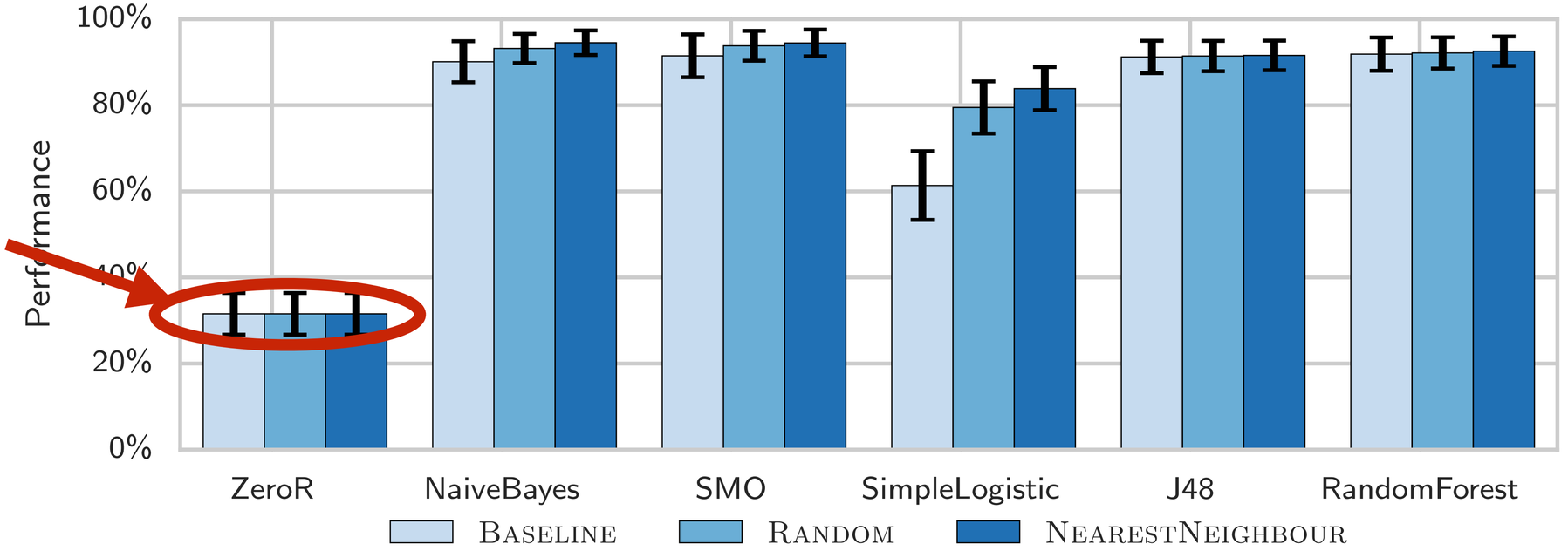


Learning an Optimization Heuristic

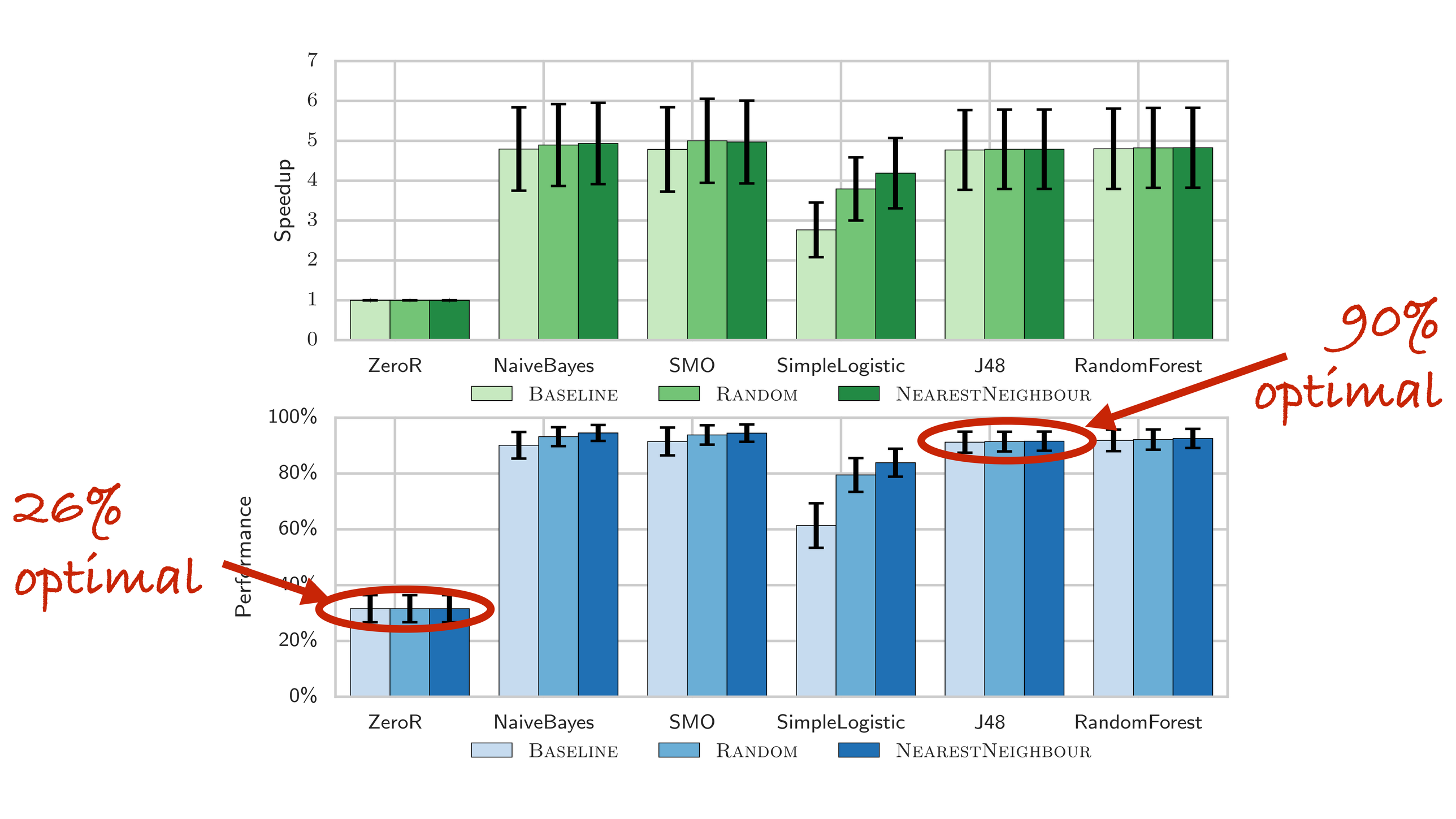



Speedup
7
6
5
4
3
2
1
0
ZeroR
NaiveBayes
SMO
SimpleLogistic
J48
RandomForest
Baseline
Random
NearestNeighbour
Performance
100%
80%
60%
40%
20%
0%
ZeroR
NaiveBayes
SMO
SimpleLogistic
J48
RandomForest
Baseline
Random
NearestNeighbour

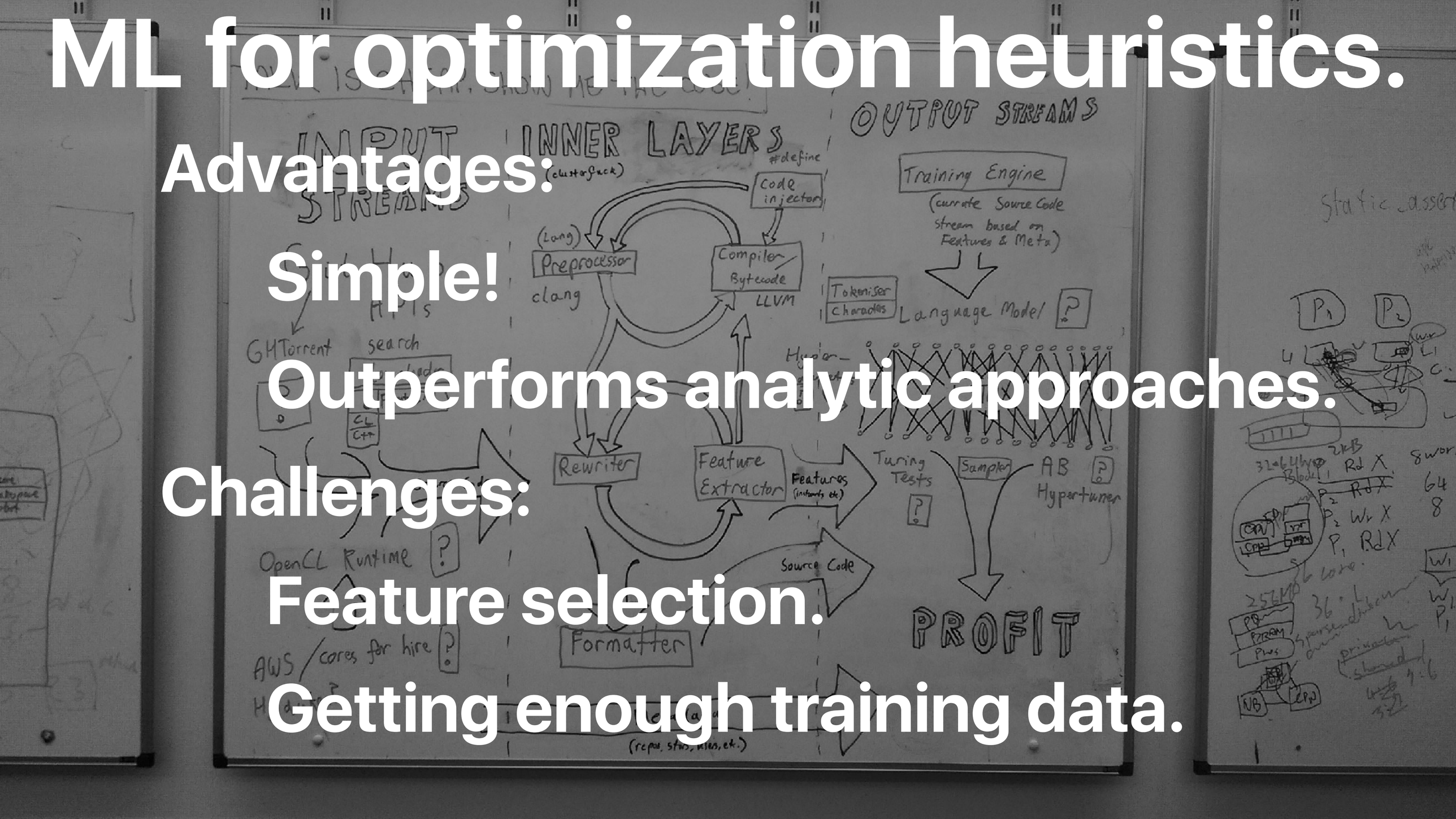

Speedup
7
6
5
4
3
2
1
0
ZeroR
NaiveBayes
SMO
SimpleLogistic
J48
RandomForest
Baseline
Random
NearestNeighbour


Performance
100%
80%
60%
20%
0%
ZeroR
NaiveBayes
SMO
SimpleLogistic
J48
RandomForest
Baseline
Random
NearestNeighbour

26%
optimal


Speedup
7
6
5
4
3
2
1
0
ZeroR
NaiveBayes
SMO
SimpleLogistic
J48
RandomForest
Baseline
Random
NearestNeighbour
Performance
100%
80%
60%
20%
0%
26% optimal
90% optimal

ML for optimization heuristics.



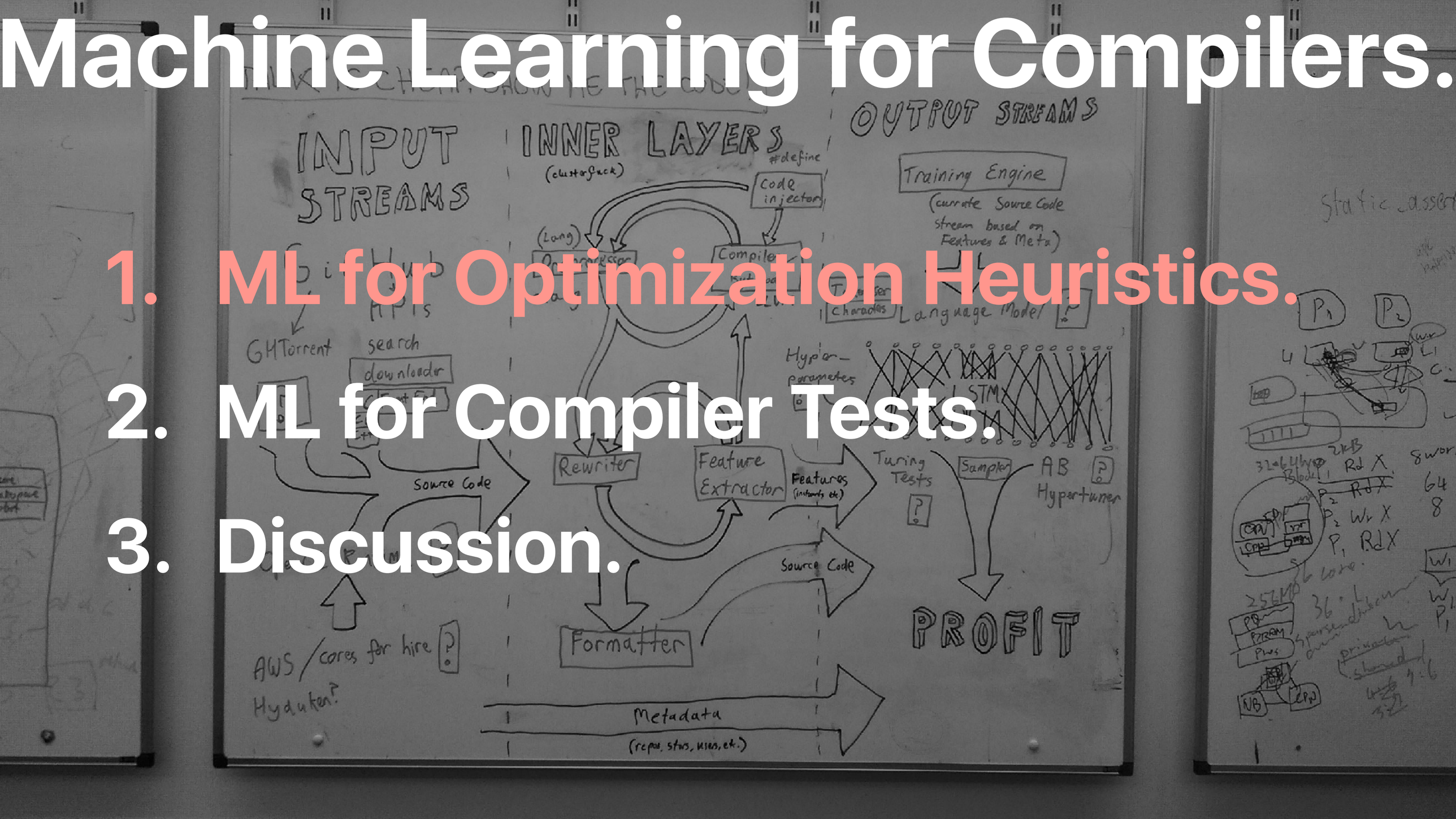
Advantages:

- Simple!
- Outperforms analytic approaches.

Challenges:

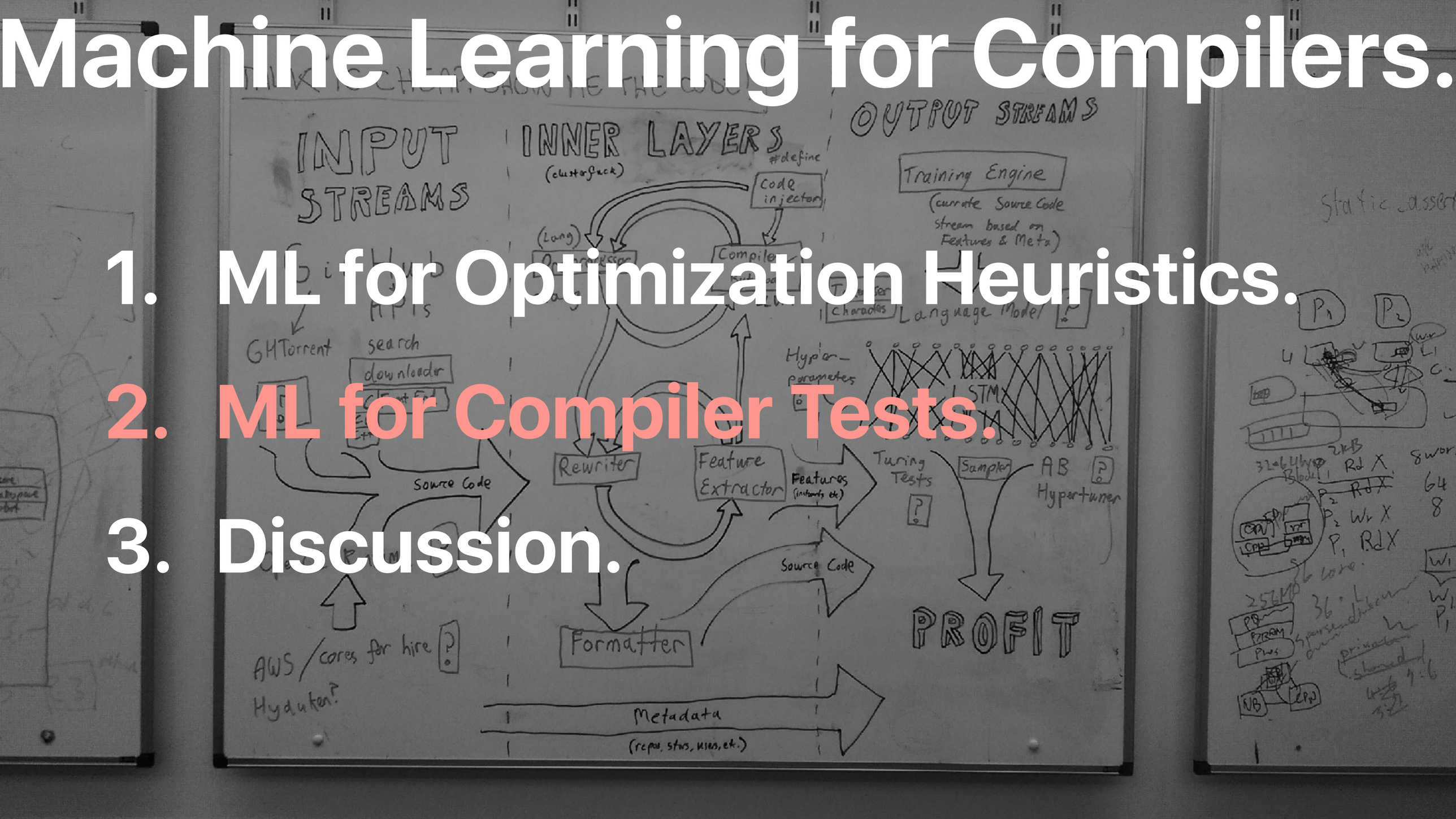
- Feature selection.
- Getting enough training data.

Machine Learning for Compilers.



1. ML for Optimization Heuristics.
2. ML for Compiler Tests.
3. Discussion.

Machine Learning for Compilers.



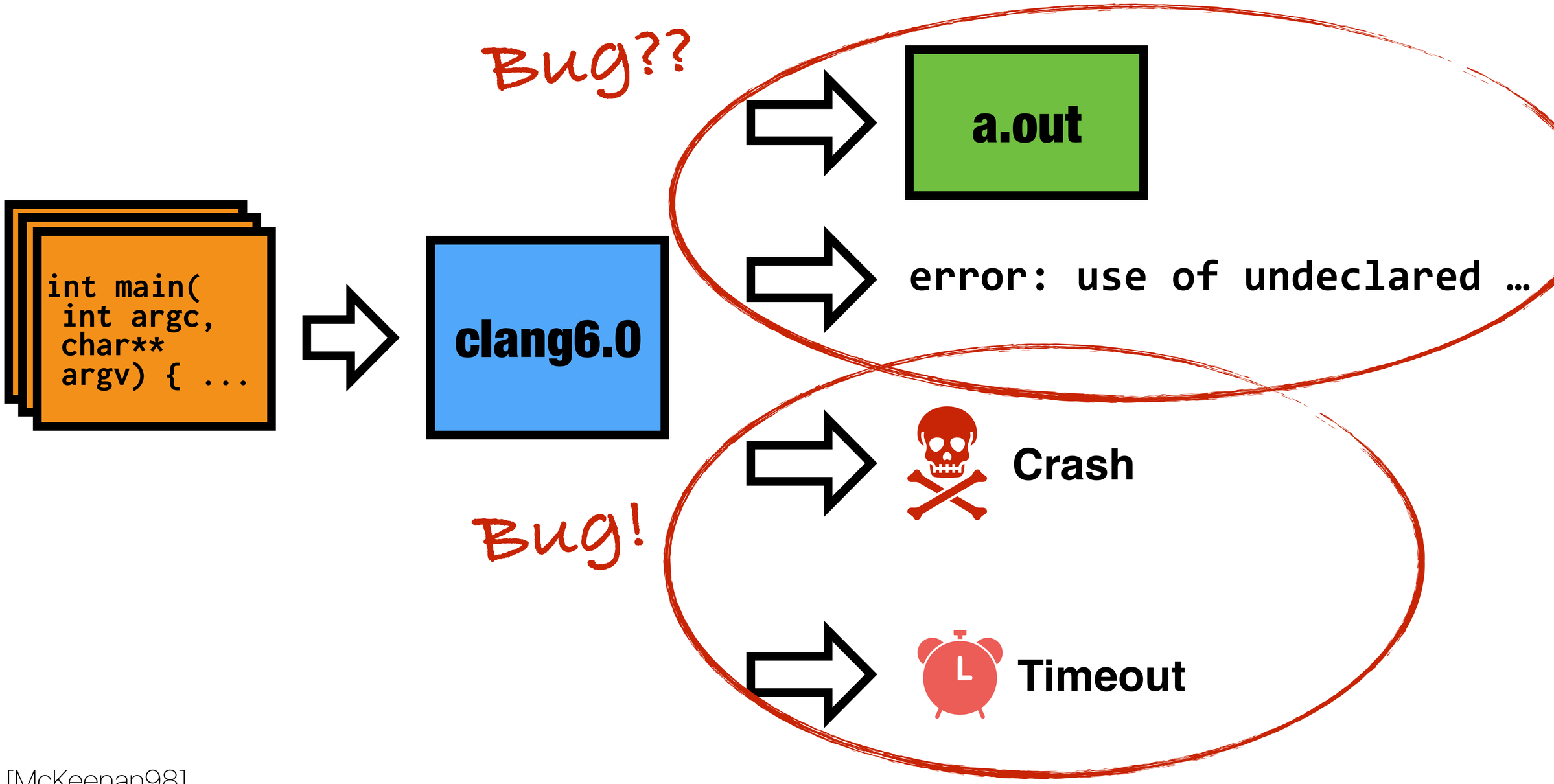
1. ML for Optimization Heuristics.
2. ML for Compiler Tests.
3. Discussion.

Problem:

Generating inputs for compiler fuzzing is hard.

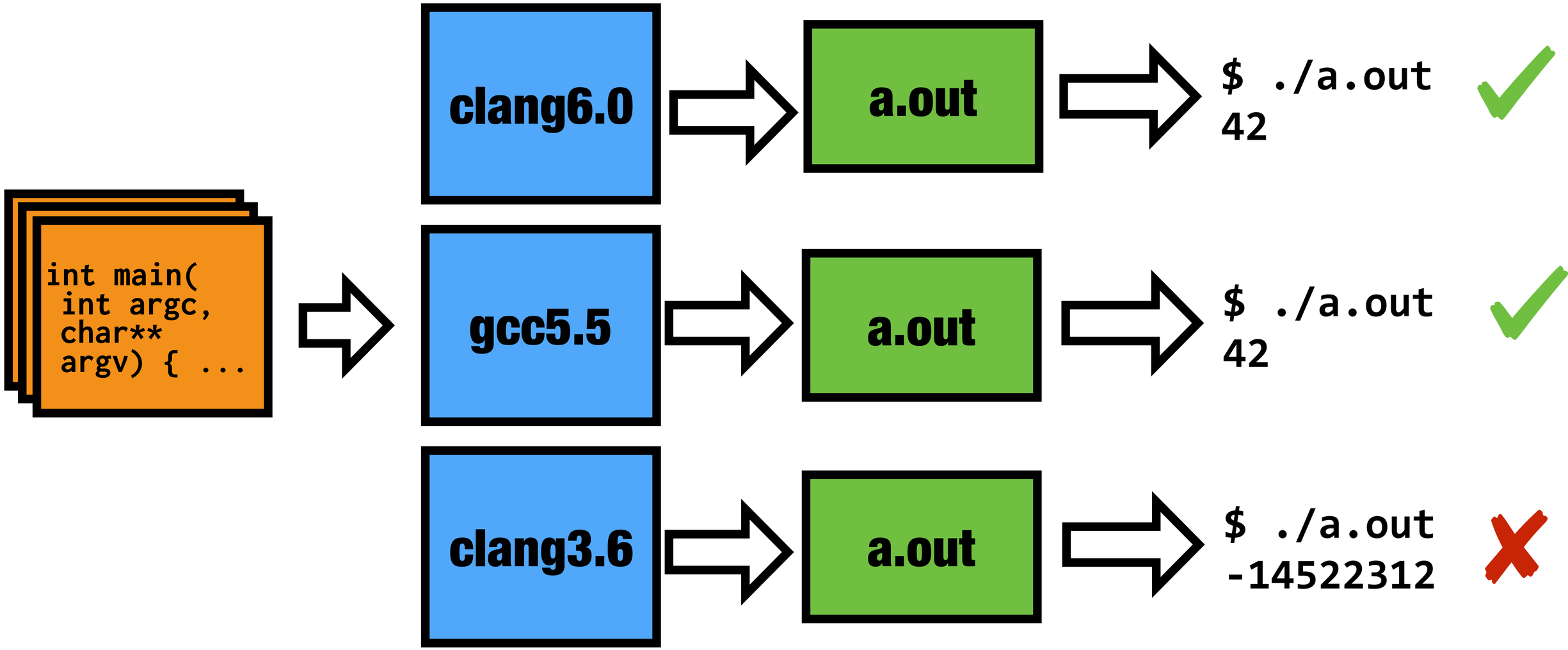
(requires complex grammar enumeration and analysis)

fuzzing a compiler



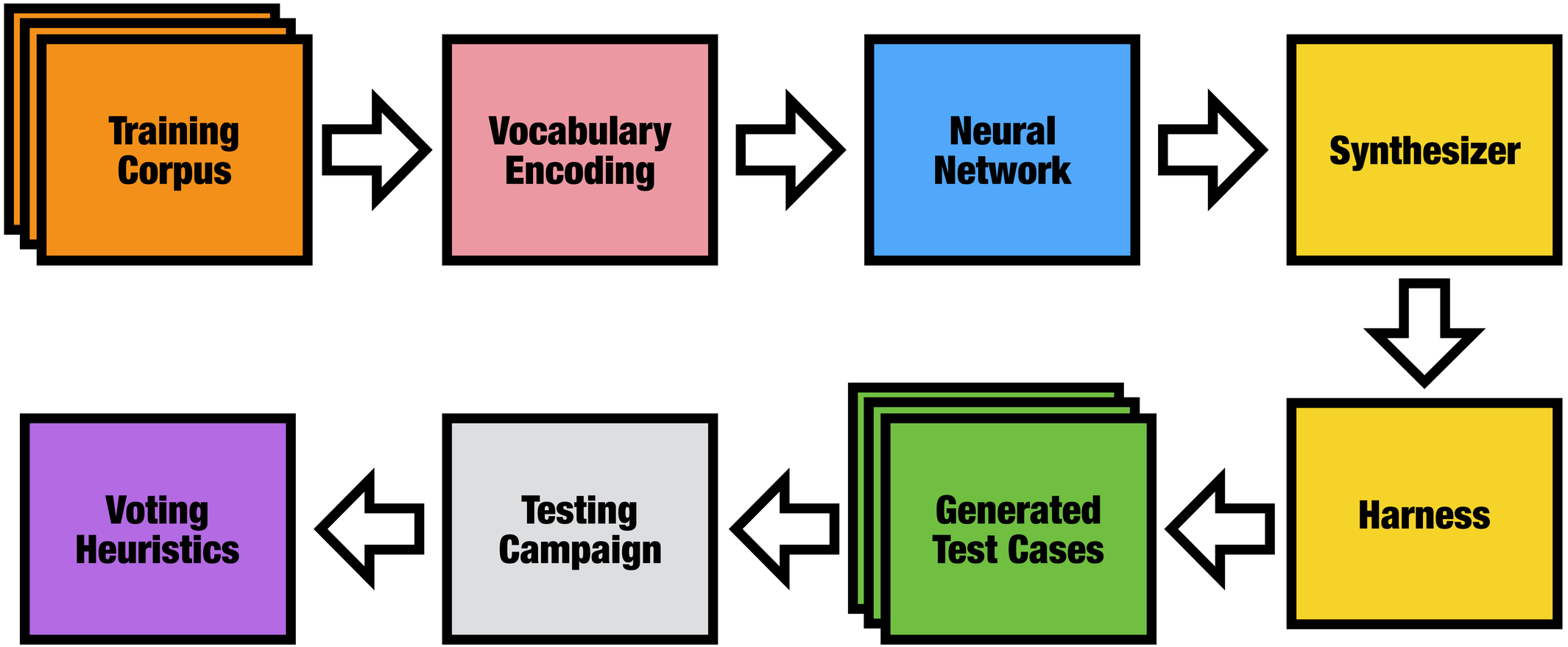
circa [McKeenan98]

differential testing compilers

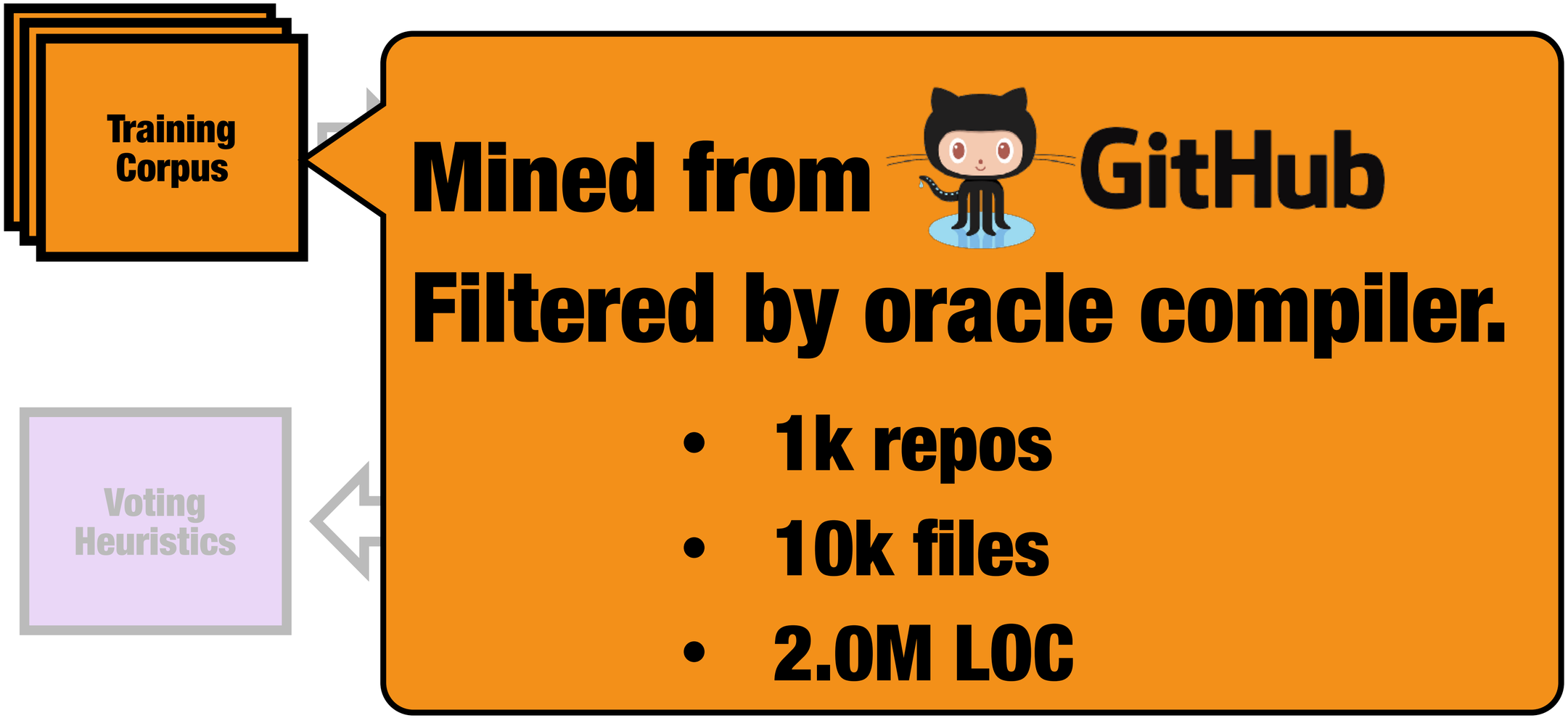


circa [McKeenan98]

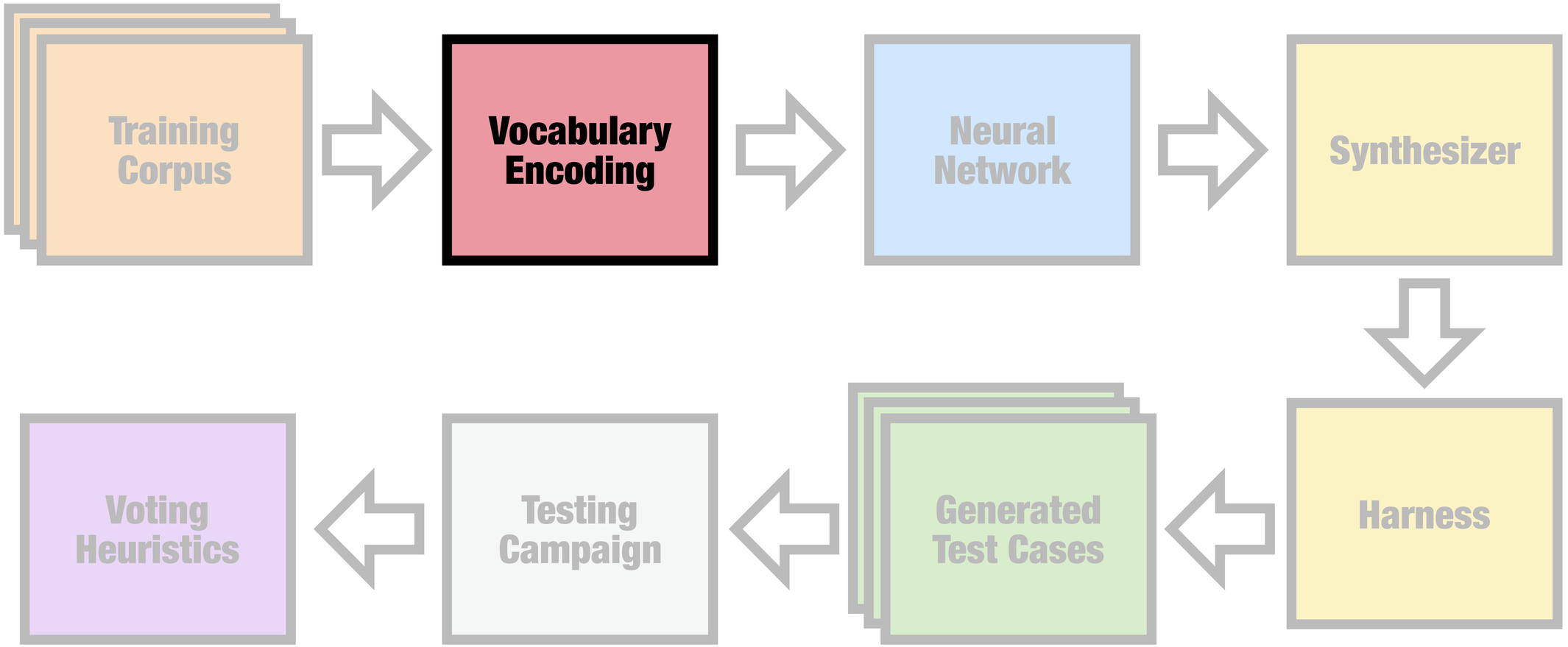
ML for compiler test generation



ML for compiler test generation



ML for compiler test generation



vocabulary encoding

```
kernel void A(global float* a, const float b) {
  a[get_global_id(0)] *= 3.14 + b;
}
```

Vocab:

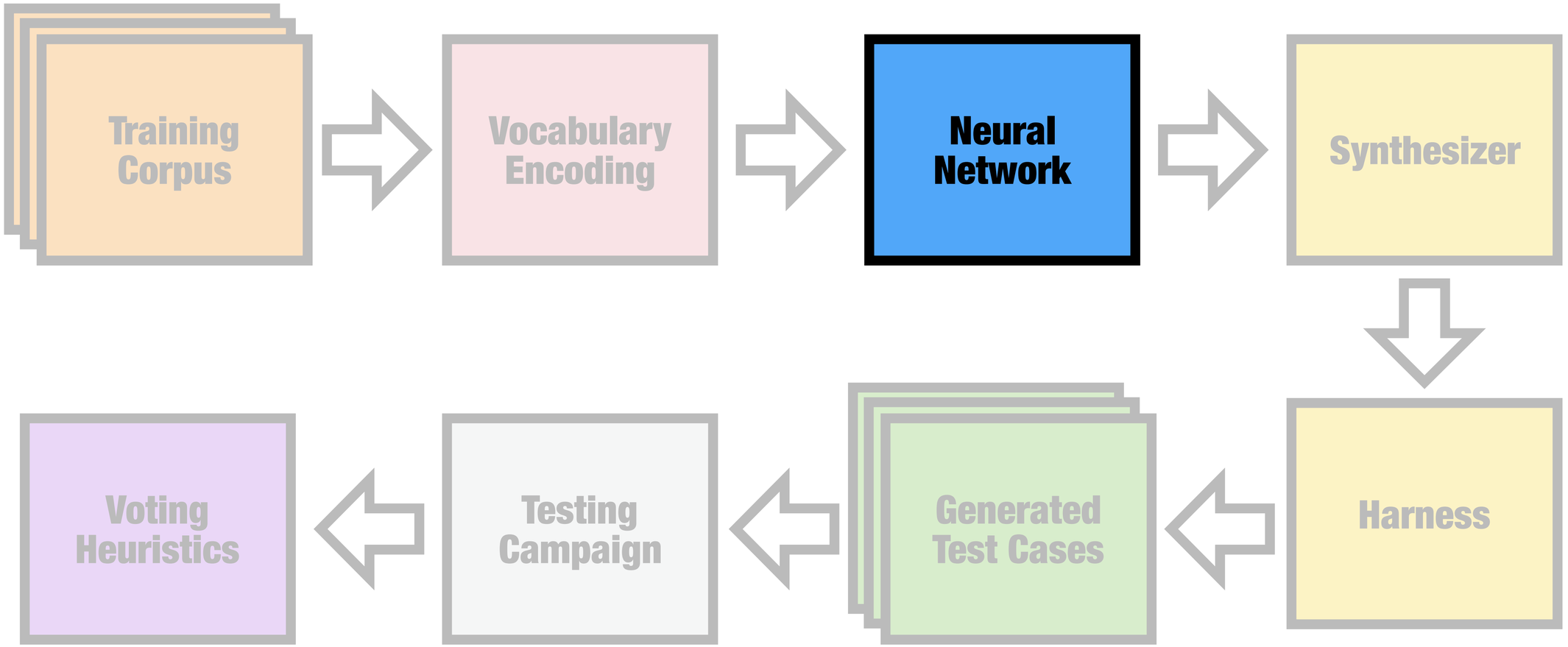
Token	Index
kernel	0
[space]	1
void	2
A	3
(	4
global	5
float	6
*	7
a	8

Token	Index
,	9
const	10
b	11
)	12
{	13
\n	14
[	15
get_global_id	16
0	17

Token	Index
]	18
=	19
3	20
.	21
1	22
4	23
+	24
;	25

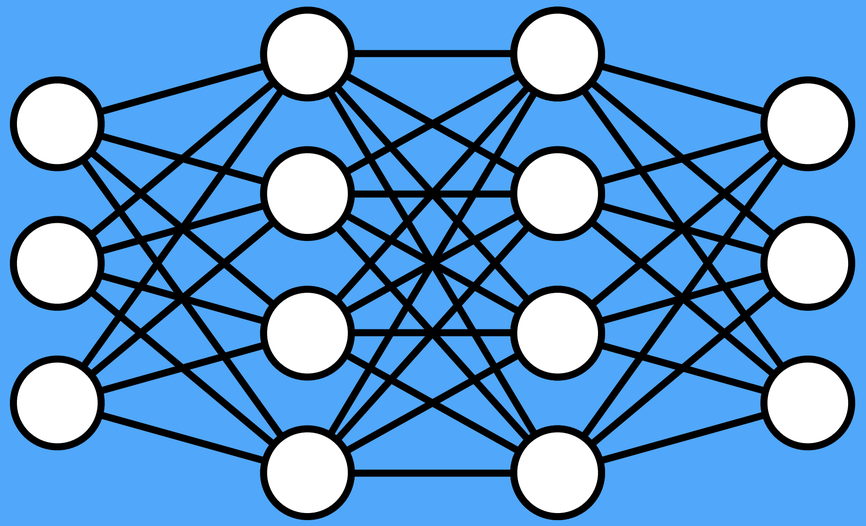
Encoded: 0 1 2 1 3 4 5 1 6 7 1 8 ...

ML for compiler test generation



neural network

181 tokens



181 tokens

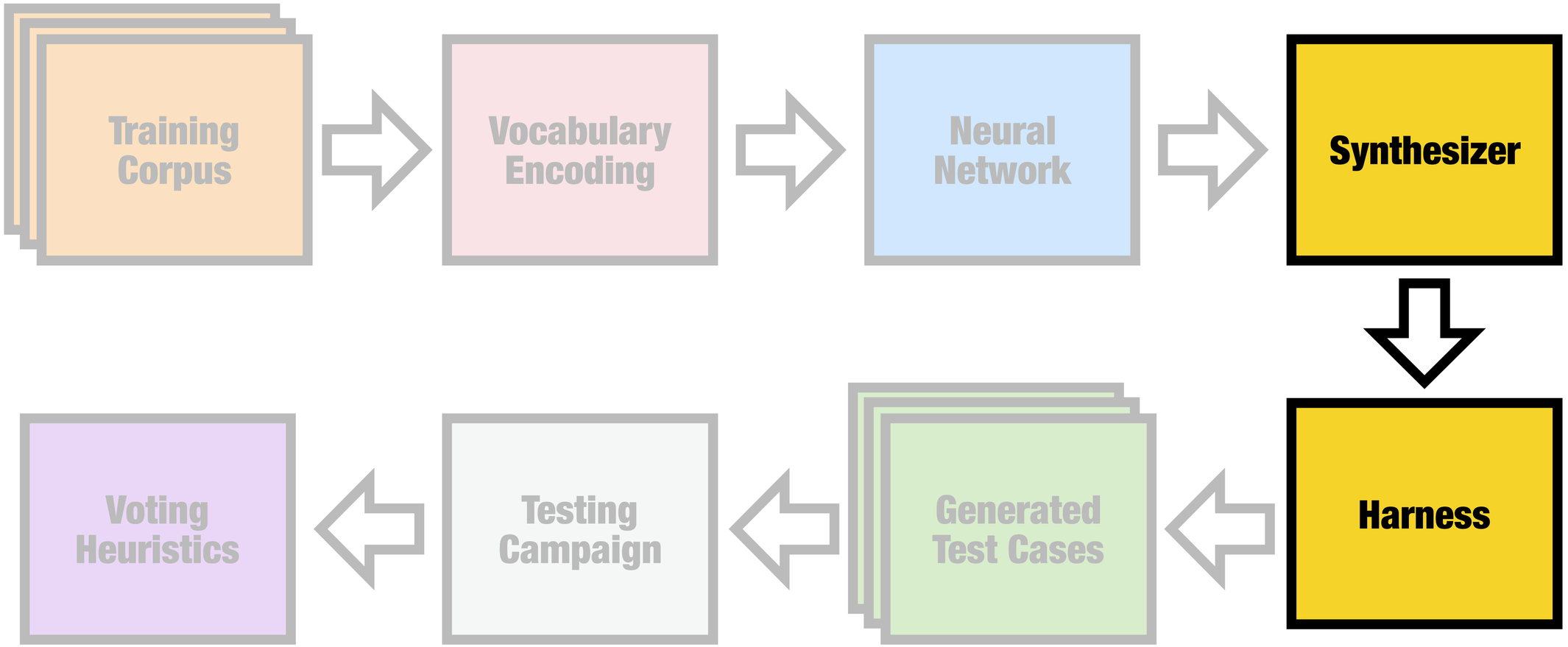
512x2 LSTM

Input: 30M token corpus 0 1 2 ...

Learns probability distribution over corpus.

< 500 lines of code, 12 hours training on GPU.

ML for compiler test generation



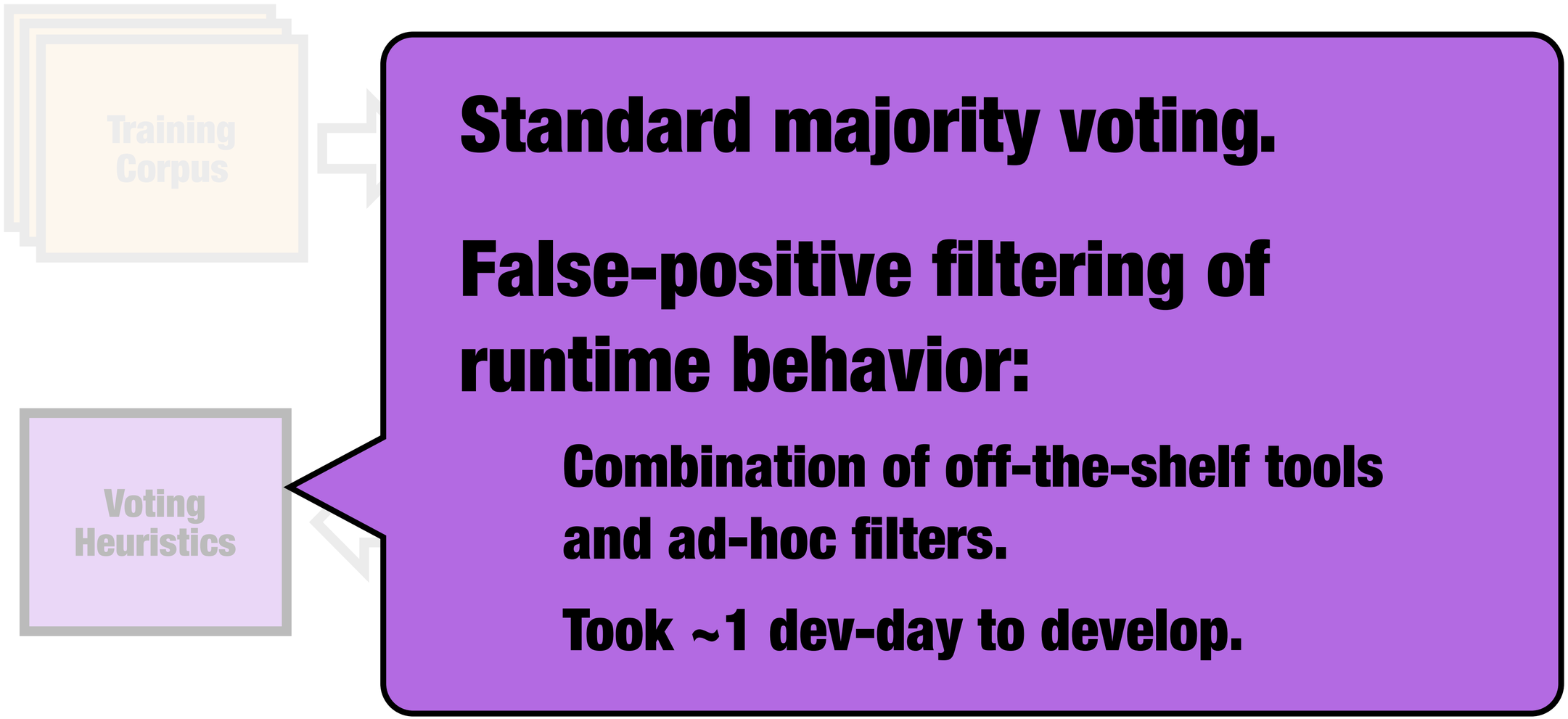
synthesizer + harness

1. Seed the model with the start of a program.
2. Predict tokens until { } brackets balance.
3. Can we parse signature?

    Yes: Generate input data, compile and run it.

    No: Compile it but don’t run it.

ML for compiler test generation



Round 1

Player: **969**, Robot: **1031**

```
__kernel void A(__global uint *a, __global uint *b, int c, int d) {
  const int e = get_global_id(0);
  const int f = get_global_id(1);
  if (e >= c) return;
  if (f >= d) return;

  int g = e + f * c;
  int h = a[g];

  float3 i[4] = {
      {1.0, 1.0, 1.0}, {1.0, 0.7, 0.3}, {1.5, 0.8, [illegible], {0.2, 0.8, 0.2}};

  int j = h & 3;
  float3 k = i[j];
  float l = convert_float((h & (255 - 7)));
  int m = l * k.x;
  if (m > 255) m = 255;
  int n = l * k.y;
  if (n > 255) n = 255;
  int o = l * k.z;
  if (o > 255) o = 255;
  b[g] = o + n * 256 + m * 65536;
}
```

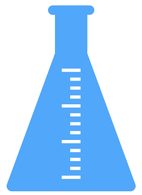
```
__kernel void A(__global int* a, __global int* b, __global int* c, const int
d) {
  unsigned int e = get_global_id(0);
  unsigned int f = get_local_size(0);

  for (unsigned int g = e; g < e + 1; g++) {
    for (unsigned int h = 0; h < f; ++h) {
      a[g * d + h] = 0;
    }
  }
}
```

52%

This side is the Robot

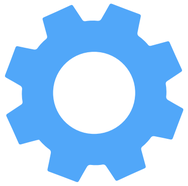
This side is the Robot

testing campaign



10 OpenCL compilers

3 GPUs, 5 CPUs, Xeon Phi, Emulator



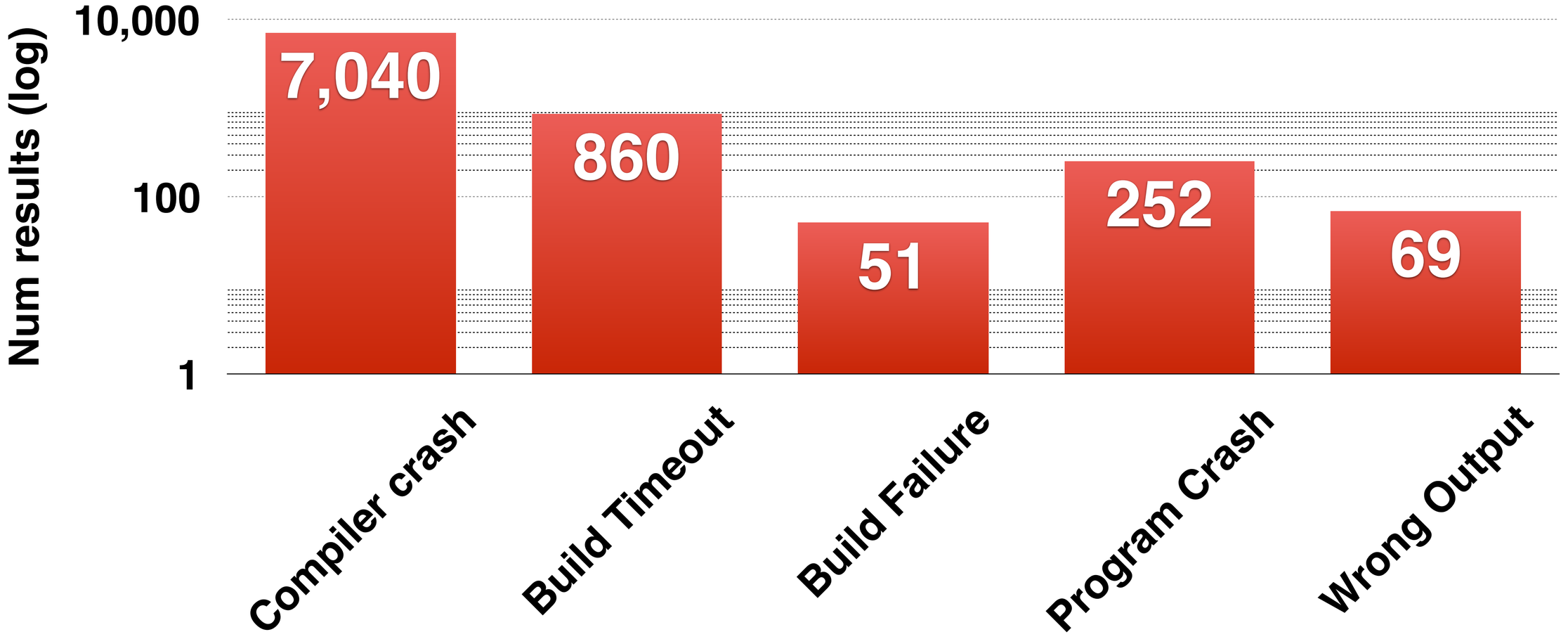
Test with optimizations on / off

Treat as separate testbeds

48 hours per testbed

results overview

Errors in ever compiler!



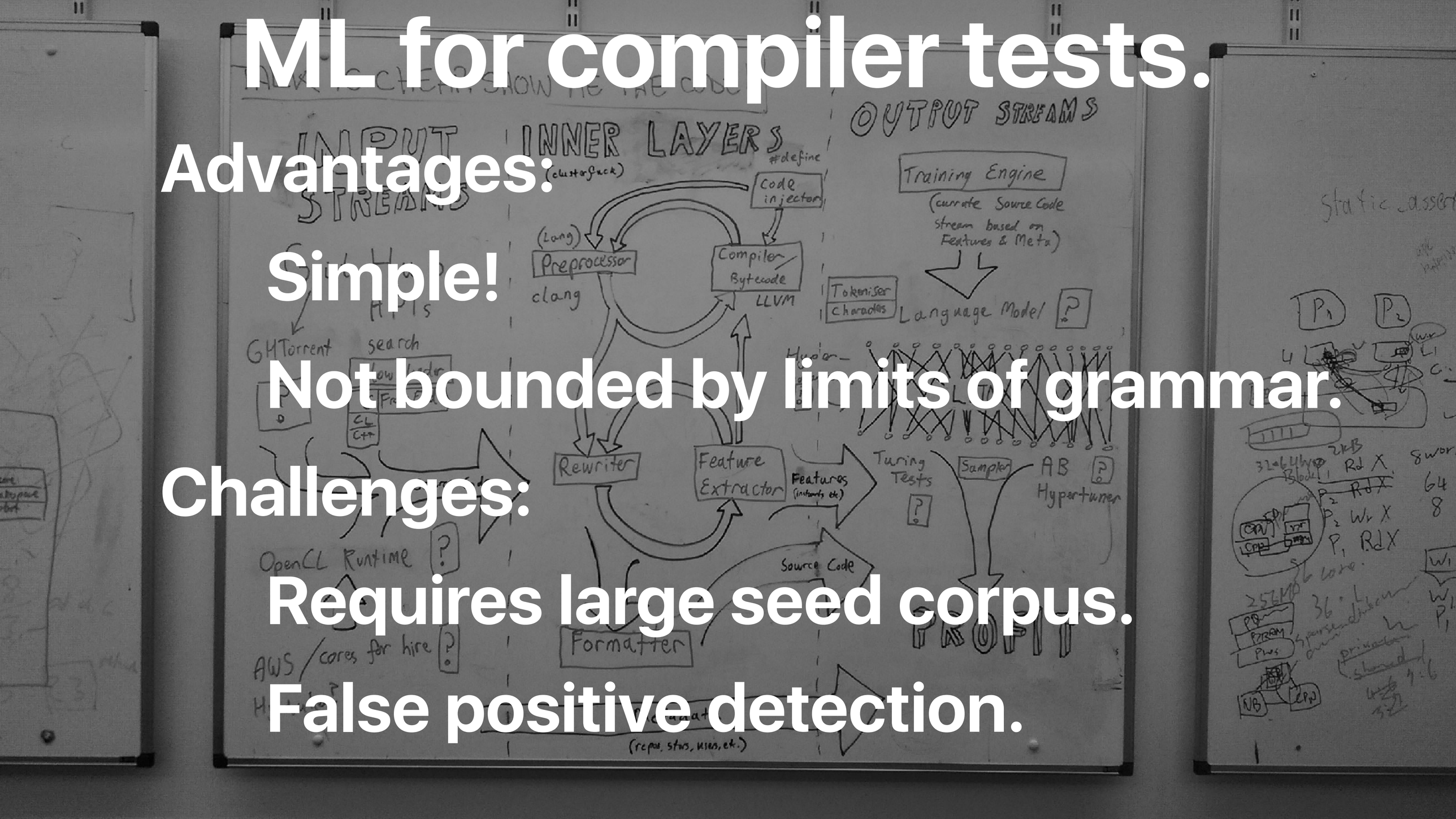
ML for compiler tests.

Advantages:

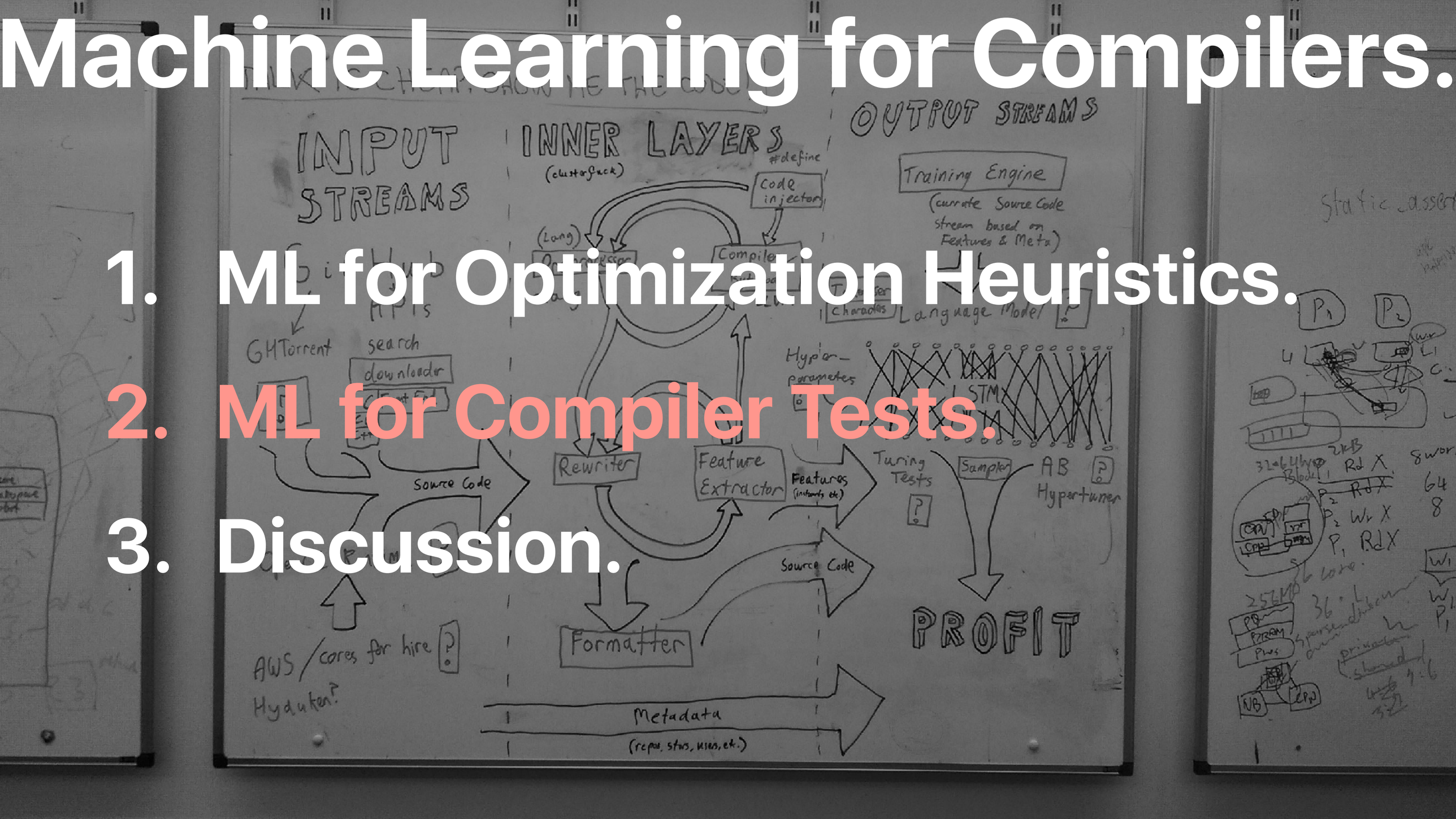
- Simple!
- Not bounded by limits of grammar.

Challenges:

- Requires large seed corpus.
- False positive detection.

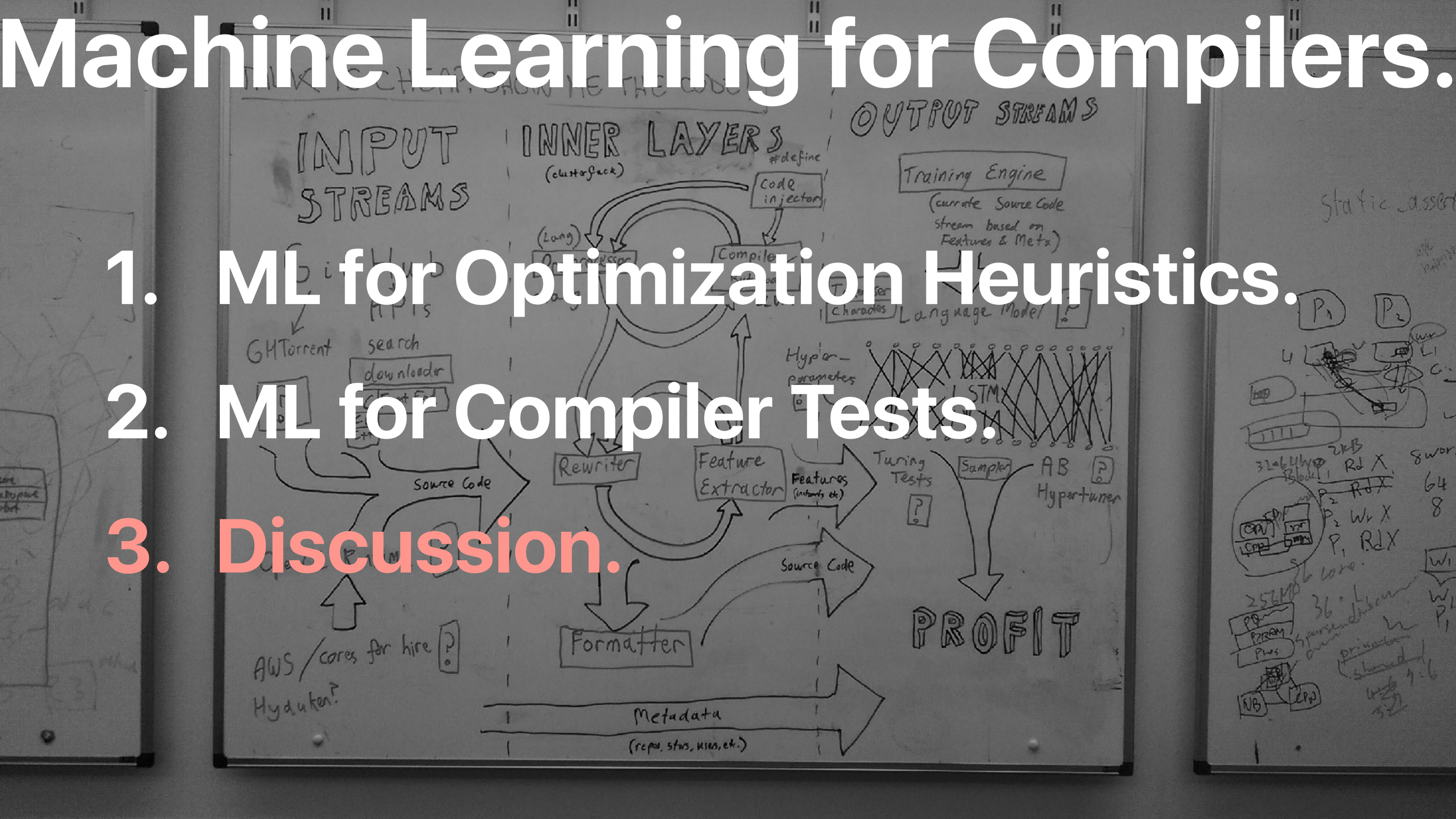


Machine Learning for Compilers.



1. ML for Optimization Heuristics.
2. ML for Compiler Tests.
3. Discussion.

Machine Learning for Compilers.



1. ML for Optimization Heuristics.
2. ML for Compiler Tests.
3. Discussion.

Discussion

What problems do you know of where:

There is room for improvement?

Currently rely on domain expertise?

Challenges with ML:

Feature design / selection.

Correctness and data distribution.