Optimizing Homomorphic Evaluation Circuit with Program Synthesis and Term Rewriting

Dongkwon Lee, Woosuk Lee, Hakjoo Oh, Kwangkeun Yi

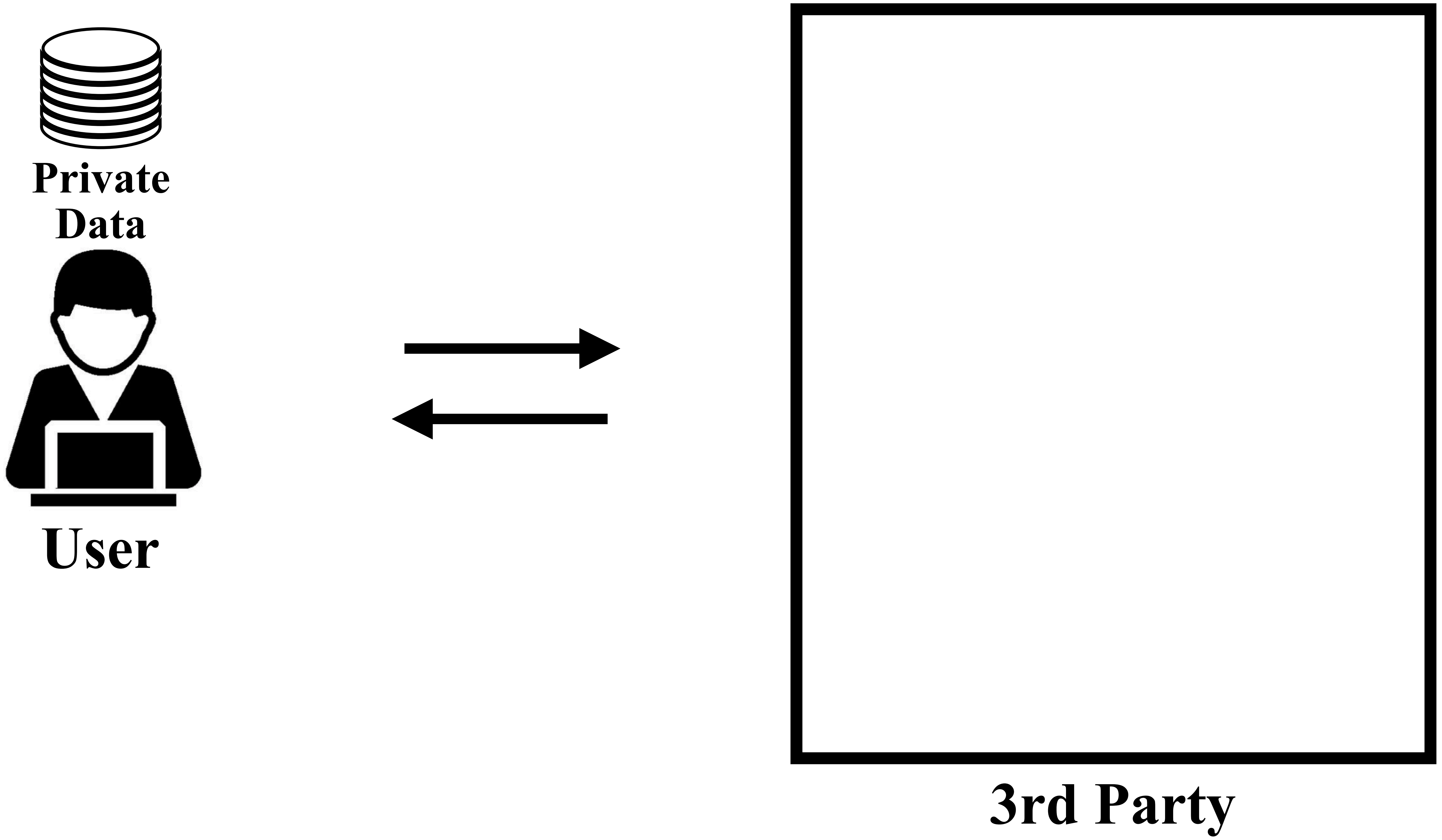
Seoul National University | Hanyang University | Korea University | Seoul National University

Homomorphic Evaluation(HE) (1/3)

Privacy Preserving Secure Computation

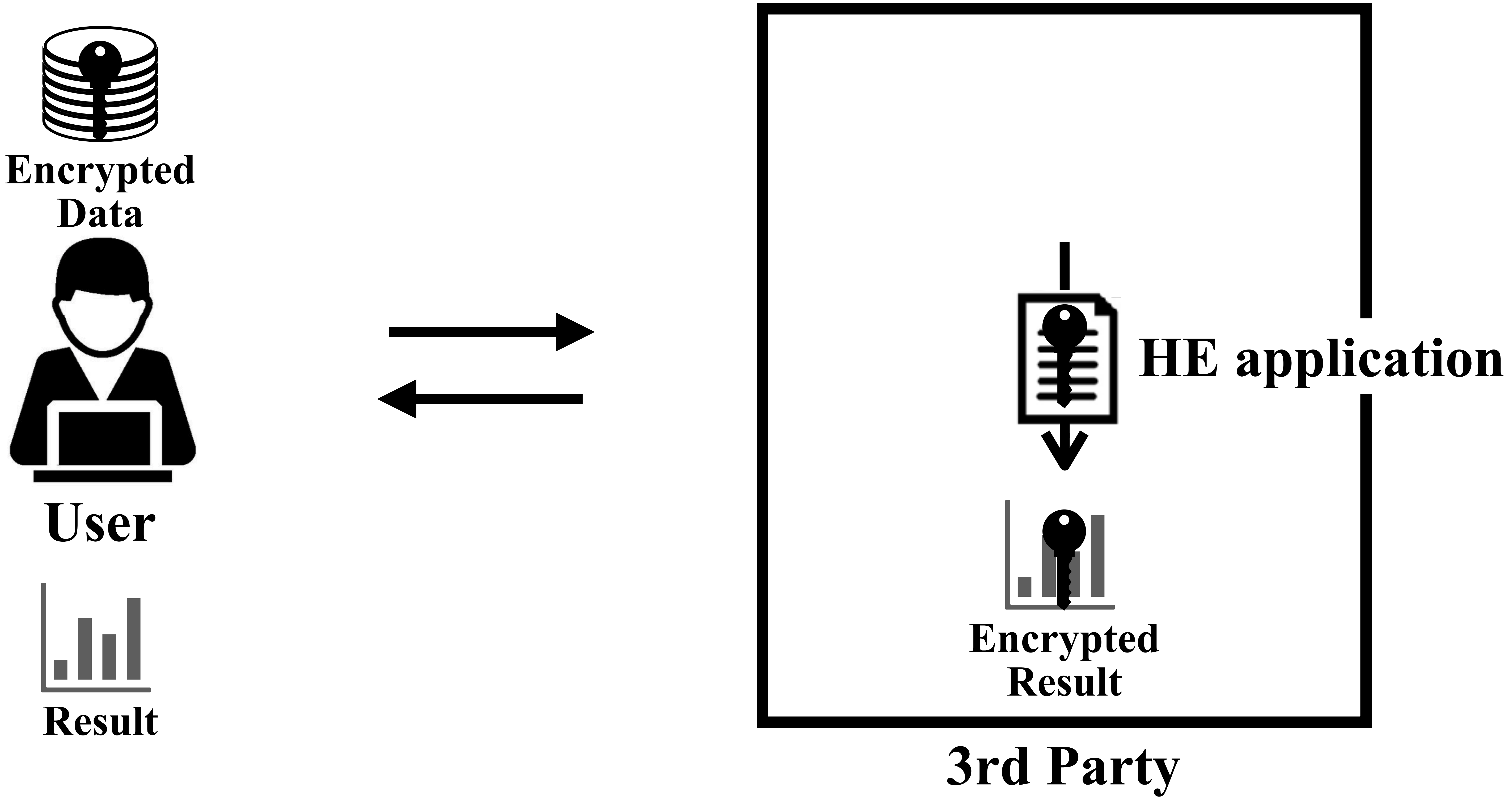
- Allows for computation on encrypted data
- Enables the outsourcing of private data storage/processing



Homomorphic Evaluation(HE) (1/3)

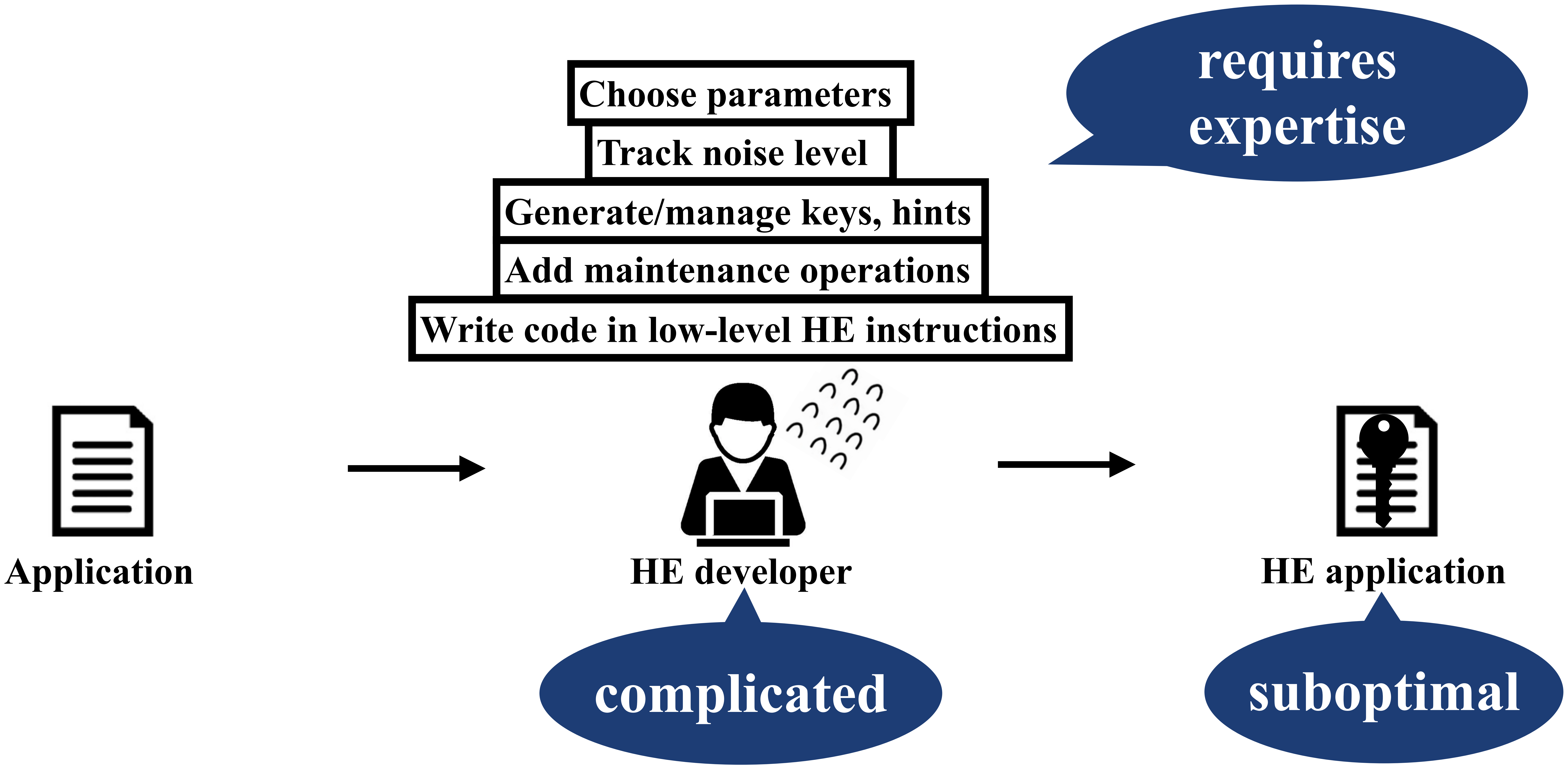
Privacy Preserving Secure Computation

- Allows for computation on encrypted data
- Enables the outsourcing of private data storage/processing



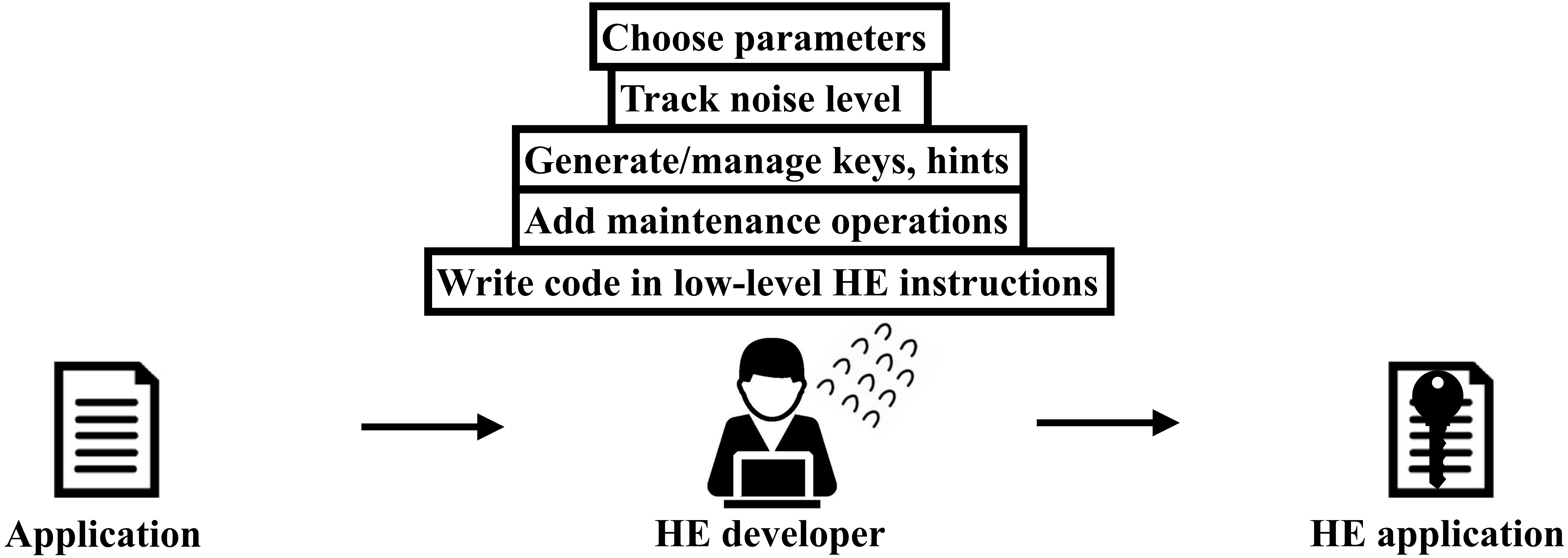
Homomorphic Evaluation(HE) (2/3)

Building HE applications



Homomorphic Evaluation(HE) (3/3)

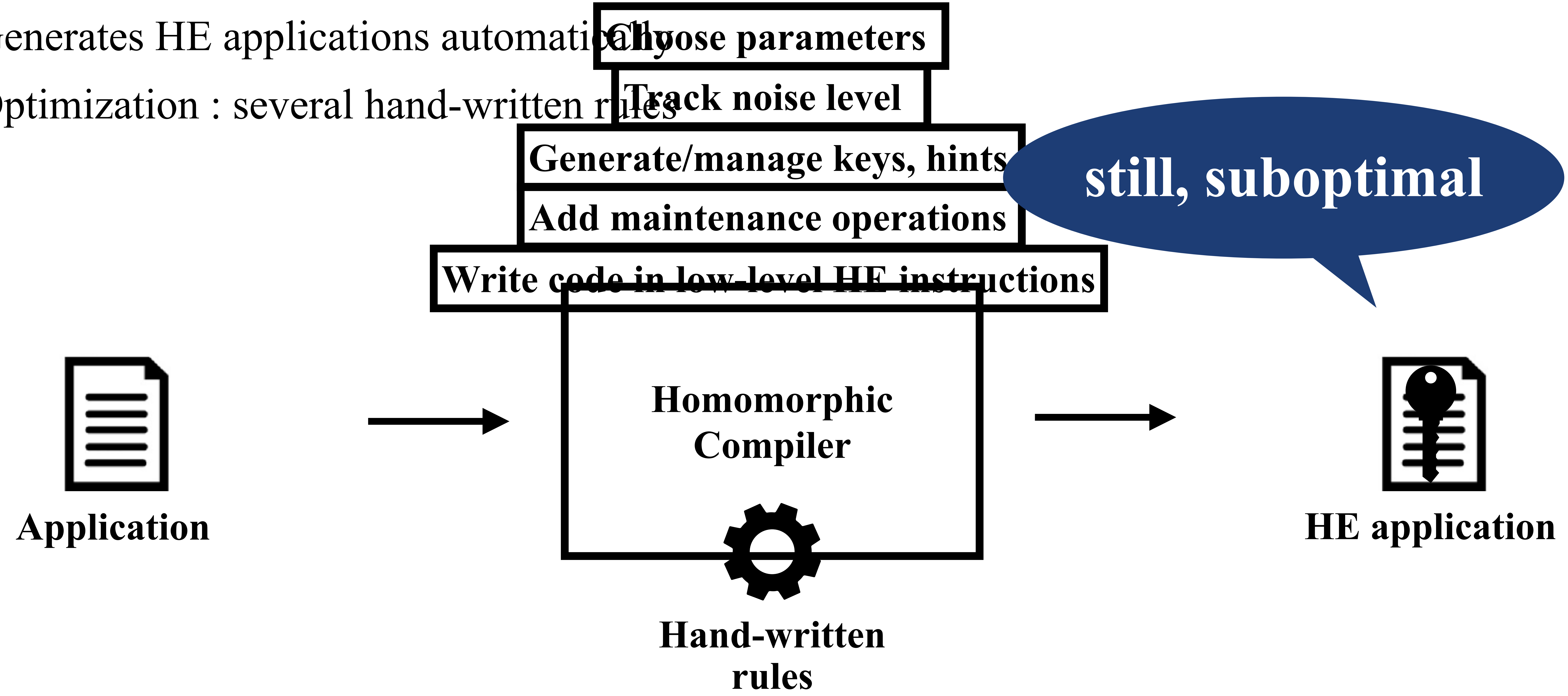
Existing Homomorphic Compiler



Homomorphic Evaluation(HE) (3/3)

Existing Homomorphic Compiler

- Generates HE applications automatically
- Optimization : several hand-written rules



Homomorphic Evaluation(HE) (2/3)

Code for homomorphic addition of two integers

```
#include "FHE.h"
#include "EncryptedArray.h"
#include <NTL/lzz_pXFactoring.h>
#include <fstream>
#include <sstream>
#include <sys/time.h>

int main(int argc, char **argv)
{
    long m=0, p=2, r=1; // Native plaintext space
                        // Computations will be 'modulo p'
    long L=16;          // Levels
    long c=3;           // Columns in key switching matrix
    long w=64;          // Hamming weight of secret key
    long d=0;
    long security = 128;
    ZZX G;
    m = FindM(security,L,c,p, d, 0, 0);
    FHEcontext context(m, p, r);
    buildModChain(context, L, c);
    FHESecKey secretKey(context);
    const FHEPubKey& publicKey = secretKey;
    G = context.alMod.getFactorsOverZZ()[0];
    secretKey.GenSecKey(w);
    addSome1DMatrices(secretKey);
    EncryptedArray ea(context, G);
    vector<long> v1;
    v1.push_back(atoi(argv[1]));
    Ctxt ct1(publicKey);
    ea.encrypt(ct1, publicKey, v1);
    v2.push_back(atoi(argv[2]));
    Ctxt ct2(publicKey);
    ea.encrypt(ct2, publicKey, v2);
    Ctxt ctSum = ct1;
    ctSum += ct2;
}
```

Manually written using HElib

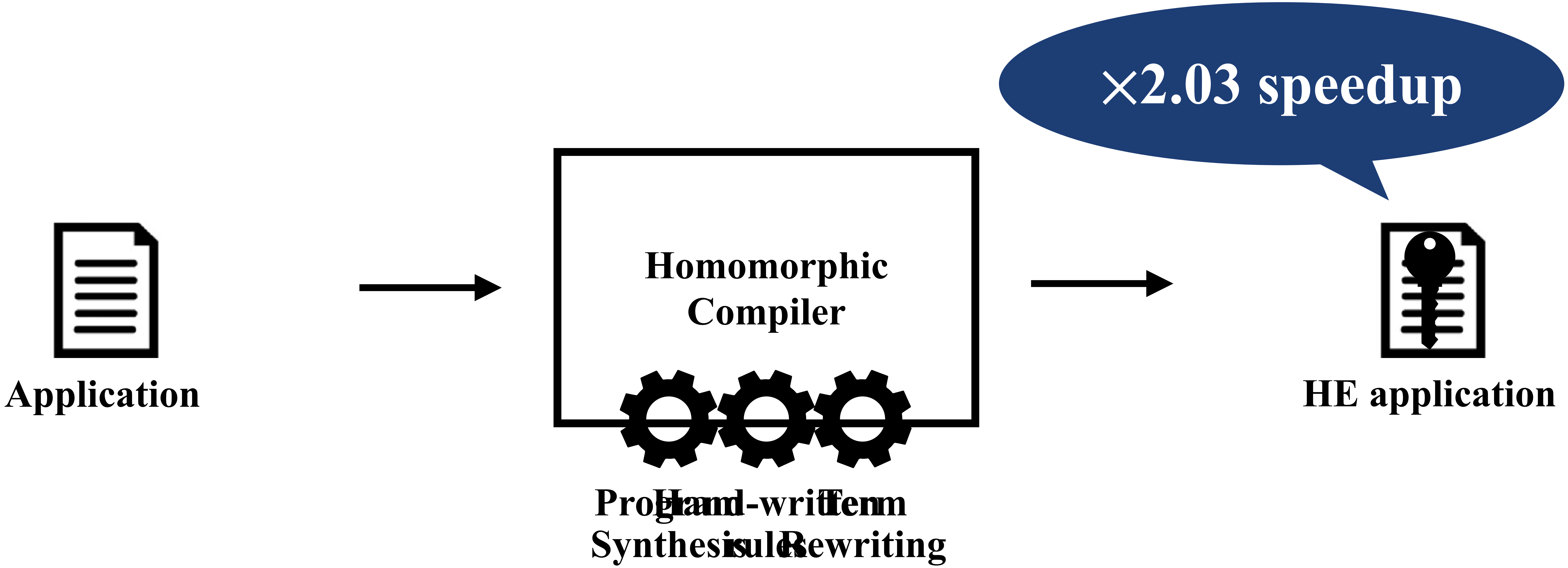
```
#include <iostream>
#include <fstream>
#include <integer.hxx>

int main()
{
        Integer8 a, b, c;

        cin >> a;
        cin >> b;
        c = a + b;

        cout << c;
        FINALIZE_CIRCUIT(blif_name);
}
```

Input to Cingulata (a HE compiler)

Our Contributions (1/2)

Automatic, Aggressive HE optimization Framework

- Generates HE applications automatically
- Optimization : search hand-written rules
- Optimization : new rules found by program synthesis + applying by term rewriting



Our Contributions (2/2)

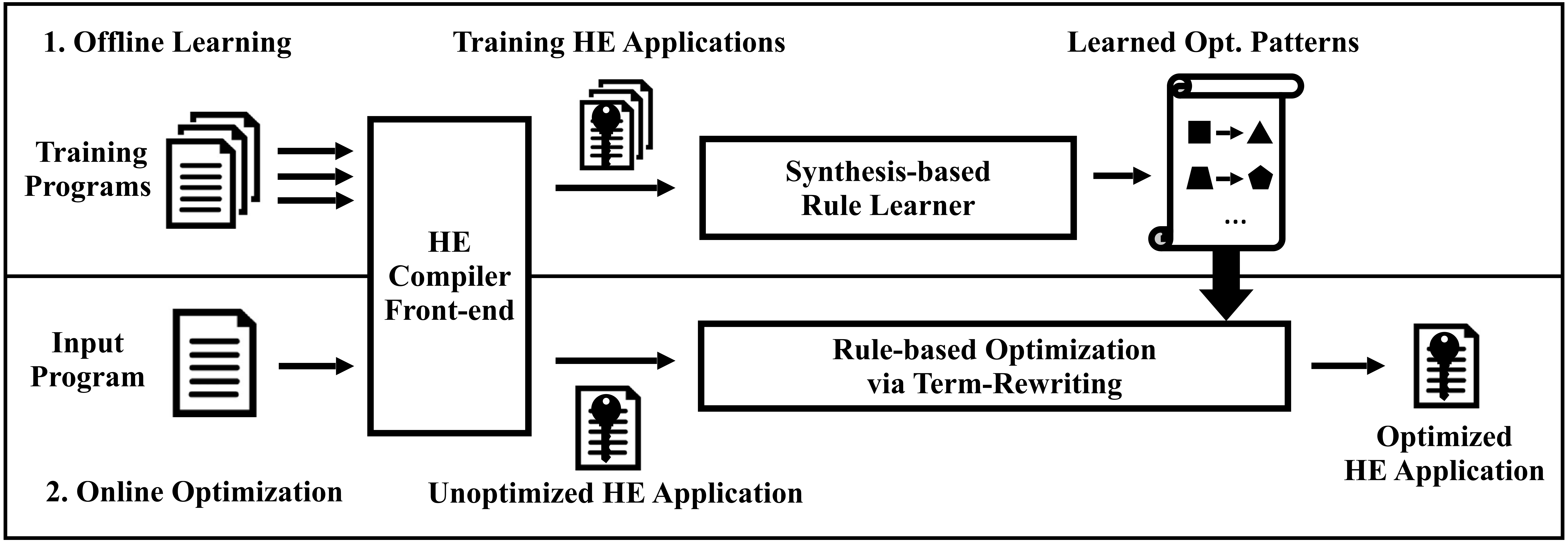
Automatic, Aggressive HE optimization Framework

- Learning Optimization Patterns by Program Synthesis
- Applying Learned Patterns by Term Rewriting
- Theorem : Semantic Preservation & Termination Guaranteed
- Performance (vs state-of-the-art HE Optimizer)
  - Optimized 19 out of 25 Applications (vs 15)
  - x3.71 Speedup in Maximum (vs x3.0)
  - x2.03 Speedup on Average (vs x1.53)
- Open Tool Available : https://github.com/dklee0501/Lobster

Our Lobster 

Learning to Optimize Boolean circuit using Synthesis and TErm Rewriting

- Offline Learning via Program Synthesis + Online Optimization via Term Rewriting



Simple HE Scheme

- Based on approximate common divisor problem
- p : integer as a secret key
- q : random integer
- r ($\ll |p|$) : random noise for security

$$Enc_p(\ \mu \in \{0,1\}\ ) = pq + 2r + \mu$$

$$Dec_p(c) = (c \bmod p) \bmod 2$$

$$Dec_p(Enc_p(\mu)) = Dec_p(\cancel{pq} + \cancel{2r} + \mu) = \mu$$

- For ciphertexts $\underline{\mu_i} \leftarrow Enc_p(\mu_i)$, the following holds

$$Dec_p(\underline{\mu_1} + \underline{\mu_2}) = \mu_1 + \mu_2$$
$$Dec_p(\underline{\mu_1} \times \underline{\mu_2}) = \mu_1 \times \mu_2$$

- The scheme can evaluate all boolean circuits as $+$ and $\times$ in $\mathbb{Z}_2 = \{0,1\}$ are equal to XOR and AND

Performance Hurdle : Growing Noise

- Noise increases during homomorphic operations.
- For $\underline{\mu_i} = pq_i + 2r_i + \mu_i$

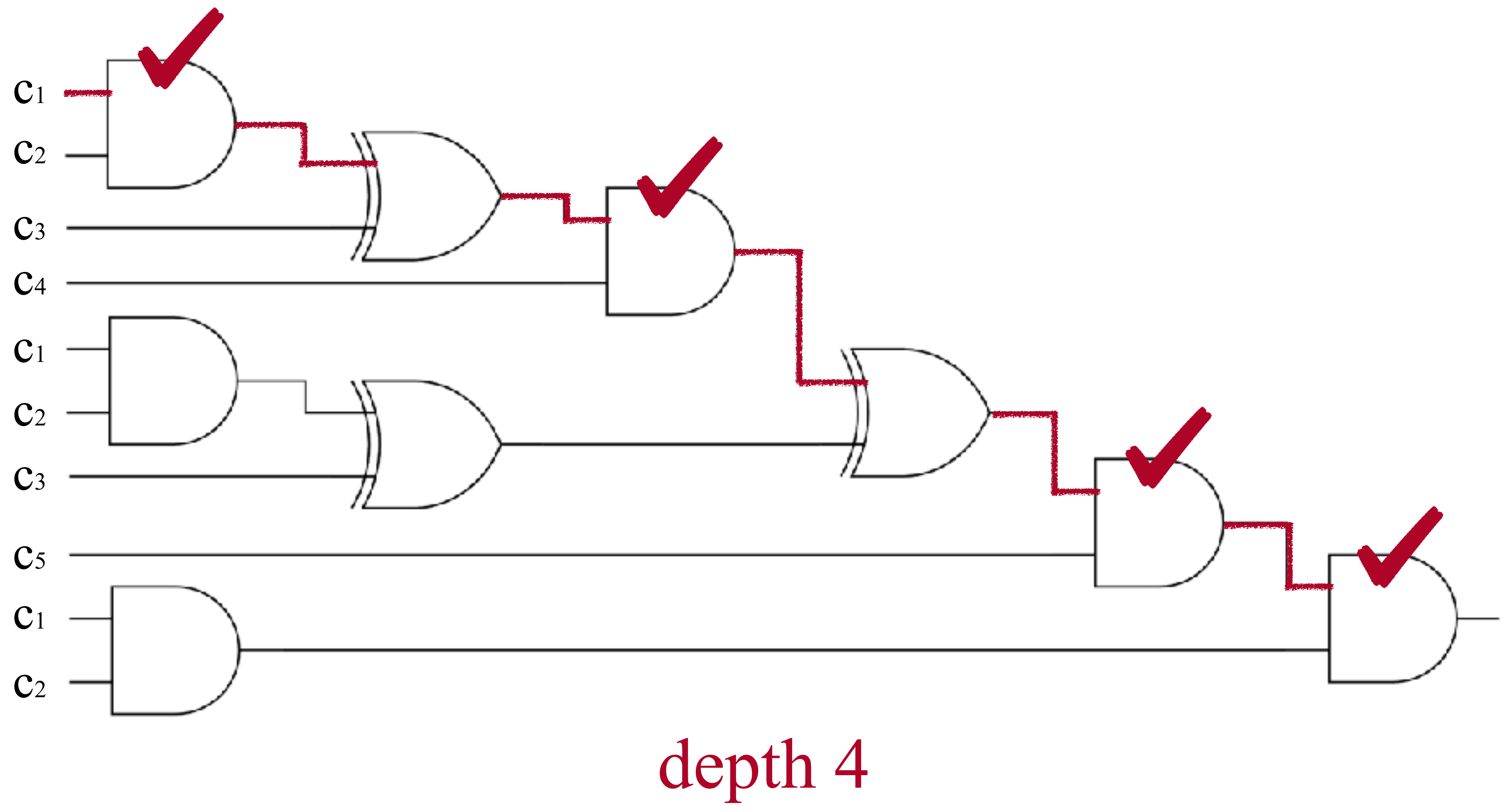
$$\begin{aligned}
\underline{\mu_1} + \underline{\mu_2} &= p(q_1 + q_2) + \boxed{2(r_1 + r_2) + (\mu_1 + \mu_2)} \text{ double increase} \\
\underline{\mu_1} \times \underline{\mu_2} &= p(pq_1q_2 + \cdots) + \boxed{2(2r_1r_2 + r_1\mu_2 + r_2\mu_1) + (\mu_1 \times \mu_2)} \text{ quadratic increase}
\end{aligned}$$

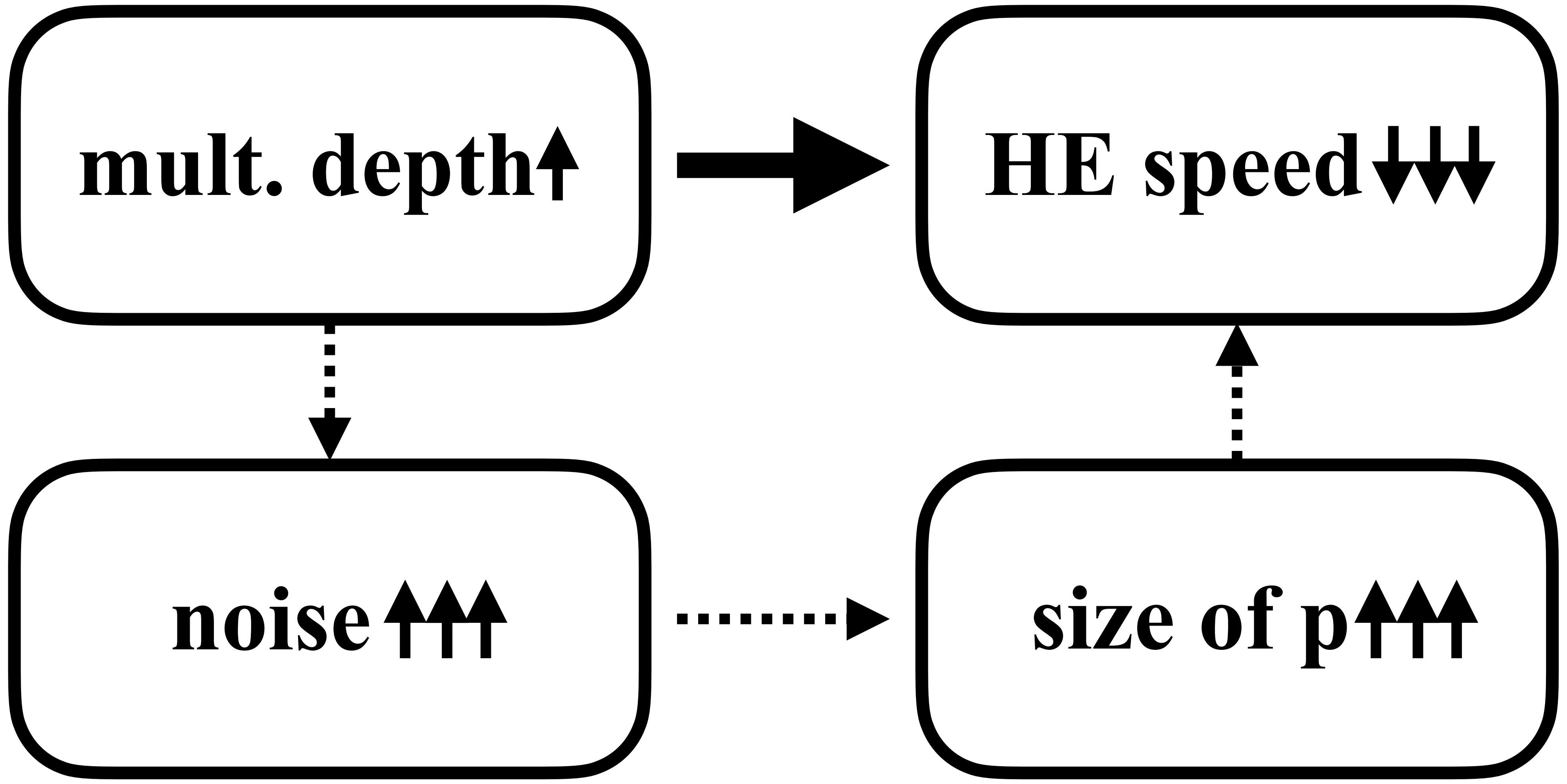
noise

- **if (noise $>$ p) then incorrect results**

Multiplicative Depth : a Decisive Performance Factor

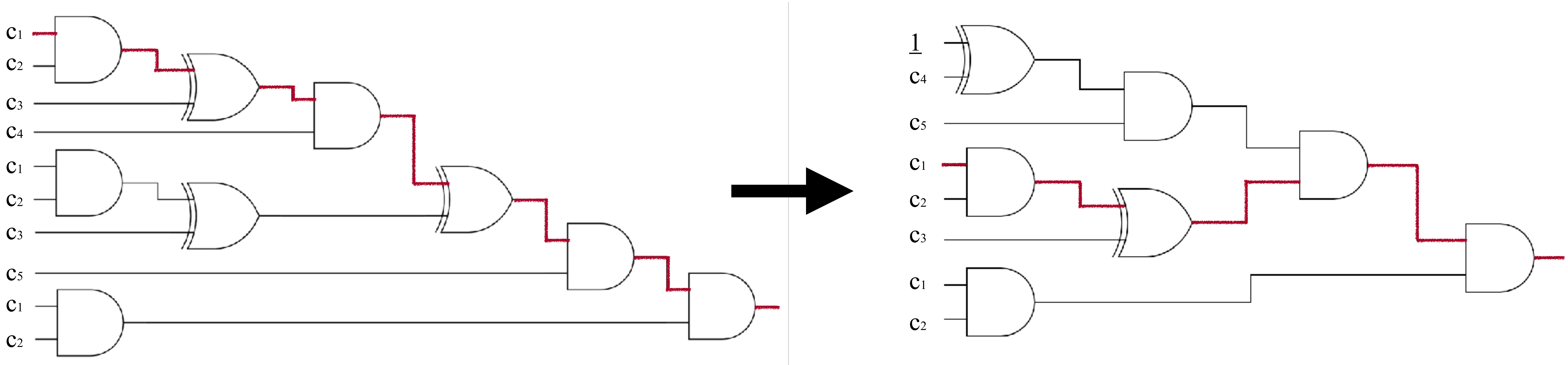
- Multiplicative depth : the maximum number of sequential multiplications from input to output





What is HE optimization?

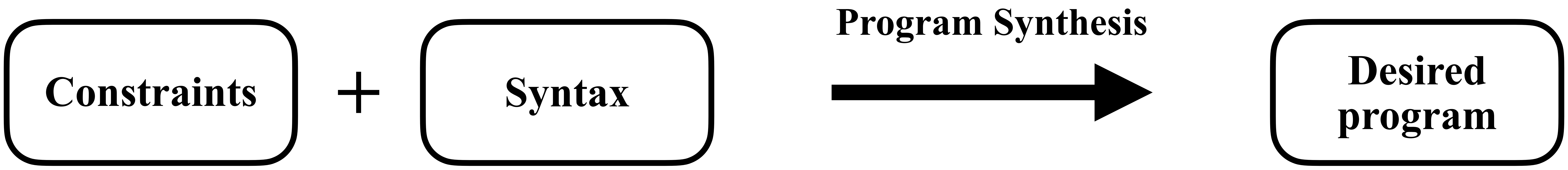
- Finding a new circuit that has smaller mult. depth



depth 4

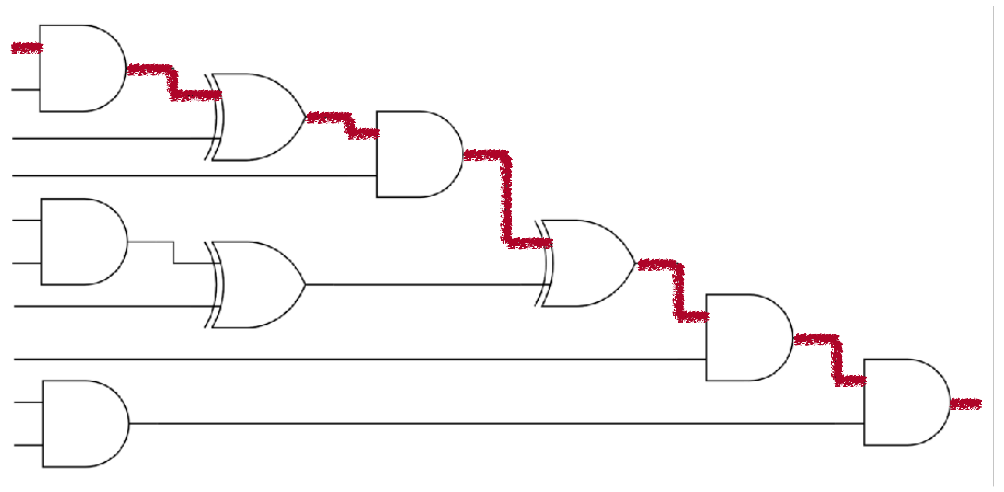
depth 3

HE optimization via Synthesis



HE optimization via Synthesis

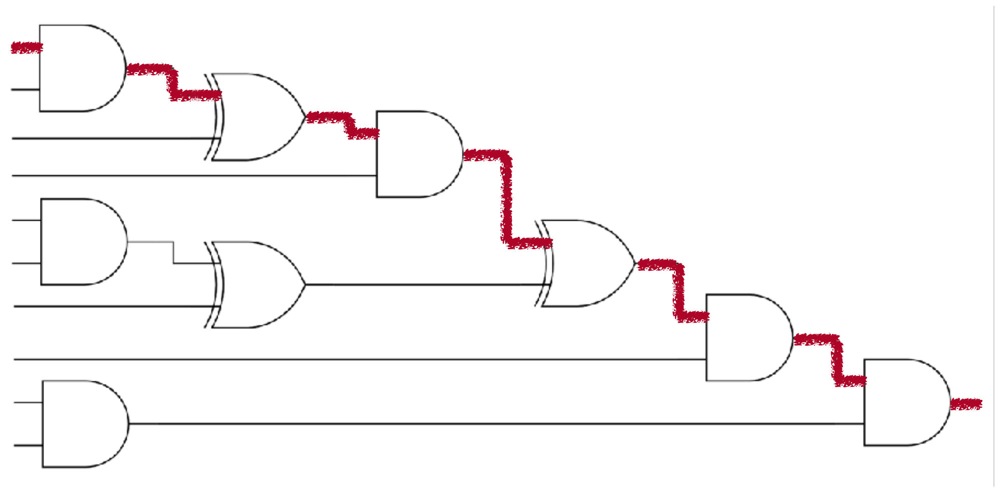
Constraints + Syntax —Program Synthesis→ Desired program



same semantics

HE optimization via Synthesis

Constraints + **Syntax** $\xrightarrow{\text{Program Synthesis}}$ **Desired program**



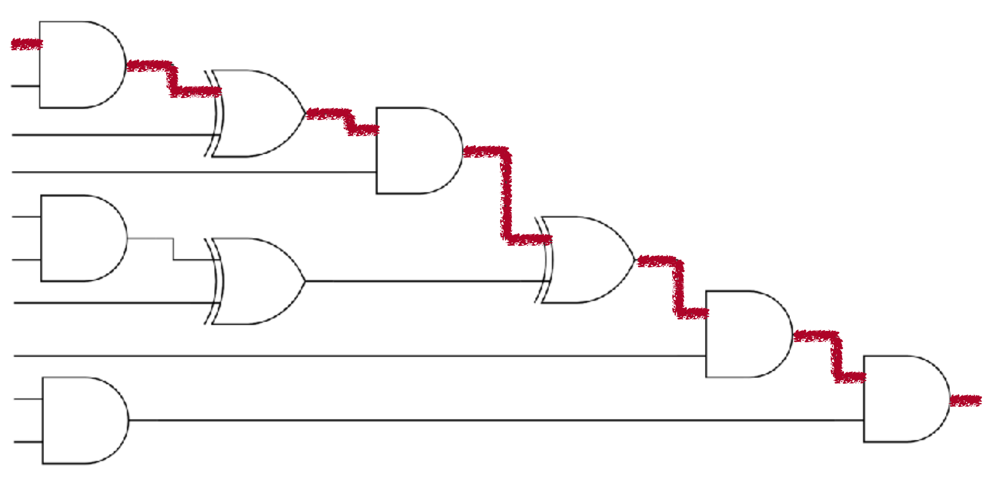
same semantics

$$
\begin{aligned}
S &\rightarrow d_3 \\
d_3 &\rightarrow d_2 \wedge d_2 \mid d_3 \oplus d_3 \mid d_2 \\
d_2 &\rightarrow d_1 \wedge d_1 \mid d_2 \oplus d_2 \mid d_1 \\
d_1 &\rightarrow d_0 \wedge d_0 \mid d_1 \oplus d_1 \mid d_0 \\
d_0 &\rightarrow 0 \mid 1 \mid c_1 \mid c_2 \mid c_3 \mid c_4 \mid c_5
\end{aligned}
$$

depth-restricting syntax

HE optimization via Synthesis

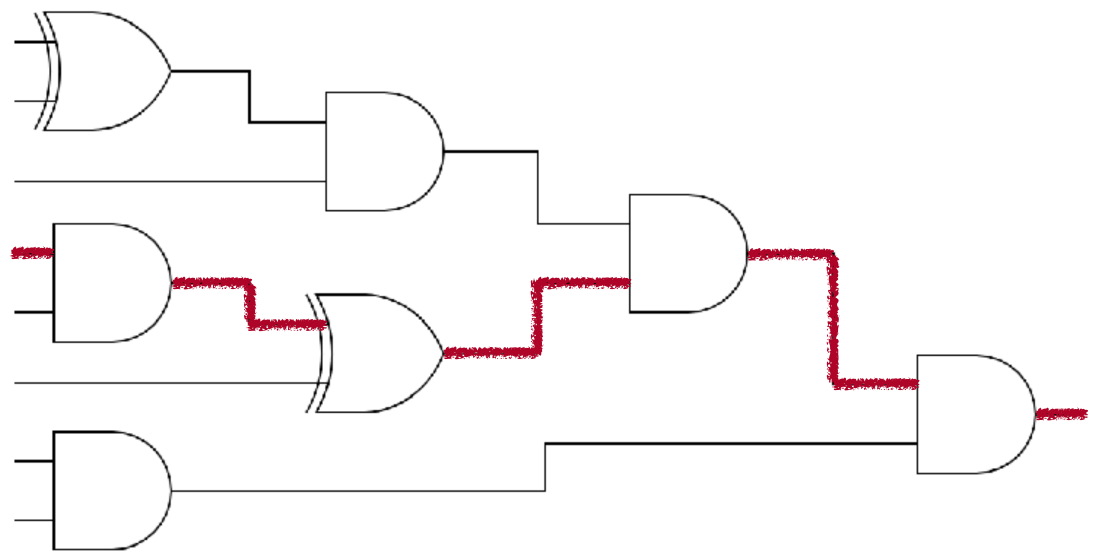
Constraints + **Syntax** —Program Synthesis→ **Desired program**



same semantics

$$
\begin{array}{lcl}
S & \rightarrow & d_3 \\
d_3 & \rightarrow & d_2 \wedge d_2 \mid d_3 \oplus d_3 \mid d_2 \\
d_2 & \rightarrow & d_1 \wedge d_1 \mid d_2 \oplus d_2 \mid d_1 \\
d_1 & \rightarrow & d_0 \wedge d_0 \mid d_1 \oplus d_1 \mid d_0 \\
d_0 & \rightarrow & 0 \mid 1 \mid c_1 \mid c_2 \mid c_3 \mid c_4 \mid c_5
\end{array}
$$

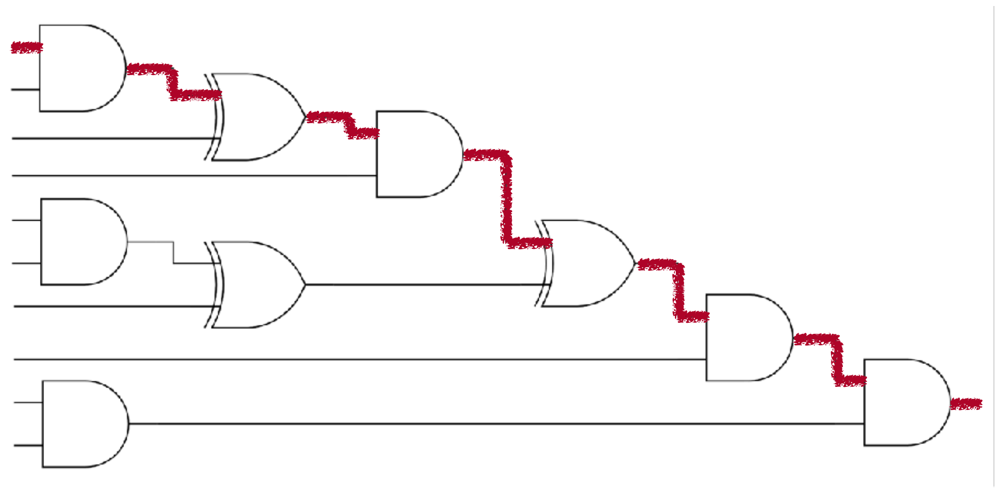
depth-restricting syntax



optimized HE circuit

HE optimization via Synthesis

Constraints + **Syntax** $\xrightarrow{\text{Optimizing Synthesis}}$ **Desired program**

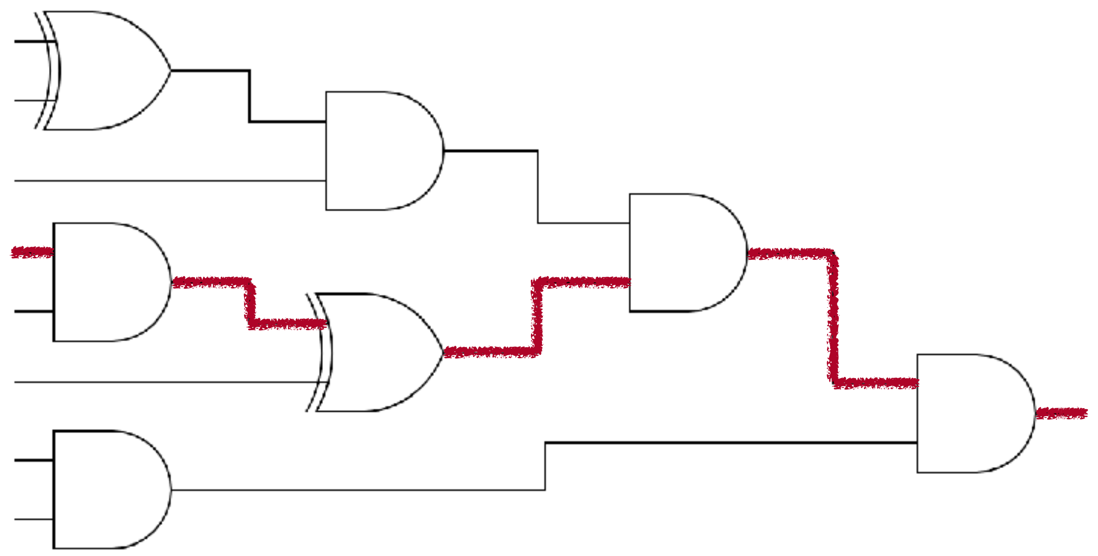


depth 4

same semantics

$$\begin{aligned} S &\rightarrow d_3 \\ d_3 &\rightarrow d_2 \wedge d_2 \mid d_3 \oplus d_3 \mid d_2 \\ d_2 &\rightarrow d_1 \wedge d_1 \mid d_2 \oplus d_2 \mid d_1 \\ d_1 &\rightarrow d_0 \wedge d_0 \mid d_1 \oplus d_1 \mid d_0 \\ d_0 &\rightarrow 0 \mid 1 \mid c_1 \mid c_2 \mid c_3 \mid c_4 \mid c_5 \end{aligned}$$

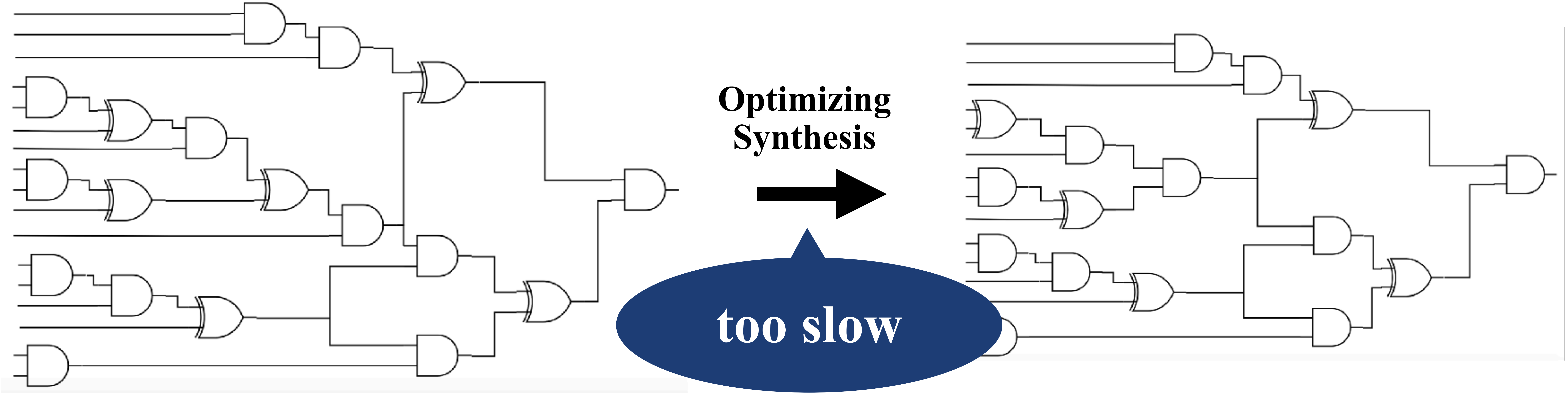
depth-restricting syntax



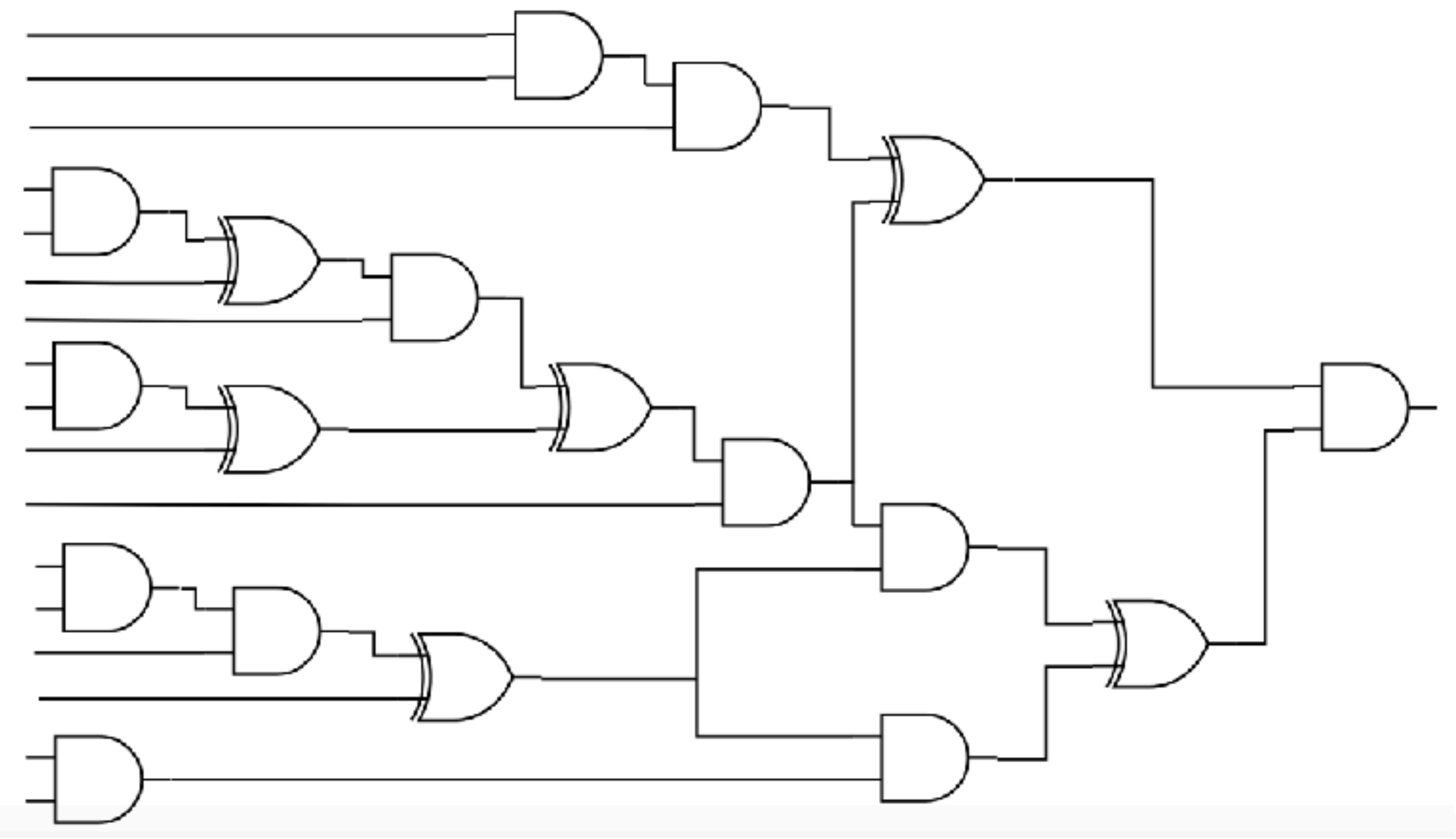
depth 3

optimized HE circuit

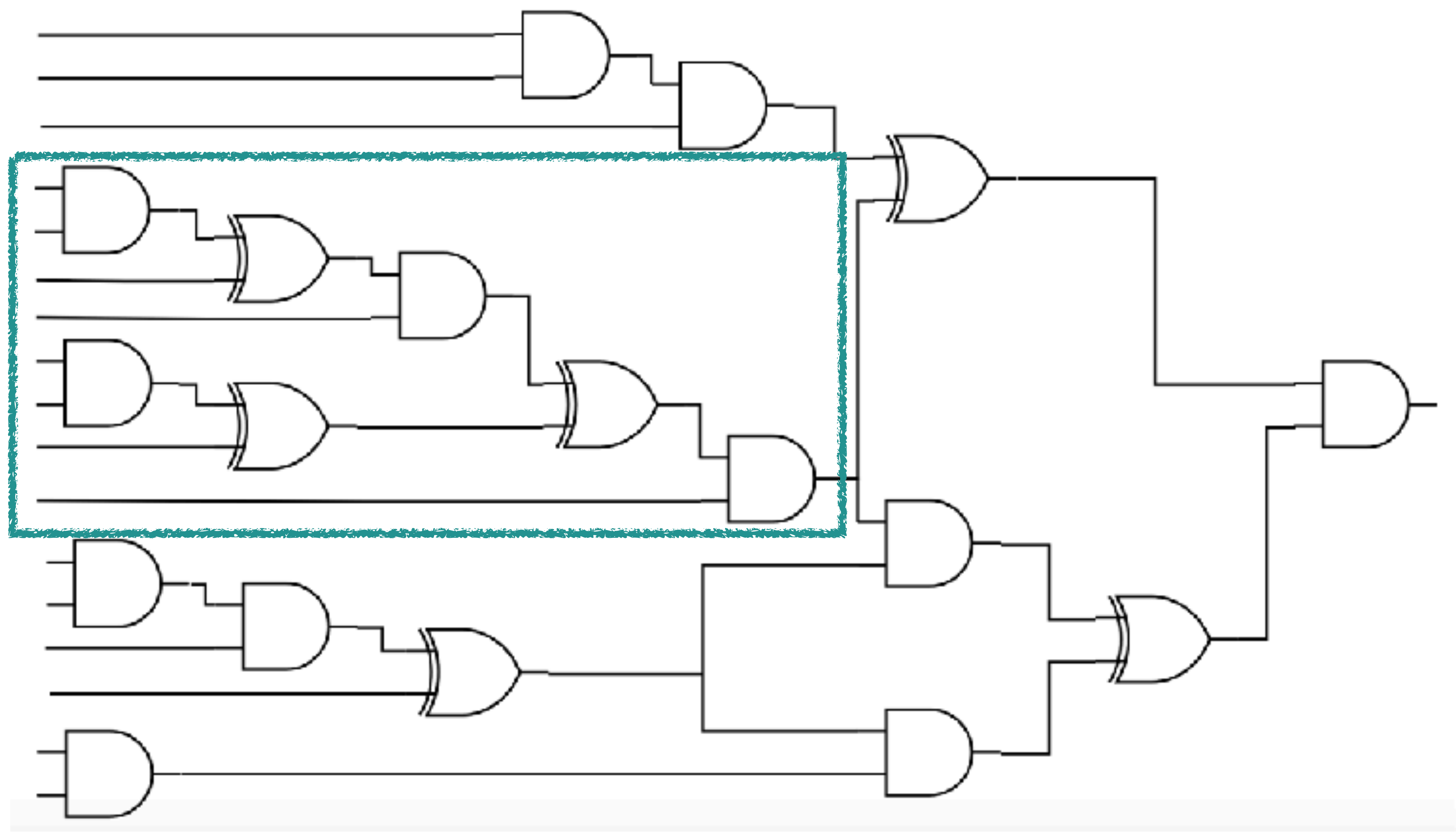
Hurdle : Synthesis Scalability



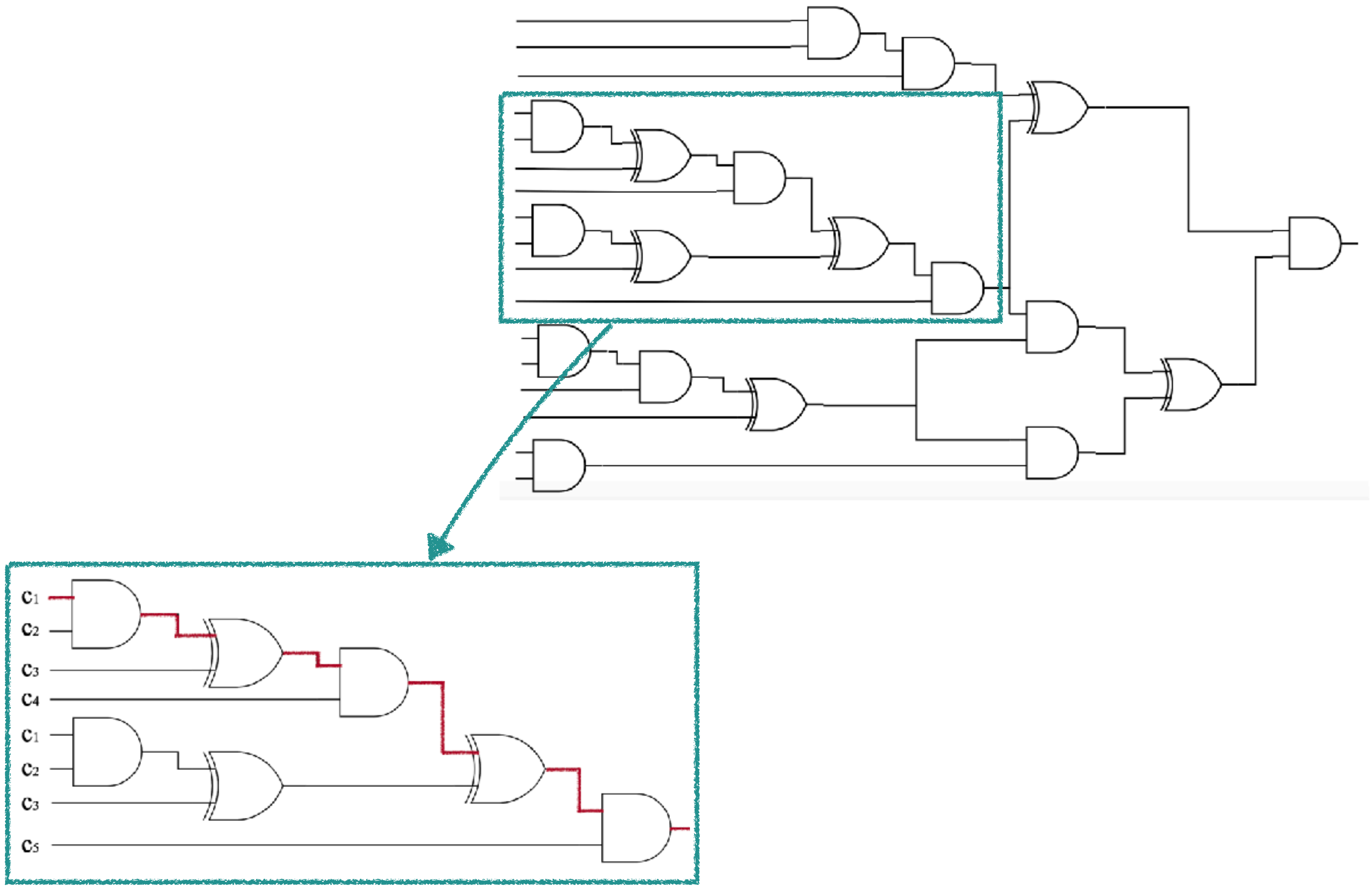
Solution1 : Synthesis via Localization



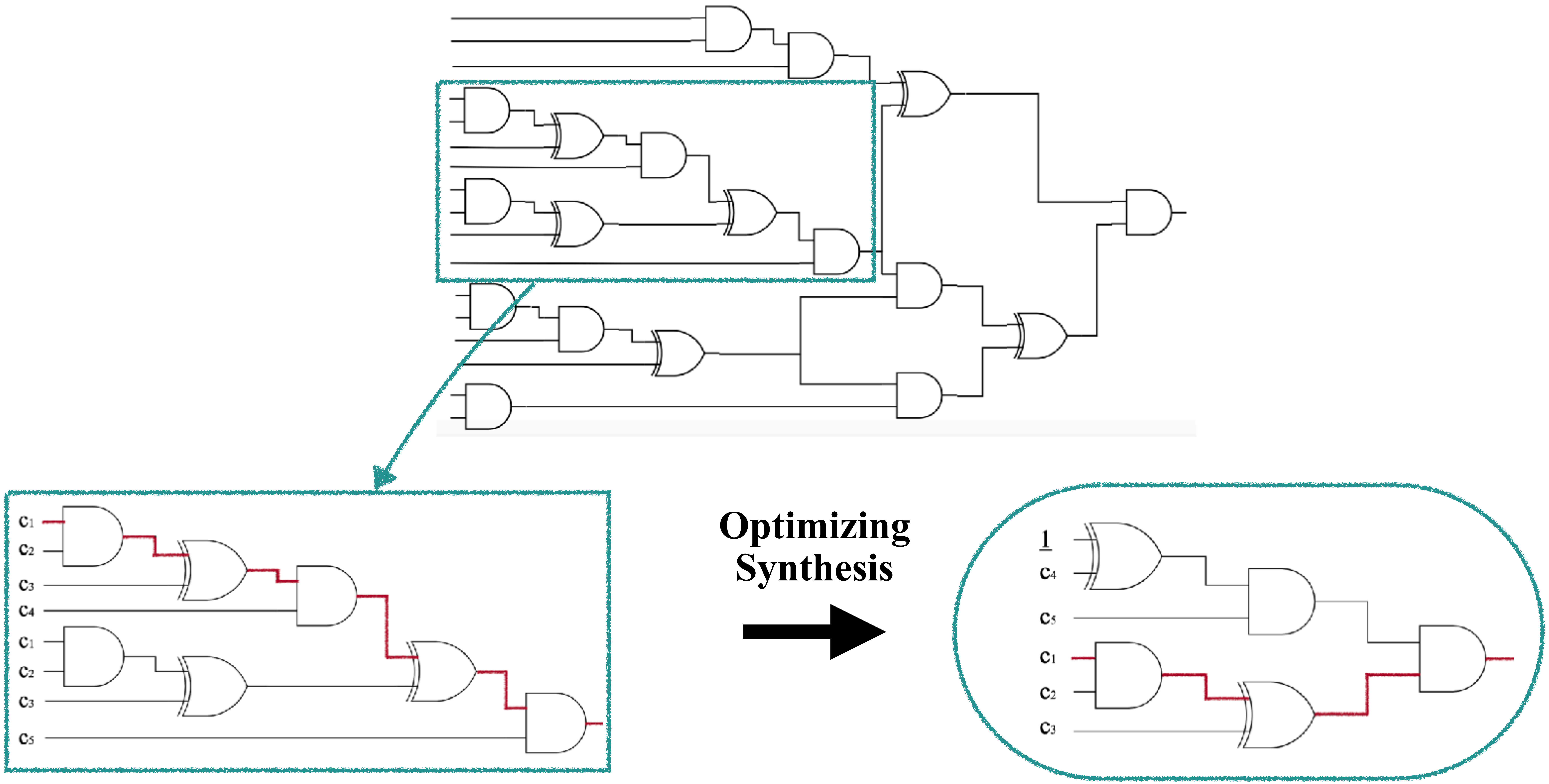
Solution1 : Synthesis via Localization



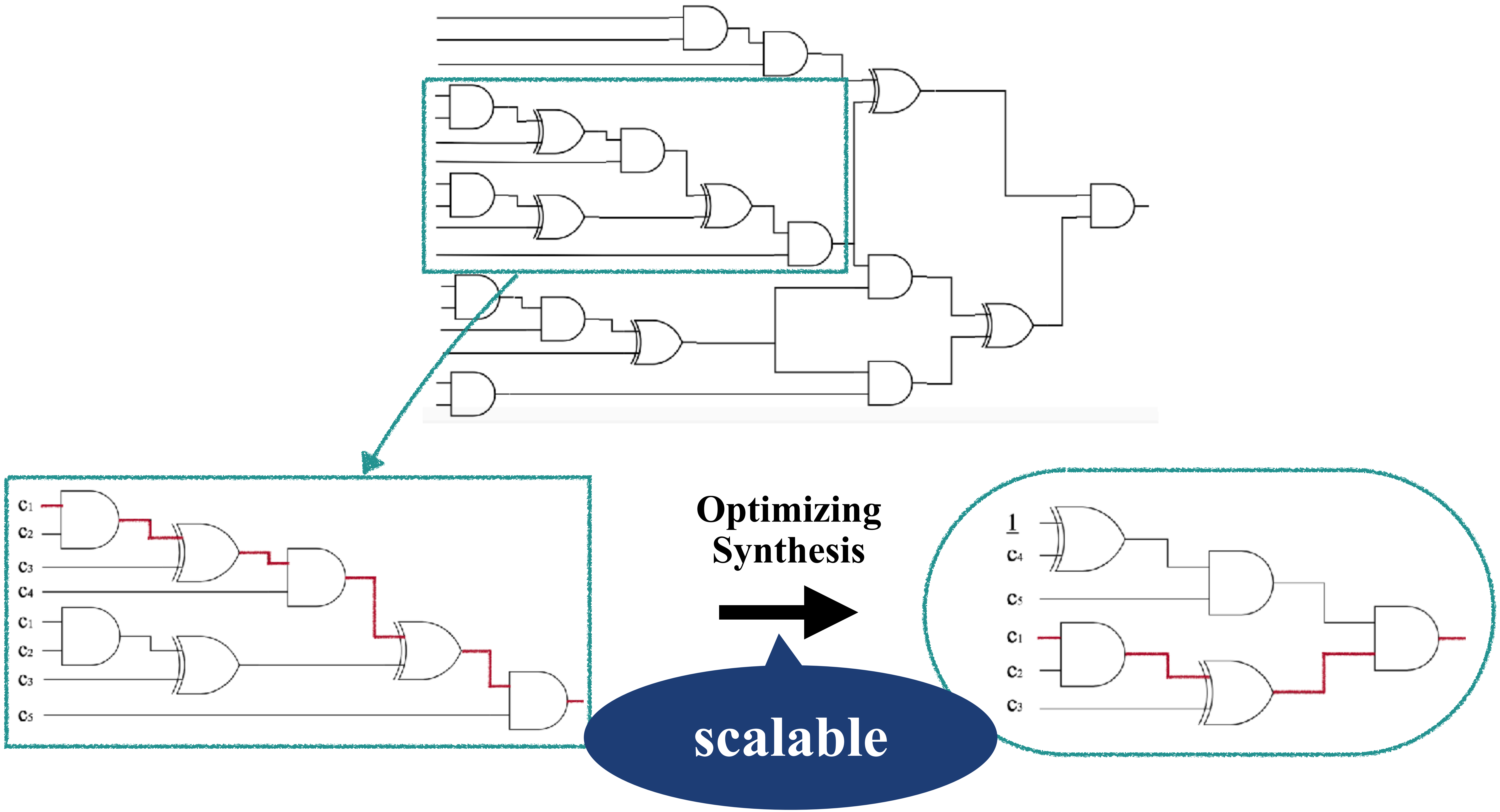
Solution1 : Synthesis via Localization



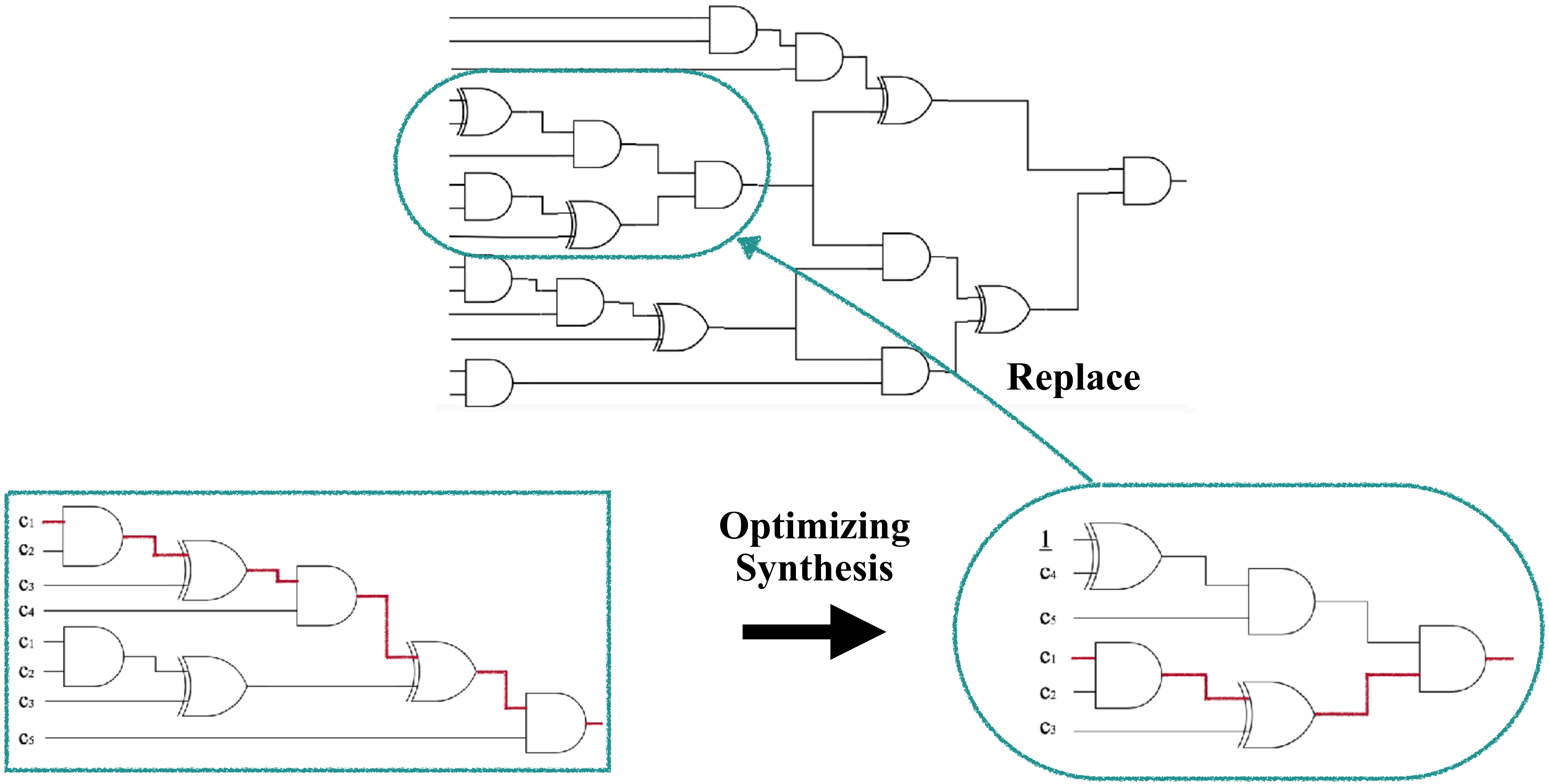
Solution1 : Synthesis via Localization



Solution1 : Synthesis via Localization



Solution1 : Synthesis via Localization

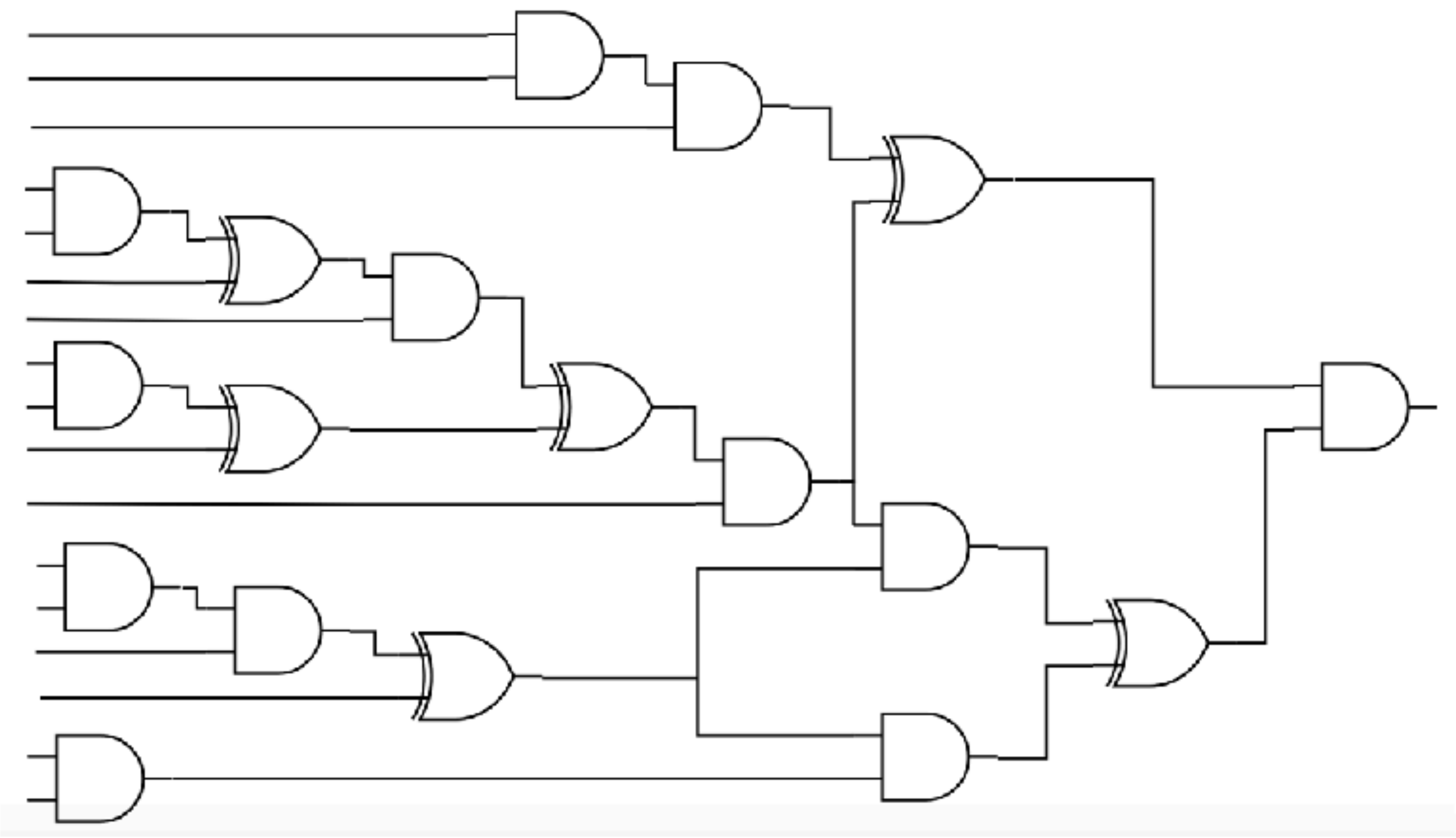


Solution 2: Learning Successful Synthesis Patterns

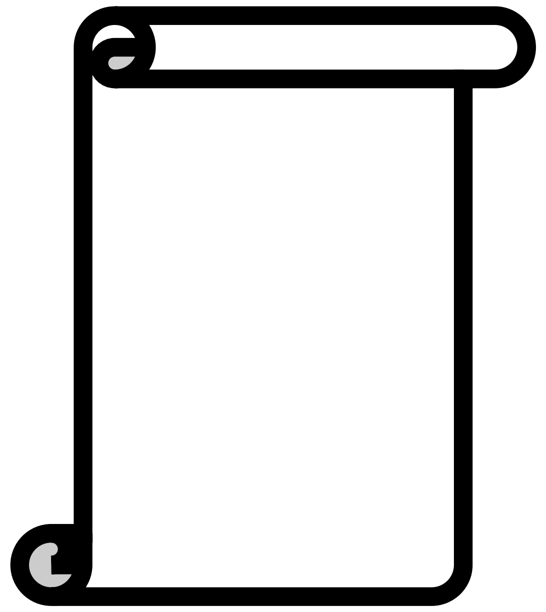
- Offline Learning
  - Collect successful synthesis patterns
- Online Optimization
  - Applying the patterns by term rewriting

Offline Learning to Collect Opt. Patterns

Training HE Applications

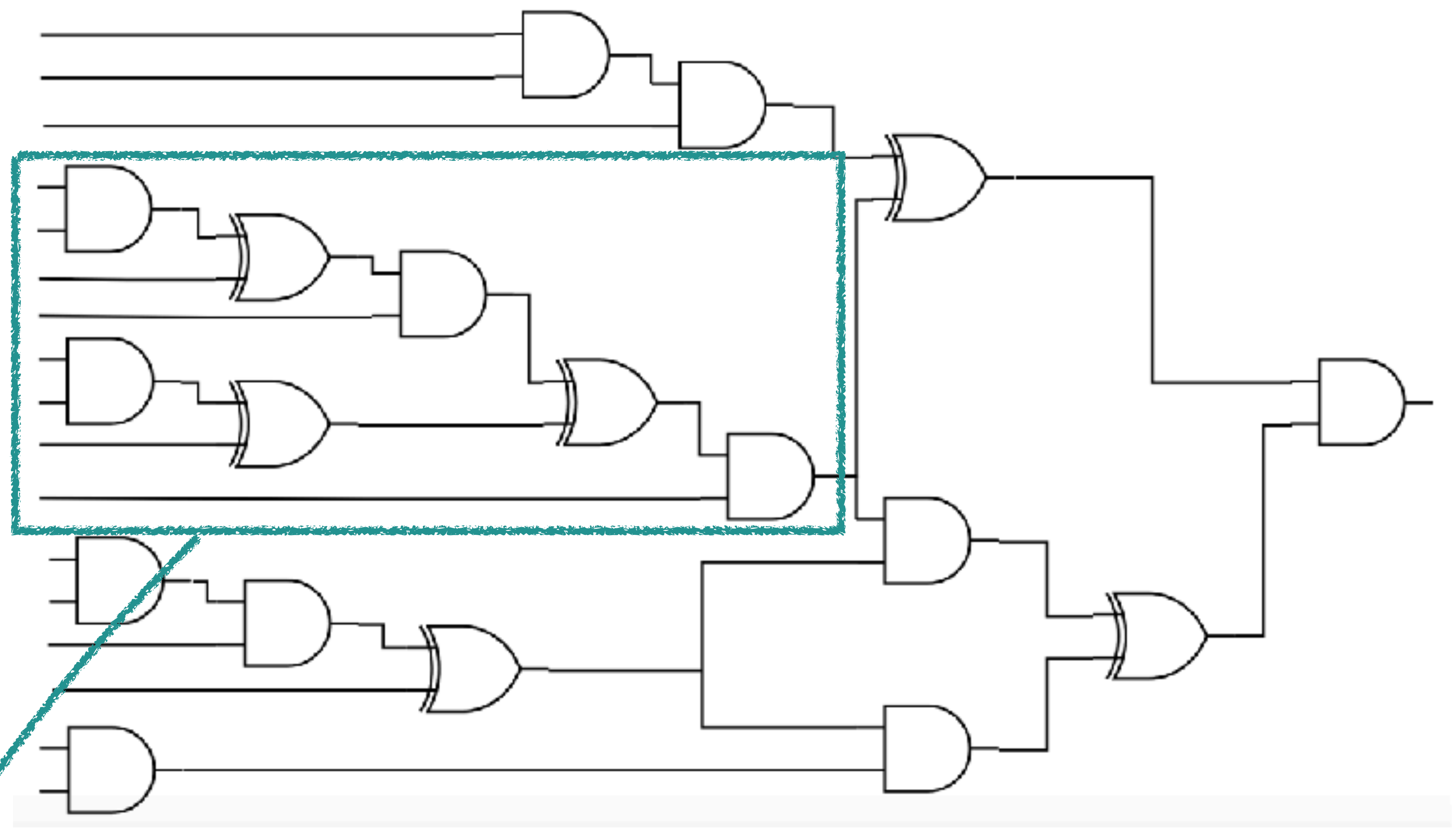


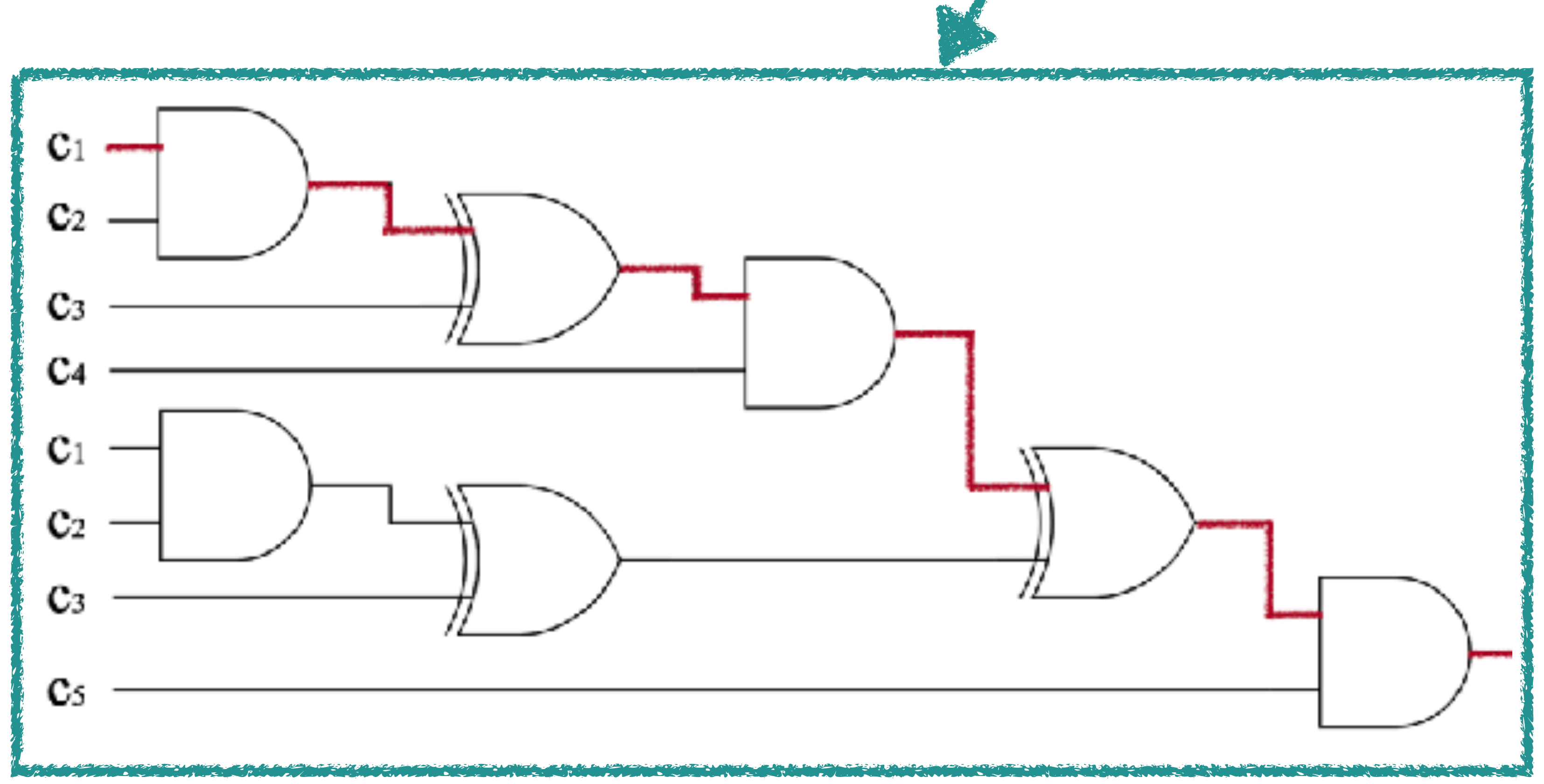
Collected Opt. Patterns



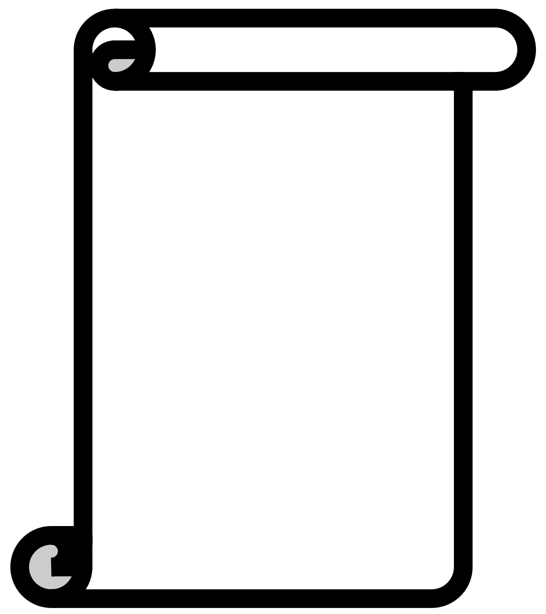
Offline Learning to Collect Opt. Patterns

Training
HE Applications

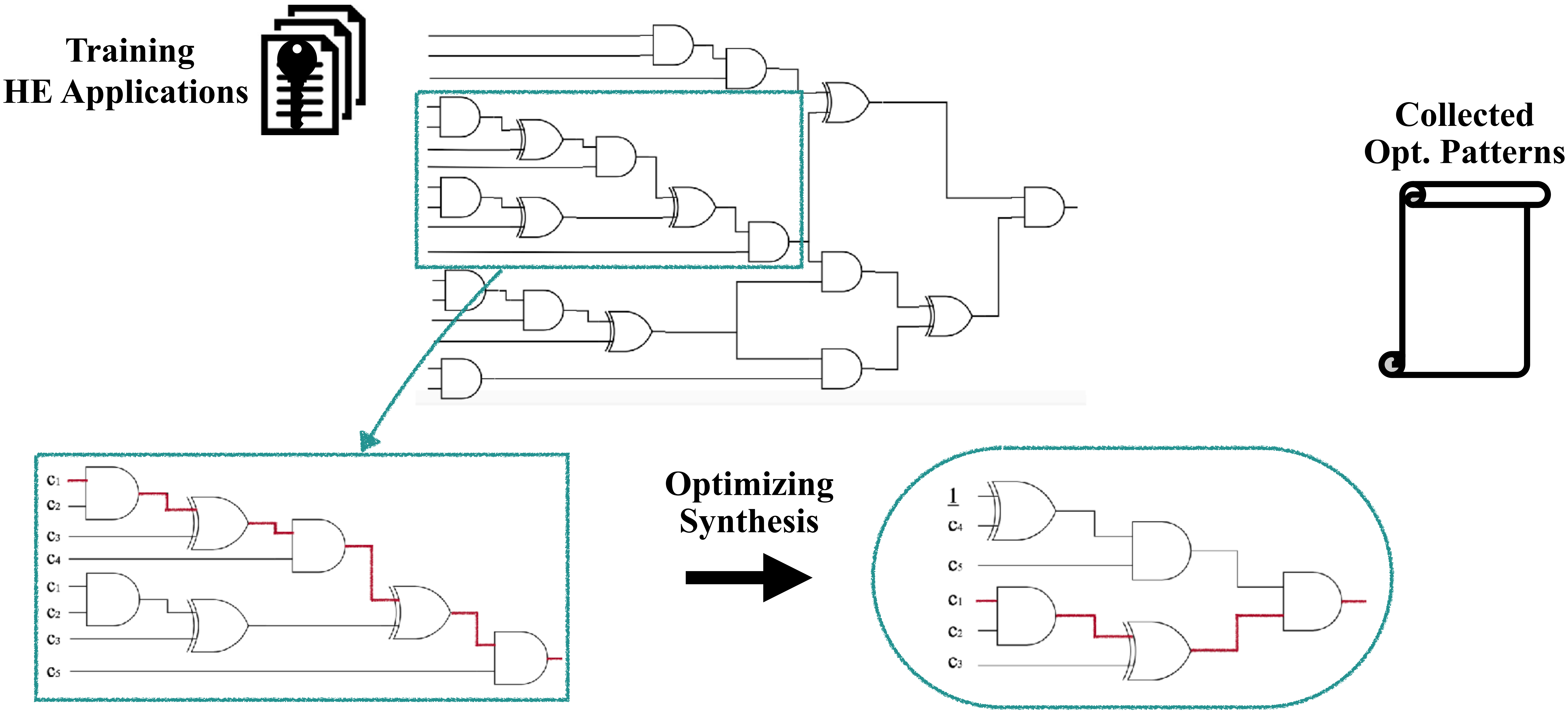




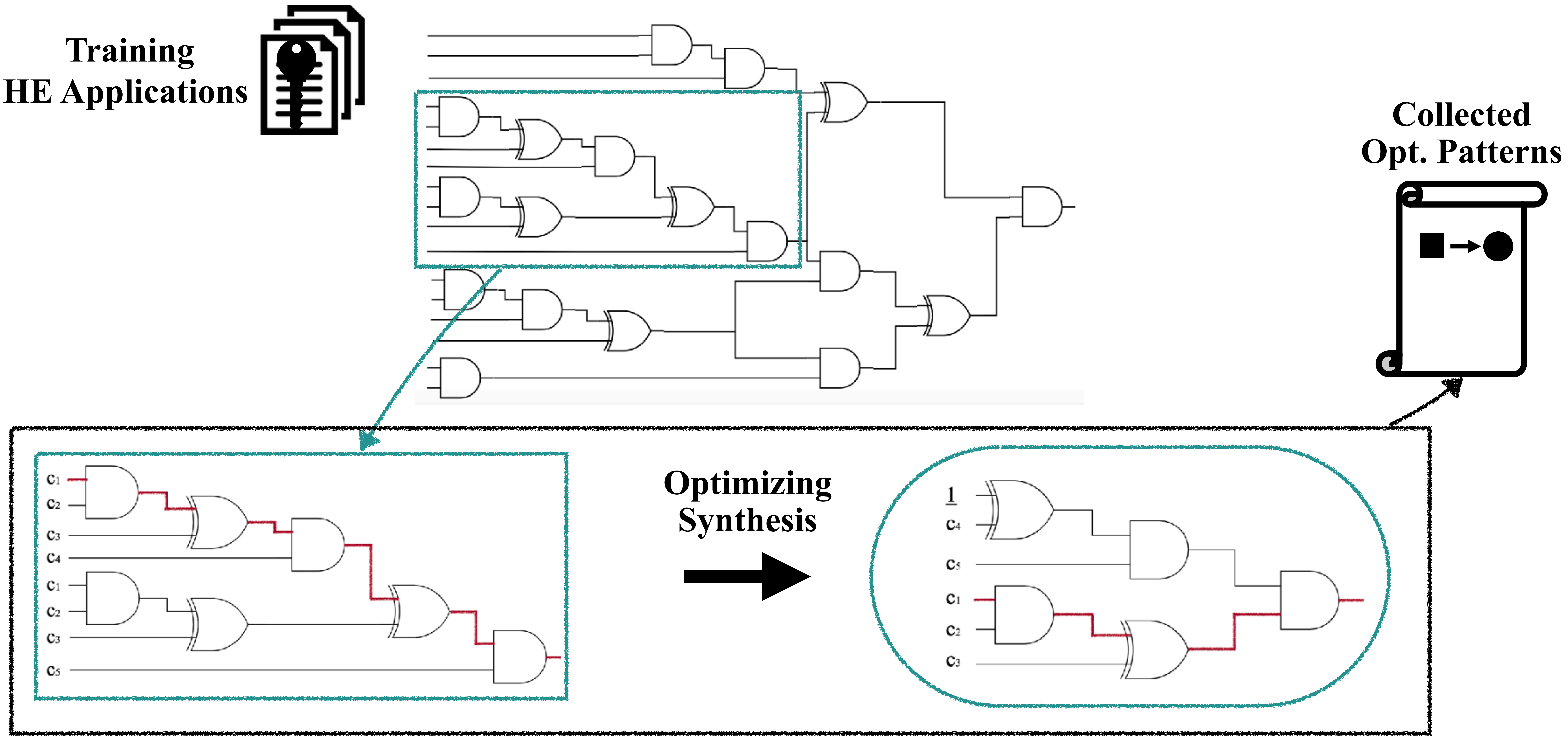
Collected
Opt. Patterns



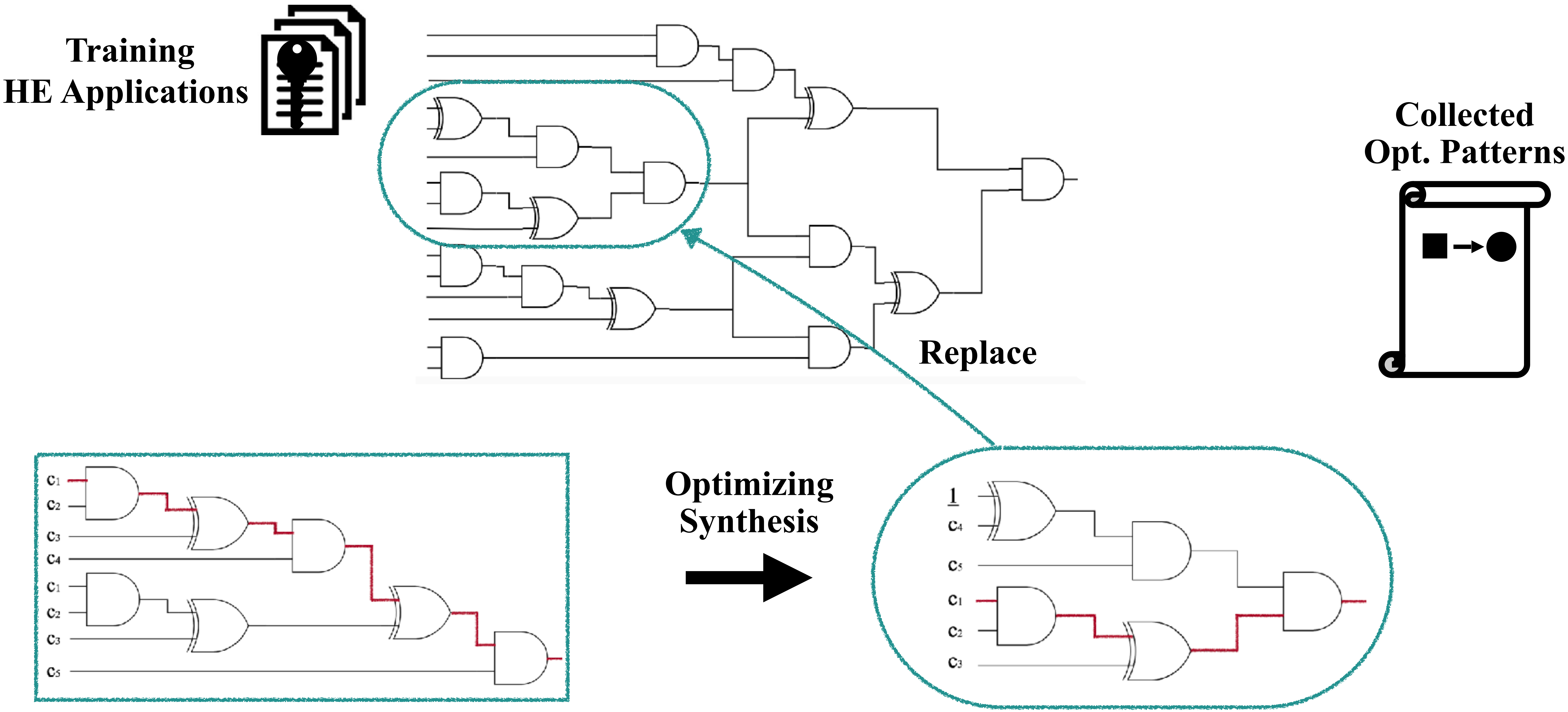
Offline Learning to Collect Opt. Patterns



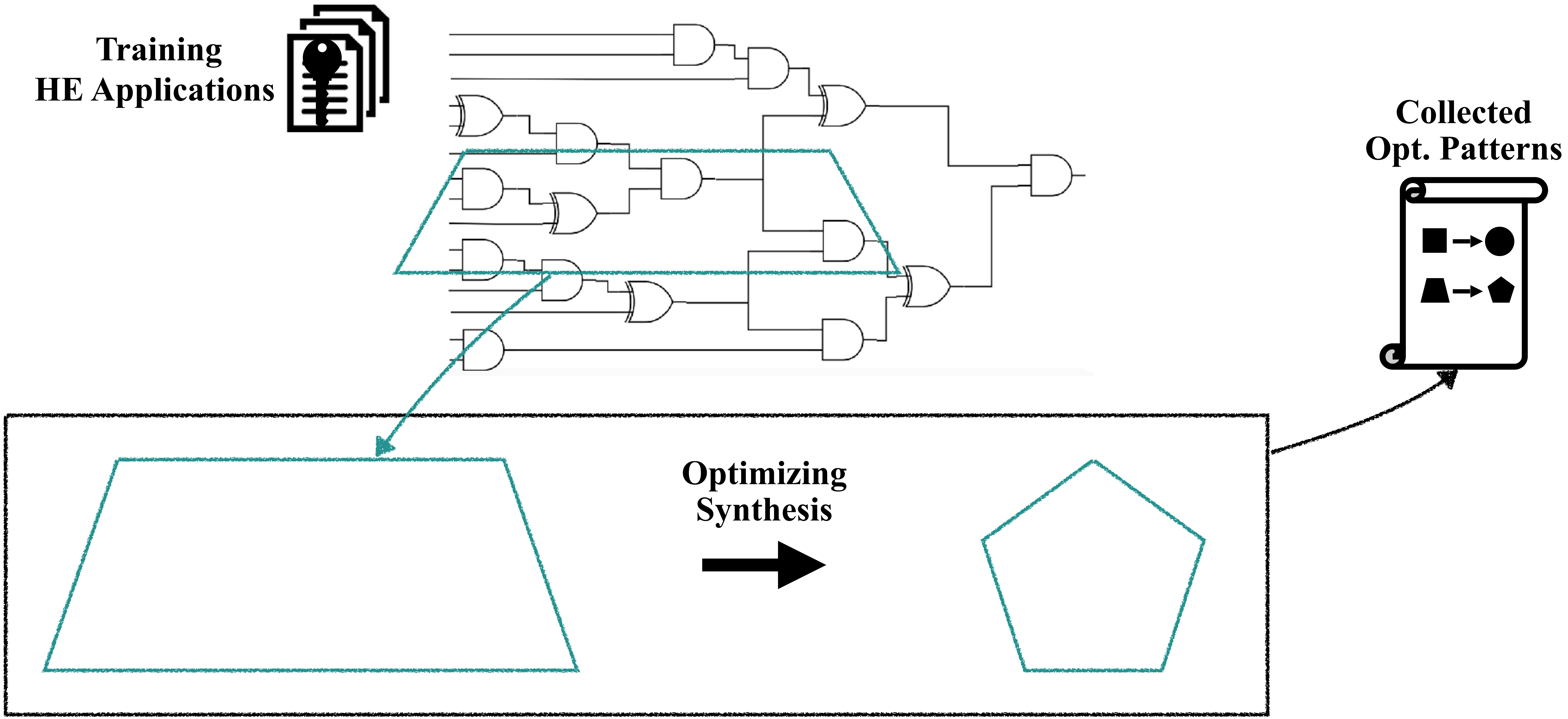
Offline Learning to Collect Opt. Patterns



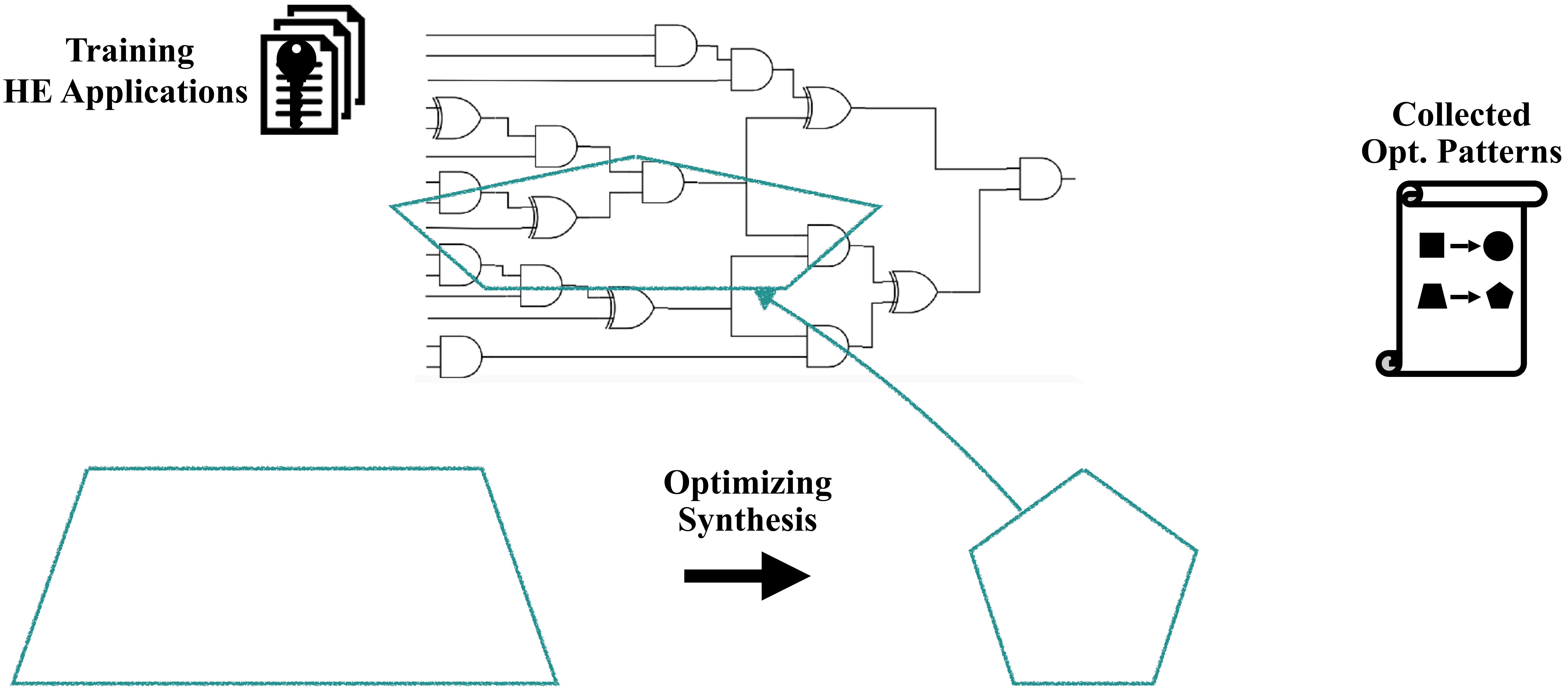
Offline Learning to Collect Opt. Patterns



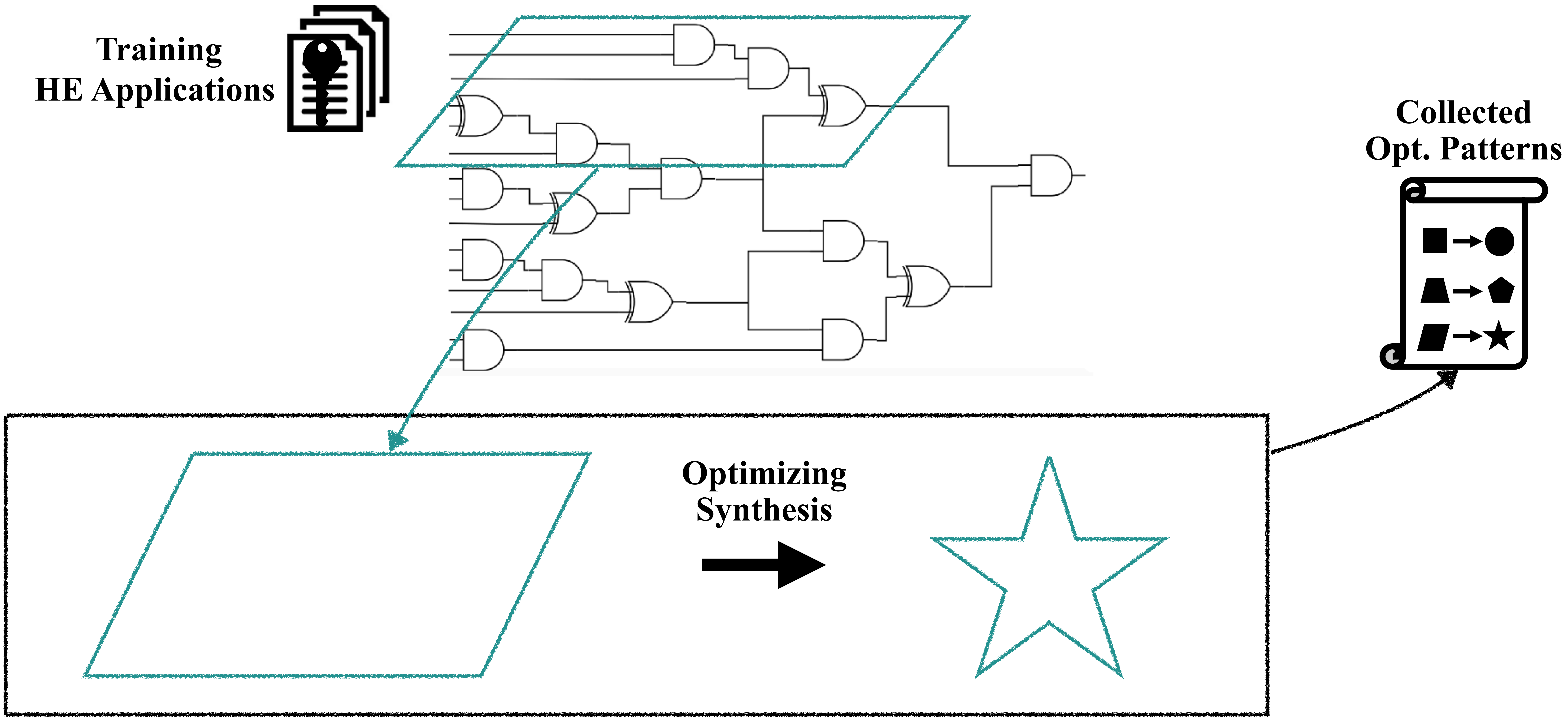
Offline Learning to Collect Opt. Patterns



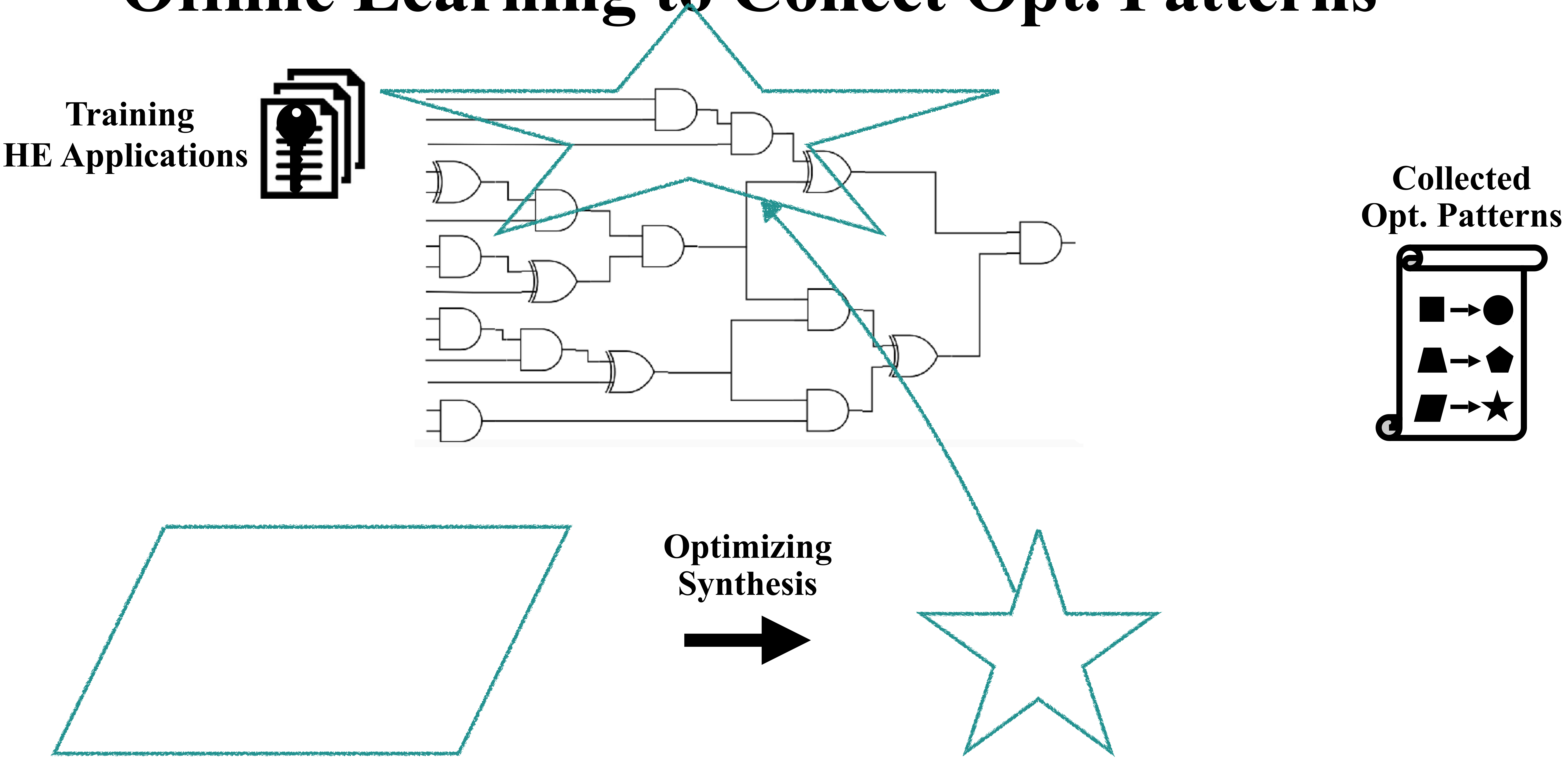
Offline Learning to Collect Opt. Patterns



Offline Learning to Collect Opt. Patterns

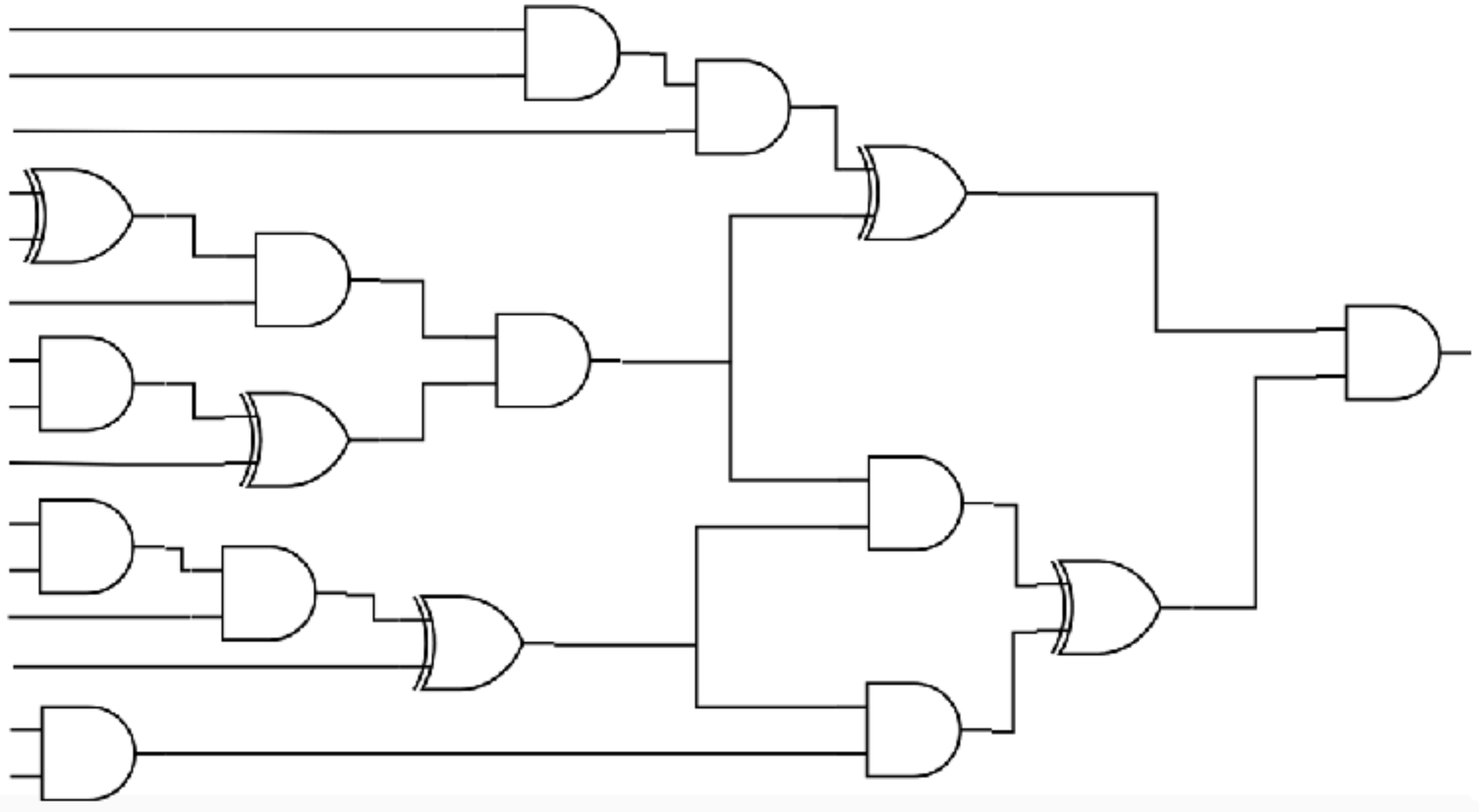


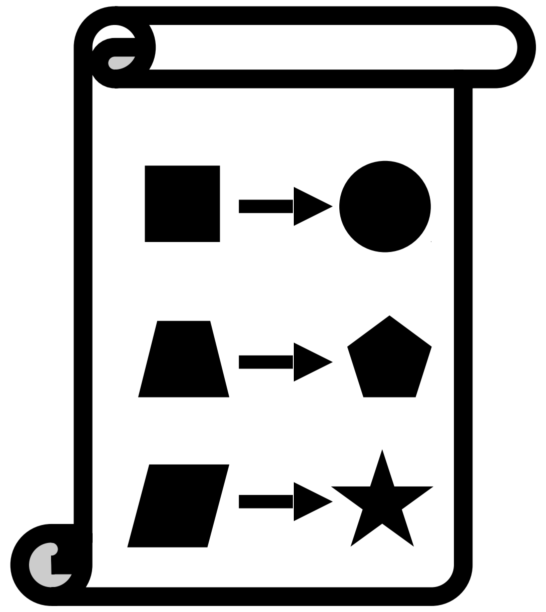
Offline Learning to Collect Opt. Patterns



Offline Learning to Collect Opt. Patterns

Training
HE Applications

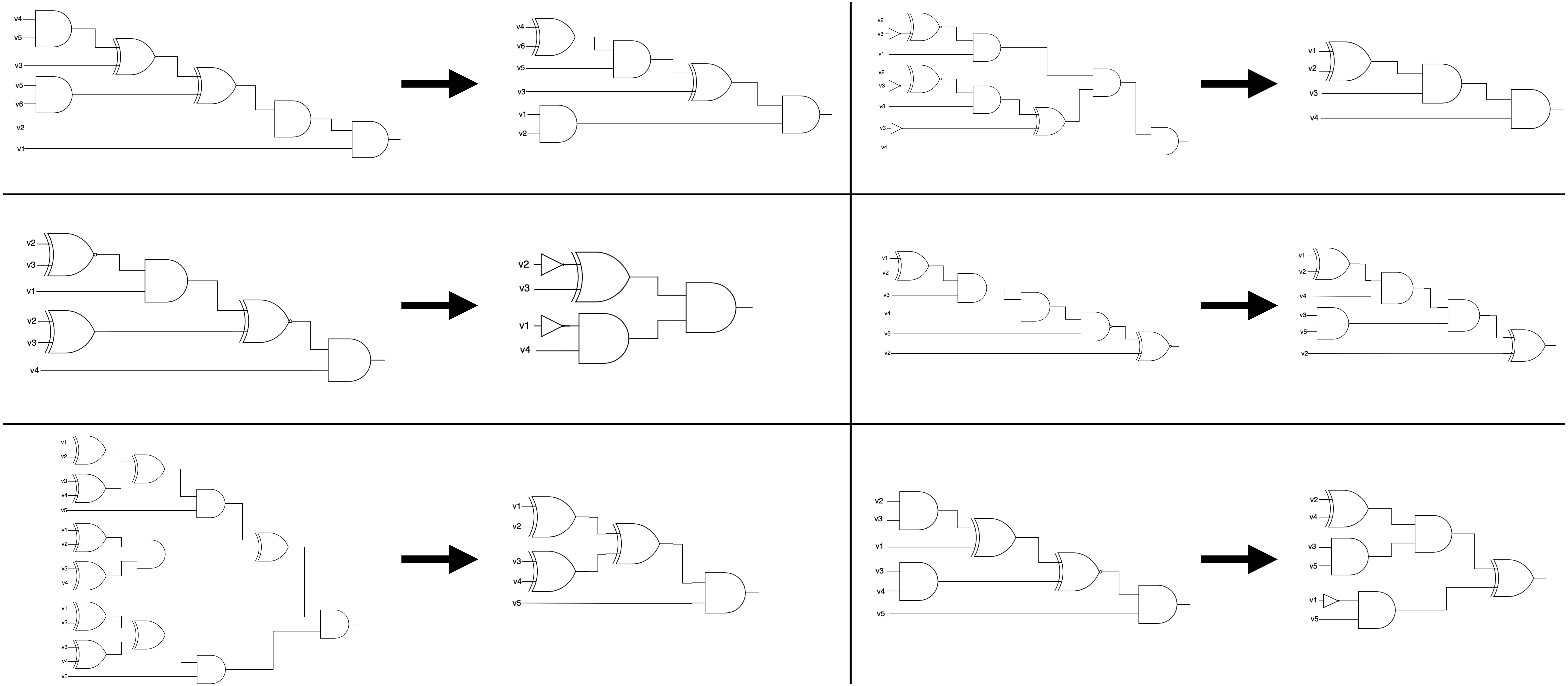




Collected
Opt. Patterns

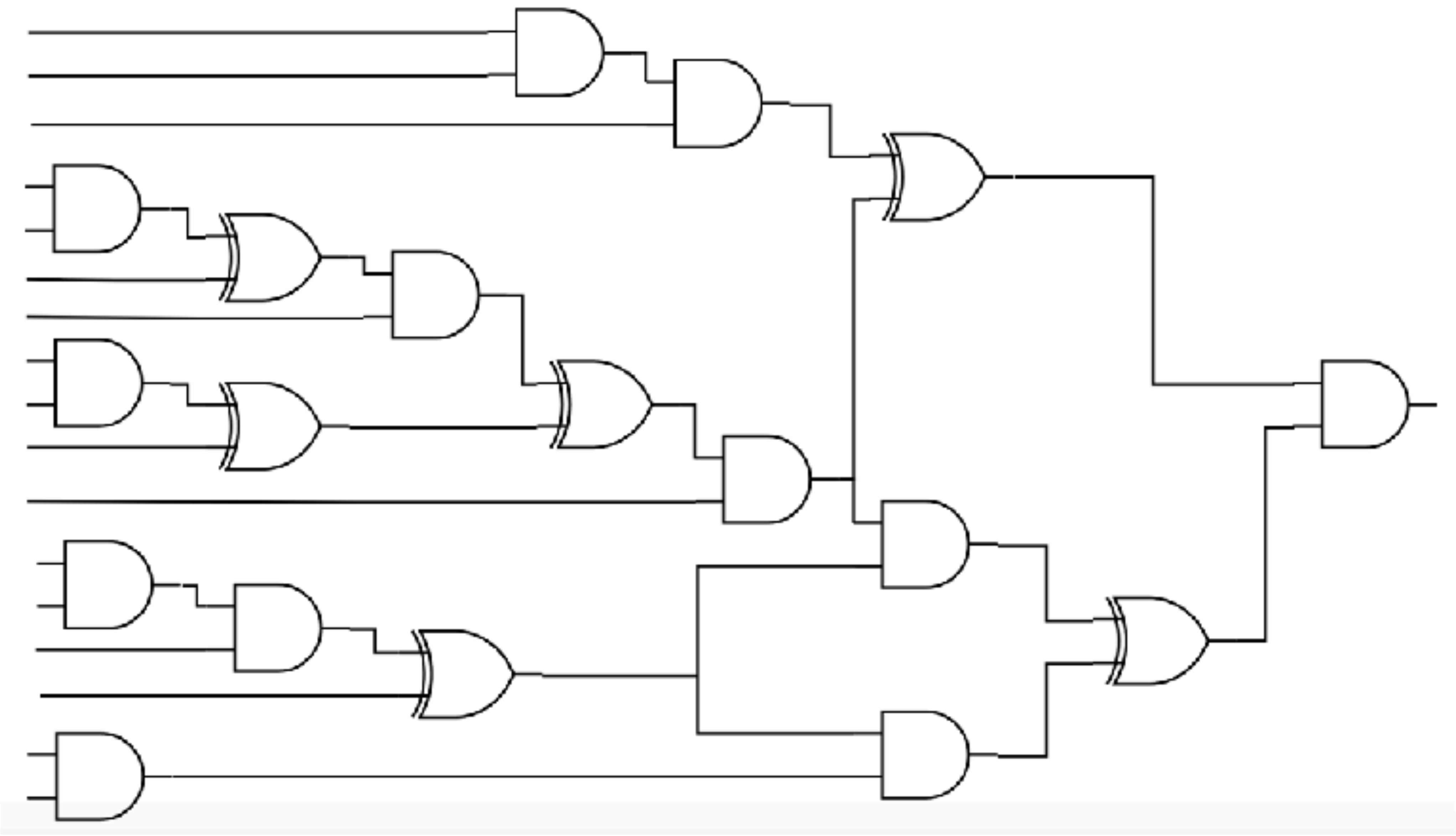
186 Opt. patterns

Learned Optimization Patterns : examples

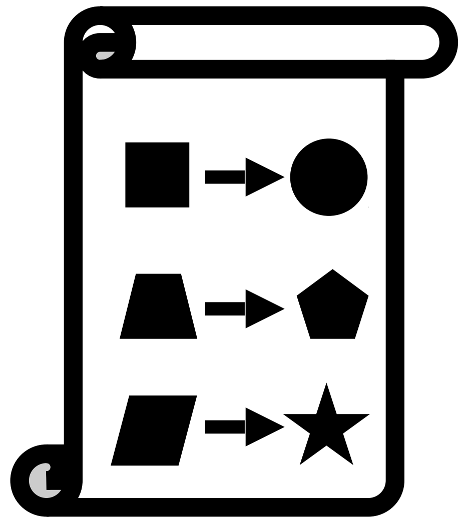


Online Rule-based Optimization

Input
HE application

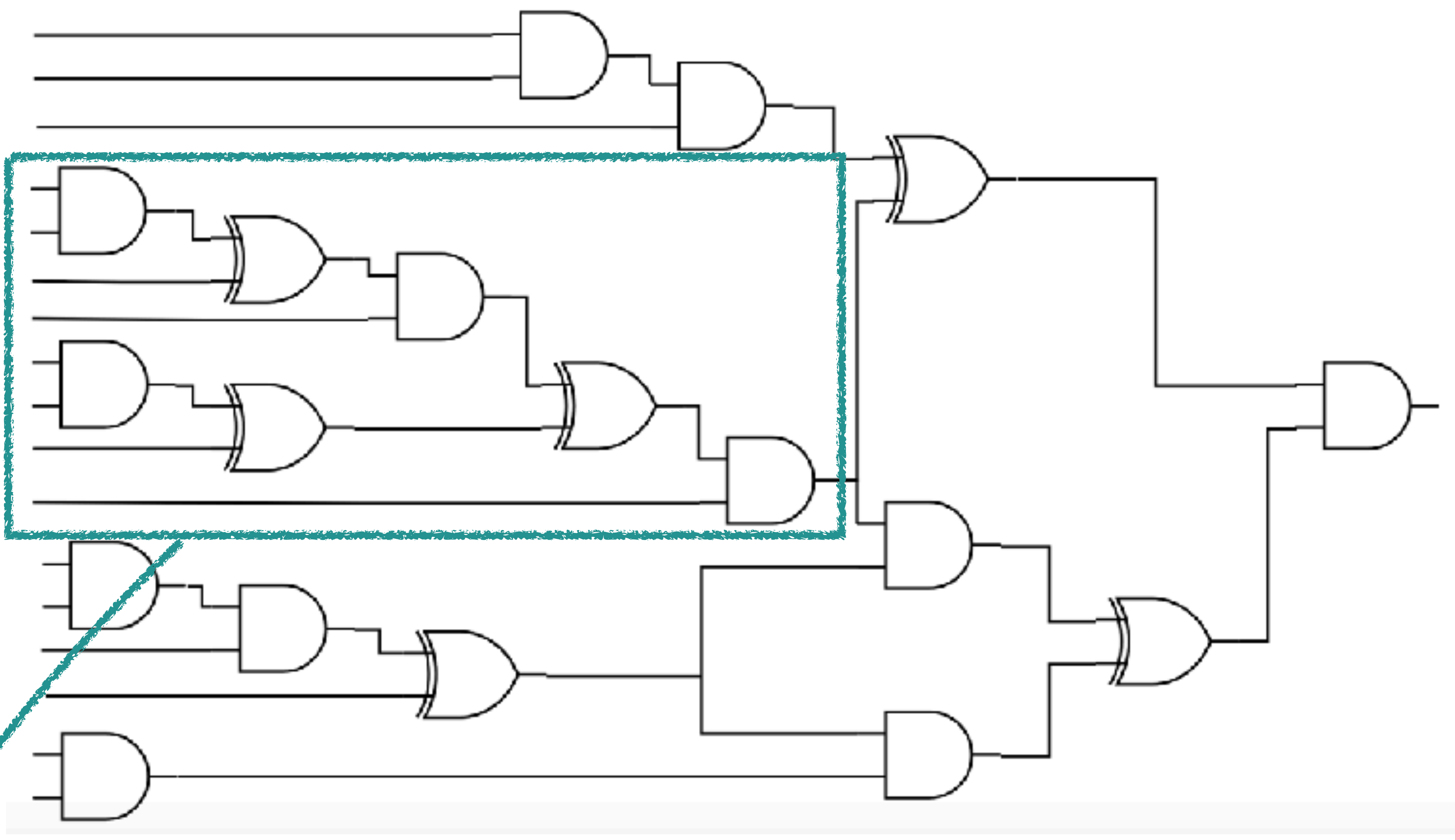


Learned
Opt. Patterns

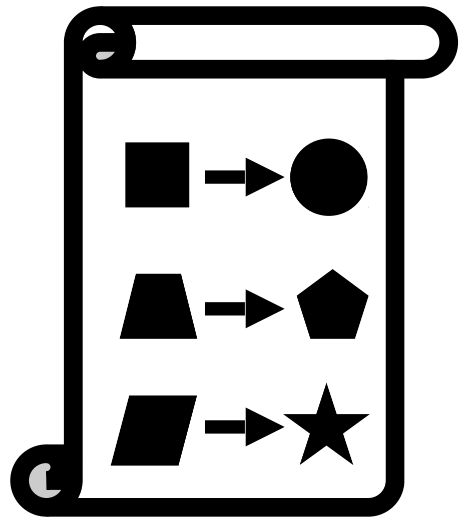


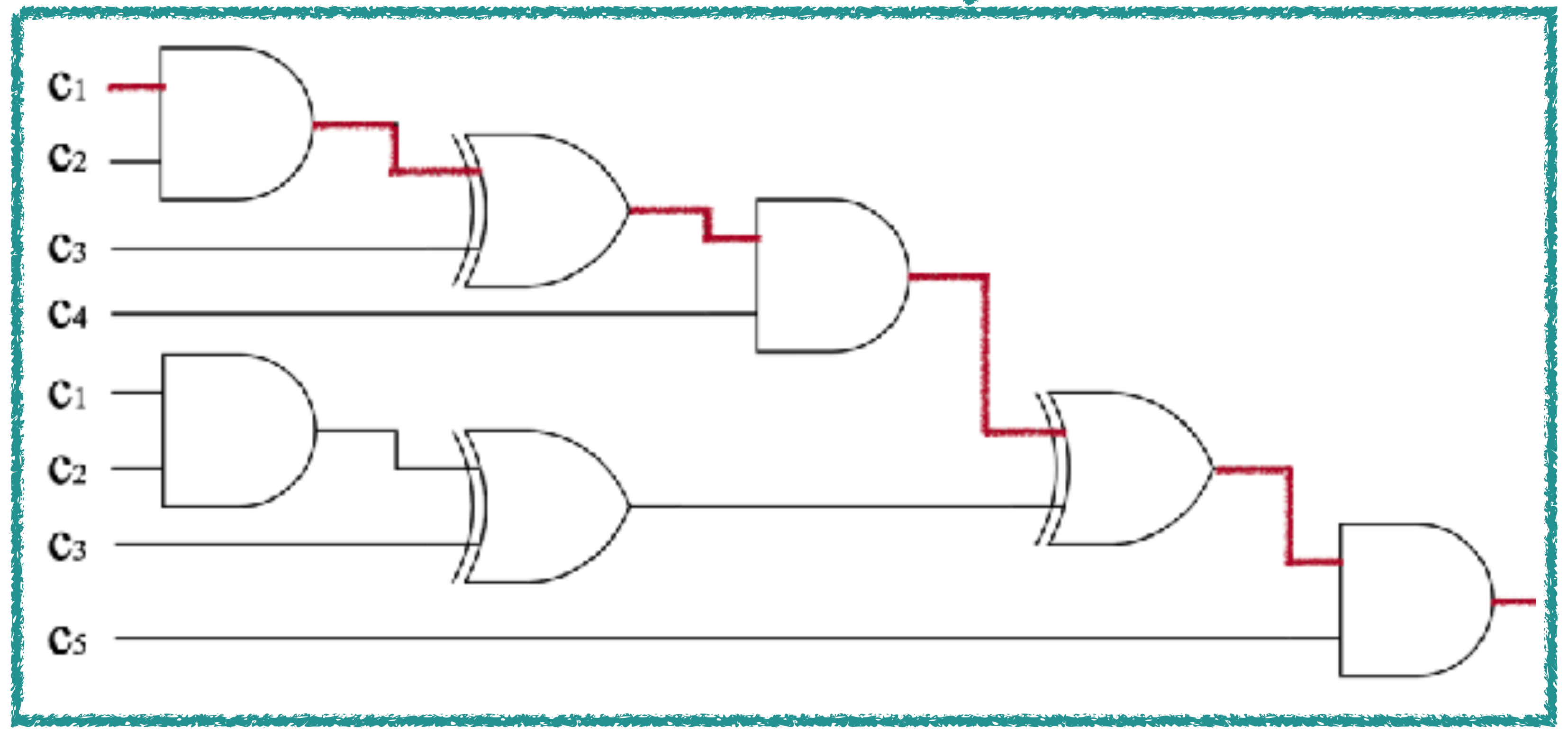
Online Rule-based Optimization

Input
HE application

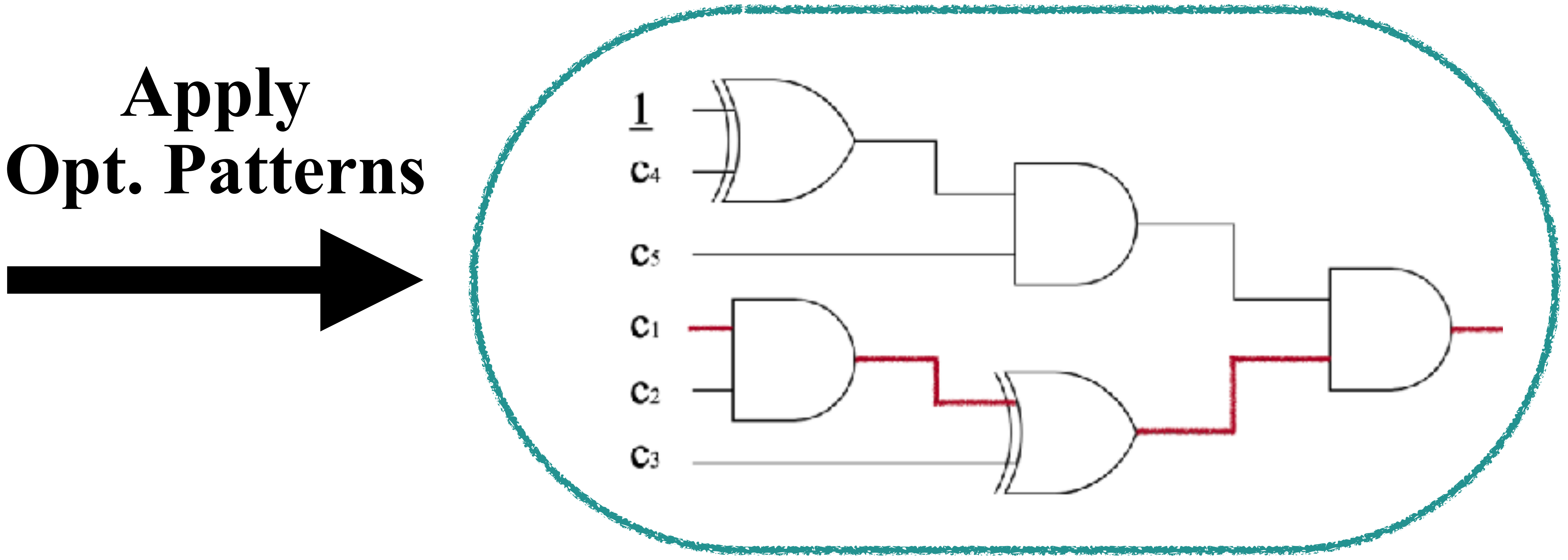


Learned
Opt. Patterns

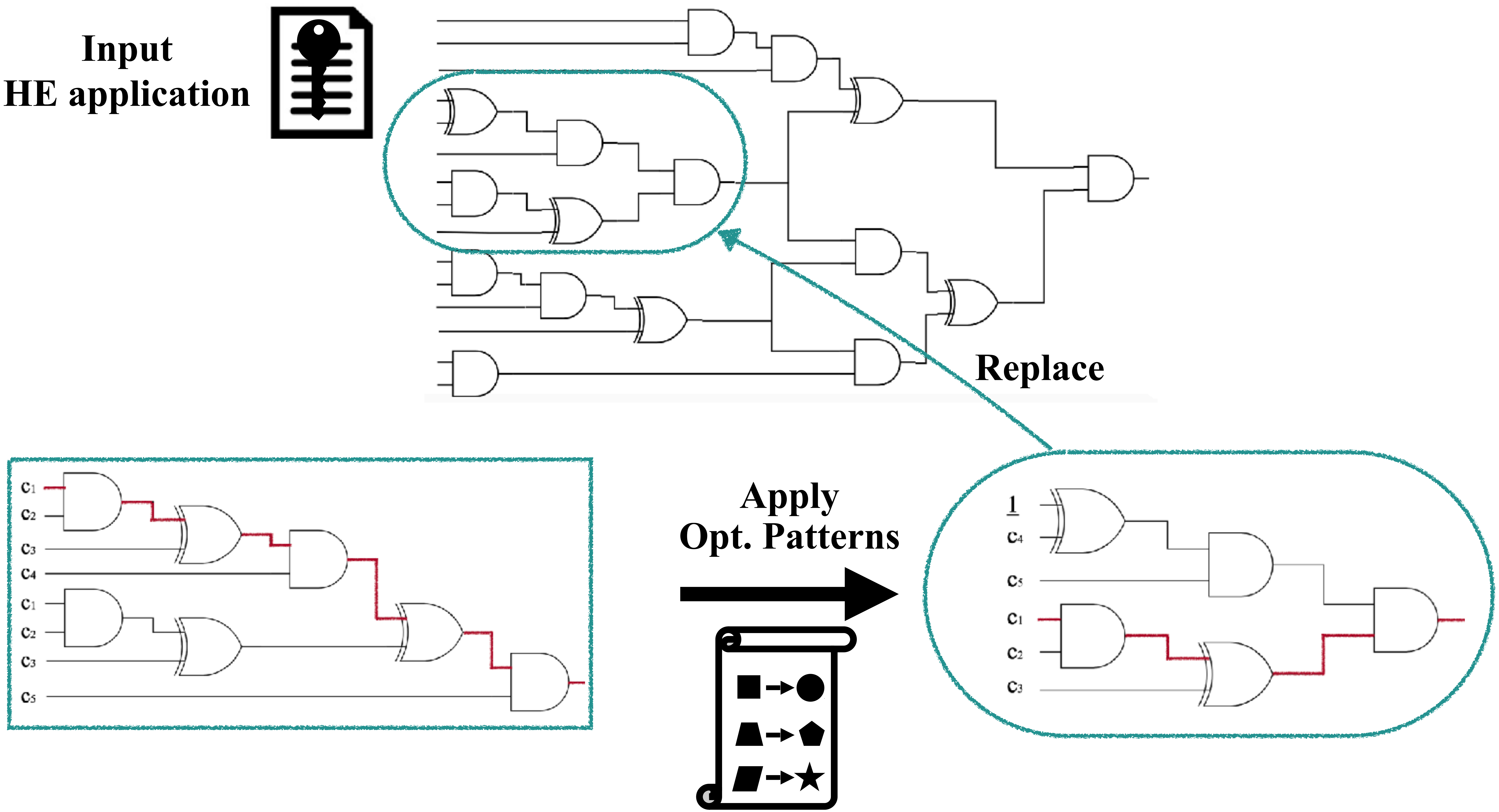




Apply
Opt. Patterns

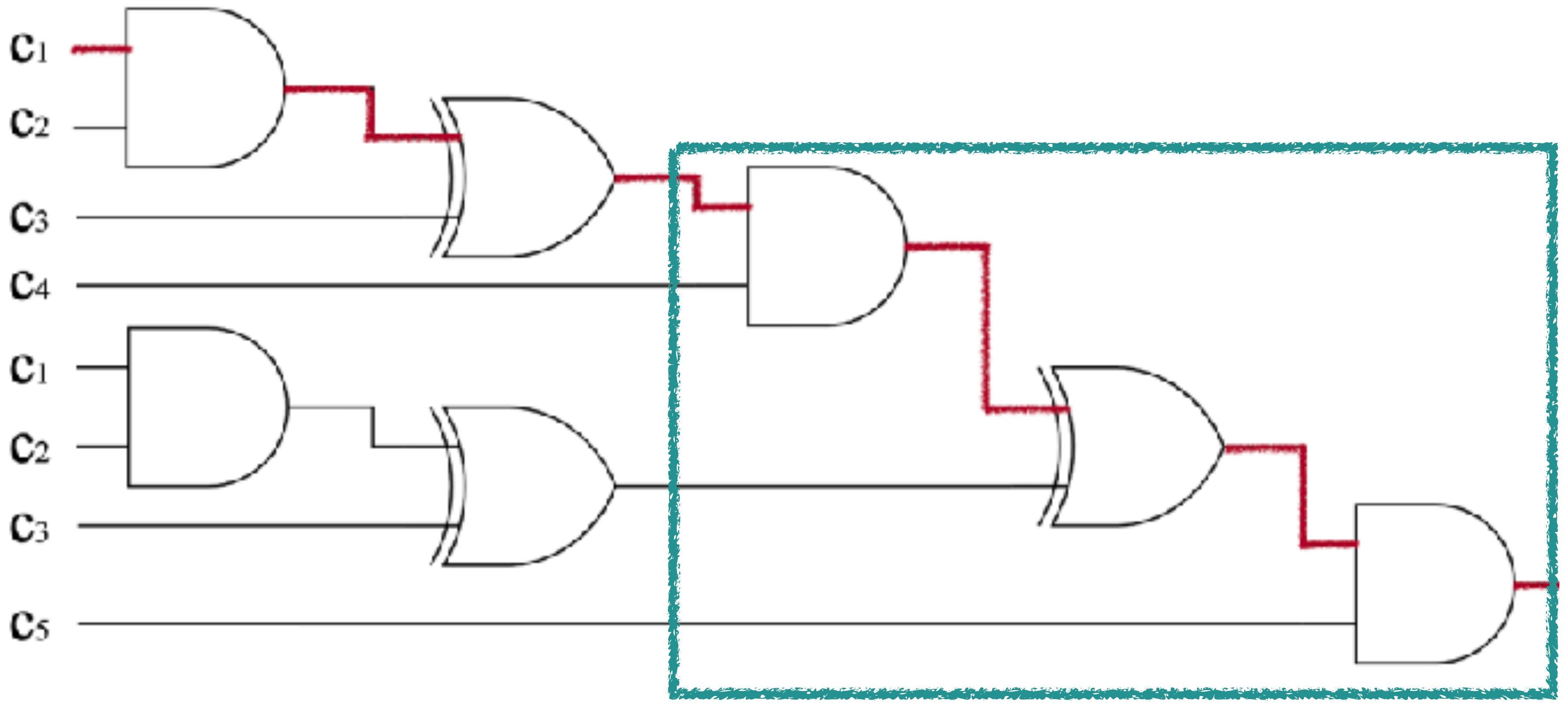


Online Rule-based Optimization

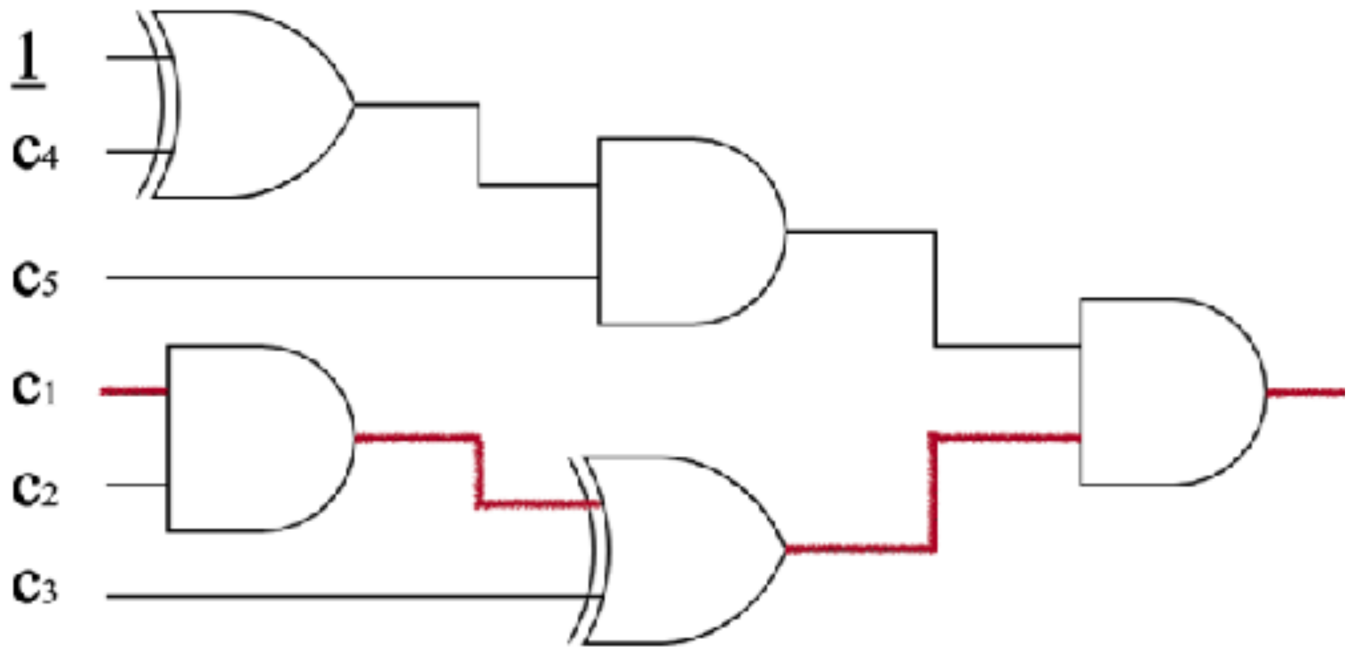


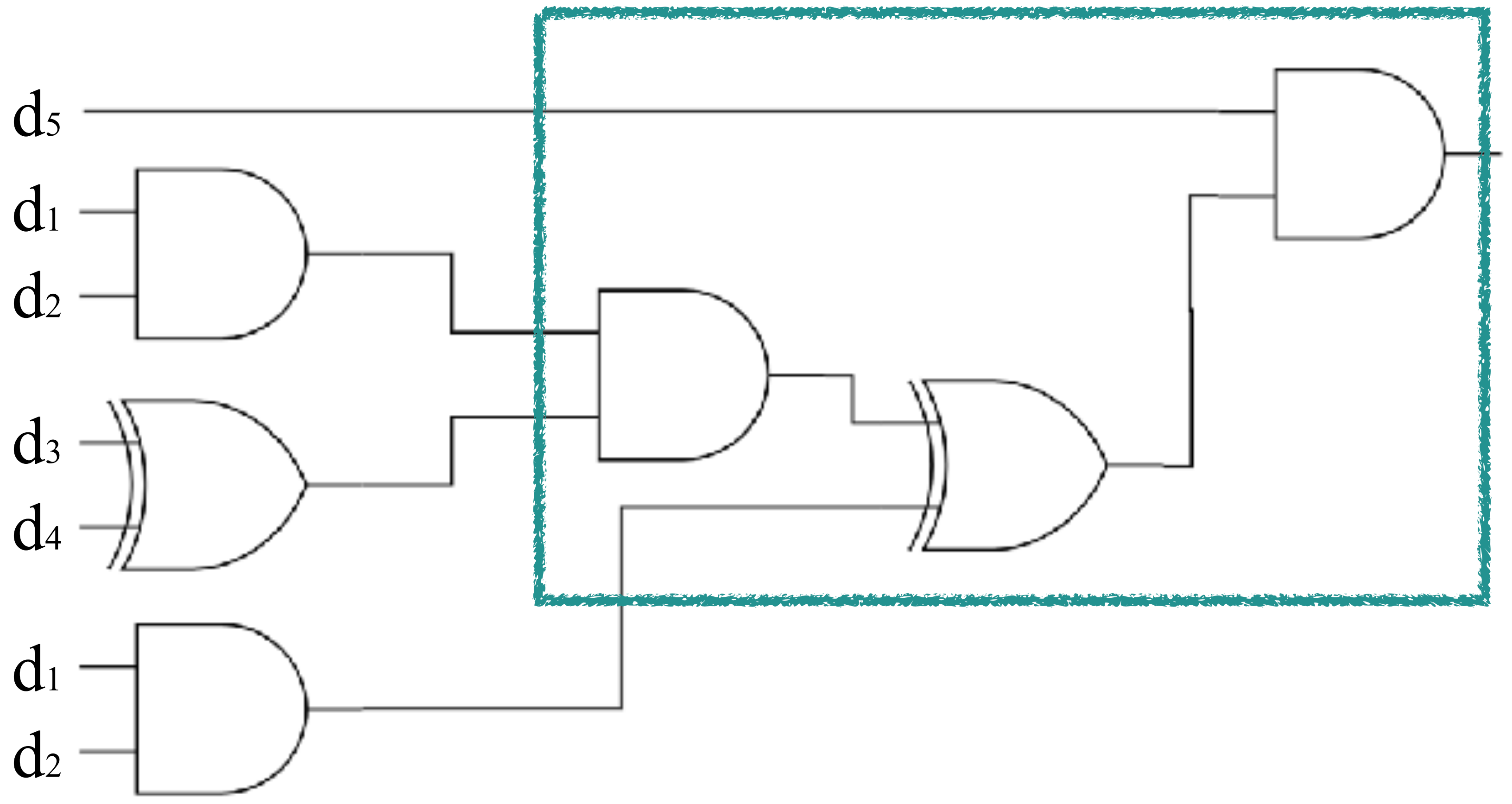
Applying Learned Optimization Patterns (1/2)

Syntactic Matching is Not Effective



Learned Opt. Patterns



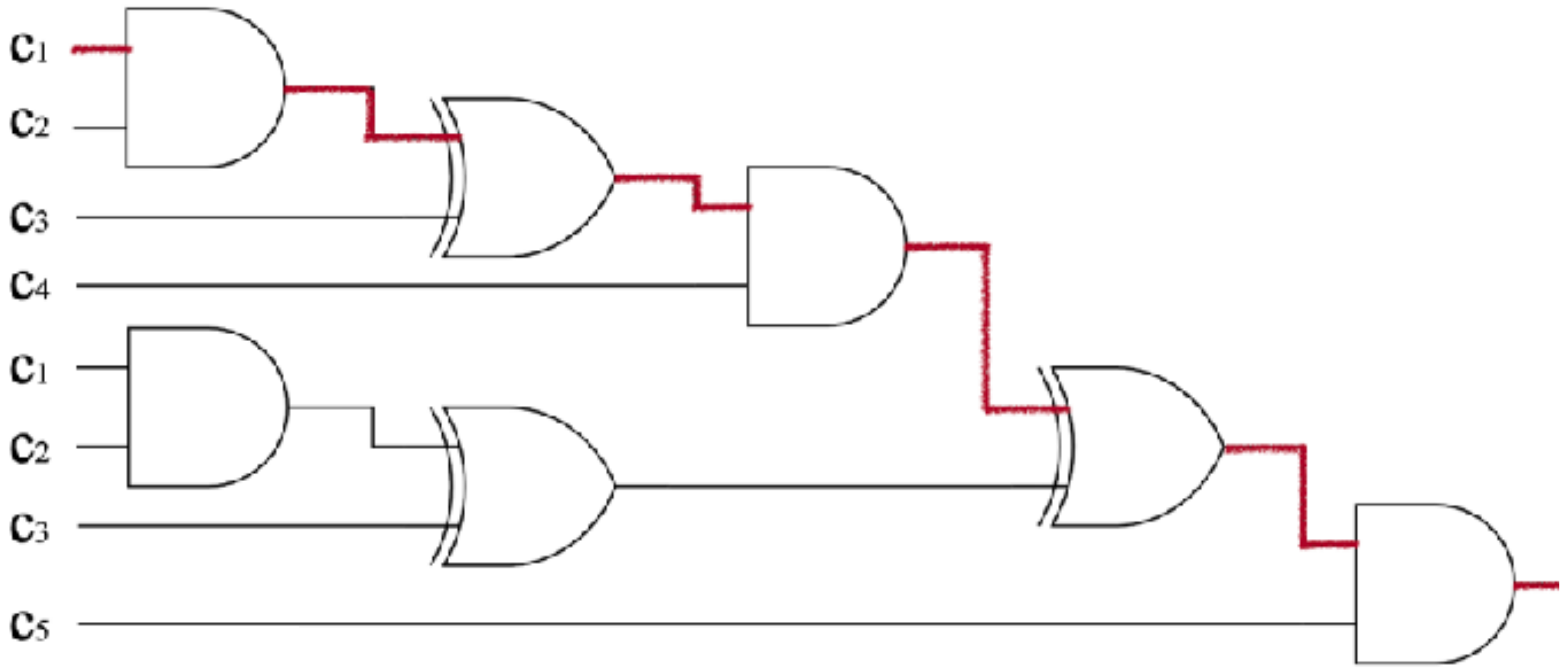


New Input Circuit Optimization

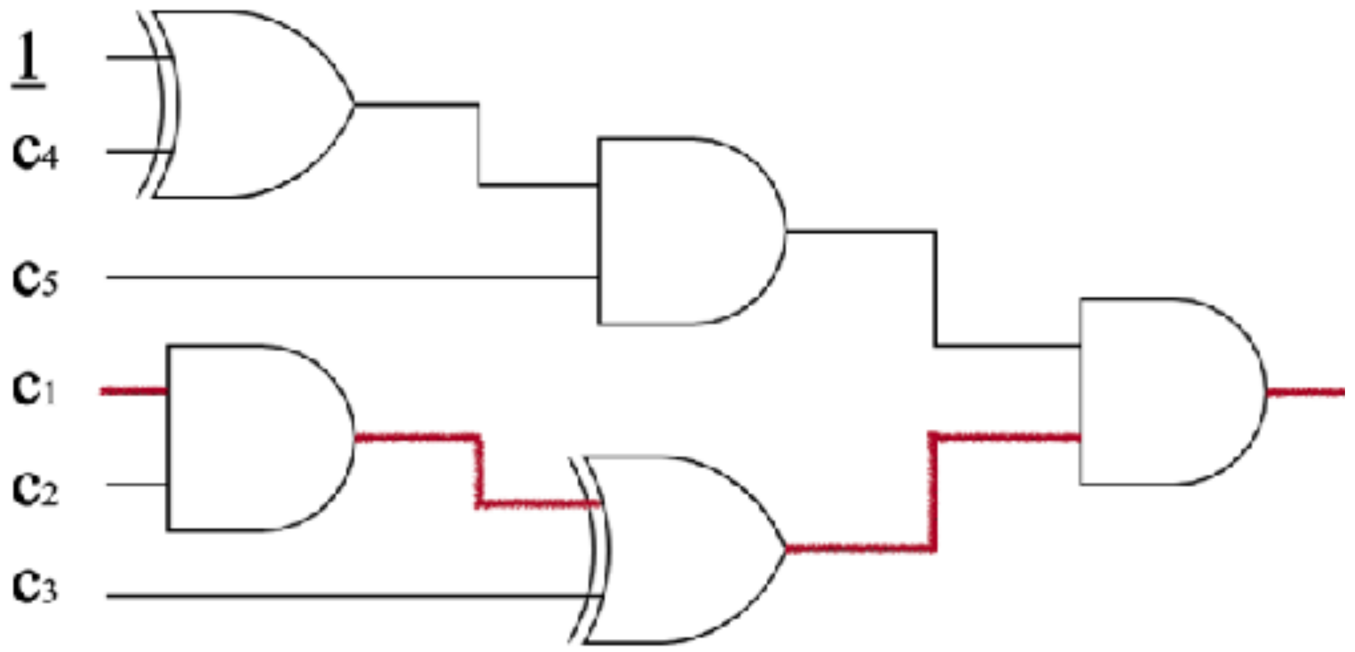
?

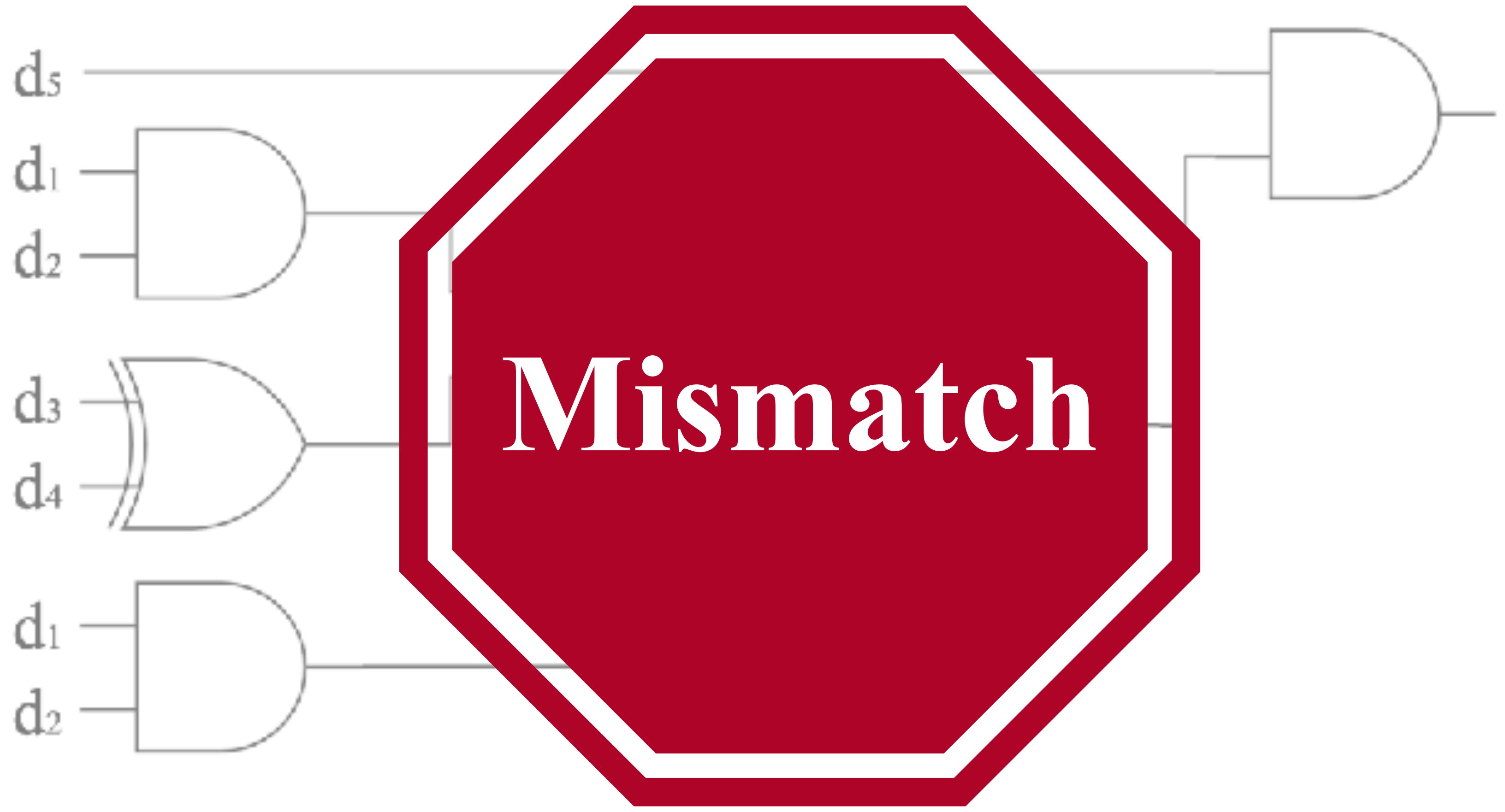
Applying Learned Optimization Patterns (1/2)

Syntactic Matching is Not Effective



Learned
Opt. Patterns



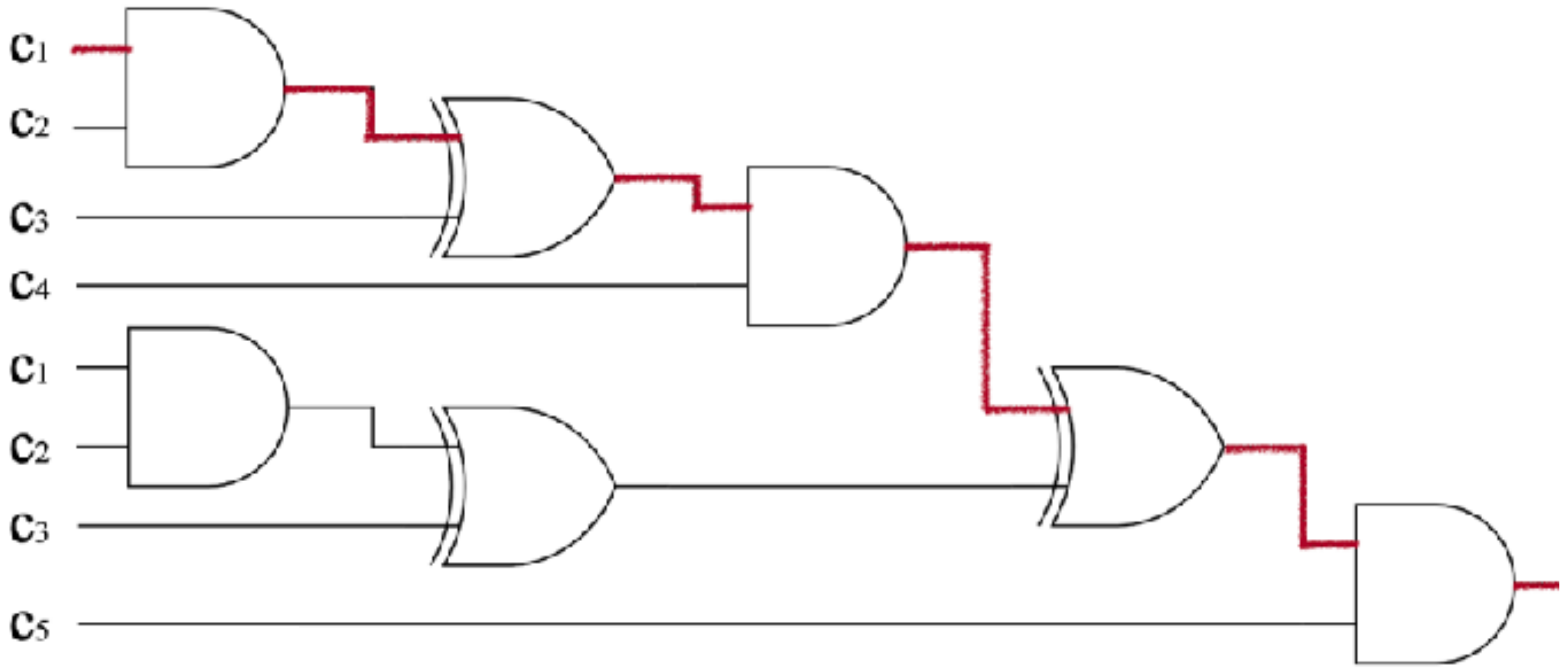


New Input Circuit
Optimization

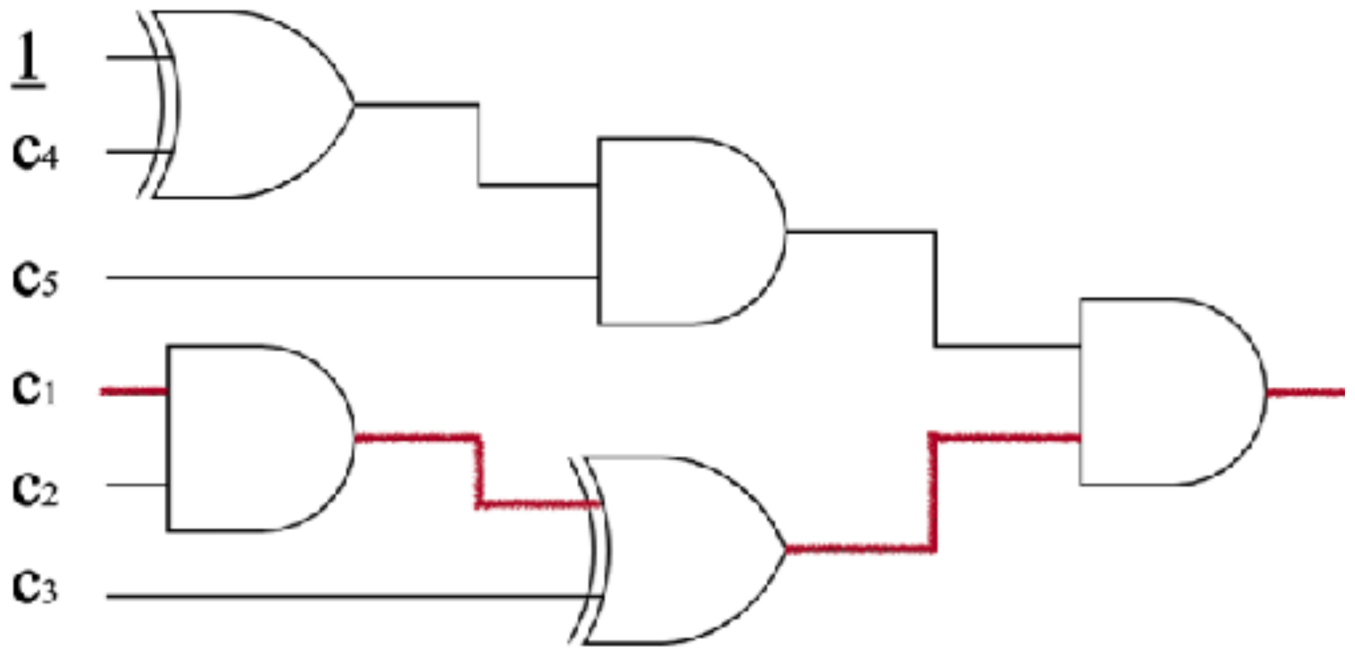
?

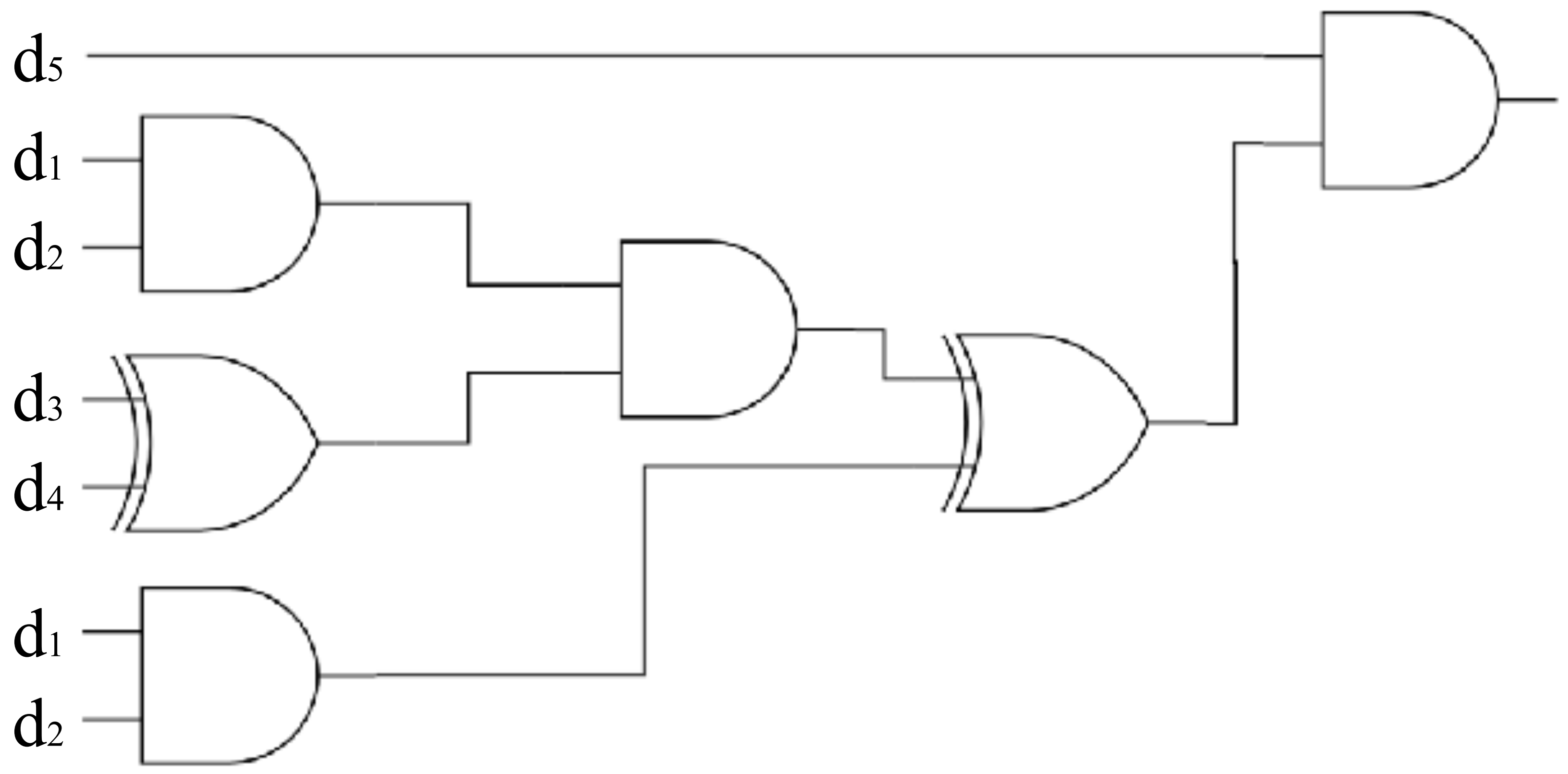
Applying Learned Optimization Patterns (2/2)

Normalization + Equational Matching



Learned Opt. Patterns



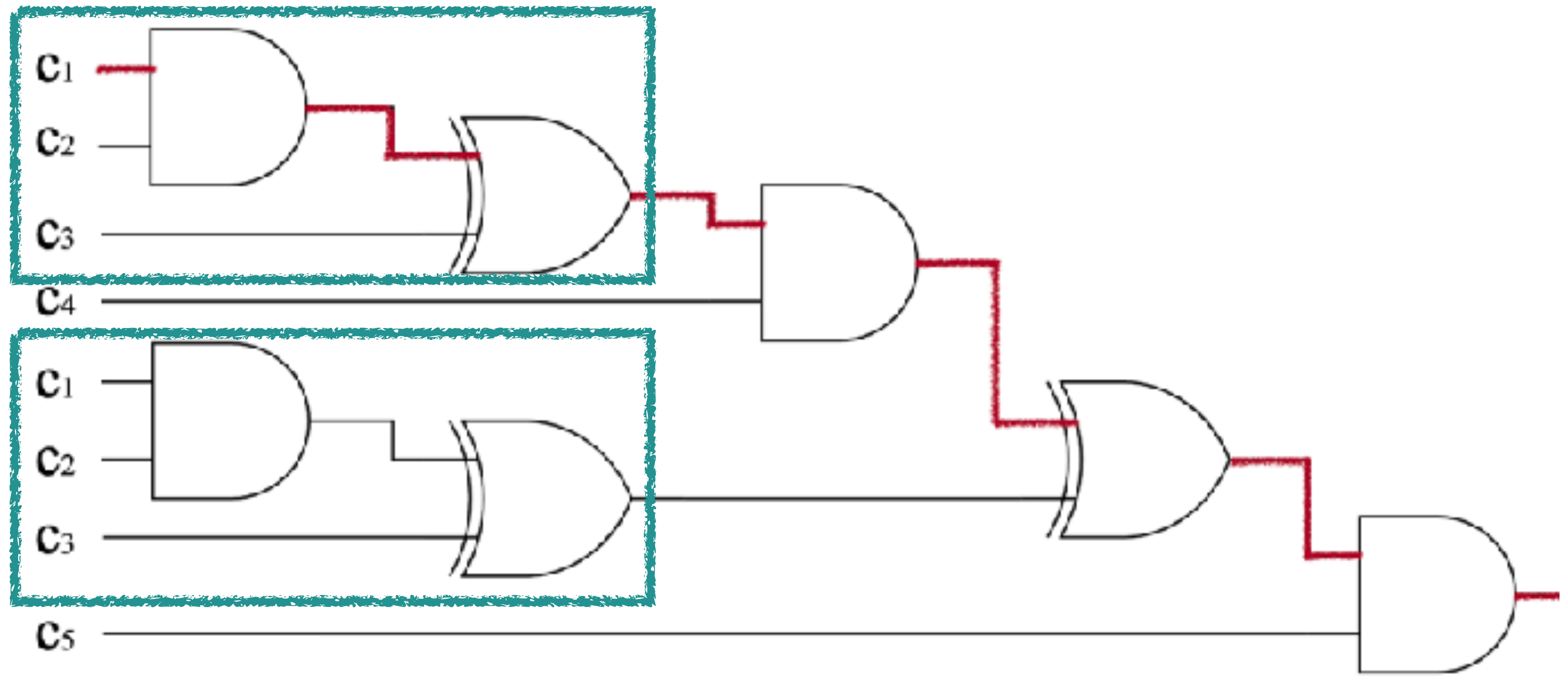


New Input Circuit Optimization

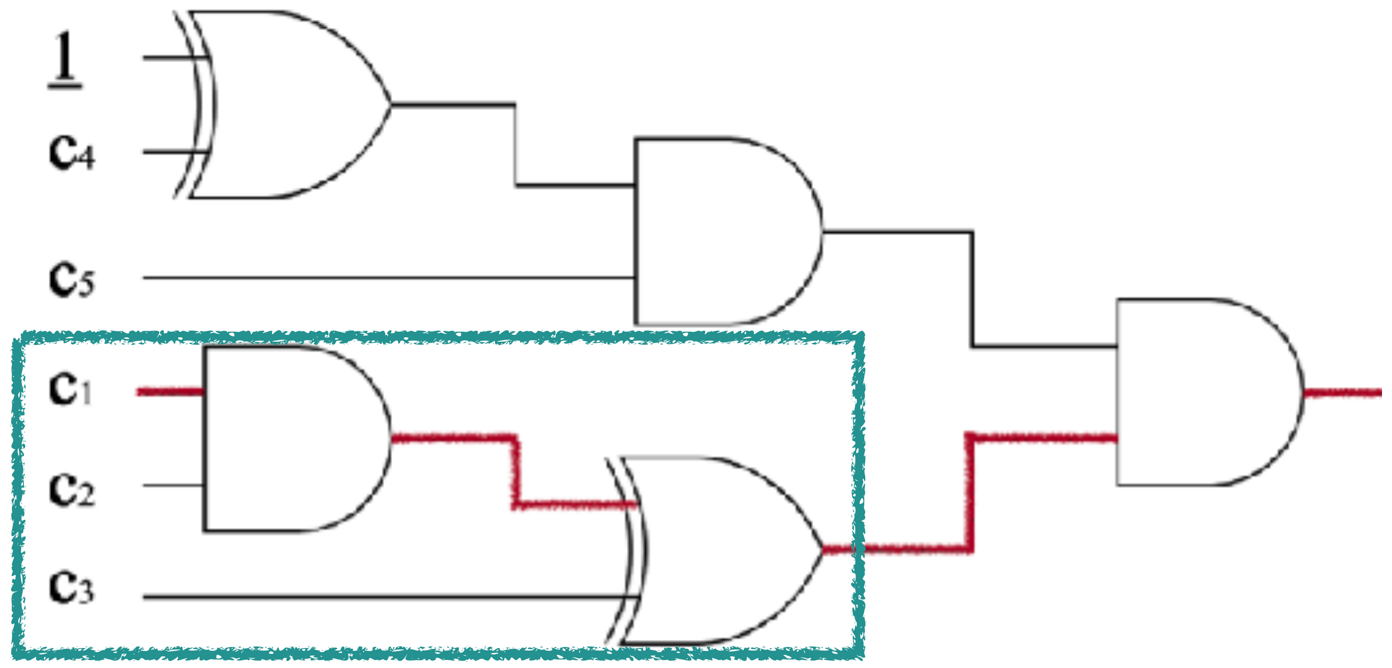
?

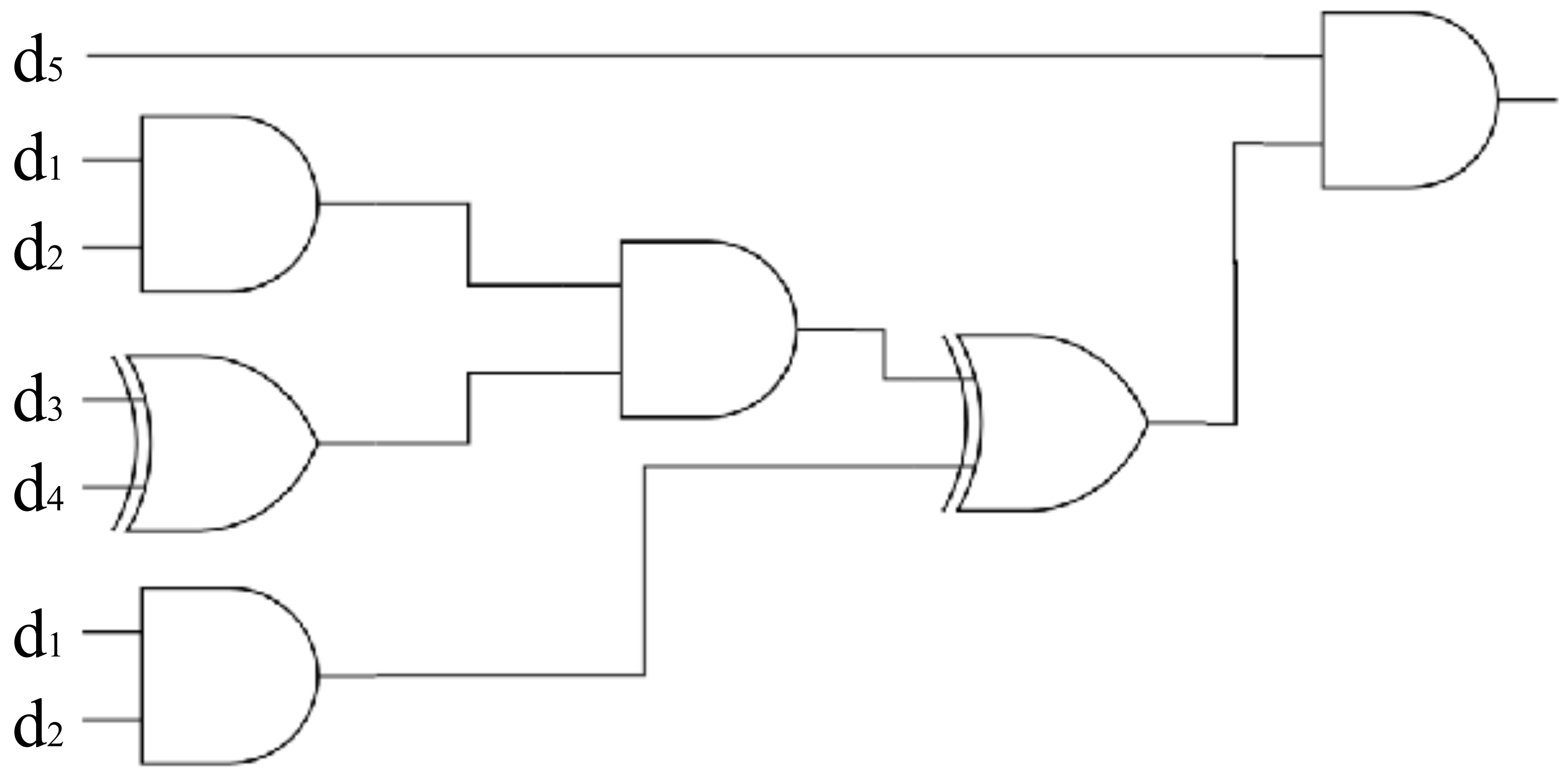
Applying Learned Optimization Patterns (2/2)

Normalization + Equational Matching



Learned Opt. Patterns



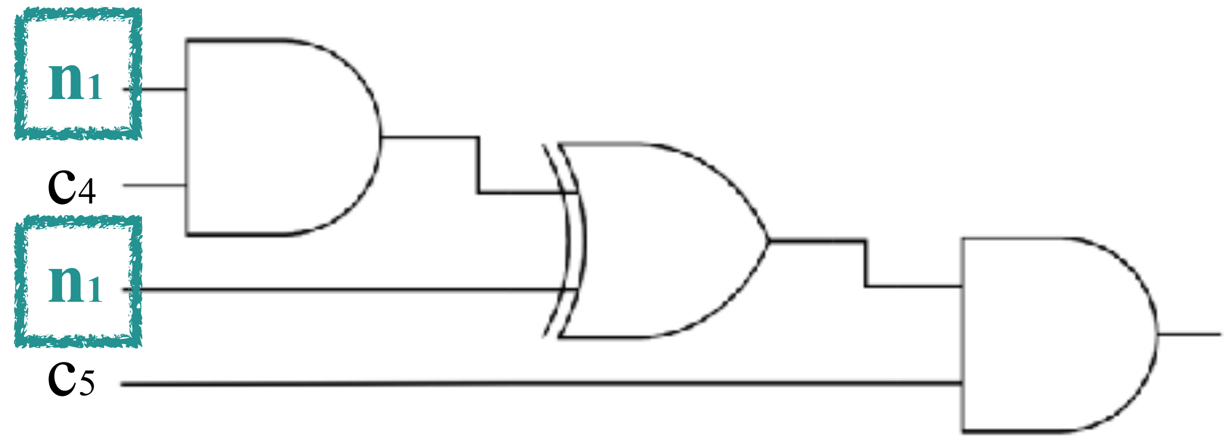


New Input Circuit Optimization

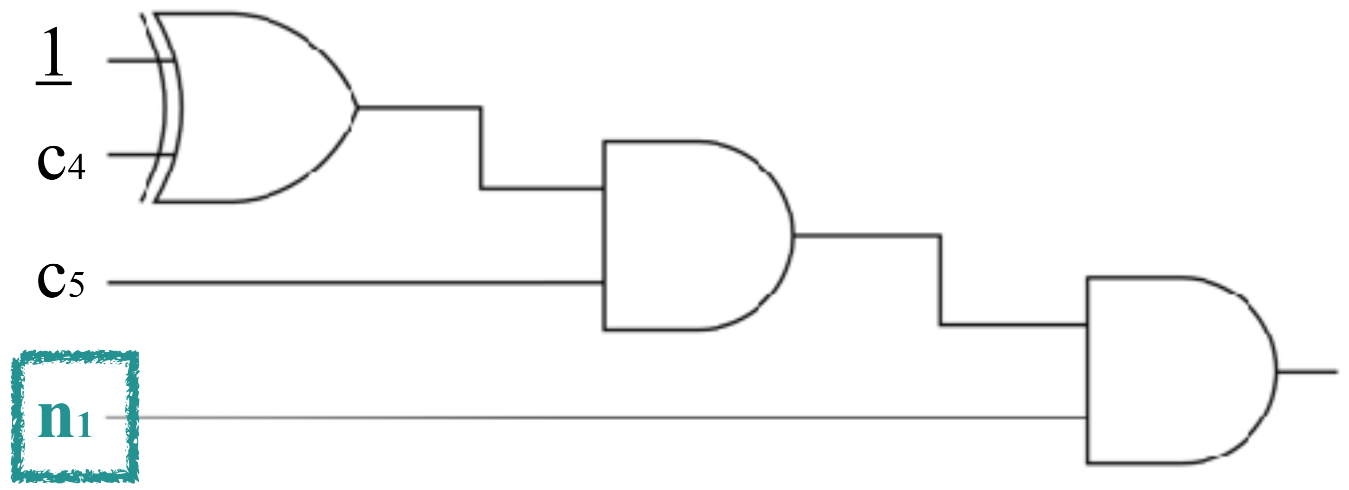
?

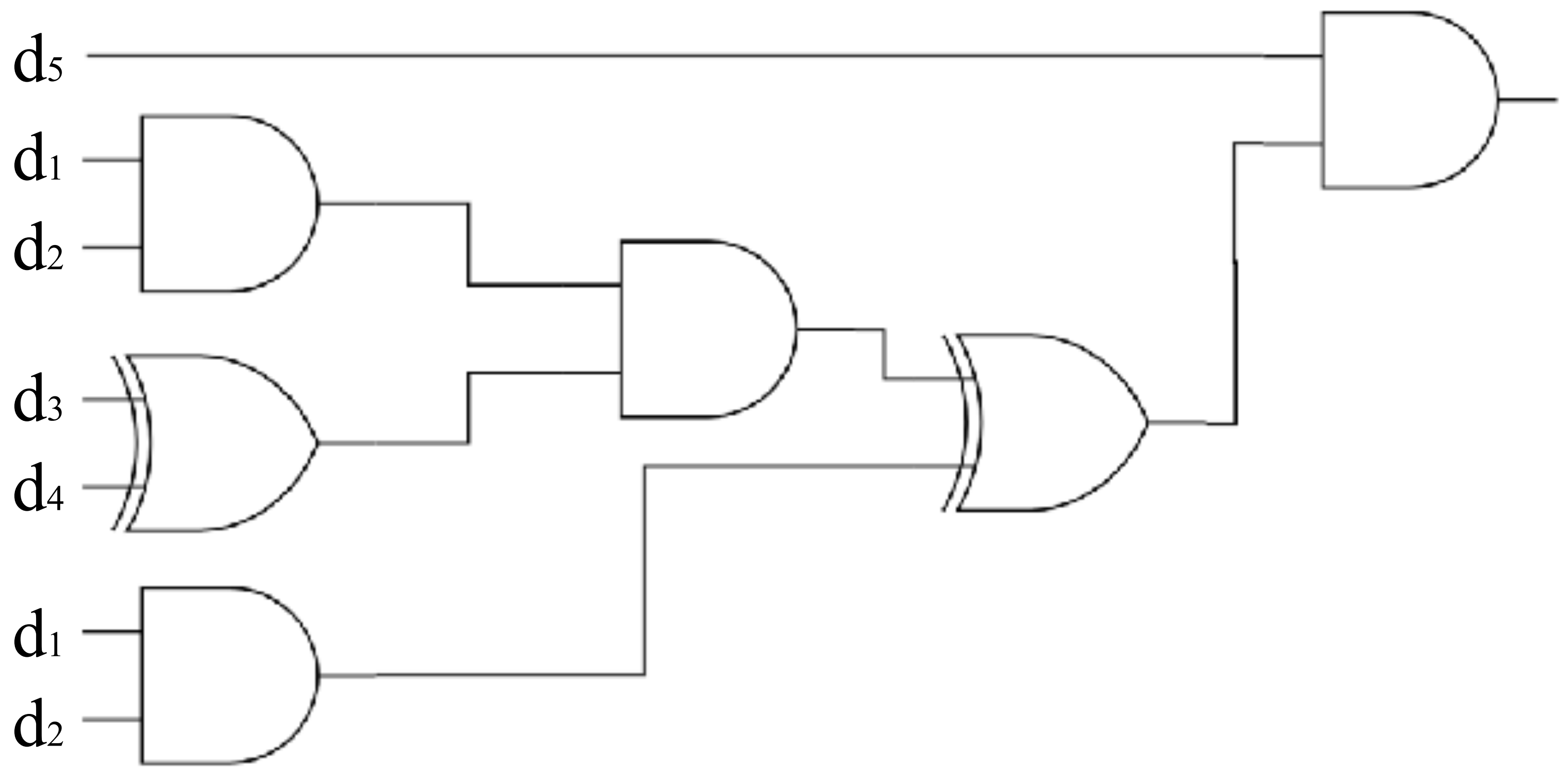
Applying Learned Optimization Patterns (2/2)

Normalization + Equational Matching



Normalized Opt. Patterns





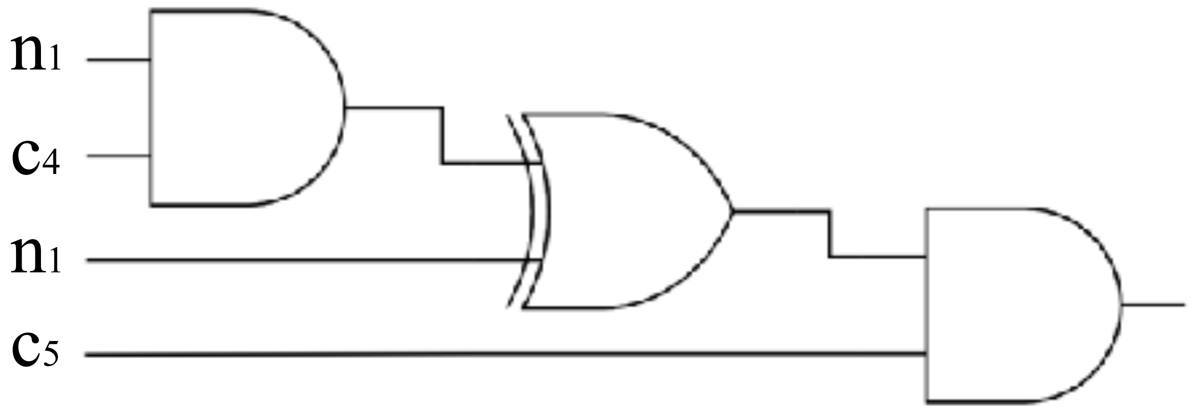
New Input Circuit Optimization

?

Applying Learned Optimization Patterns (2/2)

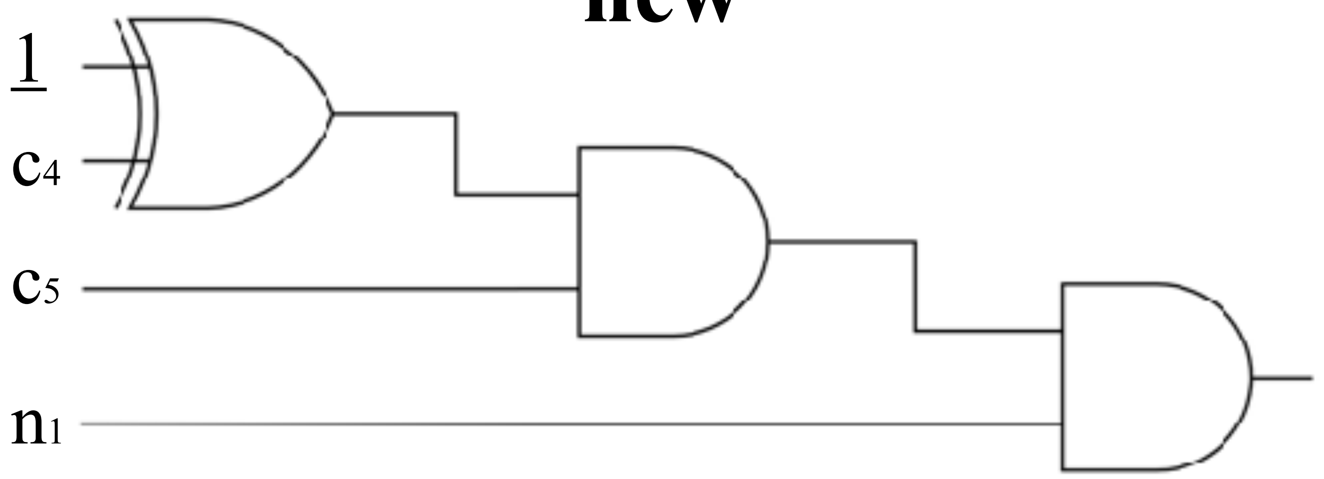
Normalization + Equational Matching

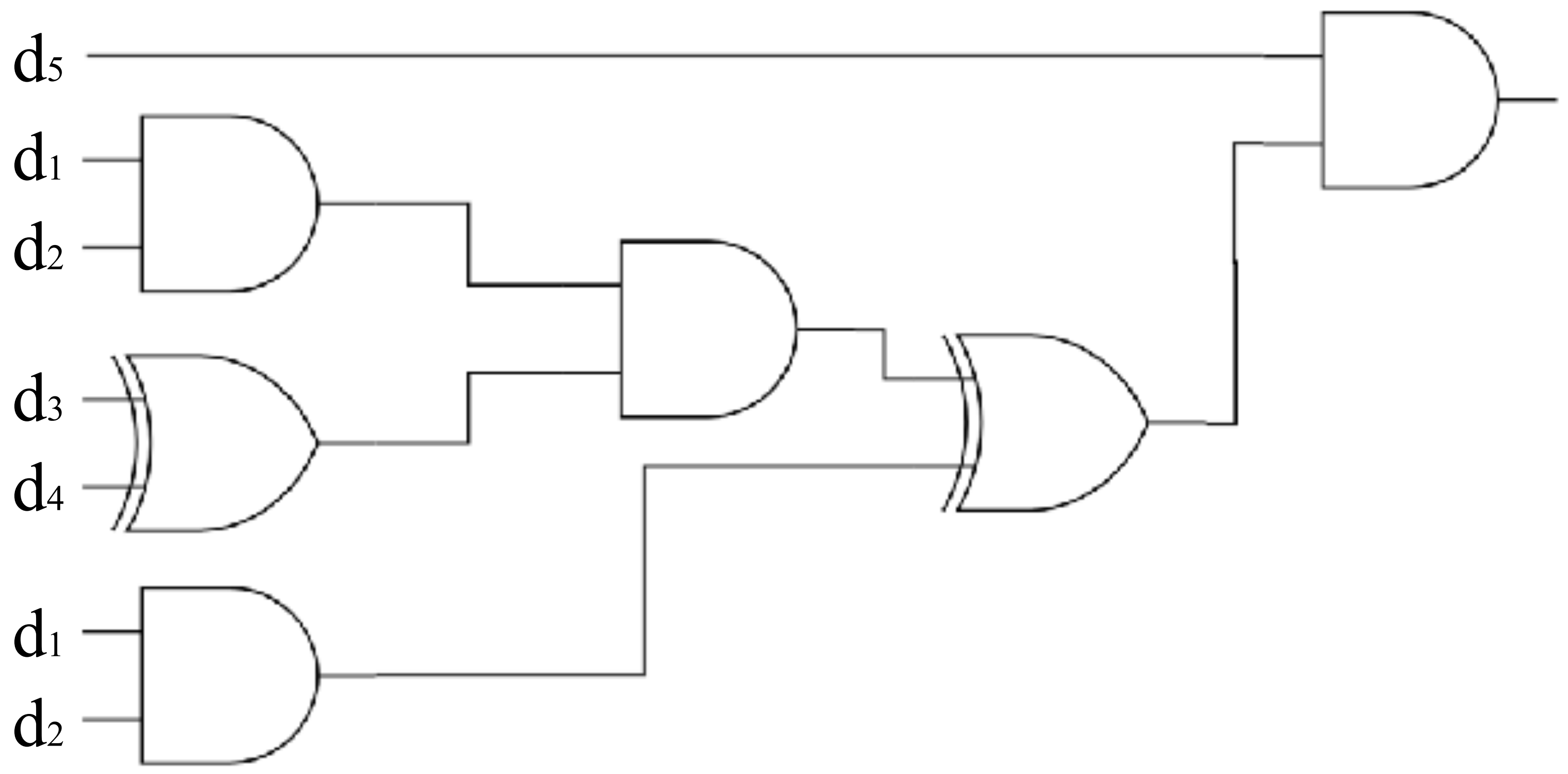
old



Normalized Opt. Patterns

new





target

New Input Circuit Optimization

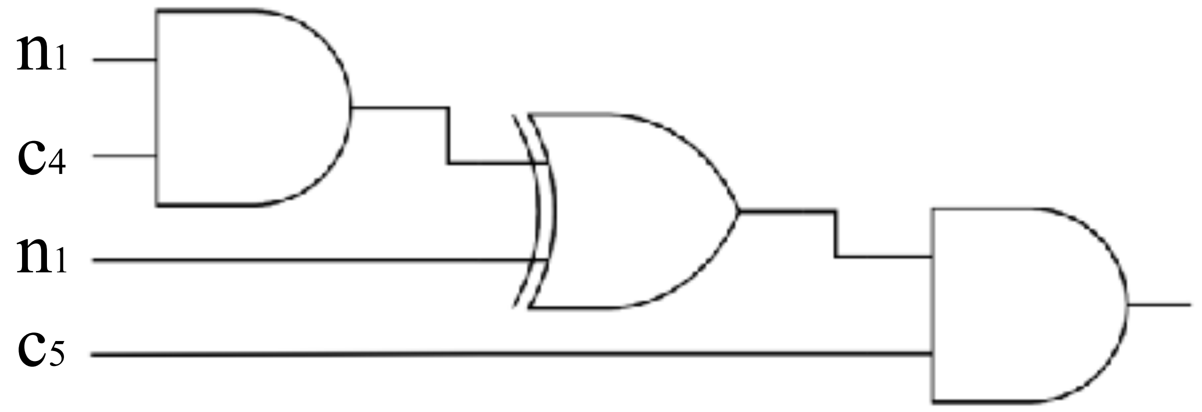
?

target'

Applying Learned Optimization Patterns (2/2)

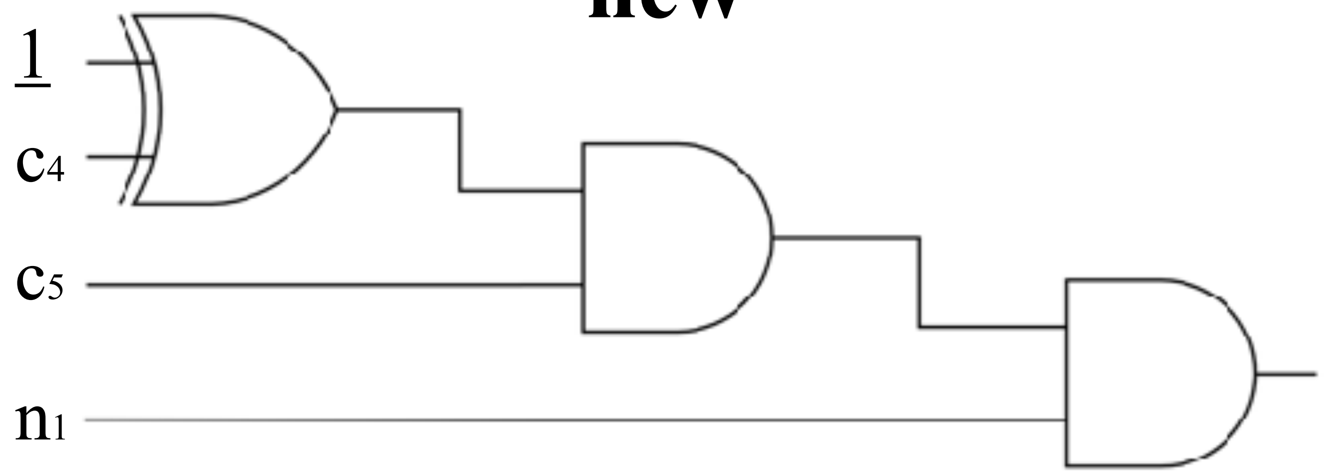
Normalization + Equational Matching

old



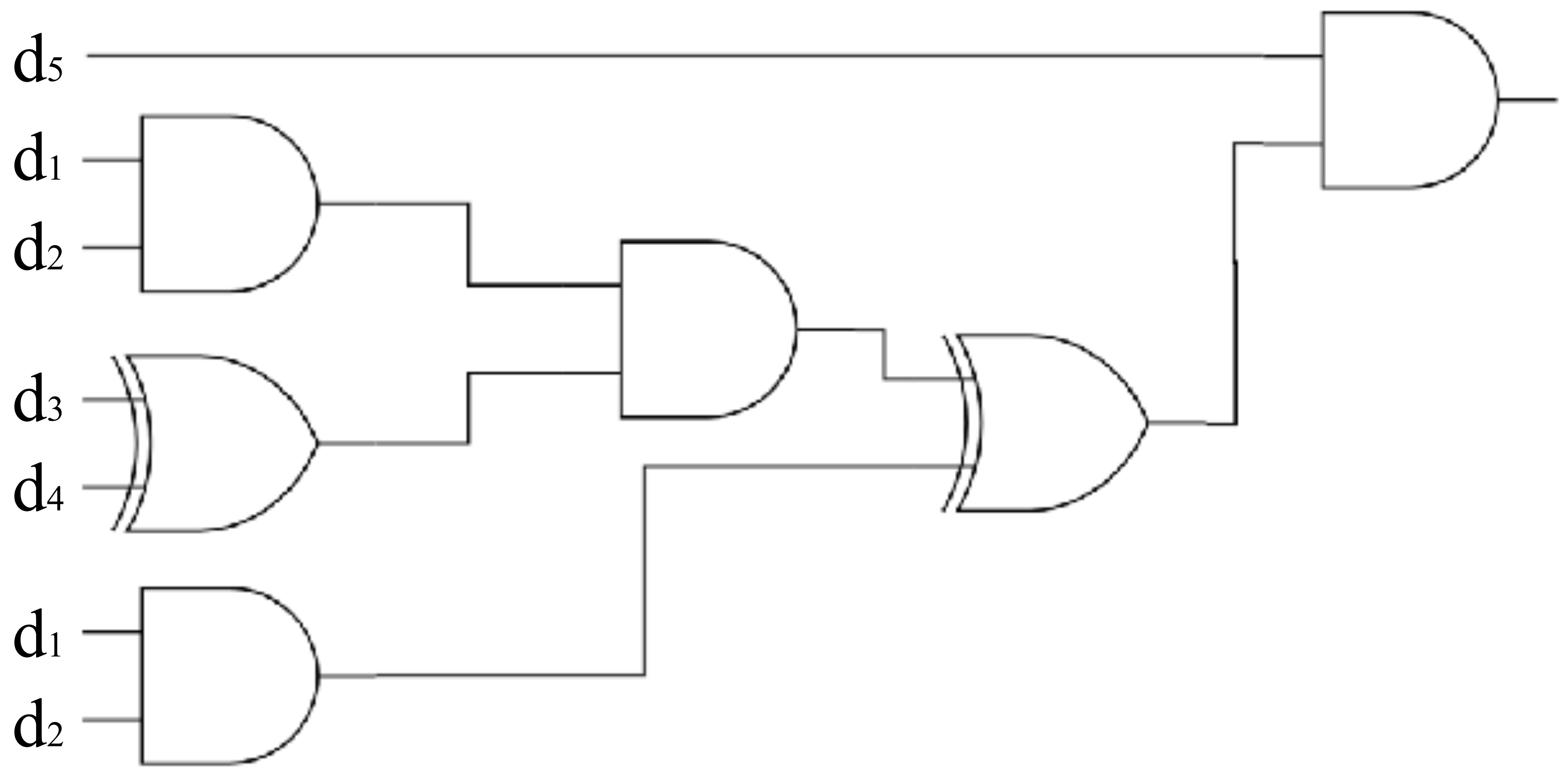
Normalized Opt. Patterns

new



Find substitution σ

(considering commutativity)



target

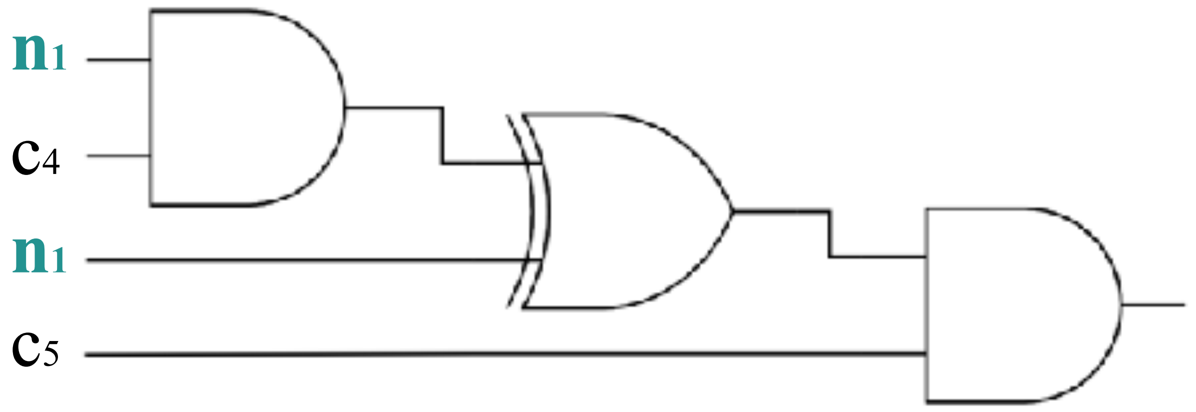
New Input Circuit Optimization

?

Applying Learned Optimization Patterns (2/2)

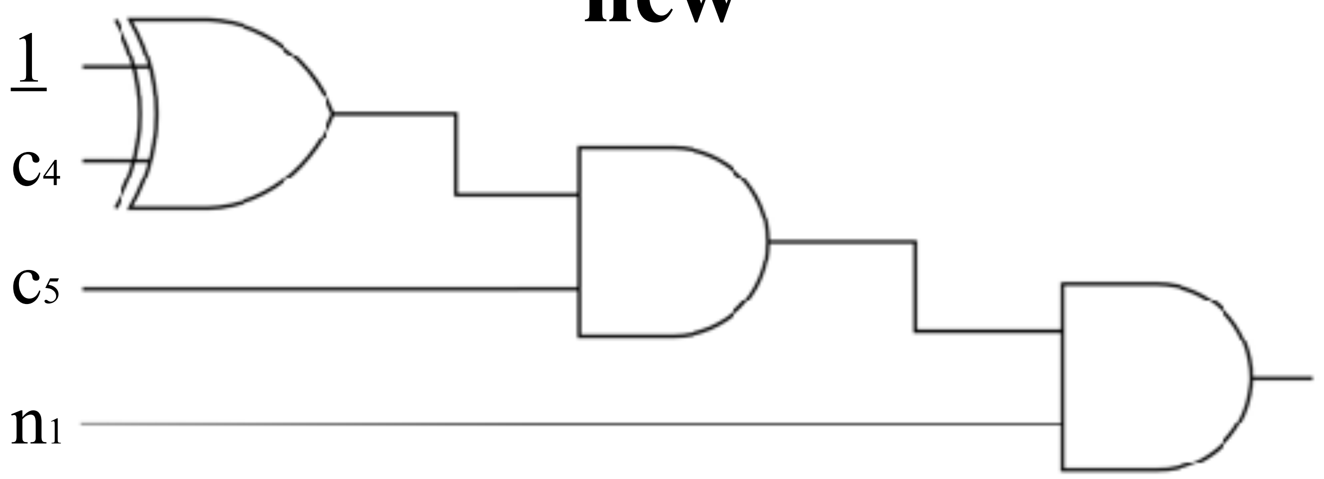
Normalization + Equational Matching

old



Normalized Opt. Patterns

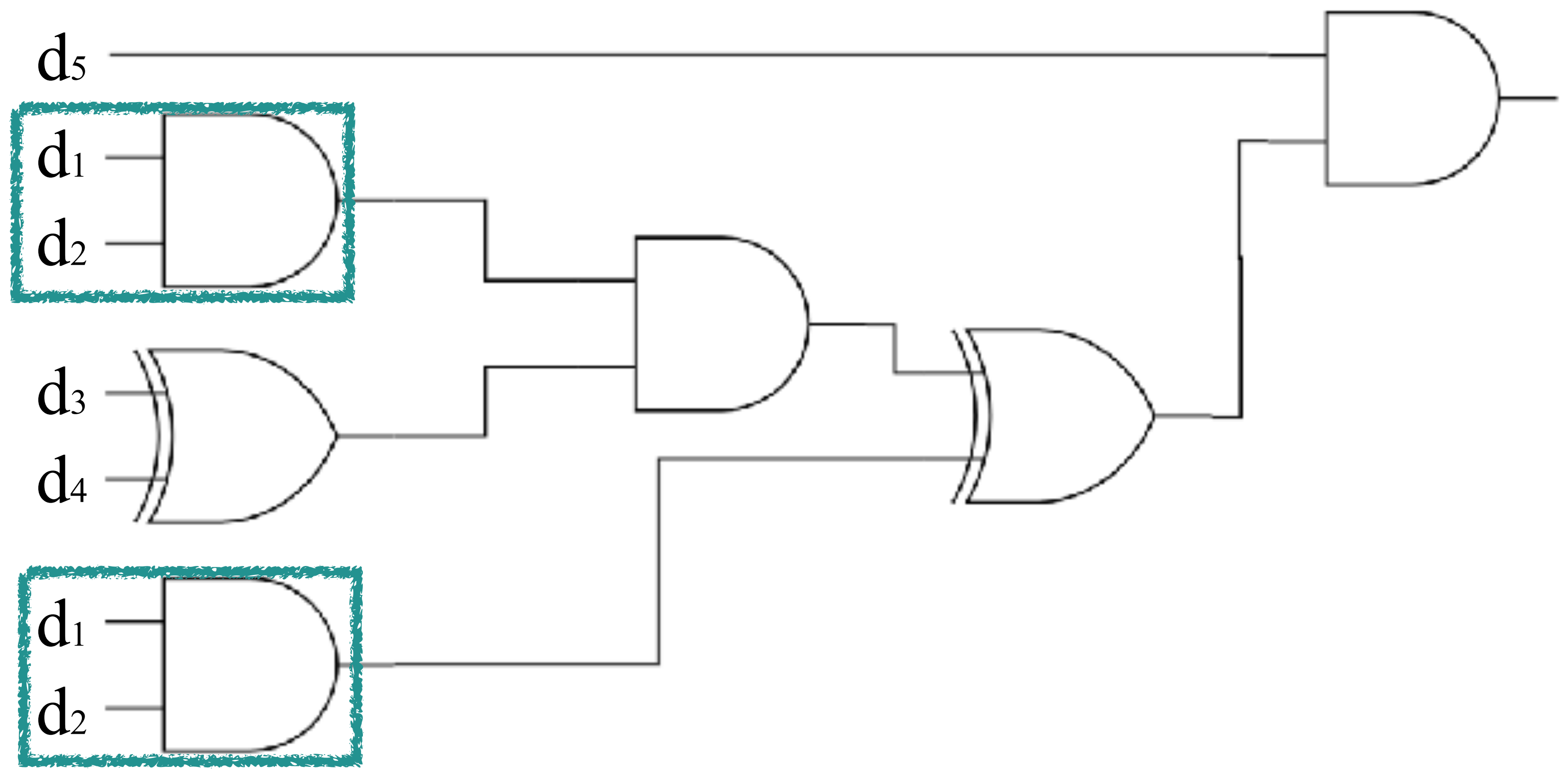
new



Find substitution σ

(considering commutativity)

σ = {n1 ↦ d1 and d2,



target

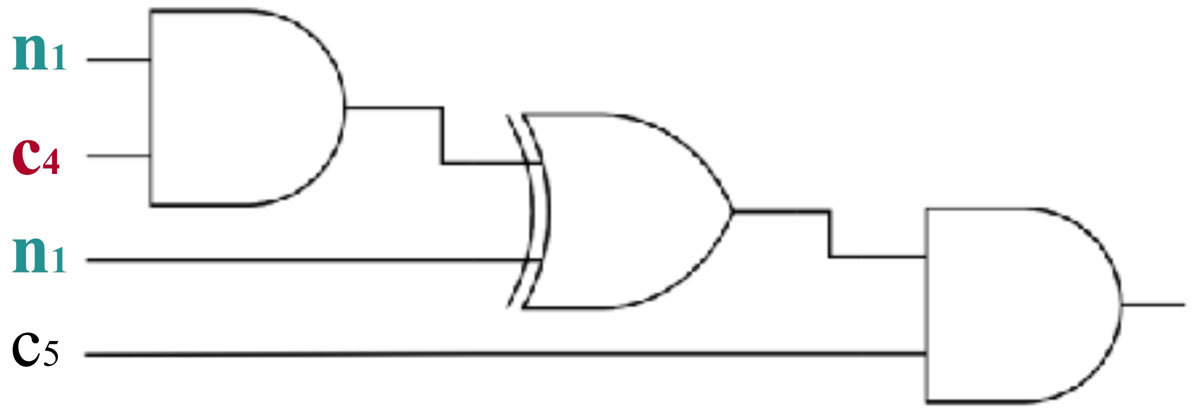
New Input Circuit Optimization

?

Applying Learned Optimization Patterns (2/2)

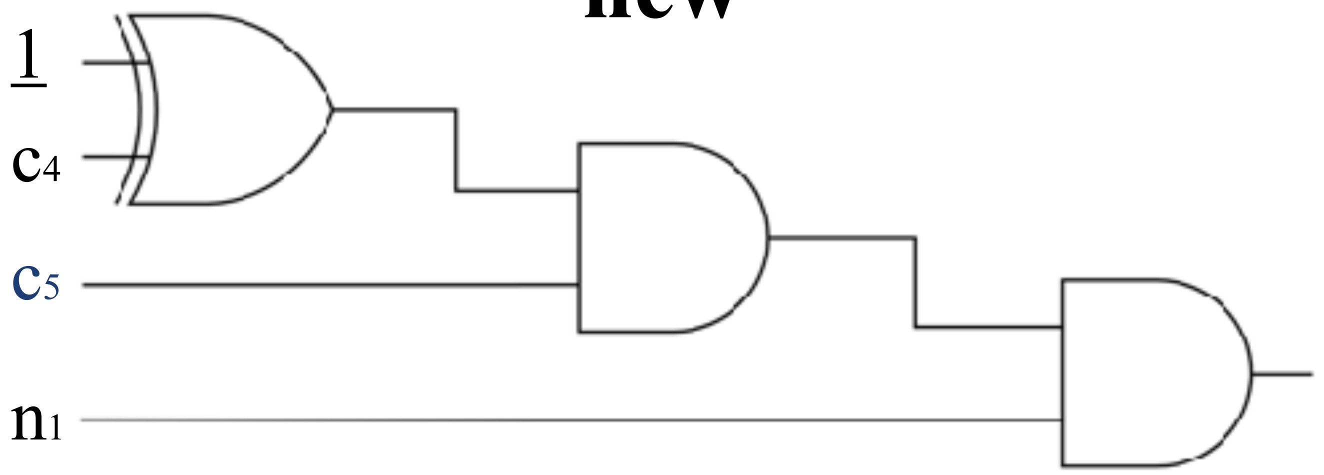
Normalization + Equational Matching

old



Normalized Opt. Patterns

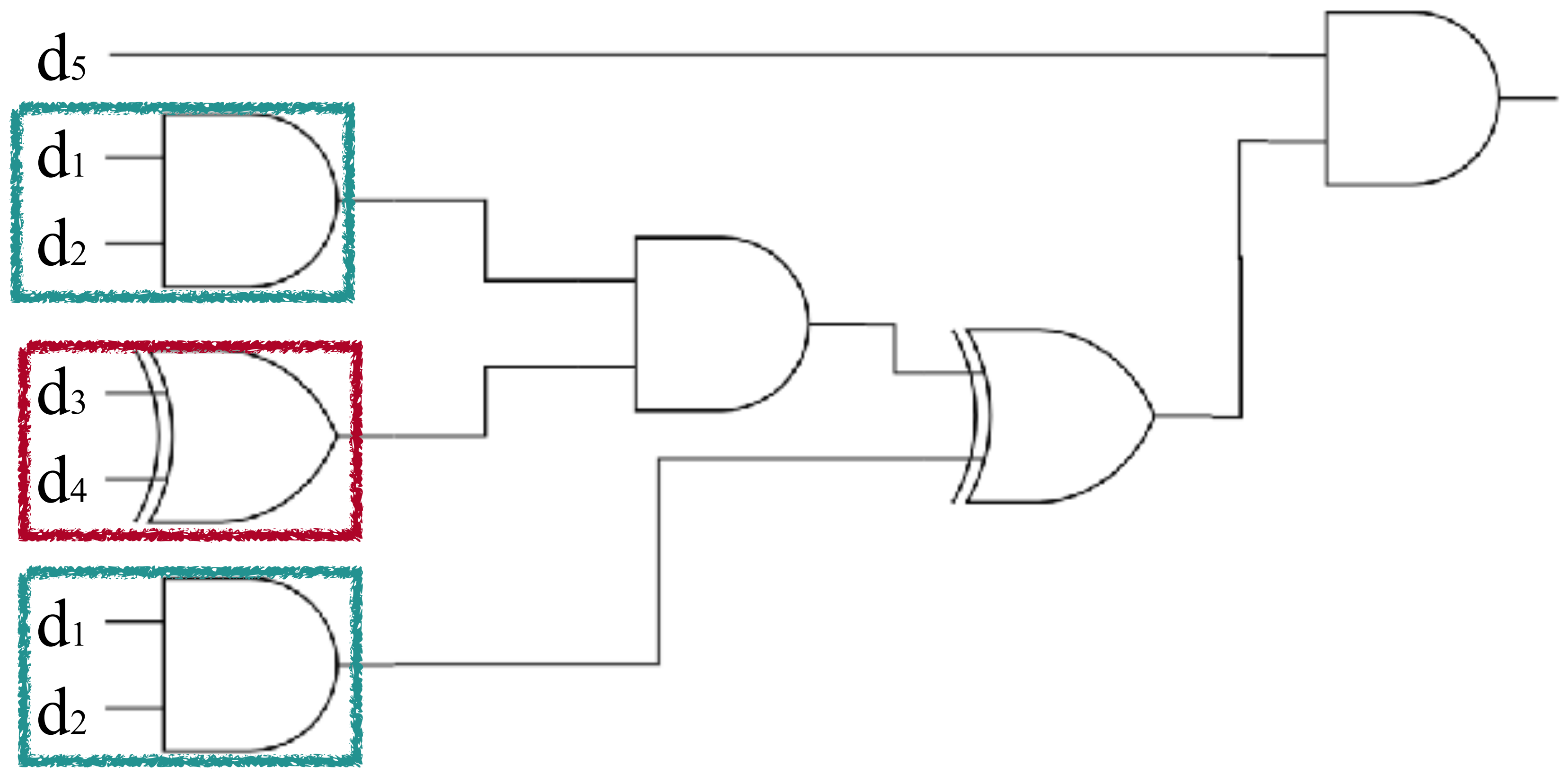
new



Find substitution σ

(considering commutativity)

σ = {n1 ↦ d1 and d2,
c4 ↦ d3 xor d4,



target

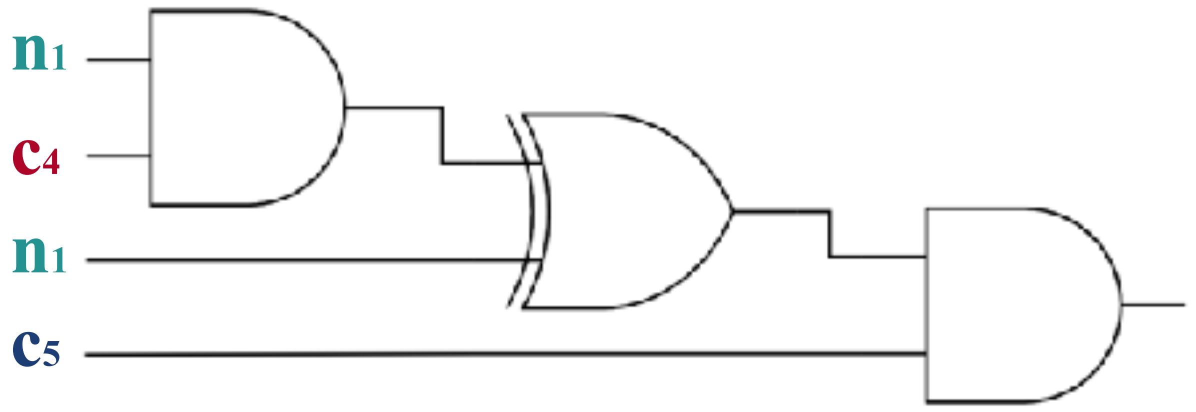
New Input Circuit Optimization

?

Applying Learned Optimization Patterns (2/2)

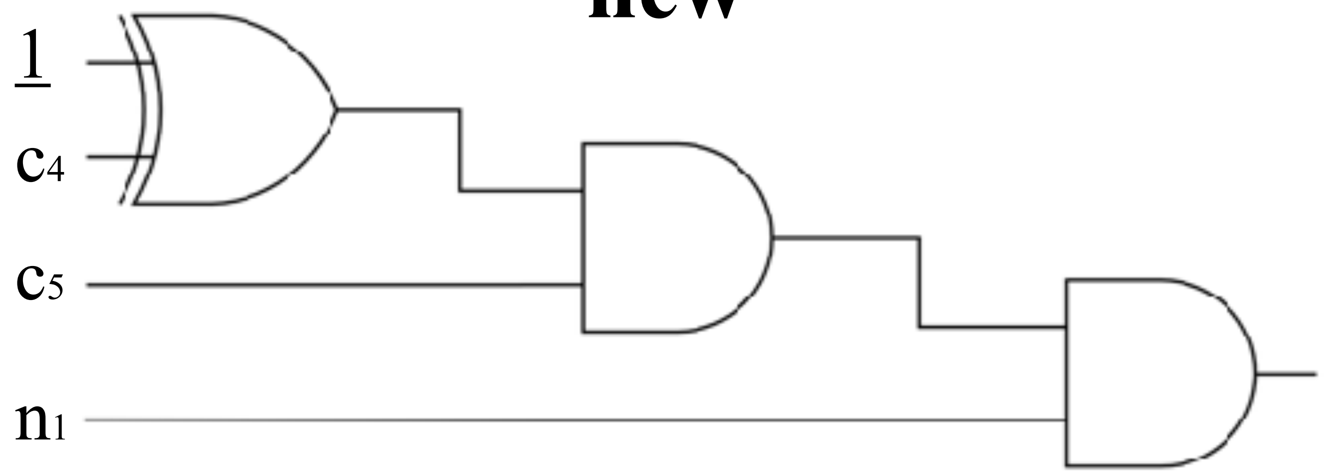
Normalization + Equational Matching

old



Normalized Opt. Patterns

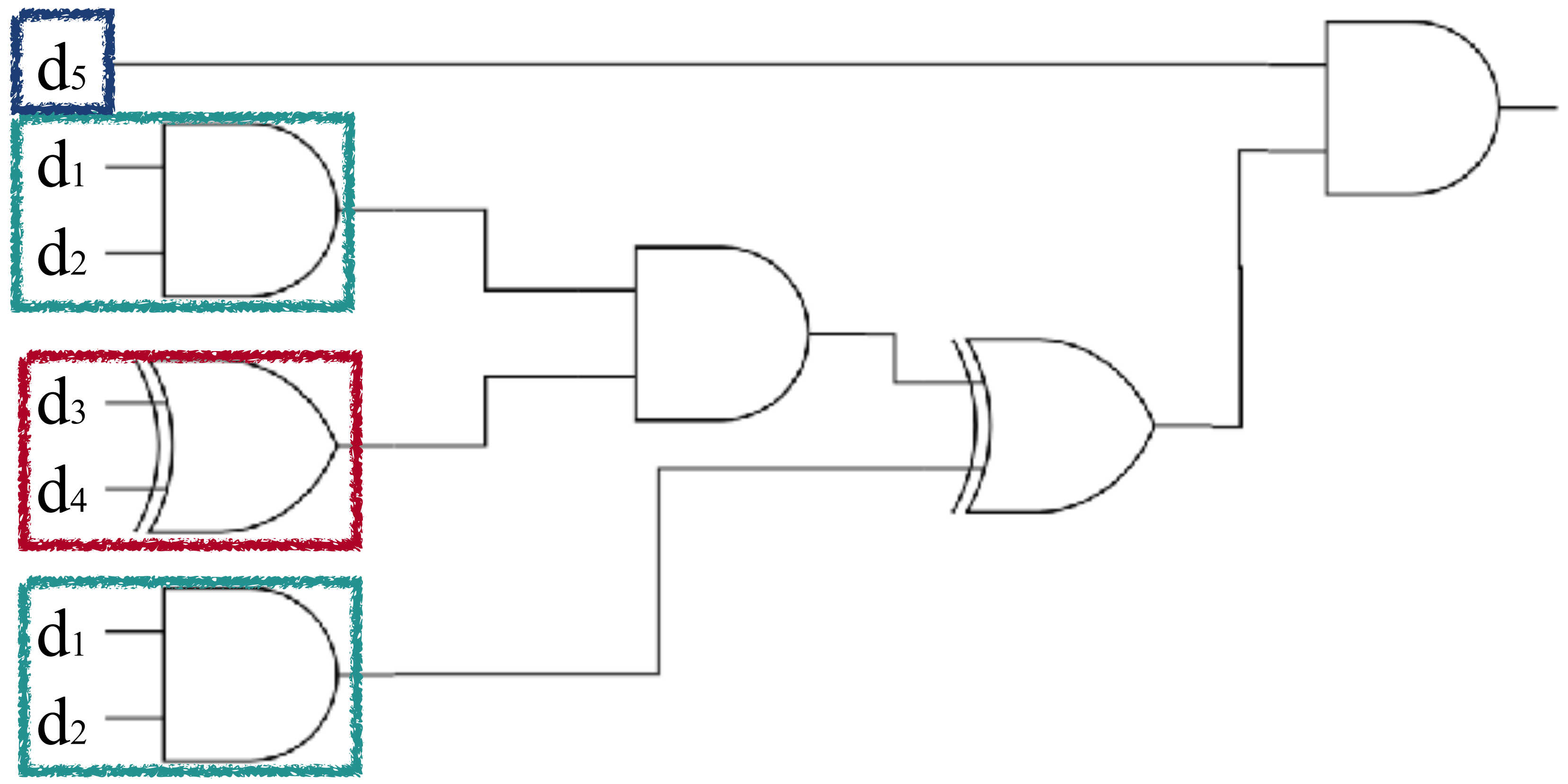
new



Find substitution σ

(considering commutativity)

σ = {n1 ↦ d1 and d2,
c4 ↦ d3 xor d4,
c5 ↦ d5}



target

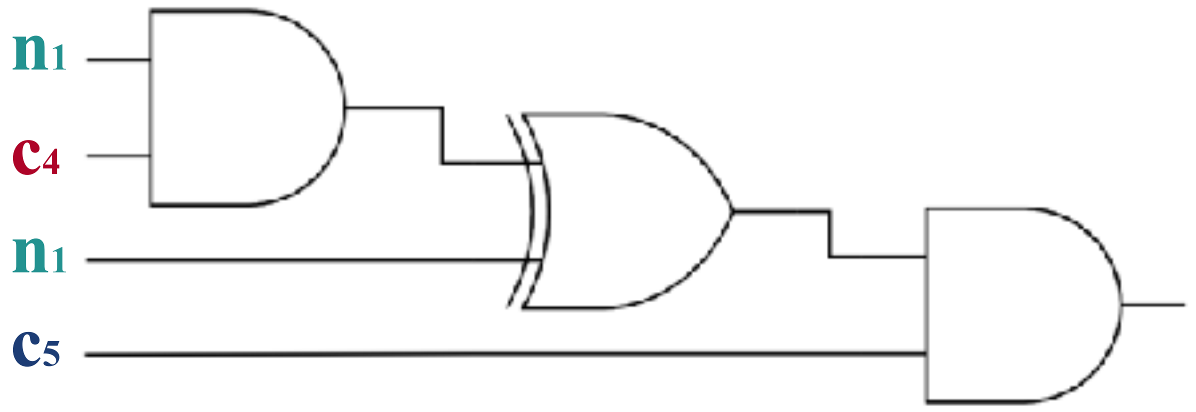
New Input Circuit Optimization

?

Applying Learned Optimization Patterns (2/2)

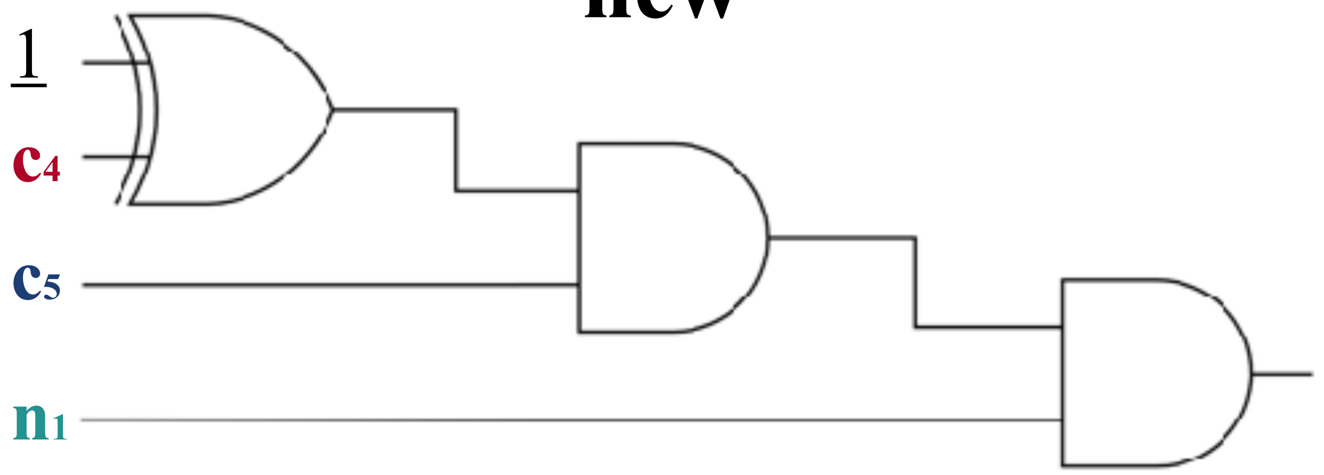
Normalization + Equational Matching

old



Normalized
Opt. Patterns

new

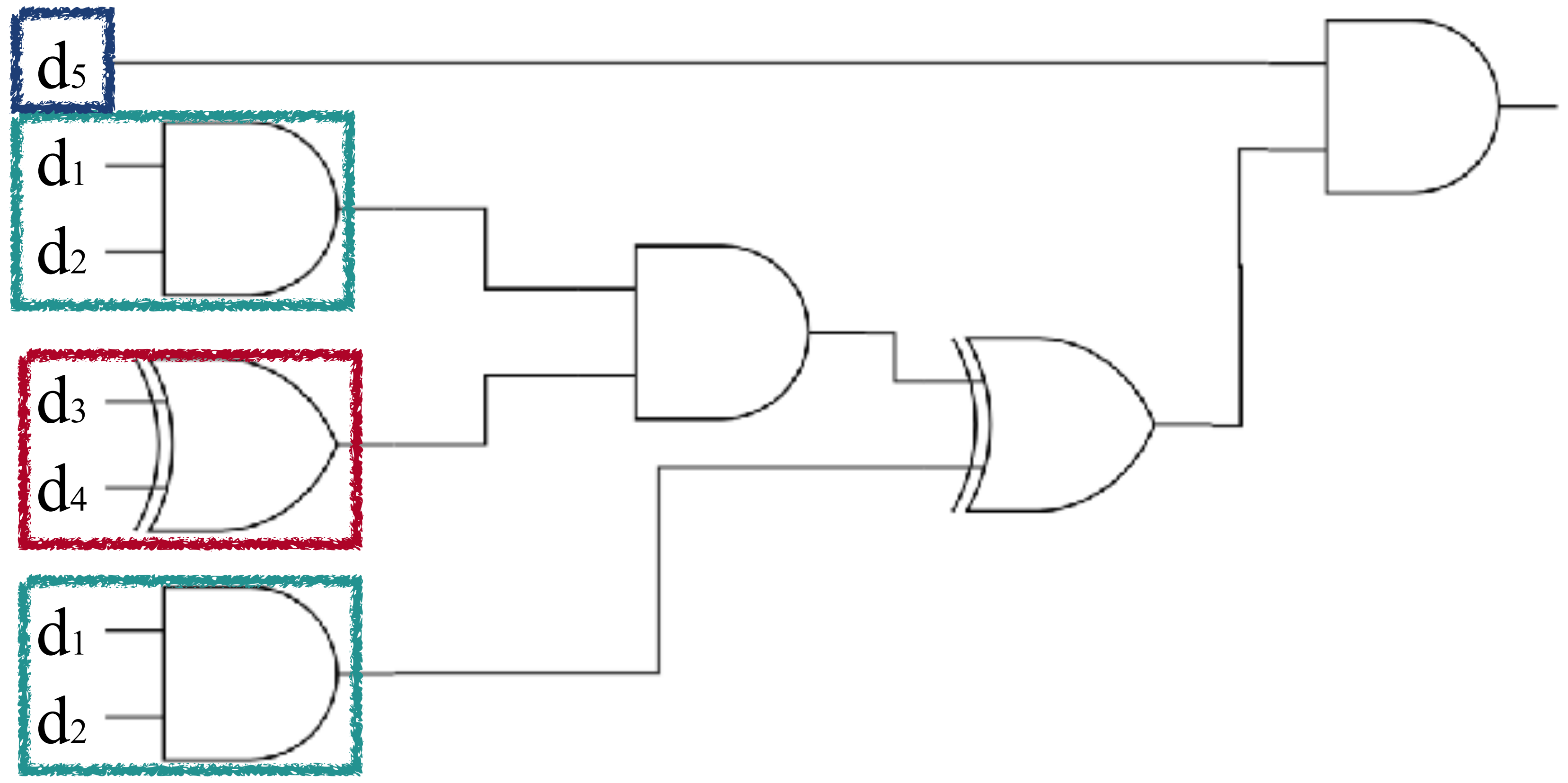


Find substitution σ

(considering commutativity)

Apply substitution σ

σ = {n1 ↦ d1 and d2,
c4 ↦ d3 xor d4,
c5 ↦ d5}



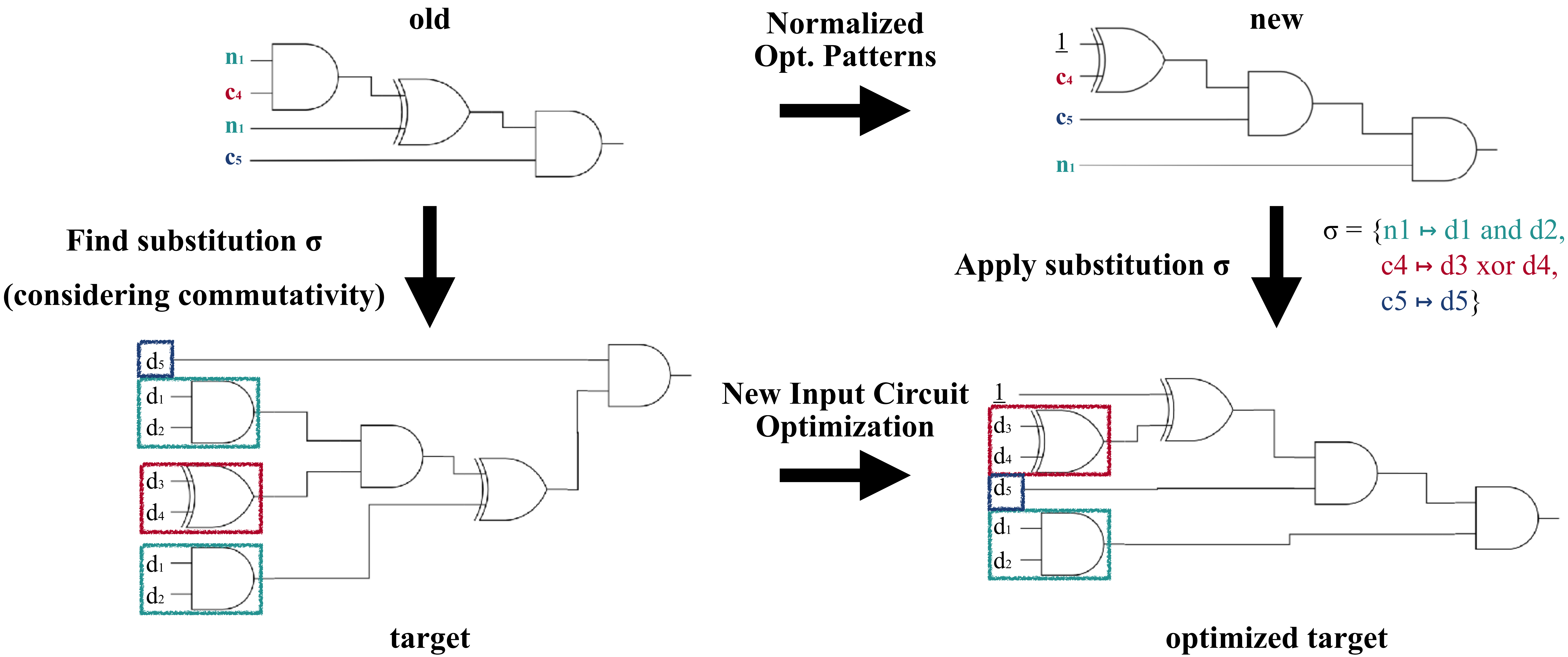
target

New Input Circuit
Optimization

?

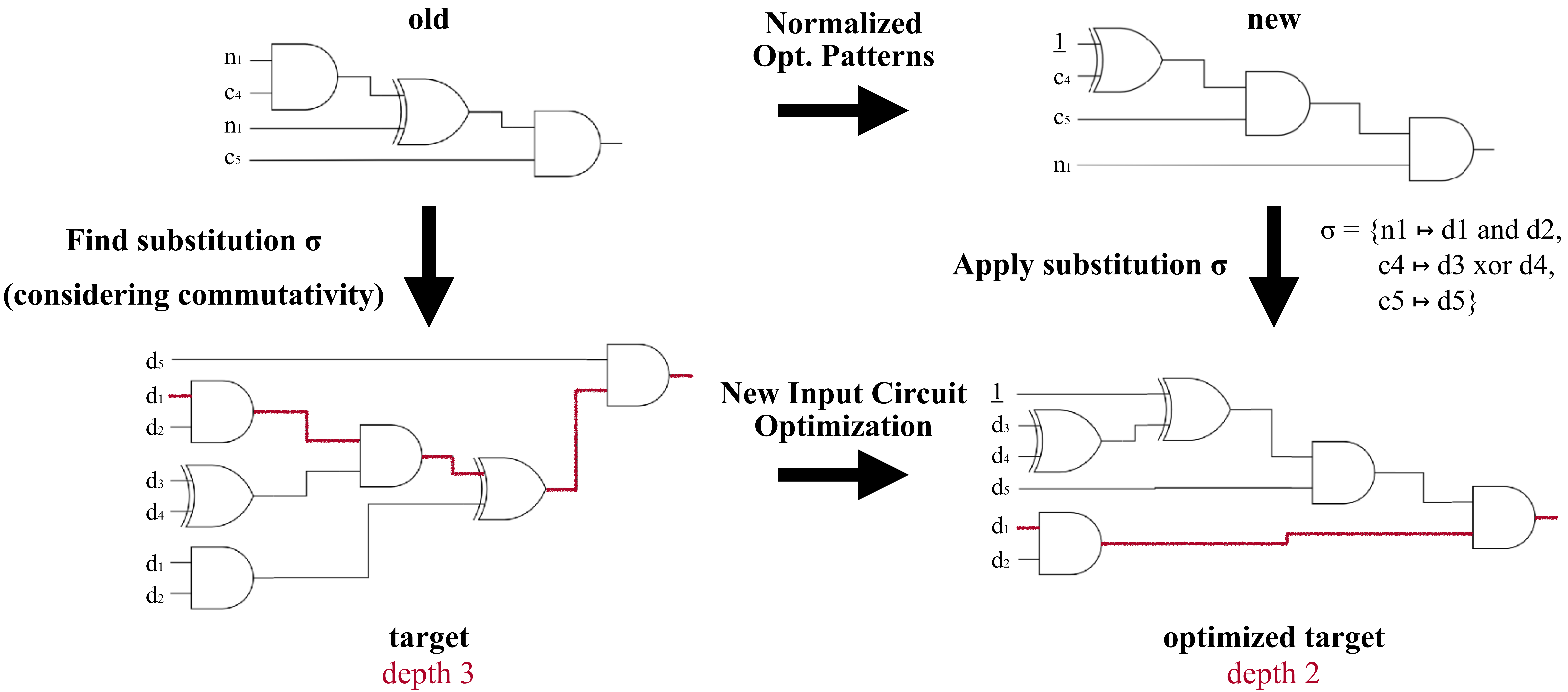
Applying Learned Optimization Patterns (2/2)

Normalization + Equational Matching



Applying Learned Optimization Patterns (2/2)

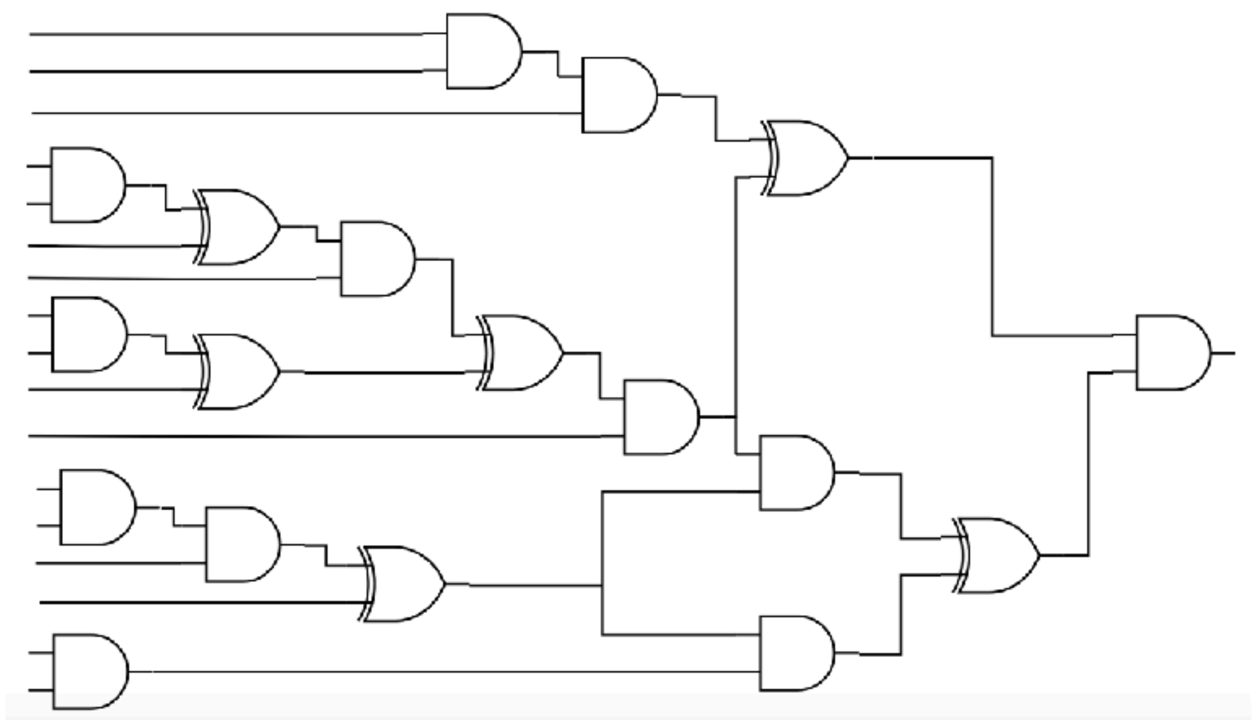
Normalization + Equational Matching



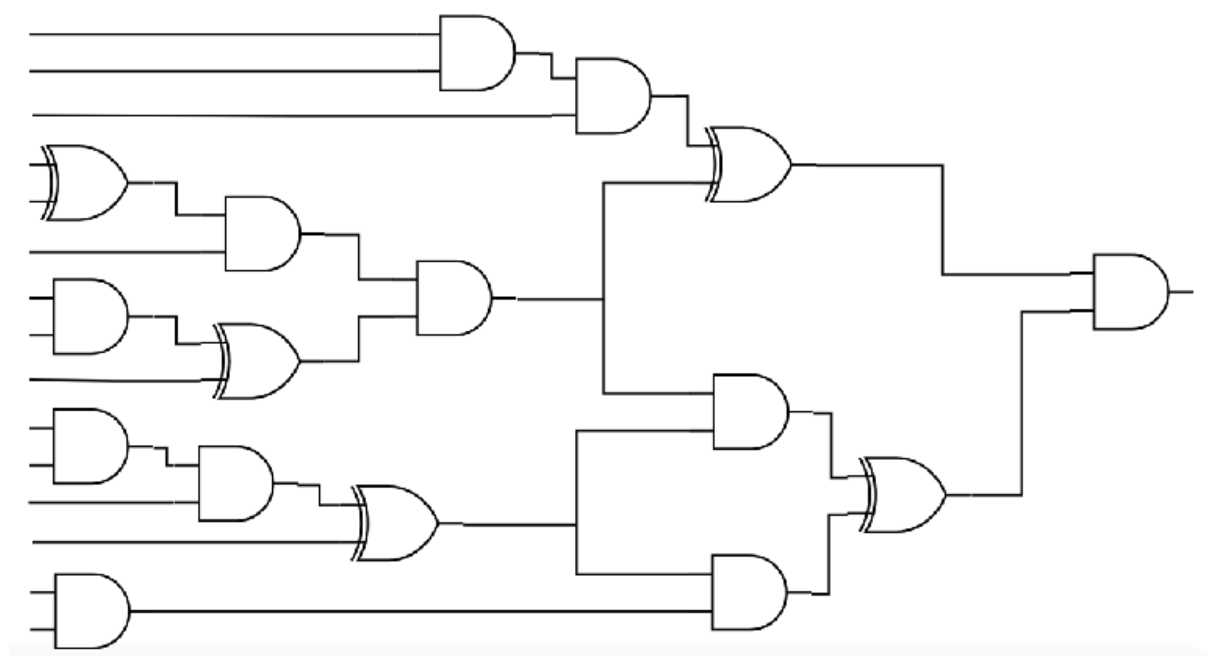
Applying Learned Optimization Patterns

Formal properties

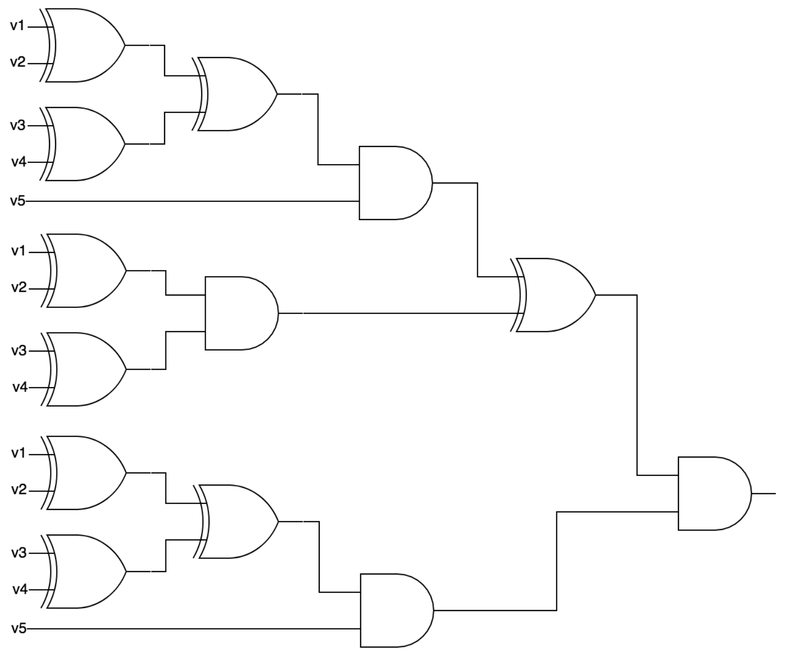
(Soundness) semantics unchanged



Applying an opt. pattern



...



(Termination) finitely many rule applications

Lobster Performance (1/5)

Benchmarks

- 25 HE algorithms from 4 sources
  - Cingulata benchmarks
  - Sorting benchmarks
  - Hackers Delight benchmarks
  - EPFL benchmarks

2 HE friendly algorithms (medical, sorting)

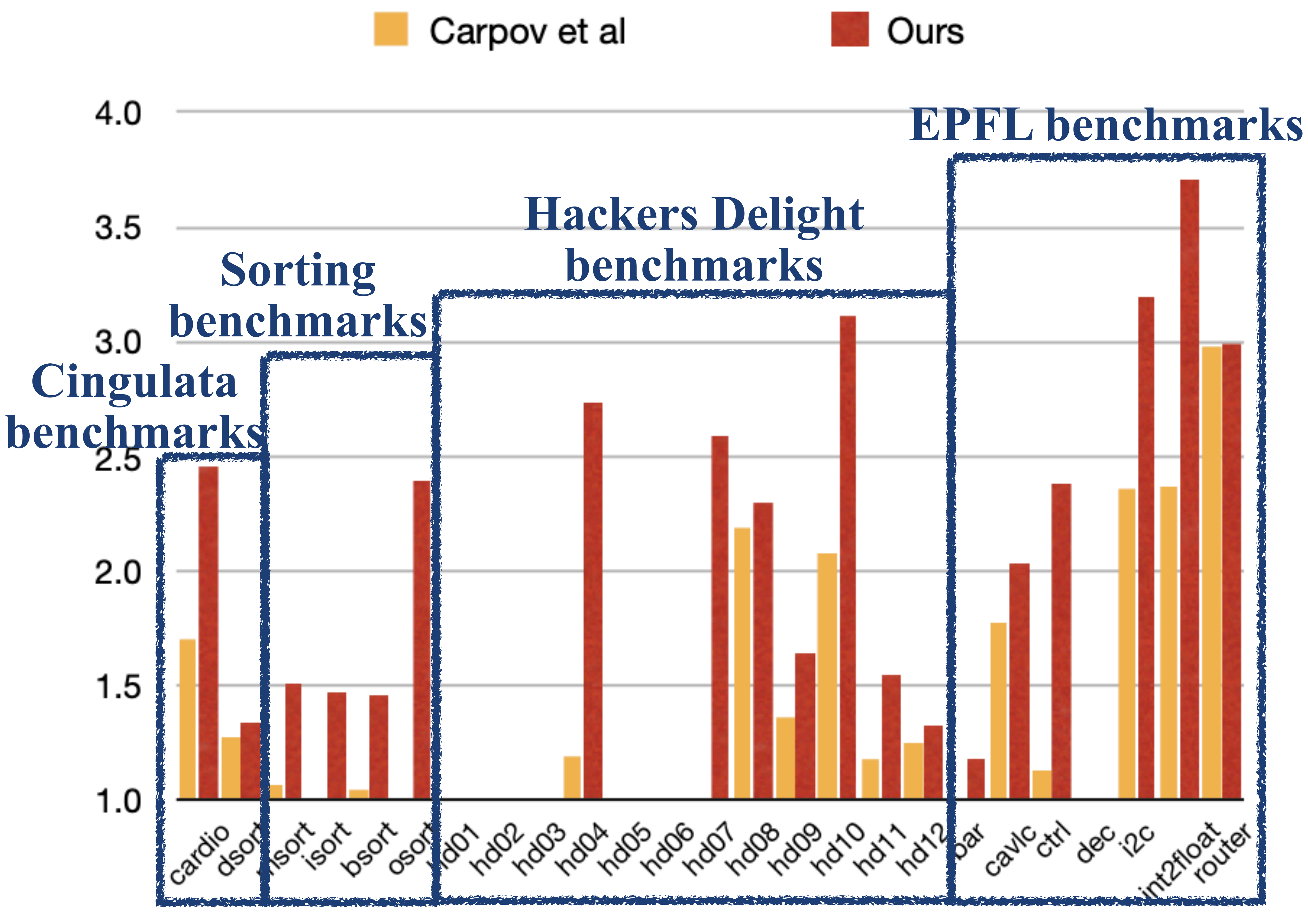
4 privacy-preserving sorting algorithms (merge, insert, bubble, odd-even)

12 Homomorphic bitwise operations

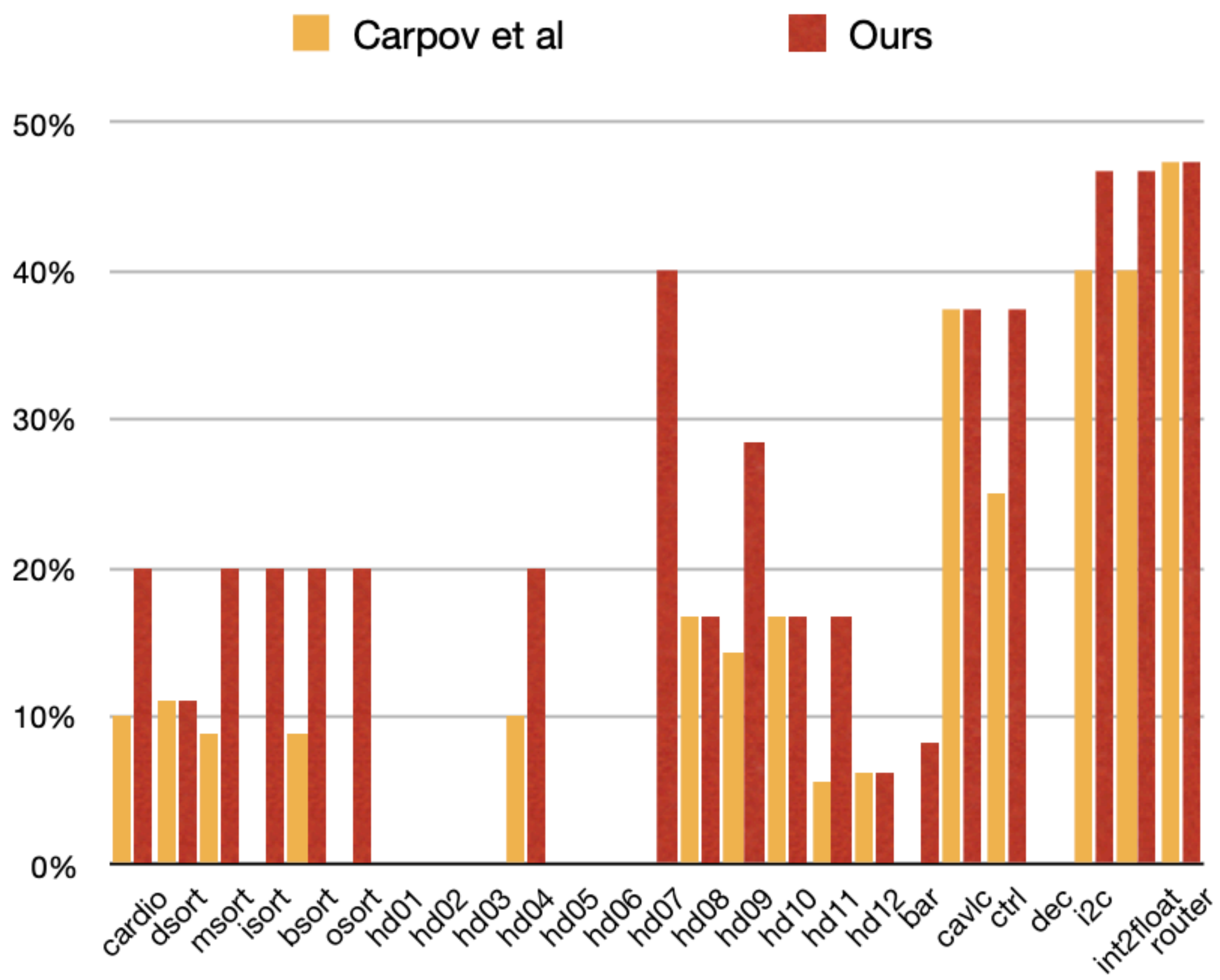
7 EPFL combinational benchmark suite (to test circuit optimizer)

Lobster Performance (2/5)

Optimization Results of Lobster and the baseline



Speedup

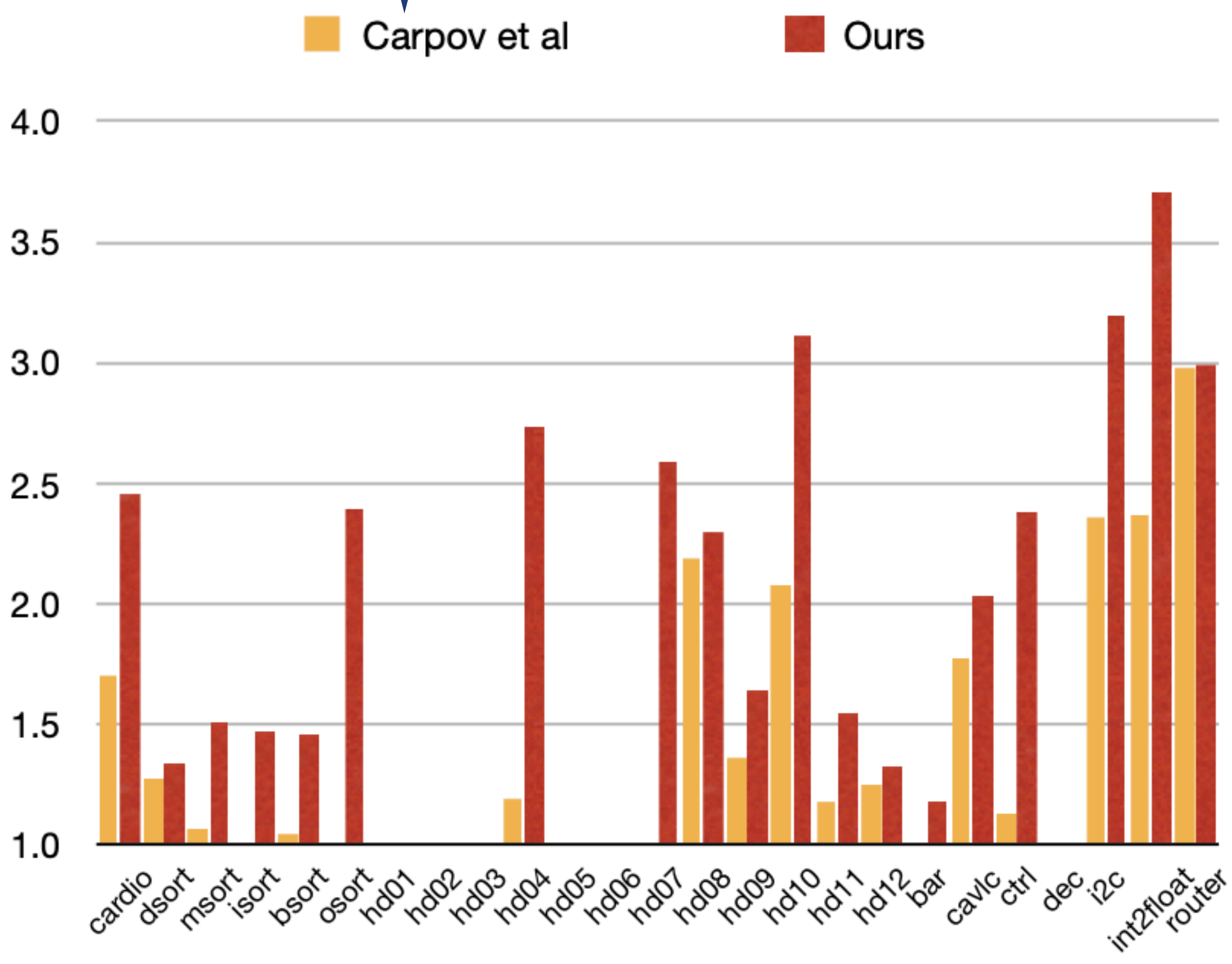


Depth Reduction

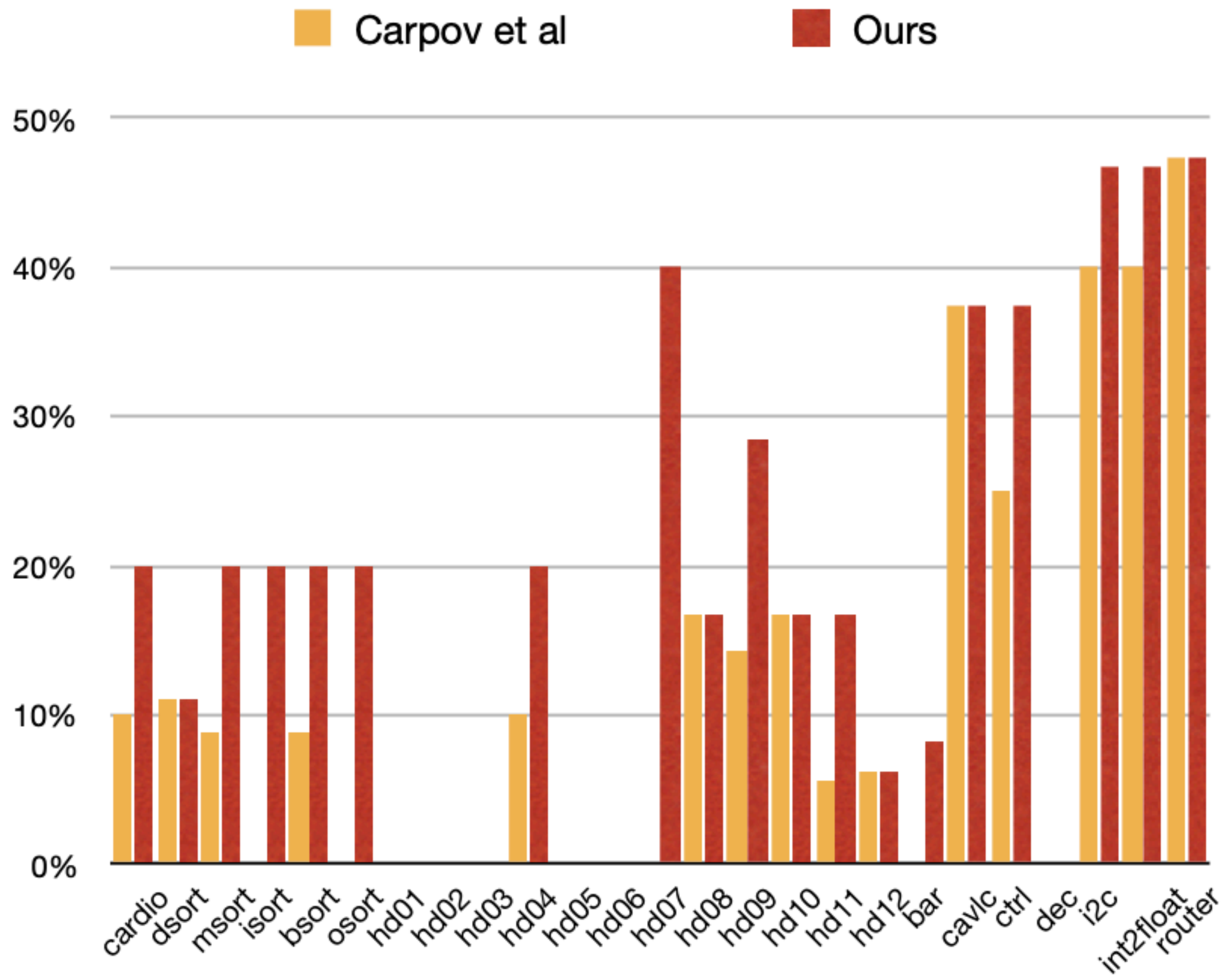
Lobster Performance (2/5)

Hand-written-rule based HE circuit optimizer

Optimization Results of Lobster and the baseline



Speedup



Depth Reduction

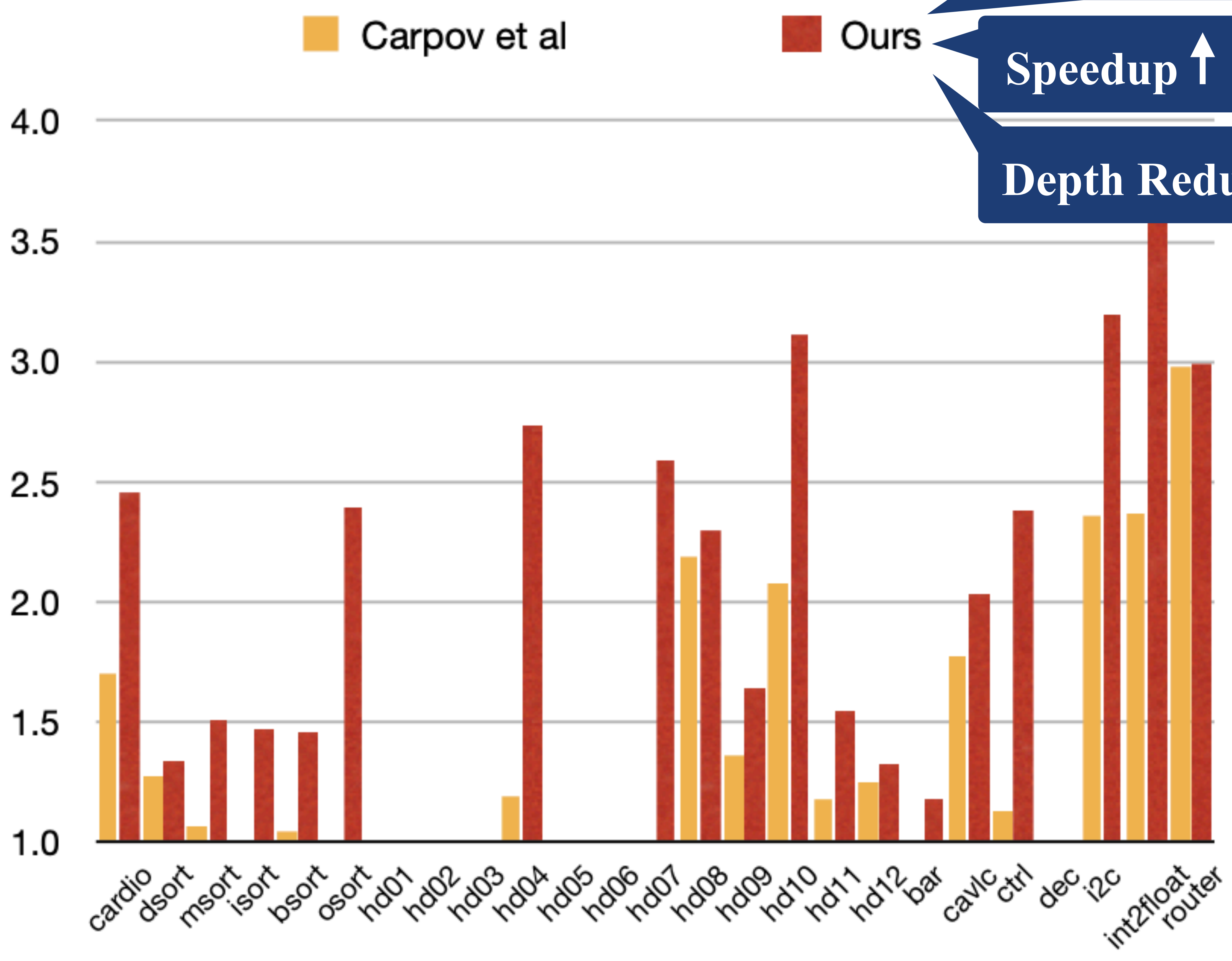
Lobster Performance (2/5)

Optimization Results of Lobster and the baseline

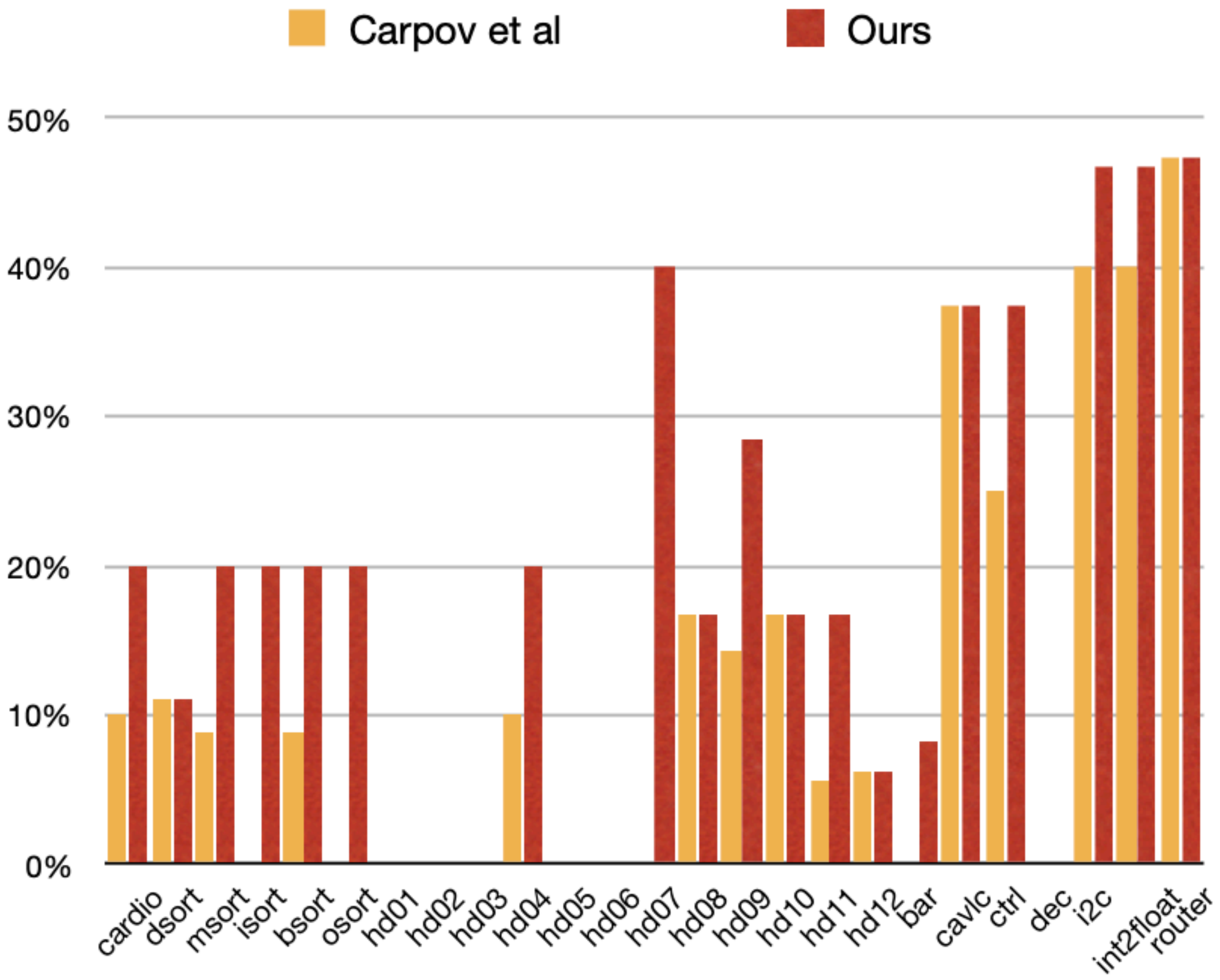
Success rate ↑

Speedup ↑

Depth Reduction ↑



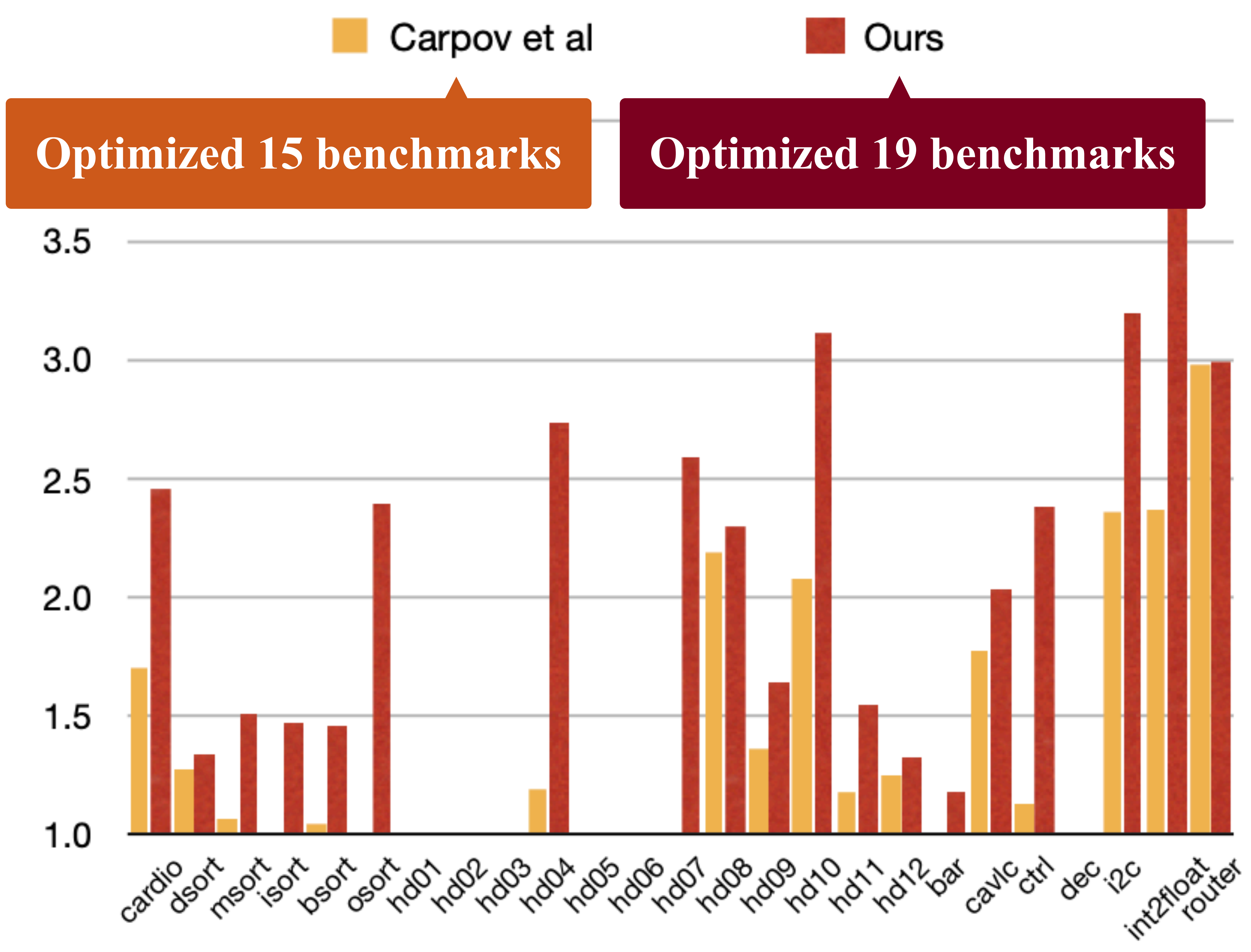
Speedup



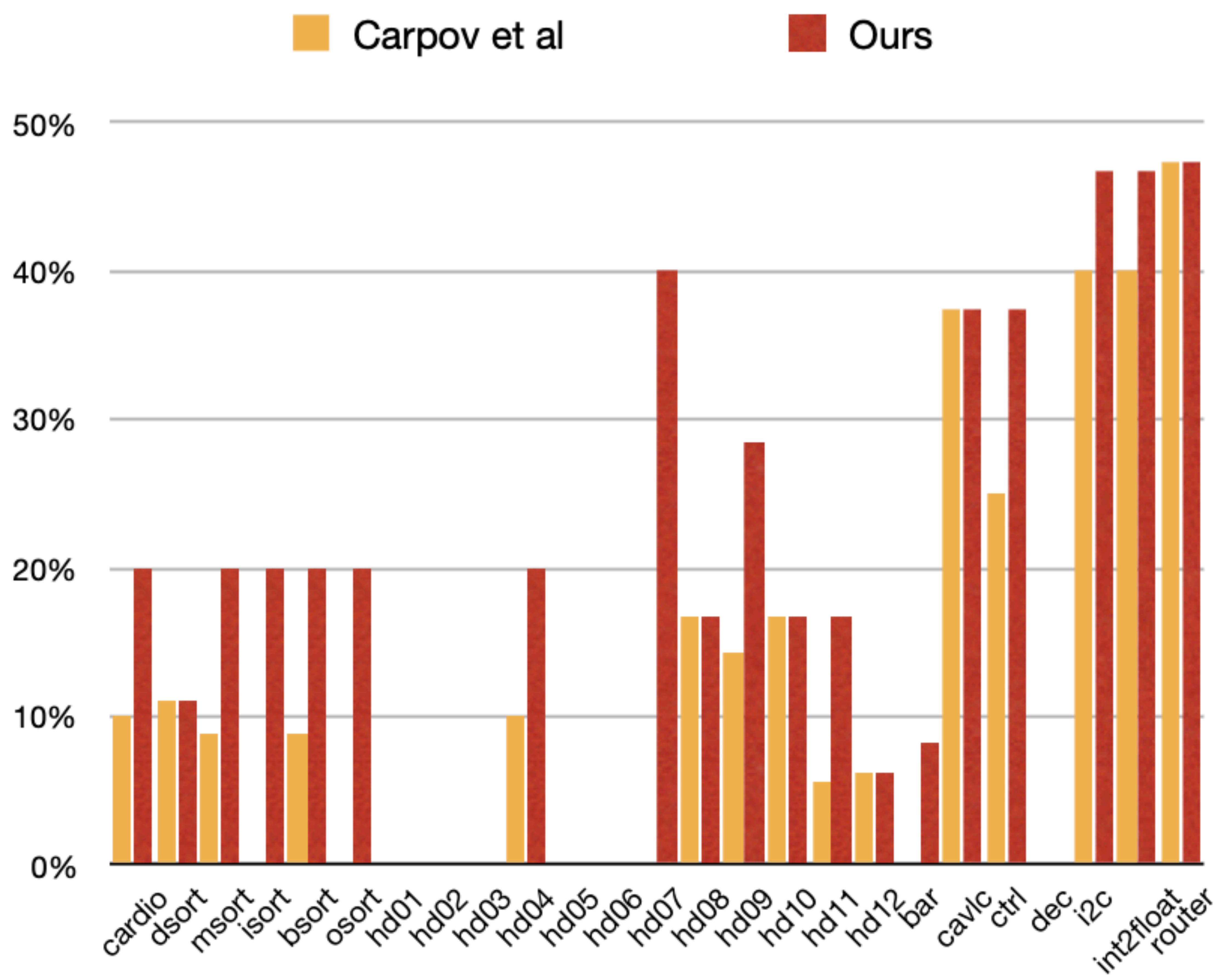
Depth Reduction

Lobster Performance (2/5)

Optimization Results of Lobster and the baseline



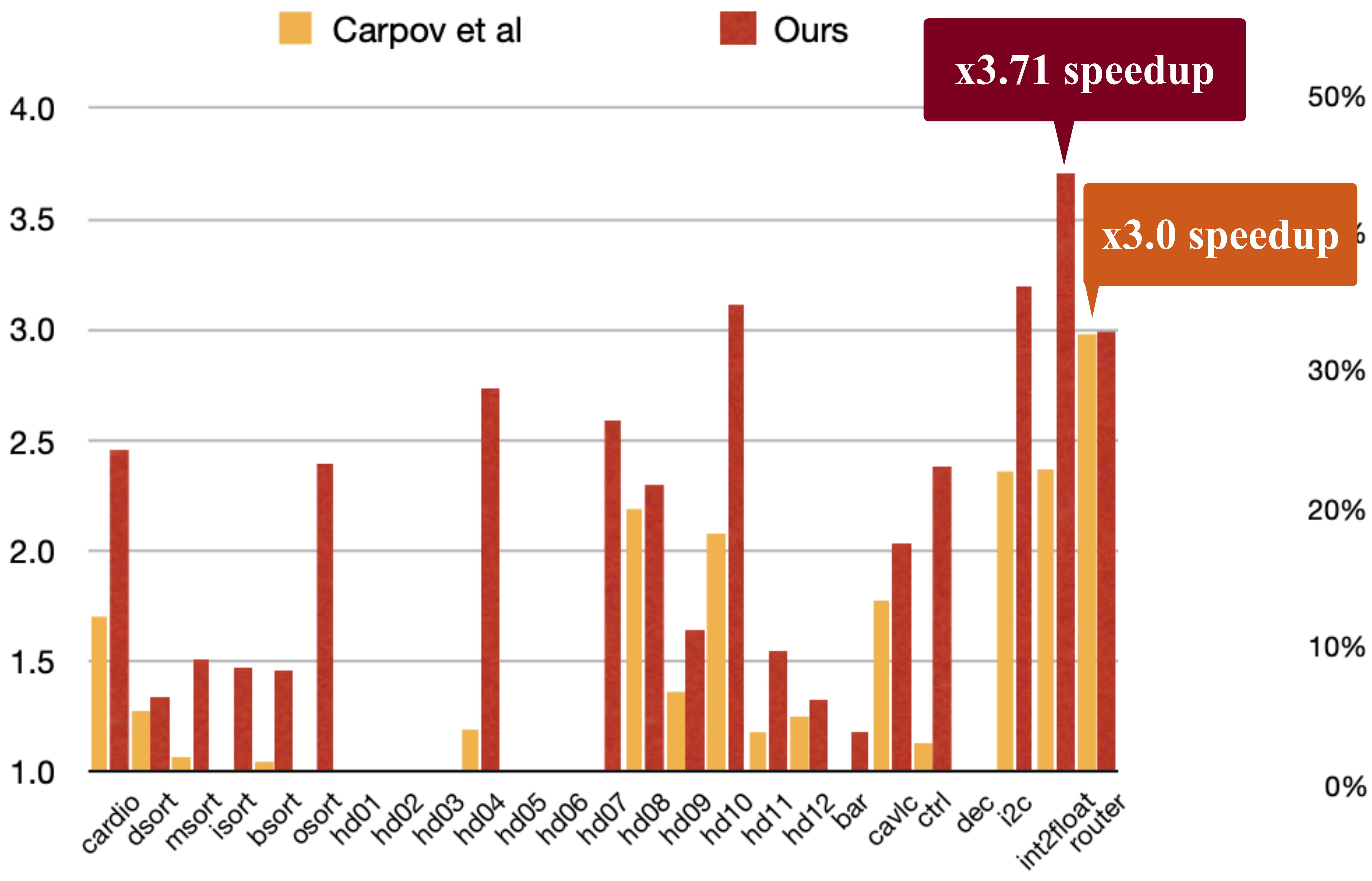
Speedup



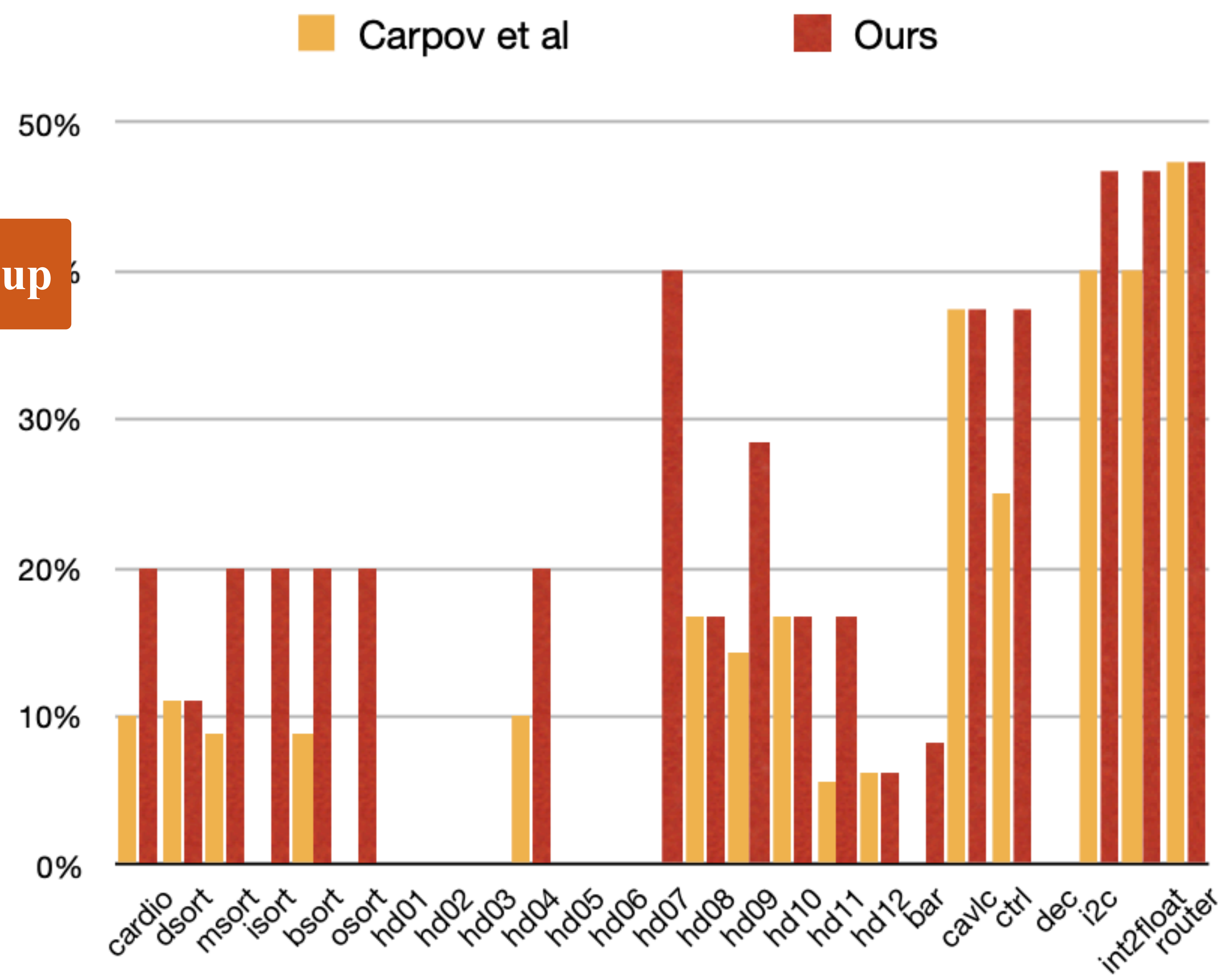
Depth Reduction

Lobster Performance (2/5)

Optimization Results of Lobster and the baseline



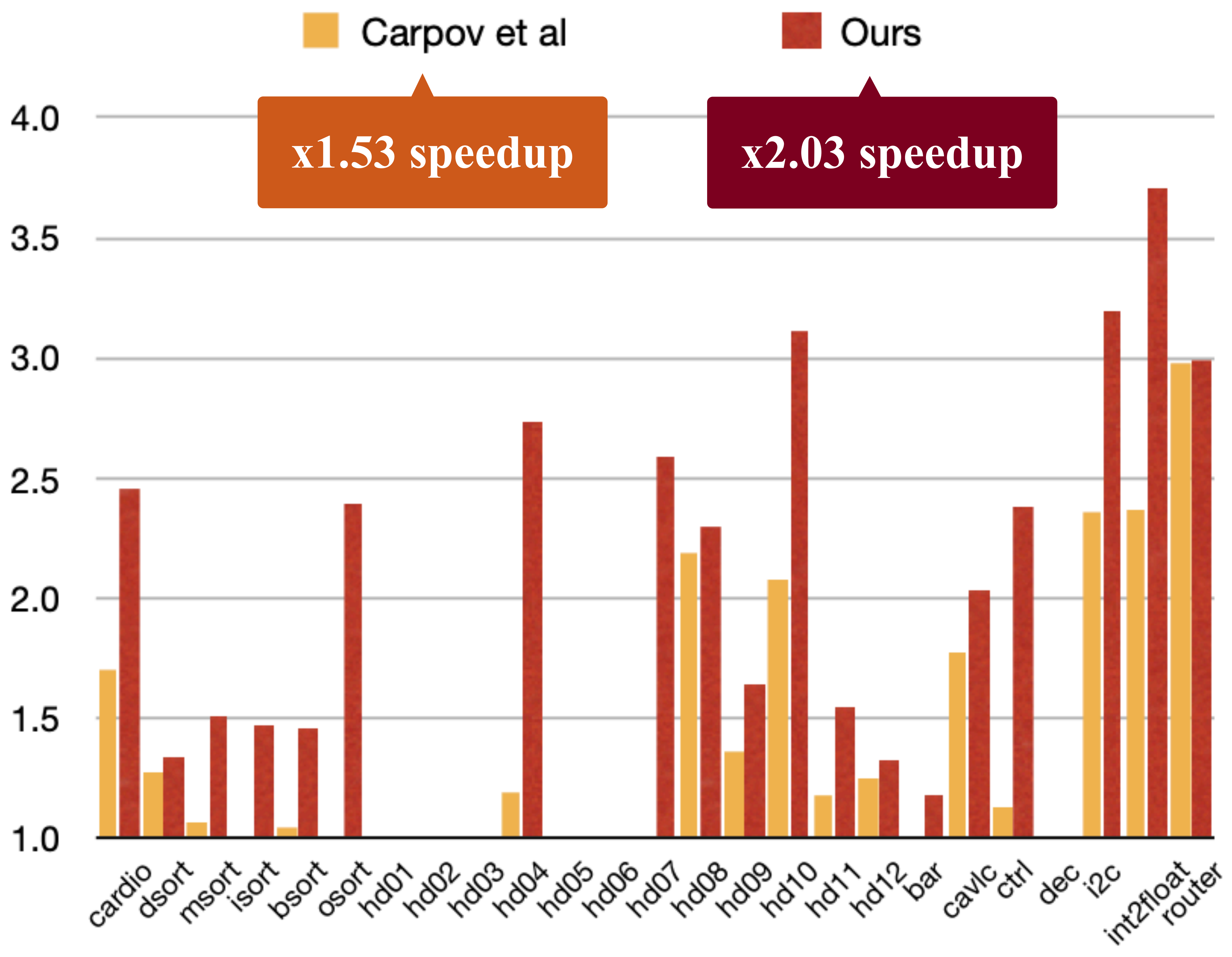
Speedup



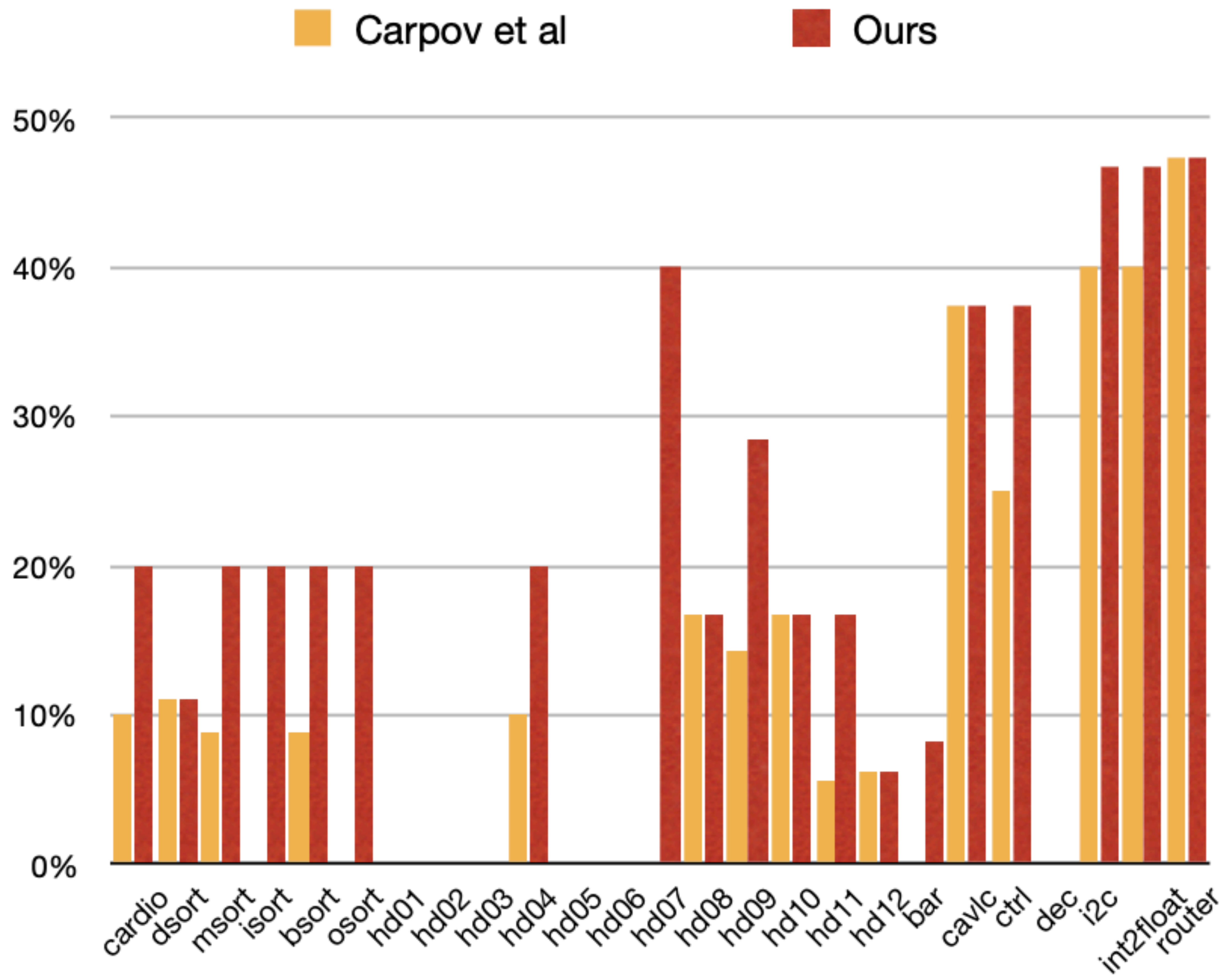
Depth Reduction

Lobster Performance (2/5)

Optimization Results of Lobster and the baseline



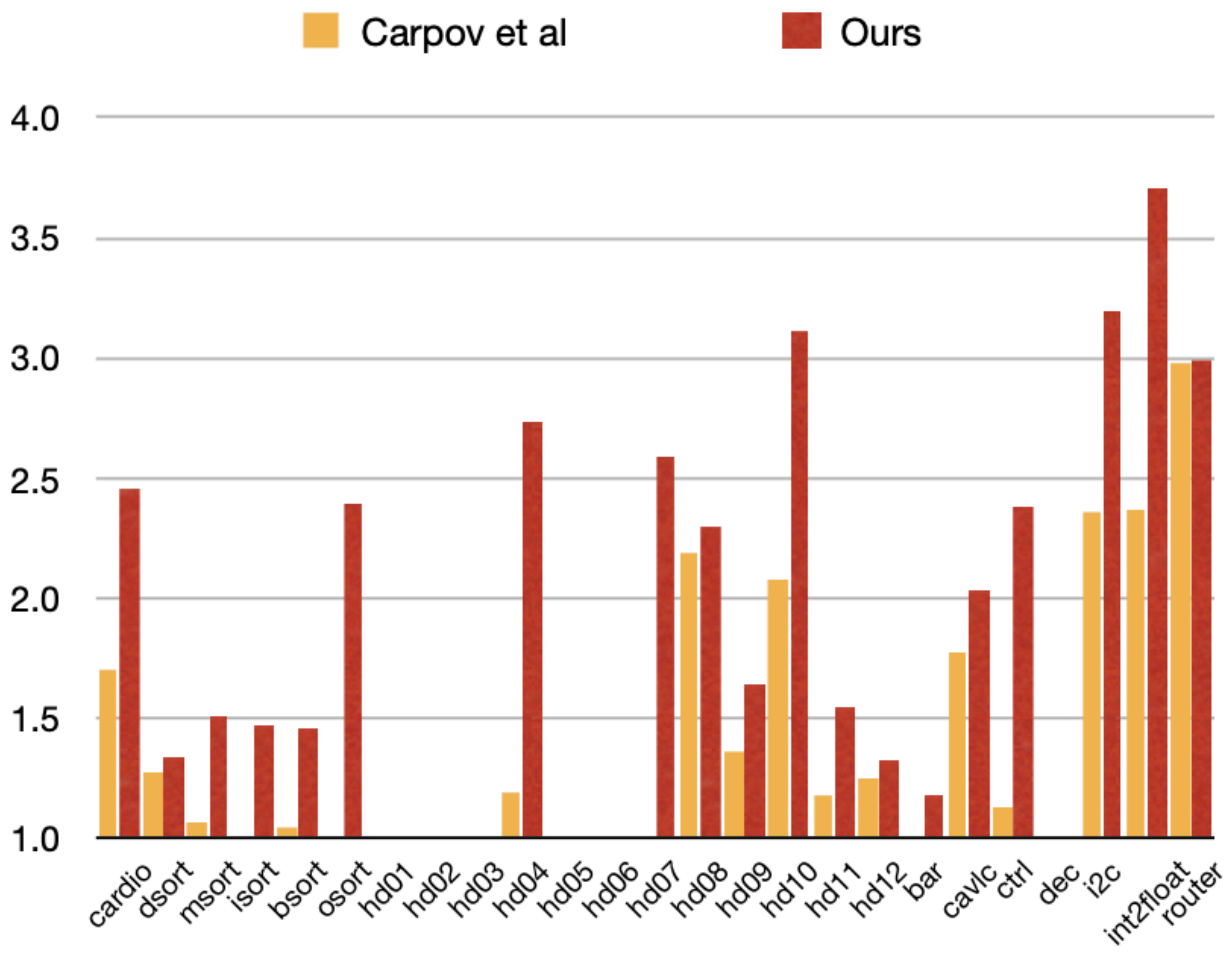
Speedup



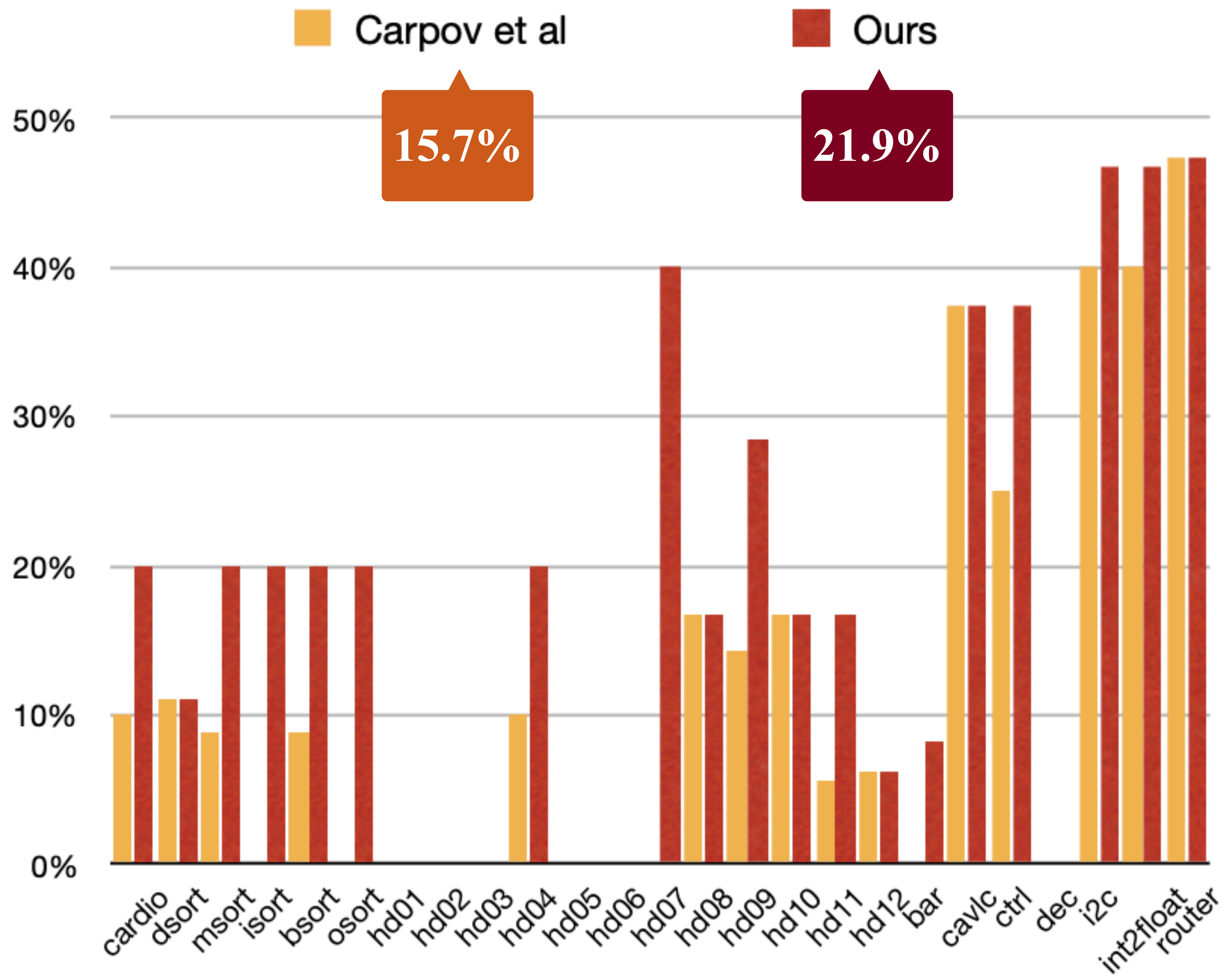
Depth Reduction

Lobster Performance (2/5)

Optimization Results of Lobster and the baseline



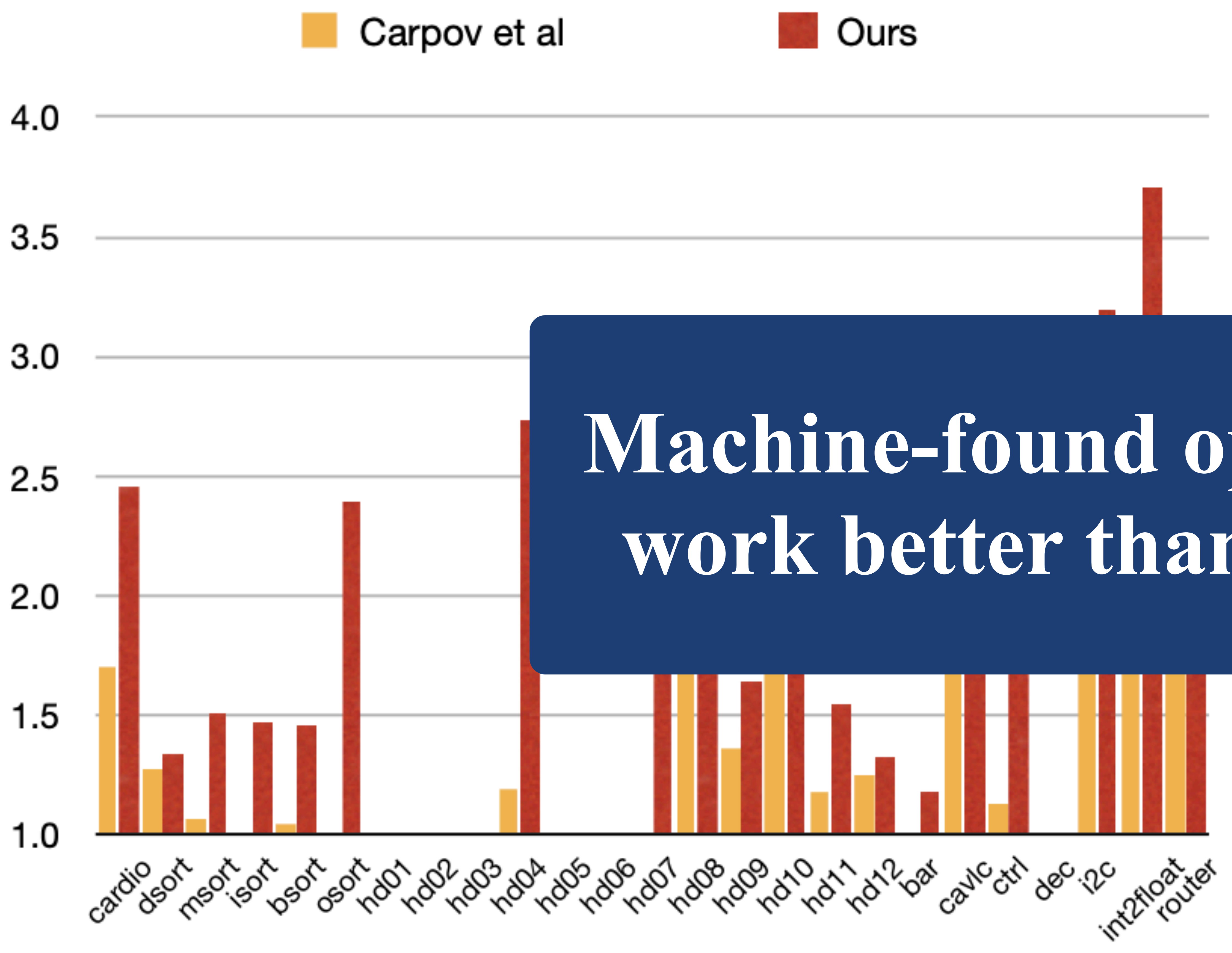
Speedup



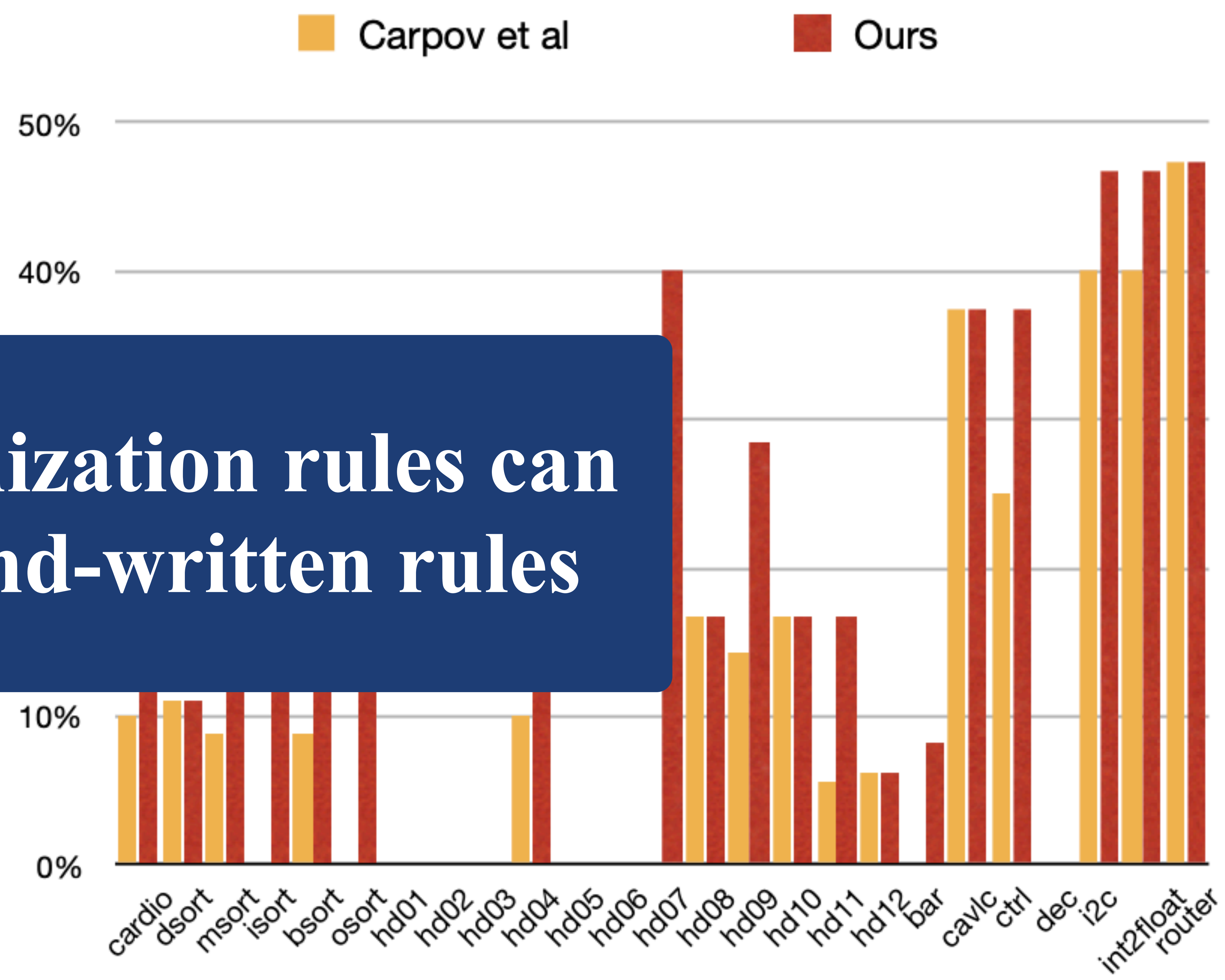
Depth Reduction

Lobster Performance (2/5)

Optimization Results of Lobster and the baseline



Speedup

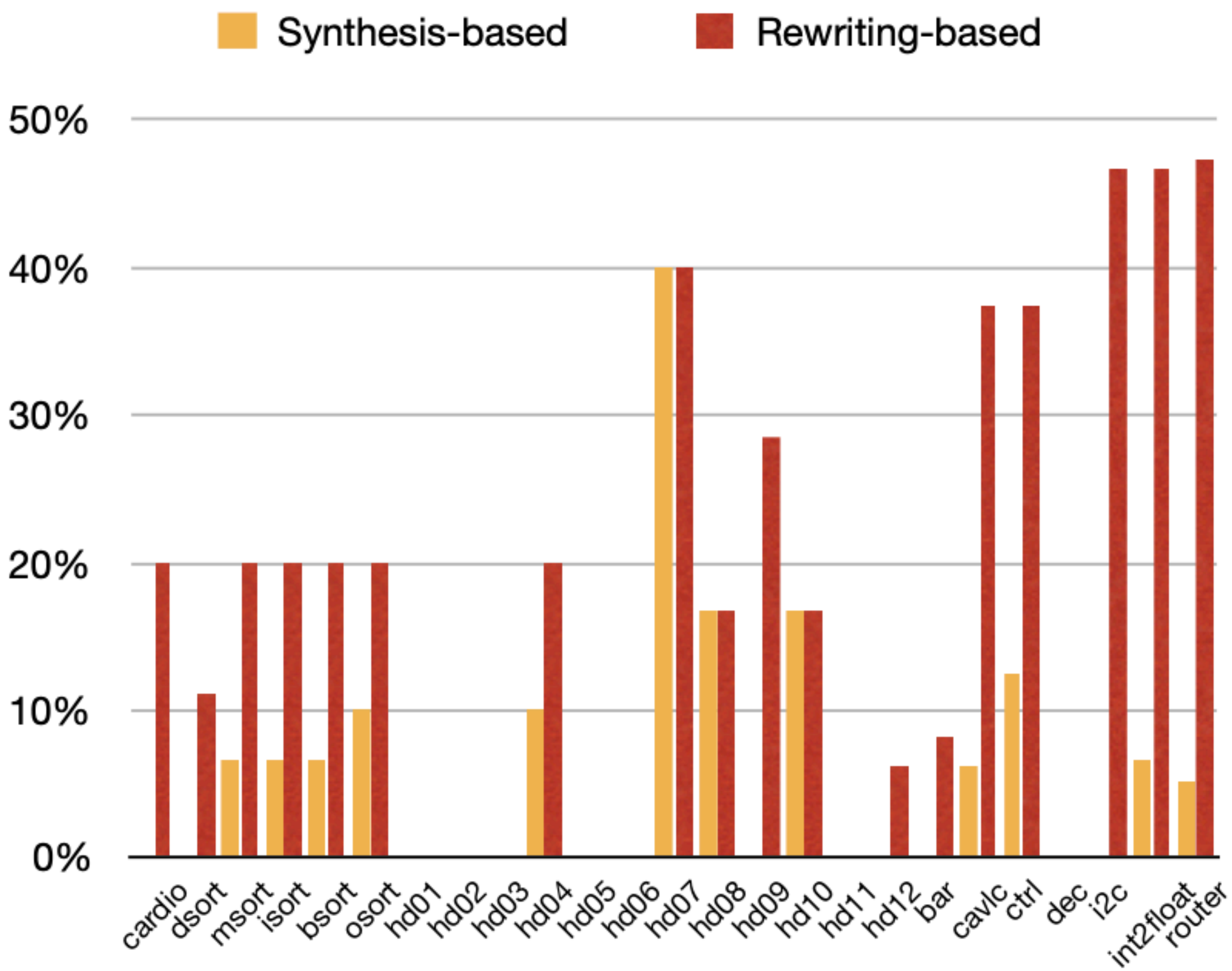


Depth Reduction

Machine-found optimization rules can work better than hand-written rules

Lobster Performance (3/5)

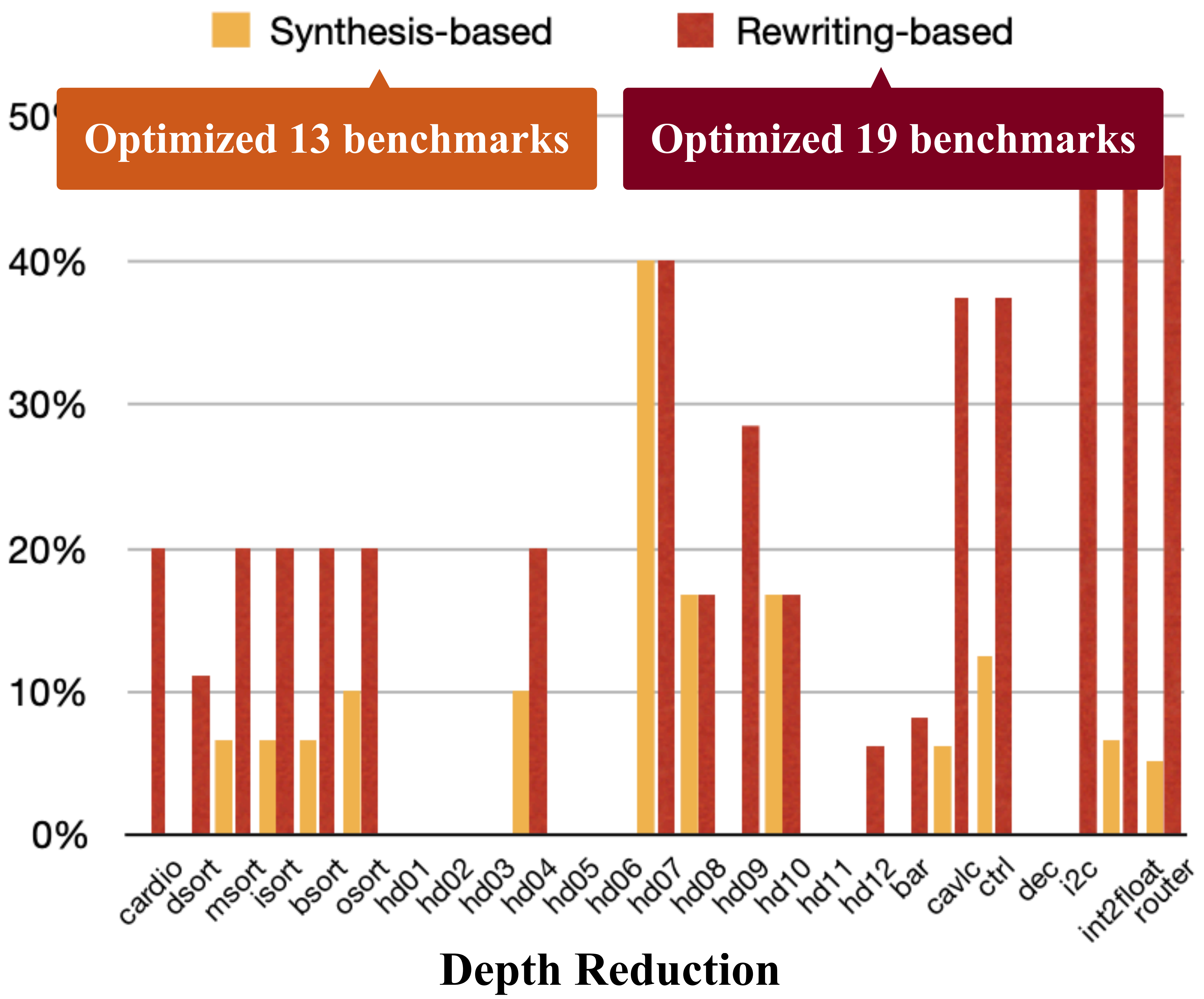
Efficacy of Reusing Learned Optimization Patterns



Depth Reduction

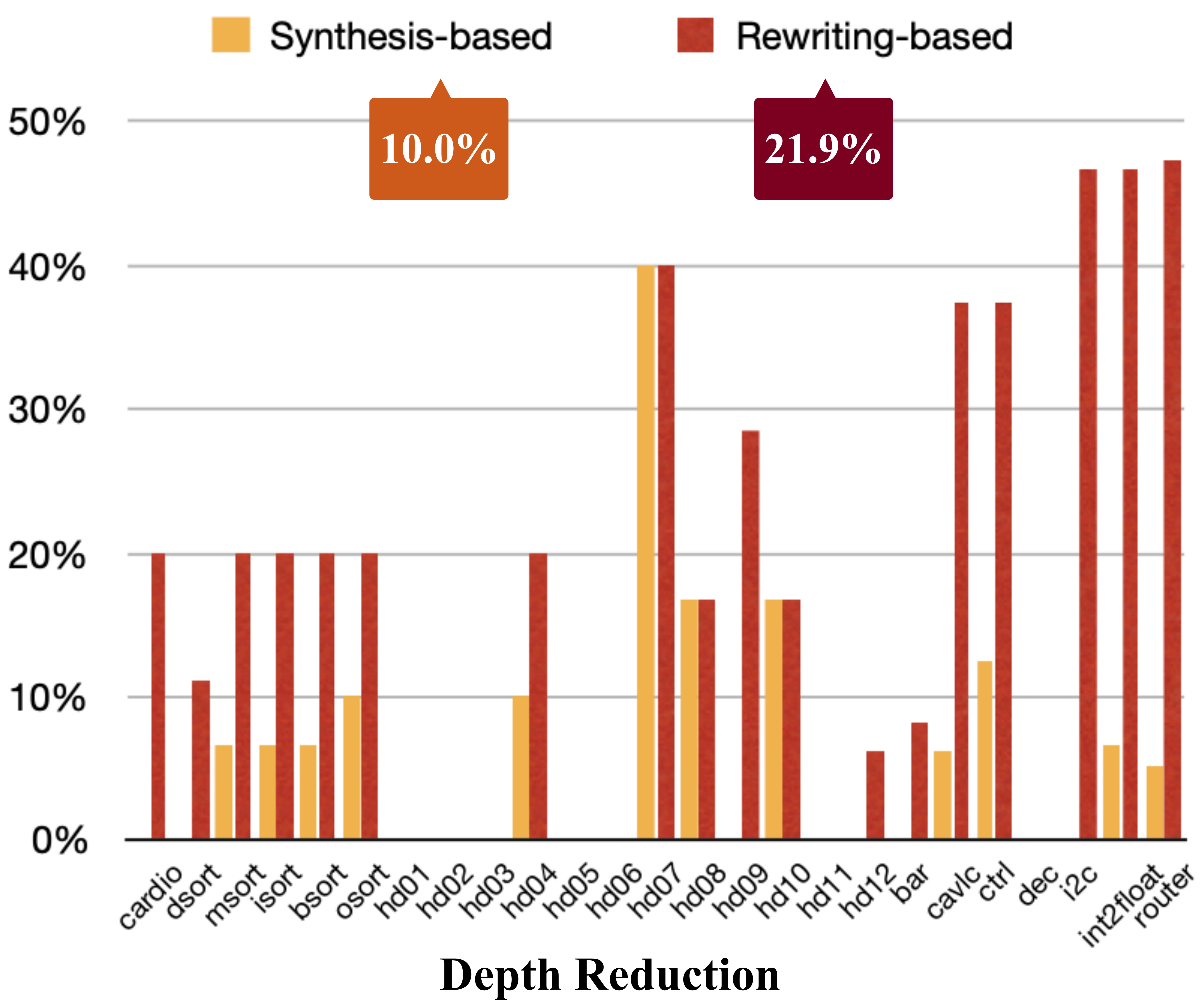
Lobster Performance (3/5)

Efficacy of Reusing Learned Optimization Patterns



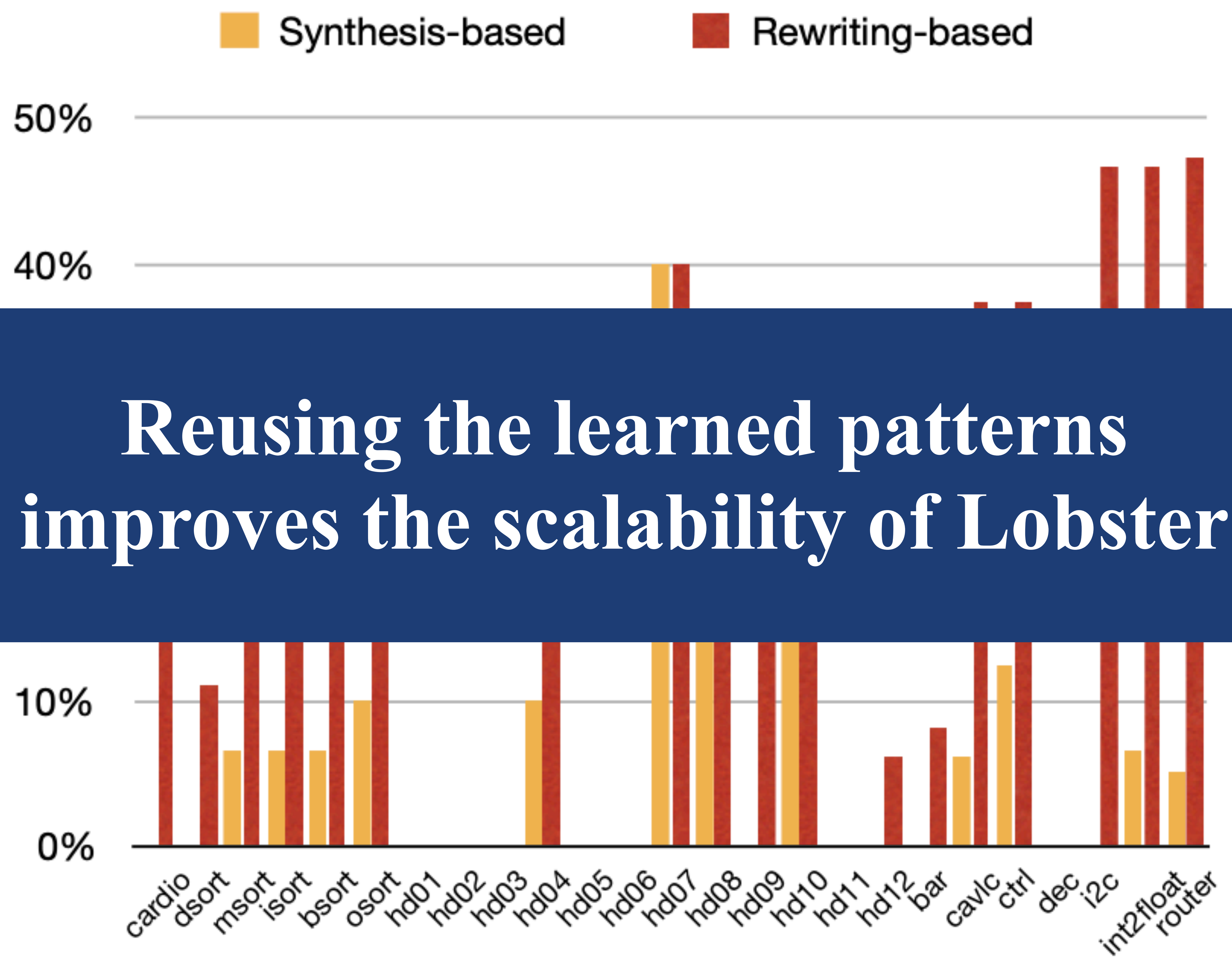
Lobster Performance (3/5)

Efficacy of Reusing Learned Optimization Patterns



Lobster Performance (3/5)

Efficacy of Reusing Learned Optimization Patterns

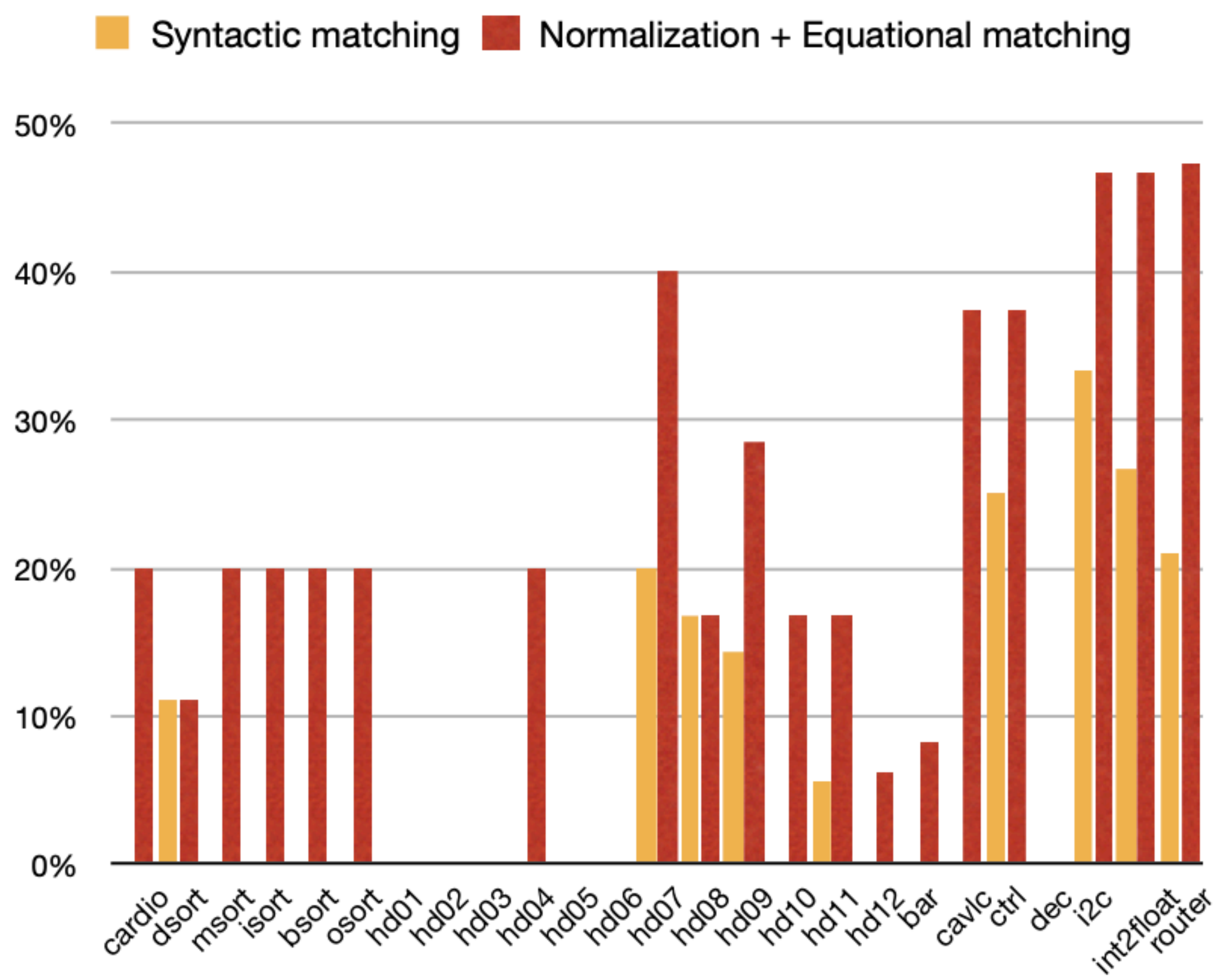


Reusing the learned patterns improves the scalability of Lobster

Depth Reduction

Lobster Performance (4/5)

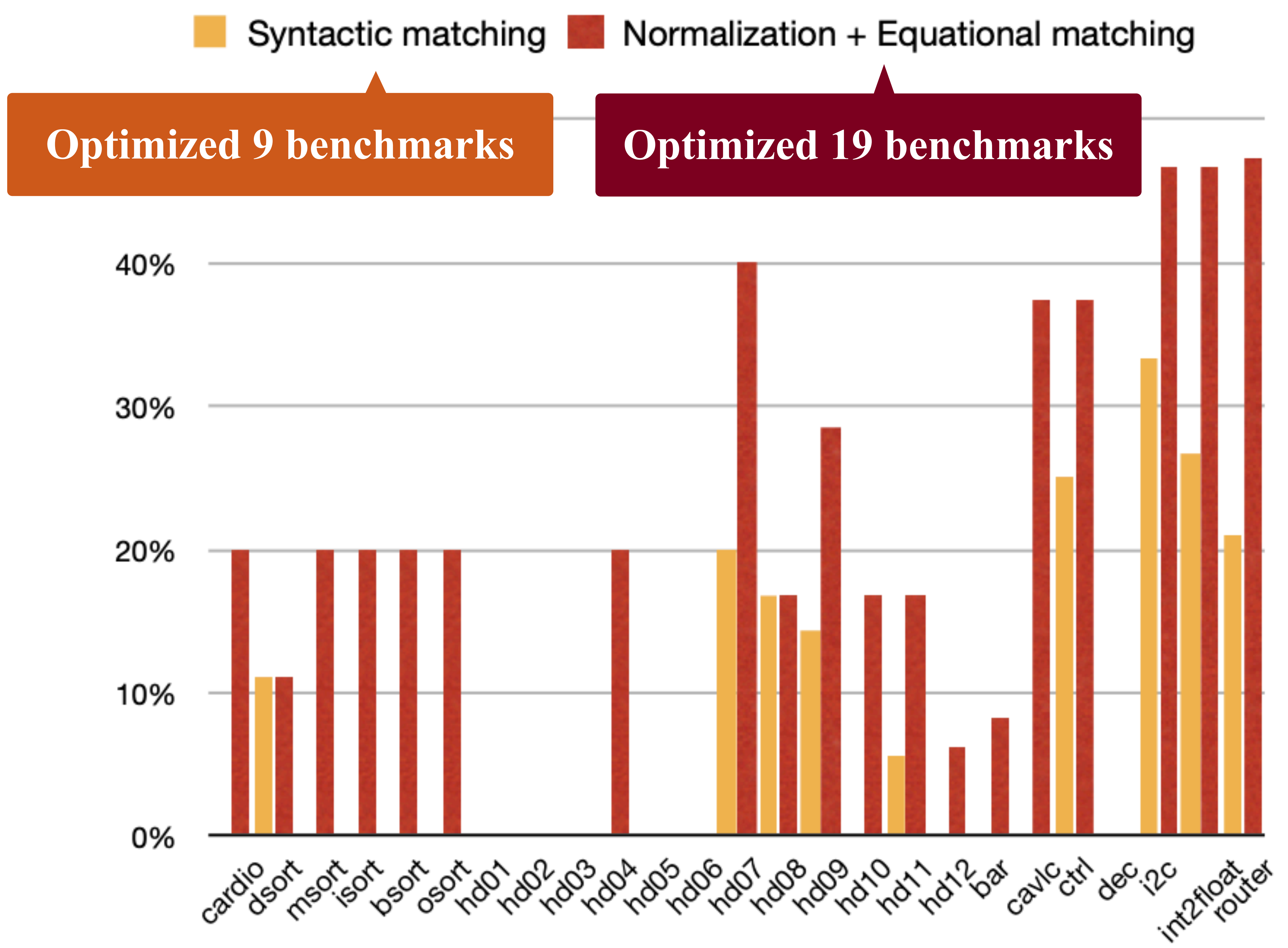
Effectiveness of Equational Term Rewriting



Depth Reduction

Lobster Performance (4/5)

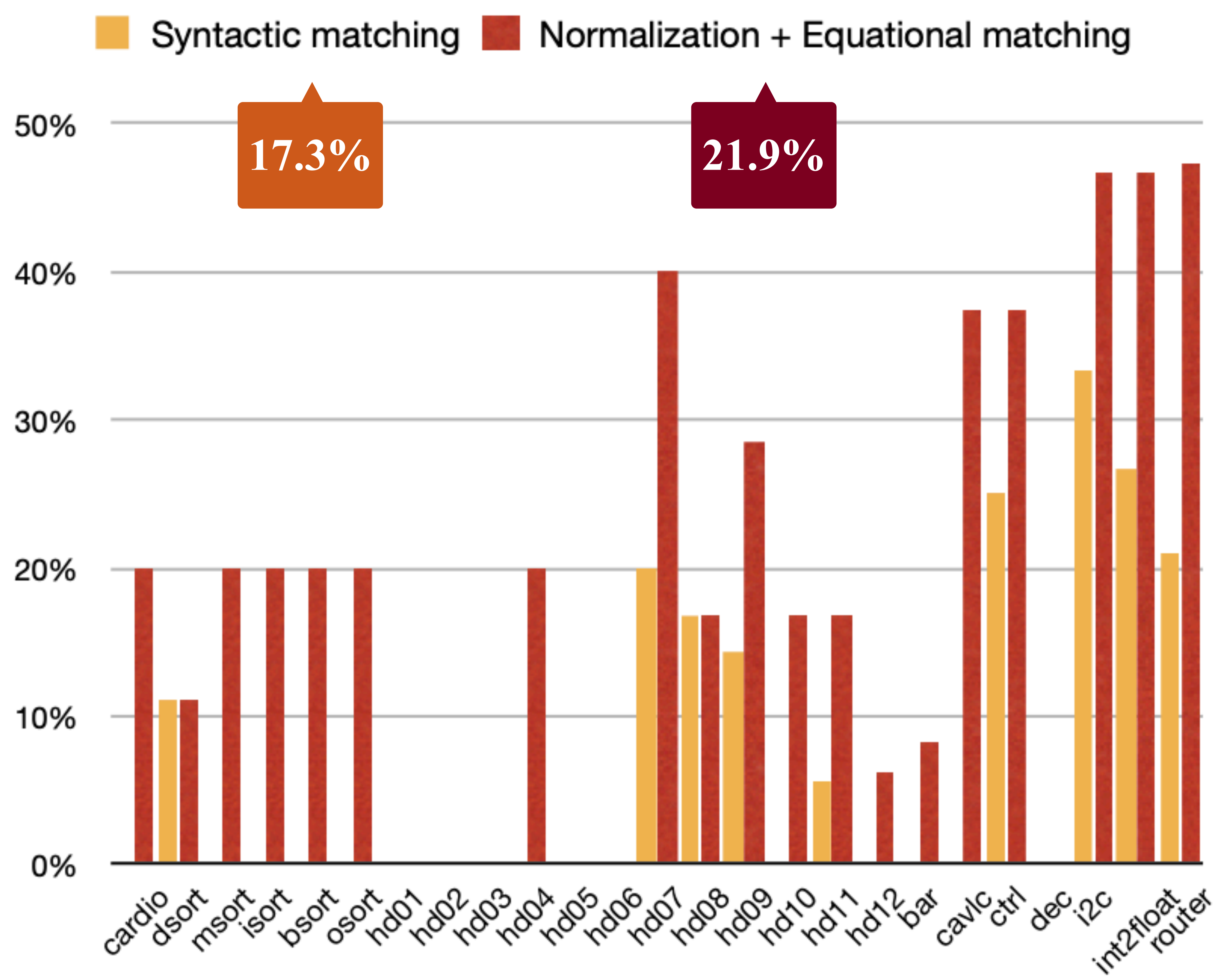
Effectiveness of Equational Term Rewriting



Depth Reduction

Lobster Performance (4/5)

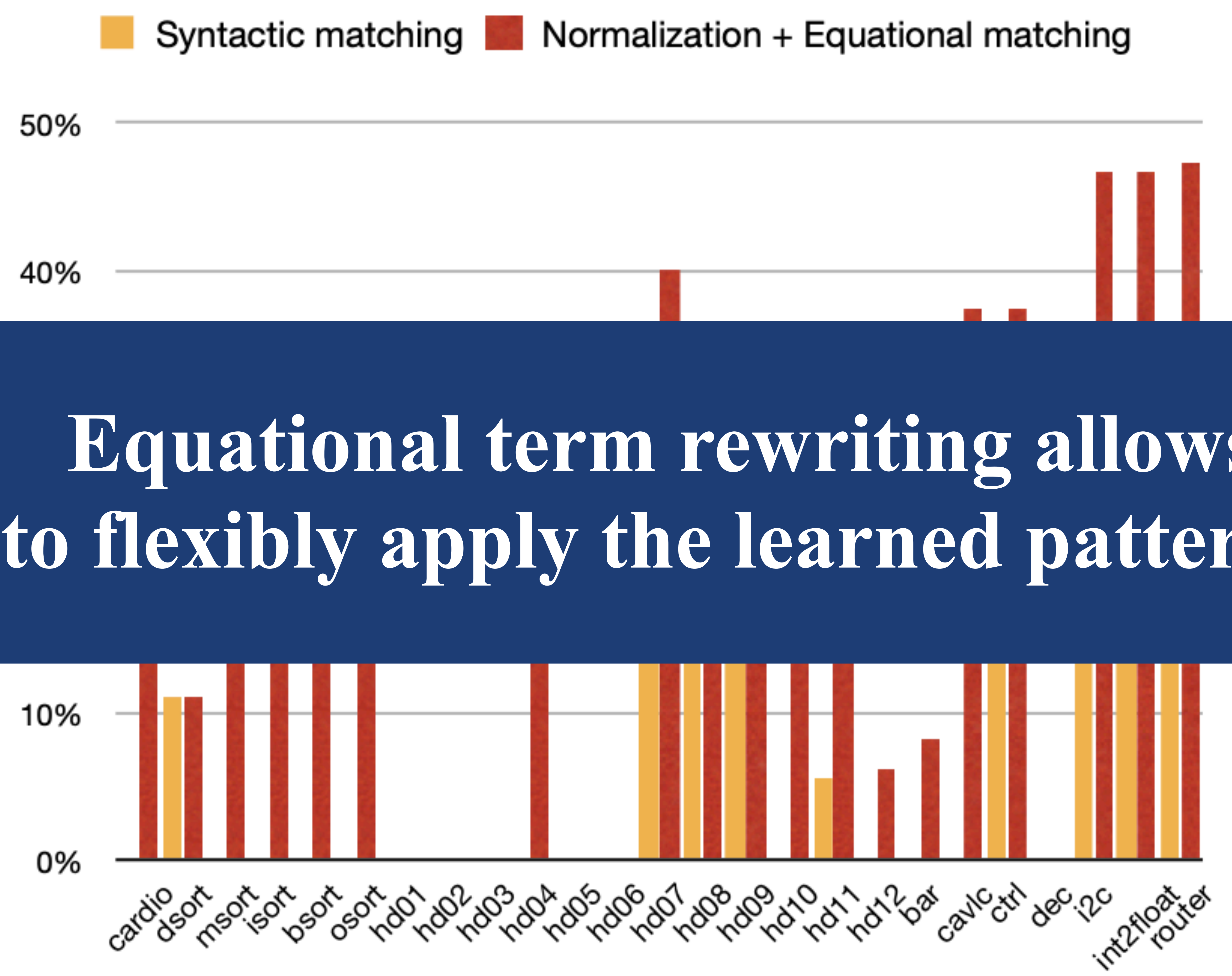
Effectiveness of Equational Term Rewriting



Depth Reduction

Lobster Performance (4/5)

Effectiveness of Equational Term Rewriting

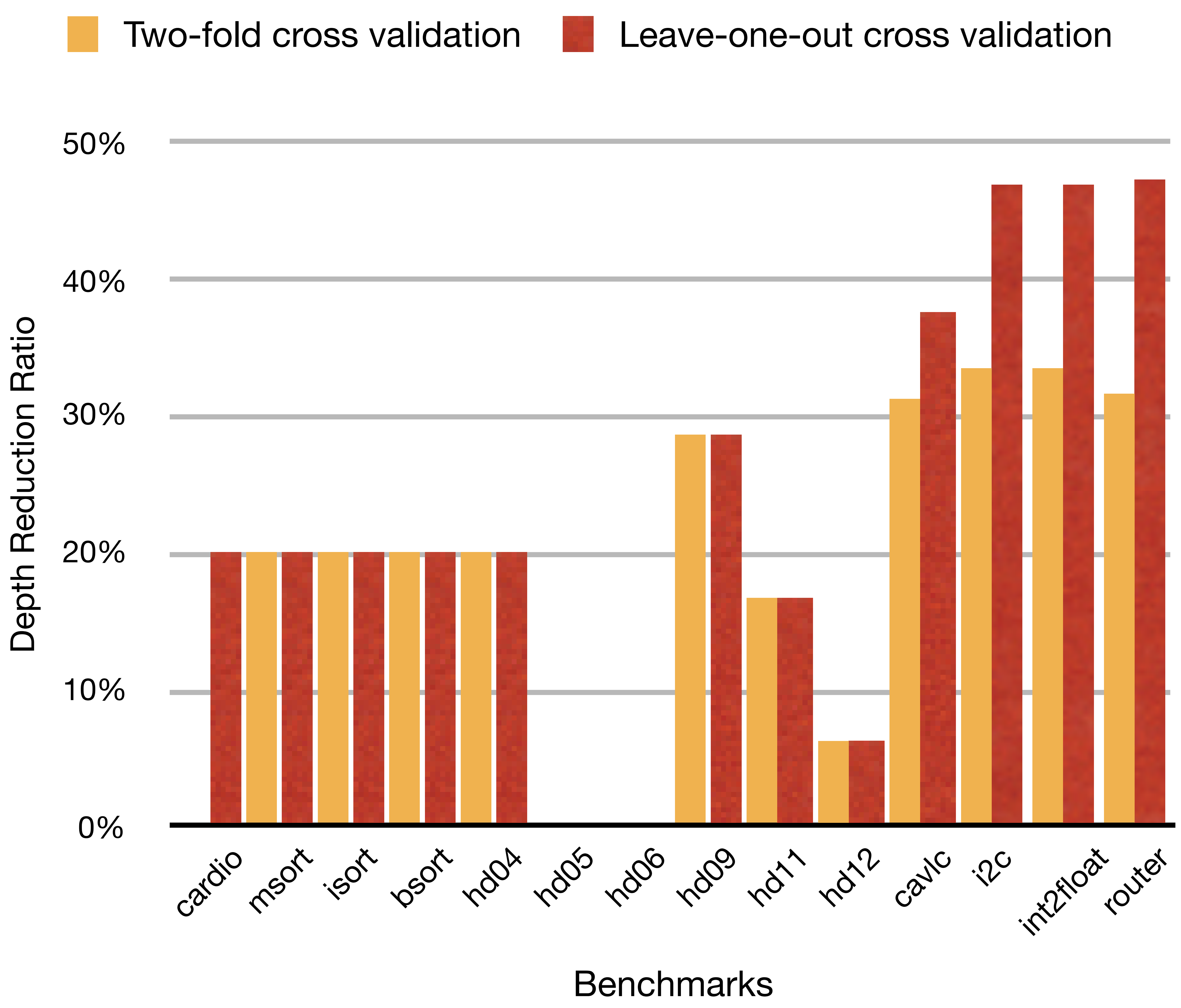


Equational term rewriting allows to flexibly apply the learned patterns

Depth Reduction

Lobster Performance (5/5)

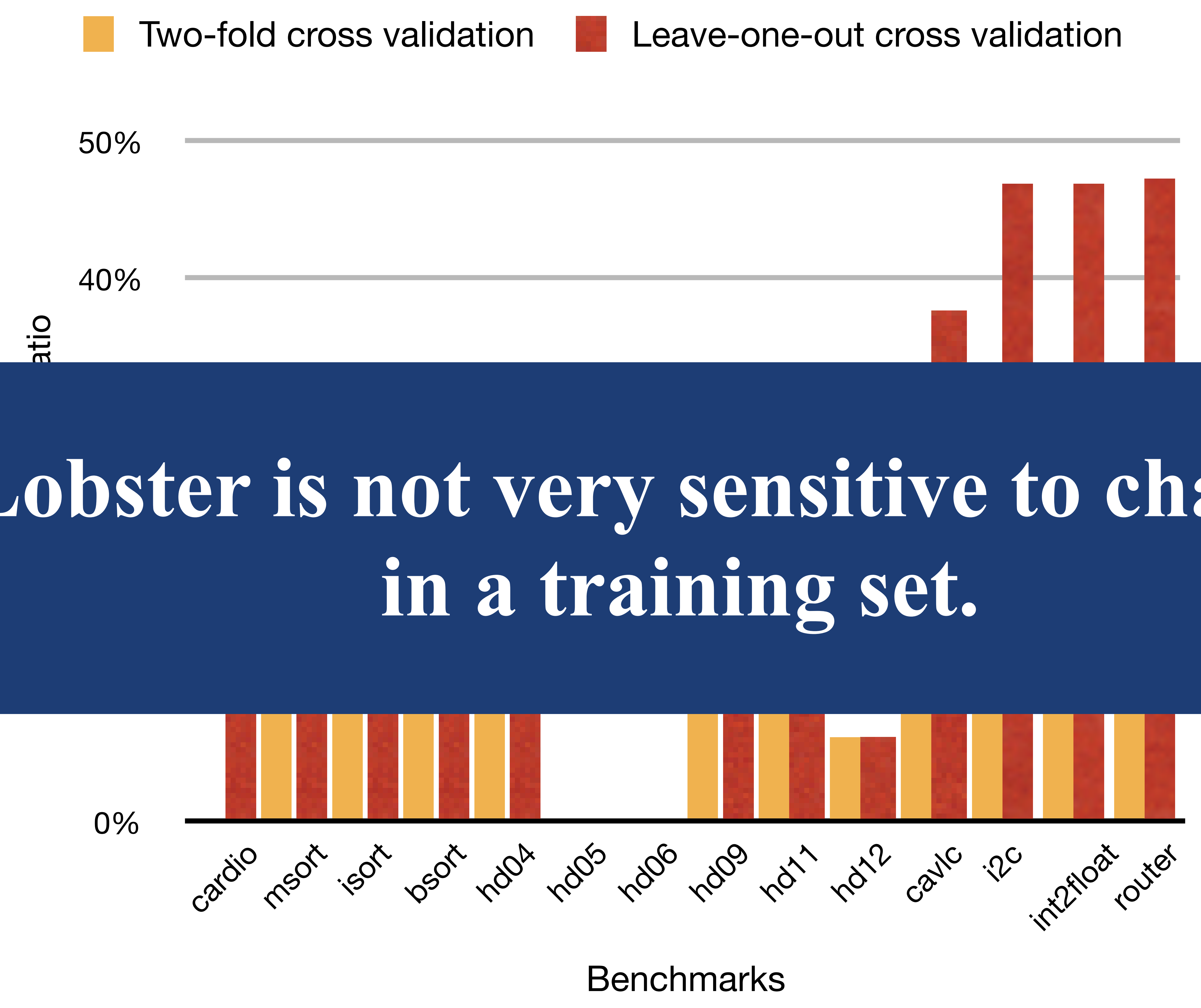
Effectiveness of Equational Term Rewriting



Depth Reduction

Lobster Performance (5/5)

Effectiveness of Equational Term Rewriting



Lobster is not very sensitive to changes in a training set.

Depth Reduction

In the Paper…

- Detailed description of synthesis via localization
- Formalized Equational Term Rewriting
- Detailed description of experiment results

Thank you!