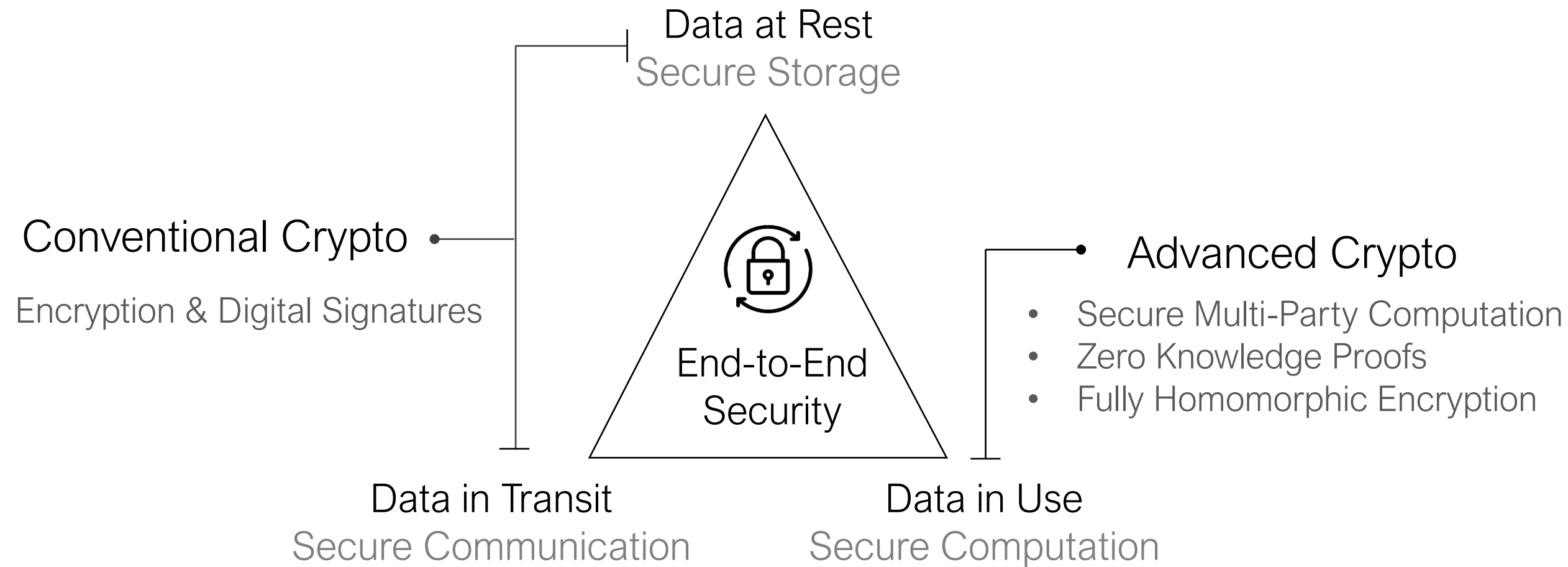


# **HECO**: Fully **H**omomorphic **E**ncryption **C**ompiler

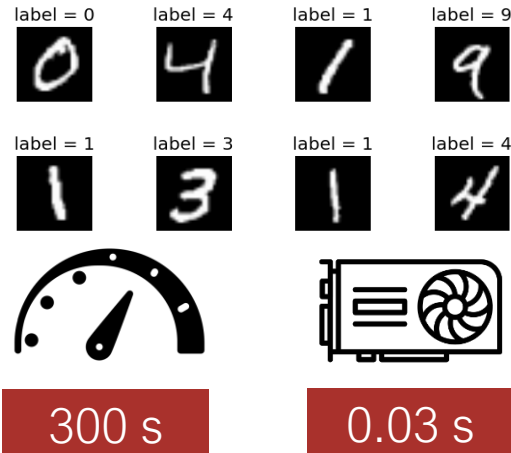
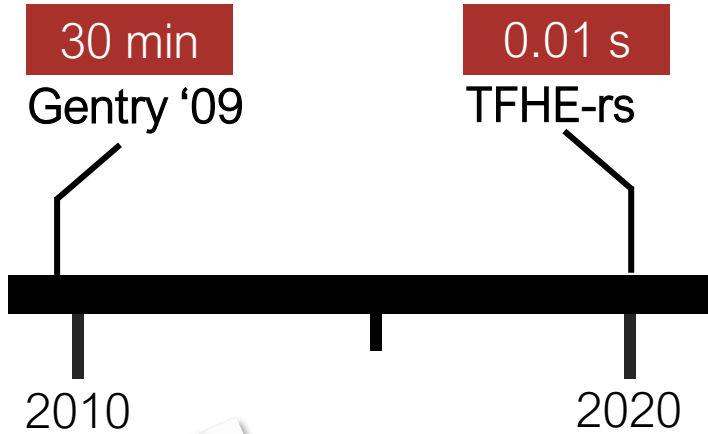
Alexander Viand, Patrick Jattke, Miro Haller, Anwar Hithnawi

***ETH*** zürich

# Modern Cryptography



# FHE in Practice



ImageNet (ResNet-18) in 16.4 s

**Fully Homomorphic Encryption Using Ideal Lattices**  
 Craig Gentry, MIT CSAIL  
 gentry@gcs.mit.edu  
 Abstract—We describe a workable fully homomorphic encryption scheme that supports a wide range of operations on encrypted data. The scheme is based on a new form of lattice-based cryptography.

**Implementing Gentry's Fully-Homomorphic Encryption Scheme**  
 Craig Gentry and Shai Halevi  
 IBM Research  
 Abstract—We describe a workable fully homomorphic encryption scheme that supports a wide range of operations on encrypted data. The scheme is based on a new form of lattice-based cryptography.

**CONCRETE: Concrete Operates on Ciphertexts Rapidly by Extending TFHE**  
 Baris Chalkitt, Mani Jeyaraman, Damien Legier, Jean-Baptiste Orfila, Samuel Tap  
 https://arxiv.org/abs/1905.02922  
 Abstract—Fully homomorphic encryption (FHE) extends traditional encryption schemes to allow one to directly compute on encrypted data without requiring access to the decryption key. This paper introduces CONCRETE, an open source library designed to run that does not require access to the decryption key. It provides a user-friendly interface for performing operations on encrypted data with support for arbitrary operations, including a programmable bootstrapping operation. CONCRETE is available at <https://github.com/IBM/CONCRETE> and on [arXiv:1905.02922](https://arxiv.org/abs/1905.02922).

**CryptoNets: Applying Neural Networks to Encrypted Data with High Throughput and Low Latency**  
 Nathan Dowling, Ran Gilad-Bachrach, Kim J. Grant, J. Alex Kluge, and Peter D. McDaniel  
 Microsoft Research, Princeton University  
 February 2019  
 Abstract—Applying machine learning to a problem which involves sensitive data often requires secure computation. This paper presents CryptoNets, a system that allows one to train and apply neural networks to encrypted data. CryptoNets is designed to be used in a cloud setting, where the data is encrypted by the client and the computation is performed by the server. CryptoNets is designed to be used in a cloud setting, where the data is encrypted by the client and the computation is performed by the server.

**GAZELLE: A Low Latency Framework for Secure Neural Network Inference**  
 Chirag Juvekar, Vinod Vaikuntanathan, MIT CSAIL  
 Anantha Chandrakasan, MIT MTL  
 anantha@mit.edu  
 Abstract—The growing popularity of cloud-based machine learning raises a natural question about the privacy guarantees that can be provided in such a setting. One work tackles this problem in the context where a client wishes to classify private information using a convolutional neural network (CNN) trained by a server. Our goal is to build efficient protocols where the client can use the server's computation power without revealing their data. Our framework, GAZELLE, is a low-latency framework for secure neural network inference.

**HyPHEN: A Hybrid Packing Method and Optimizations for Homomorphic Encryption-Based Neural Networks**  
 Donghwan Kim, Jaiyoung Park, Jongmin Kim, Sanggyo Kim, and Jung Ho Ahn  
 Seoul National University  
 {dohwan.kim, jypark}@pcslab.skku.ac.kr  
 {jongmin.kim, vob987, qa3h}@snu.ac.kr  
 Abstract—Convolutional neural network (CNN) inference using fully homomorphic encryption (FHE) is a promising private inference (PI) solution due to the capability of FHE that enables protecting the privacy of sensitive user data. However, prior FHE-based CNN (HCNN) implementations are far from being practical due to the high computational and memory overheads of FHE. To overcome this limitation, we present HyPHEN, a deep learning framework that features an efficient FHE construction algorithm, data packing methods (hybrid packing and image tiling), and FHE-specific optimizations. Such enhancements enable HyPHEN to substantially reduce the memory footprint, ciphertext rotation and bootstrapping. As a result, HyPHEN brings the latency of HCNN CIFAR-10 inference to the level of practical level. In this paper, we first, the users of such a service will rightfully be given away, and secondly, the model, which has been trained on private patient data, and may reveal information about particular patients, violating their privacy and perhaps even HIPAA. A second solution that comes to mind is for the hospital to adopt the "machine learning as a service" paradigm and build a web service that hosts the model and provides predictions for a small fee. However, this is also undesirable for at least two reasons: first, the users of such a service will rightfully be given away, and secondly, the model, which has been trained on private patient data, and may reveal information about particular patients, violating their privacy and perhaps even HIPAA.

# FHE Deployment



# FHE Deployment

The Register

SECURITY

### Microsoft Edge goes homomorphic: Nobody will see your credentials... but you'll need to sign in to use it

Has yo

Richard

Micro

versio

The r

prev

"tra

The

an

W

M

**Protect your passwords**

Let Microsoft Edge check passwords you've saved in the browser and alert you if they've been compromised on the internet.  On

As well as pe

release alongside improved controls fo

microphone, and camera. Other browsers, such as Google's C

available too.

It has also been made easier to enable tracking prevention in Strict mode while InPrivate because "No one wants a personalized ad based on browsing history ruining all the fun," said Microsoft, using the example of a surprise gift's recipient perhaps seeing an ad over the

SKT 9:10

SKT 5:39

57%

03월 02일부터 03

안전하

지내

보호합니다

호화되어

에게도

드립니다.

수 있지만,

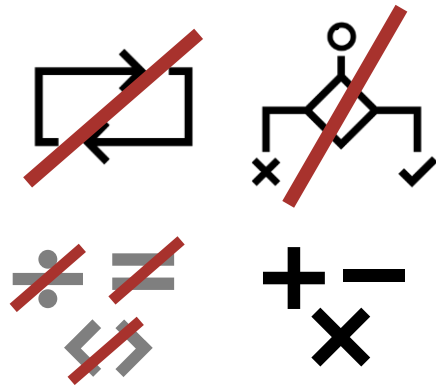
있습니다.

Developing and Deploying FHE Applications is  
**Notoriously Hard**

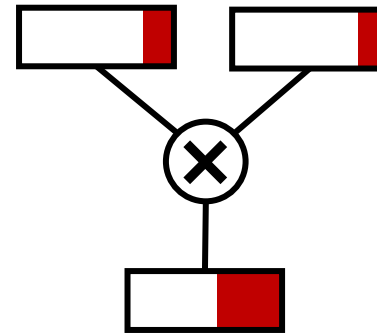
# FHE Programming Paradigm

```
void hd(vector<bool>u,  
        vector<bool>v)  
{  
    int sum = 0;  
    for(int i = 0;  
        i < v.size();  
        ++i)  
    {  
        sum += (v[i]!=u[i]);  
    }  
}
```

Application Dependent



Limited Expressiveness

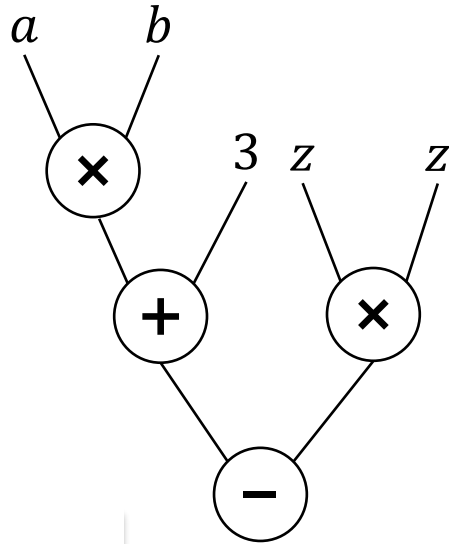


Noise Management



SIMD Batching

# Evolution of FHE Development



```
void f(...)  
{  
  mul_inp(a,b);  
  relin_inp(a);  
  add_plain_inp(a,3);  
  square_inp(z,z);  
  relin_inp(a);  
  sub_inp(a,z);  
  return a;  
}
```

Implementing Geometric  
Encryption

Algorithms in HElib

Simple Encrypted Arithmetic Library 2.3.1

## 1 Introduction

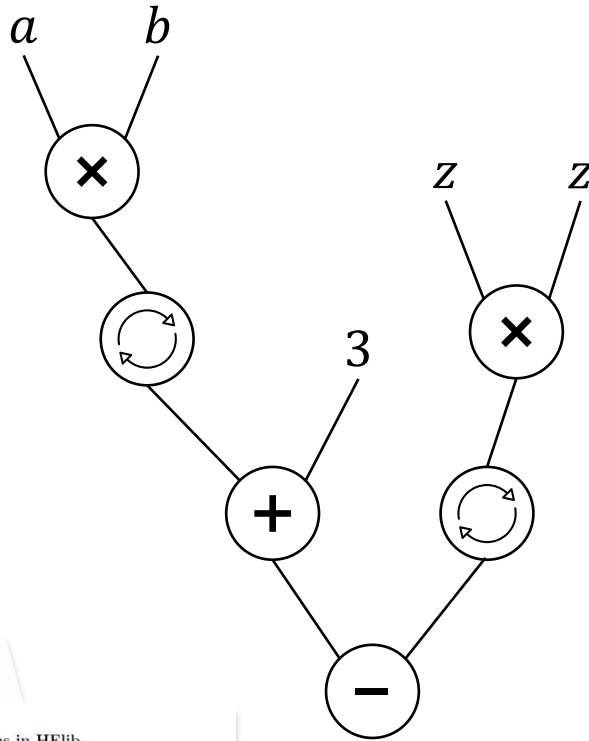
Traditional encryption schemes, both symmetric and asymmetric, were not designed to respect any algebraic structure of the plaintext and ciphertext spaces, i.e. no computations can be performed on the ciphertext in a way that would pass through the encryption to the underlying plaintext without using the secret key, and such a property would in many contexts be considered a vulnerability. Nevertheless, this property has powerful applications, e.g. in outsourced (cloud) computation scenarios the cloud provider could use this to guarantee customer data privacy in the presence of both internal (malicious employees) and external (outside attacker) threats. An encryption scheme that allows computations to be done directly on encrypted data is said to be a *homomorphic encryption scheme*.

Some schemes, such as ElGamal (resp. e.g. Paillier), are multiplicative (resp. additively homomorphic), i.e. one algebraic operation (resp. multiplication (resp. addition) on plaintext data. The restriction to a single operation is due to the fact that arbitrary binary operations are not supported.

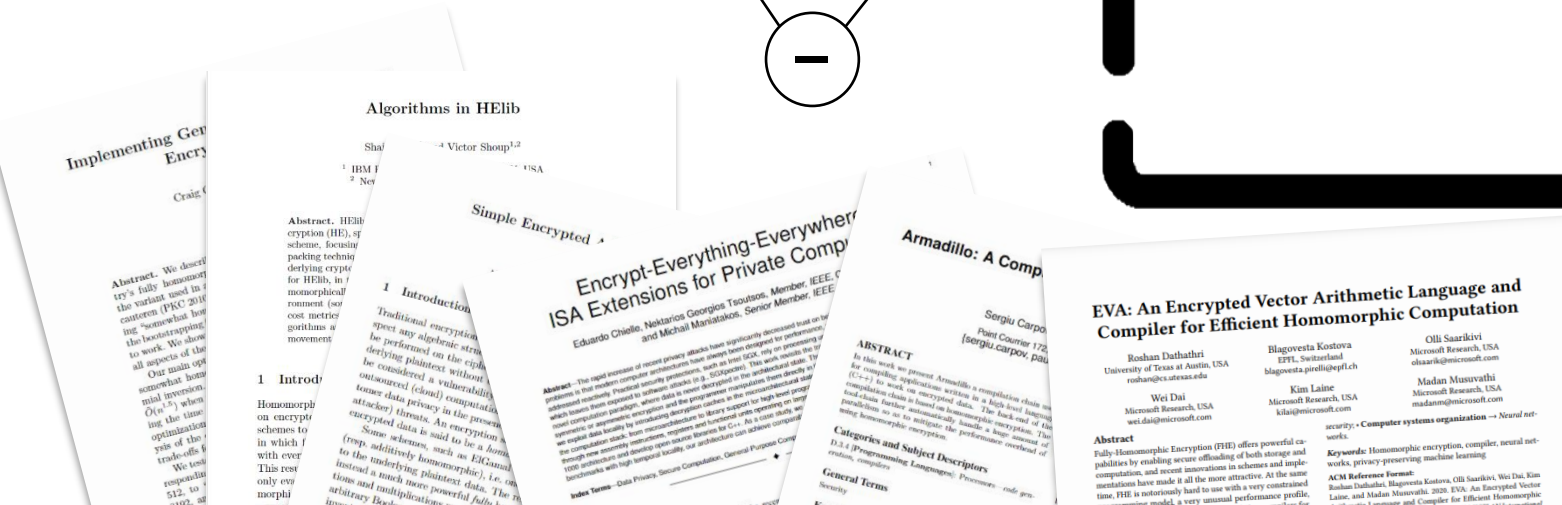
1 Introduction  
Homomorphi  
on encrypt  
in which f  
with ever  
This rest  
only ev  
morphi



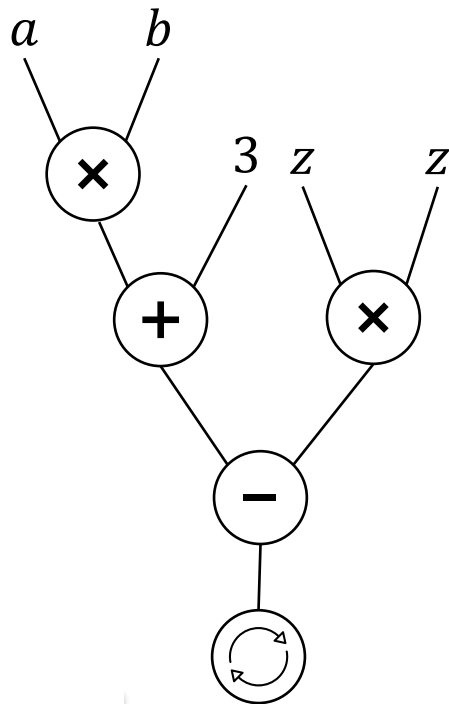
# Evolution of FHE Development



```
void f(...)
{
  ctxt ab = a*b + 3;
  return ab - z*z;
}
```



# Evolution of FHE Development



```
void f(...)
{
  ctxt ab = a*b + 3;
  return ab - z*z;
}
```

Algorithms in HElib

Simple Encrypted

Encrypt-Everything-Everywhere  
ISA Extensions for Private Computation

Armadillo: A Compiler for Efficient Homomorphic Encryption

EVA: An Encrypted Vector Arithmetic Language and Compiler for Efficient Homomorphic Computation

HECATE: Performance Analysis and Compiler for Homomorphic Encryption

ELASM: Error-Latency-Aware Scale Management for Fully Homomorphic Encryption

# Contributions

## Architecture

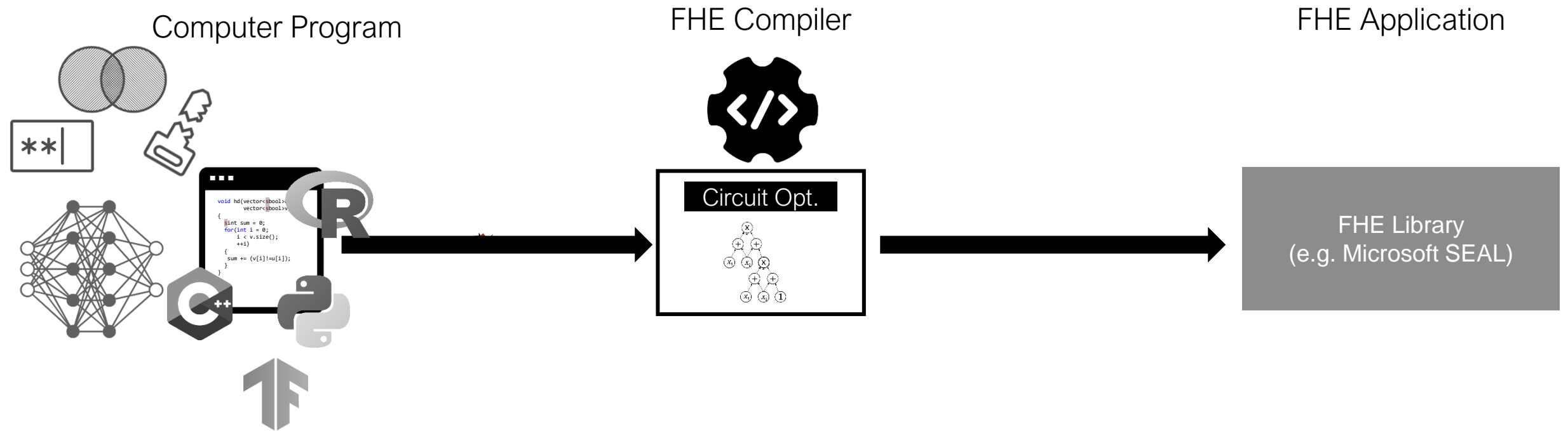
Address FHE development as an end-to-end problem

## Optimizations

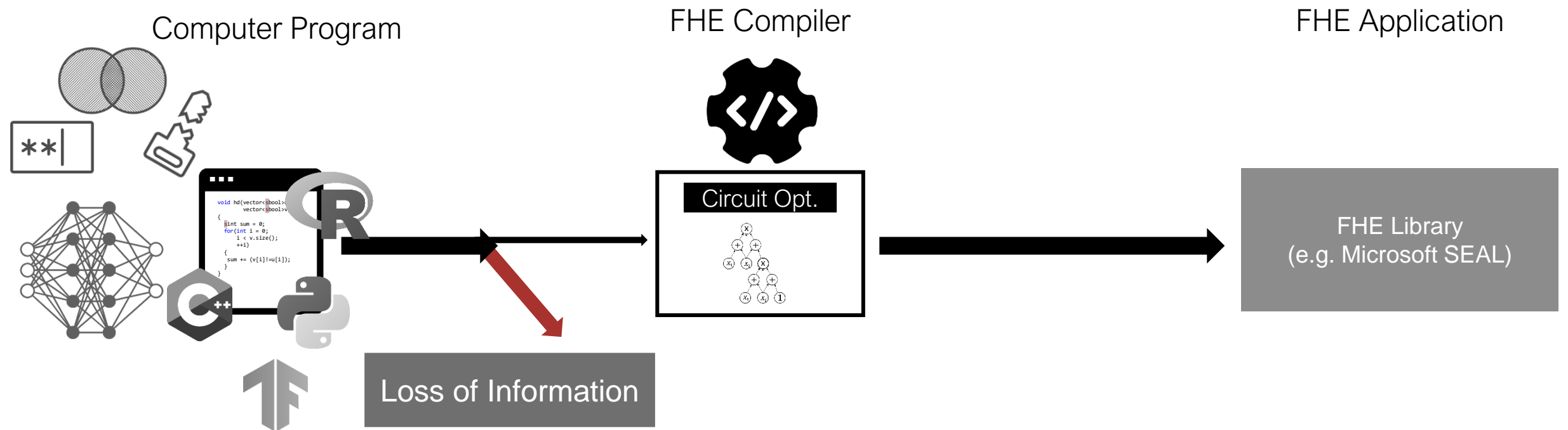
Order-of-magnitude speedups via high-level transformations

# End-to-End FHE Toolchain Architecture

# End-to-End FHE Toolchain



# End-to-End FHE Toolchain



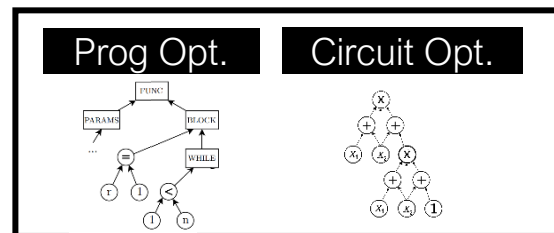
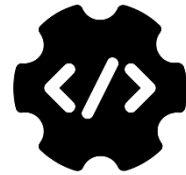
# End-to-End FHE Toolchain

Computer Program

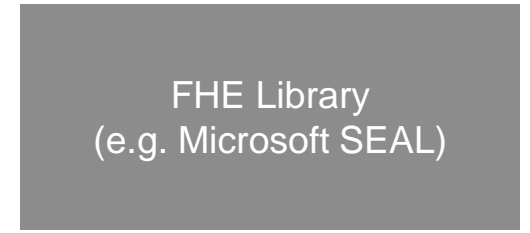
```
void hd(vector<bool>&u,  
vector<bool>&v)  
{  
    int sum = 0;  
    for(int i = 0;  
        i < v.size();  
        ++i)  
    {  
        sum += (v[i]==u[i]);  
    }  
}
```



FHE Compiler



FHE Application



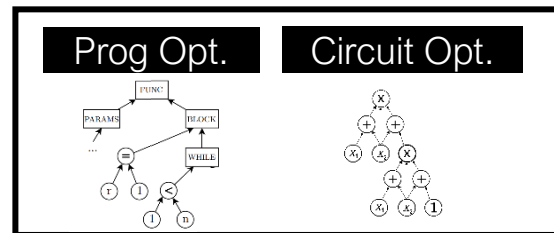
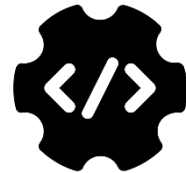
# End-to-End FHE Toolchain

Computer Program

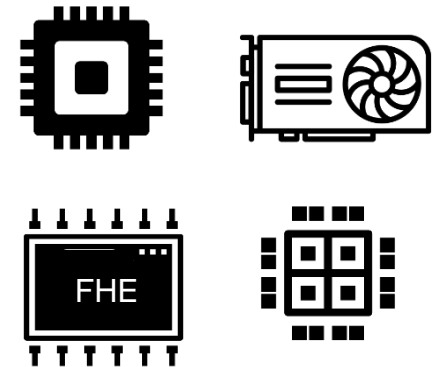
```
void hd(vector<bool>&u,  
vector<bool>&v)  
{  
  int sum = 0;  
  for(int i = 0;  
      i < v.size();  
      ++i)  
  {  
    sum += (v[i]==u[i]);  
  }  
}
```



FHE Compiler



Backend Targets





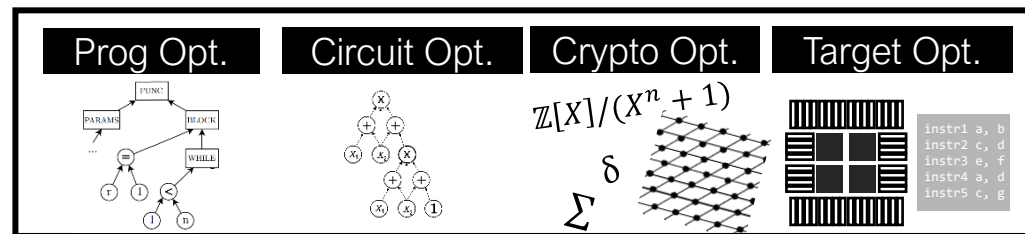
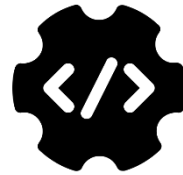
# End-to-End FHE Toolchain

Computer Program

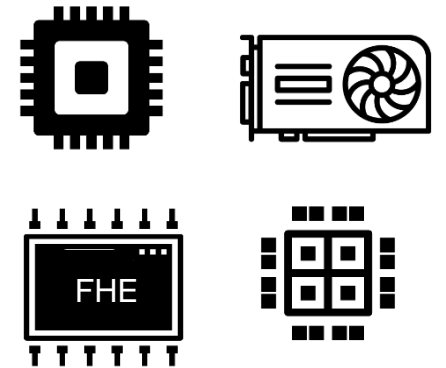
```
void hd(vector<bool>&u,  
vector<bool>&v)  
{  
    int sum = 0;  
    for(int i = 0;  
        i < v.size();  
        ++i)  
    {  
        sum += (v[i]==u[i]);  
    }  
}
```



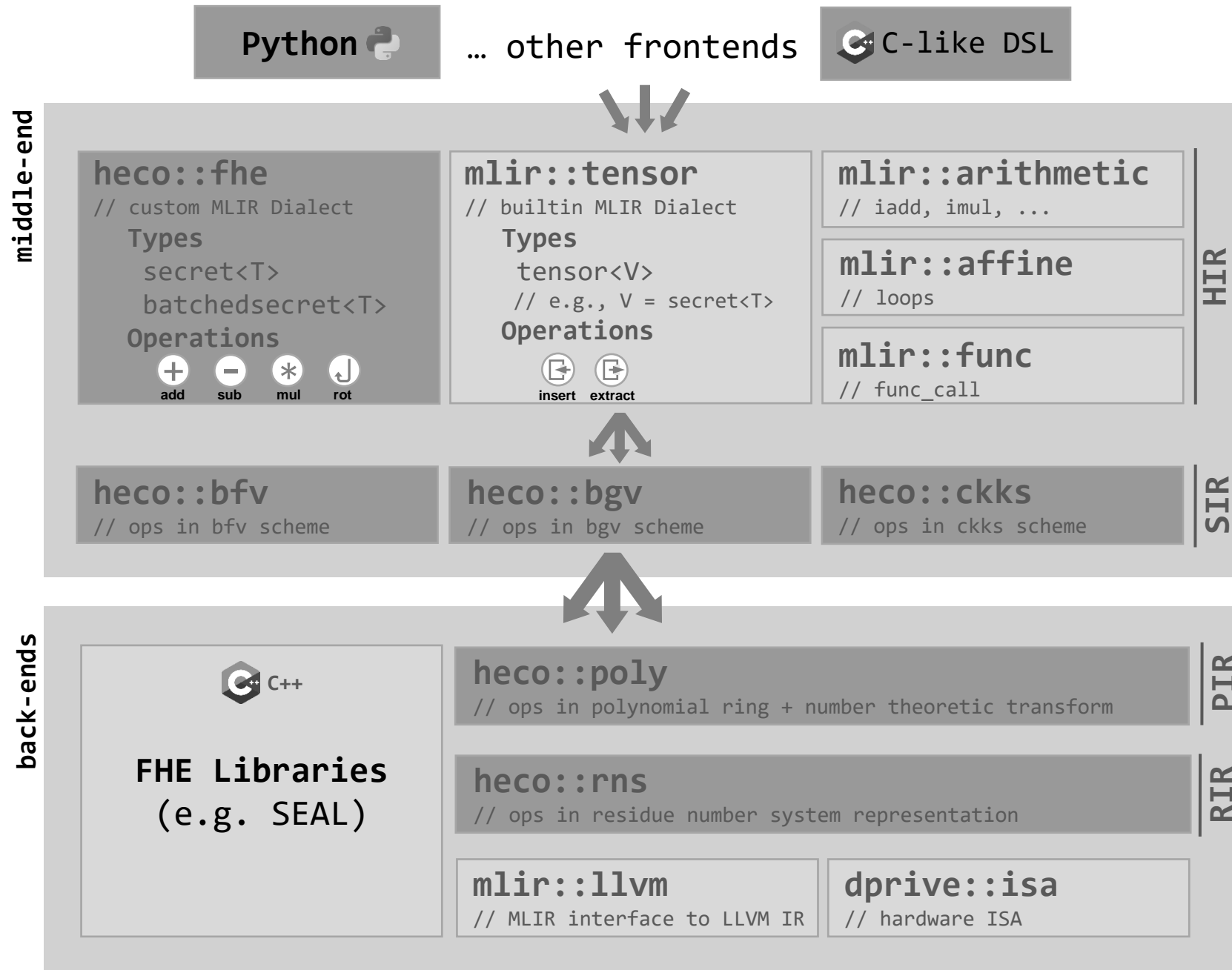
FHE Compiler



Backend Targets



# HECO Architecture

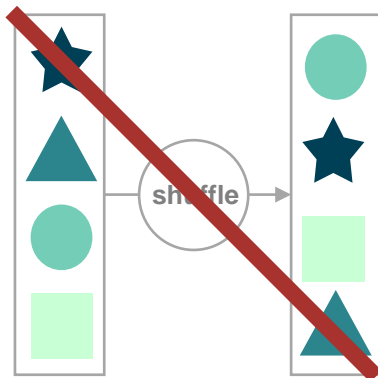


# SIMD Batching Optimization

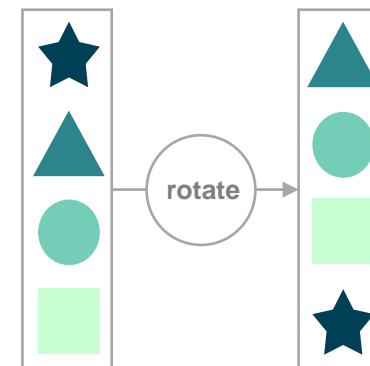
# SIMD-like Parallelism

```
Standard C++
int[] foo(int[] x,int[] y){
    int[] r;
    for(i = 0; i < 6; ++i){
        r[i] = x[i] * y[i]
    }
    return r;
}
```

```
Batched FHE
int[] foo(int[] a,int[] b){
    return a * b;
}
```



No efficient free permutation or scatter/gather

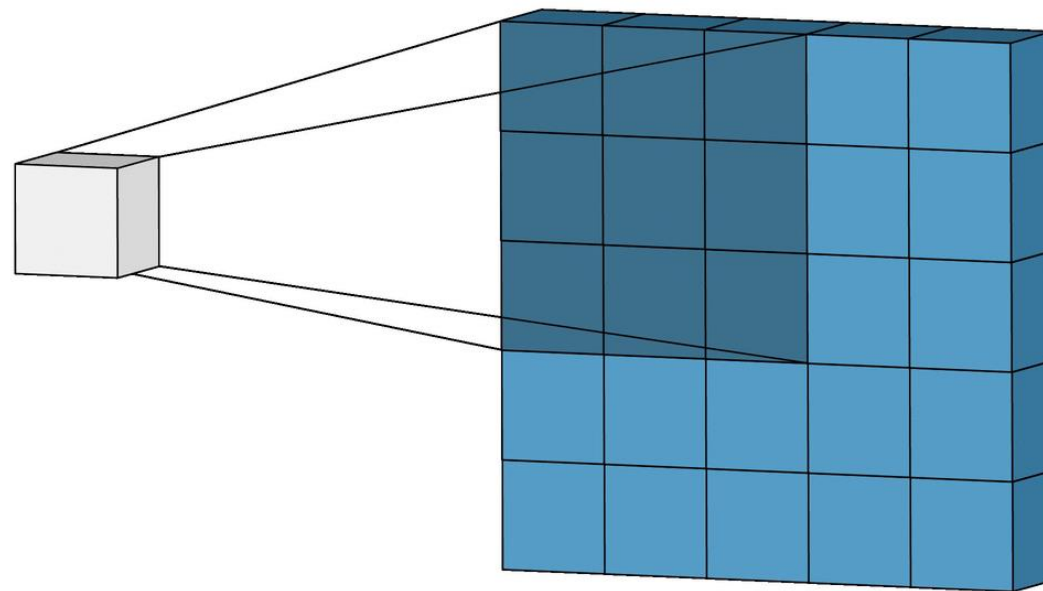


Only cyclical rotations

# Example: Simple Image Processing

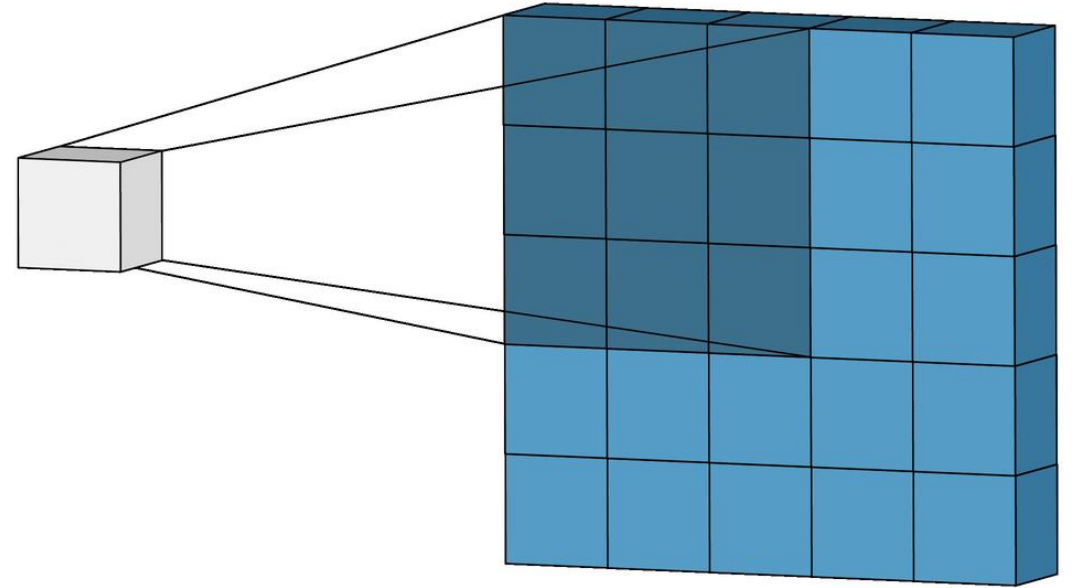
```
1 @template(N)
2 def LaplaceSharpening(img: Tensor[N, Secret[f64]]):
3   img_out = img.copy()
4   w = [[1, 1, 1], [1, -8, 1], [1, 1, 1]]
5   for x in range(N): # loop over pixels
6     for y in range(N): t = 0
7       for j in range(-1, 2): # apply kernel
8         for i in range(-1, 2):
9           t += w[i+1][j+1] * img[((x+i)*N+(y+j))%N]
10          img_out[(x*N+y)%N] = 2*img[(x*N+y)%N] - t
11   return img_out
```

$9N^2$  Homomorphic Multiplications



# Example: Simple Image Processing

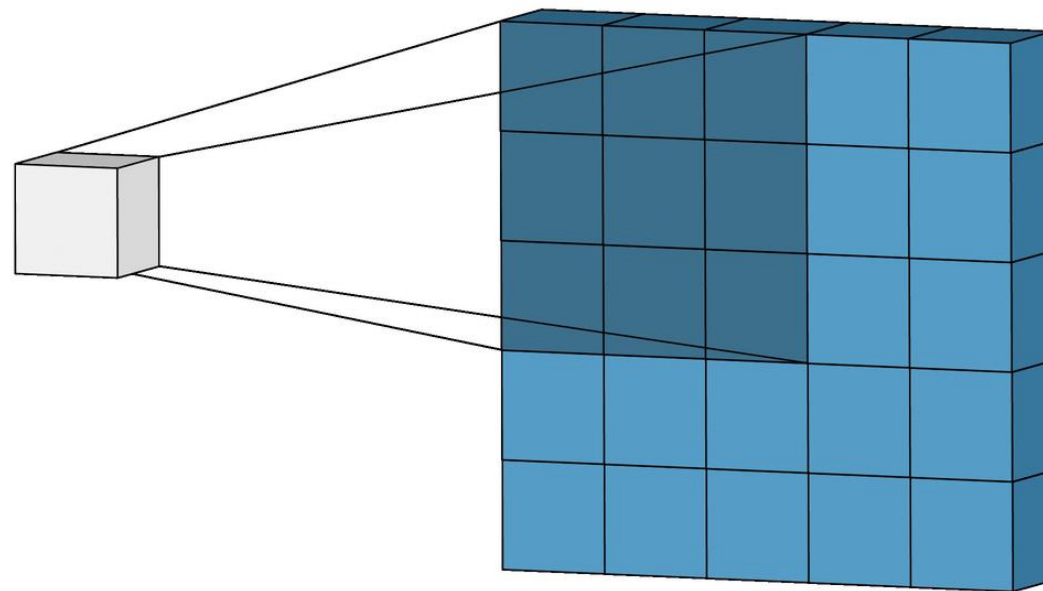
```
1 @template(N)
2 def LaplaceSharpening(img: Tensor[N, Secret[f64]]):
3   img_out = img.copy()
4   w = [[1, 1, 1], [1, -8, 1], [1, 1, 1]]
5   for x in range(N): # loop over pixels
6     for y in range(N): t = 0
7       t += w[0][0] * img[((x-1)*N+(y-1))%N]
8       t += w[1][0] * img[(x*N+(y-1))%N]
9       t += w[2][0] * img[((x+1)*N+(y-1))%N]
10      t += w[0][1] * img[((x-1)*N+y)%N]
11      t += w[1][1] * img[(x*N+y)%N]
12      t += w[2][1] * img[((x+1)*N+y)%N]
13      t += w[0][2] * img[((x-1)*N+(y+1))%N]
14      t += w[1][2] * img[(x*N+(y+1))%N]
15      t += w[2][2] * img[((x+1)*N+(y+1))%N]
16      img_out[(x*N+y)%N] = 2*img[(x*N+y)%N] - t
17   return img_out
```



# Example: Simple Image Processing

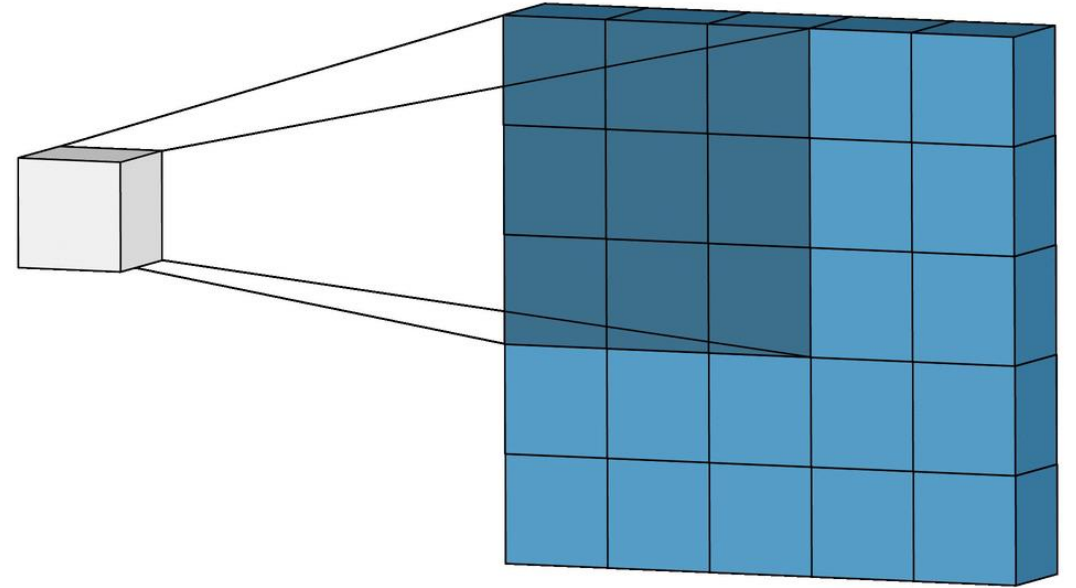
```
1 @template(N)
2 def LaplaceSharpening(img: Tensor[N, Secret[f64]]):
3   img_out = img.copy()
4   w = [[1, 1, 1], [1, -8, 1], [1, 1, 1]]
5   for x in range(N): # loop over pixels
6     for y in range(N): t = 0
7
8
9
10
11     t += w * temp
12
13
14
15
16     img_out[(x*N+y)%N] = 2*img[(x*N+y)%N] - t
17   return img_out
```

$N^2$  Batched Homomorphic Multiplications?



# Example: Simple Image Processing

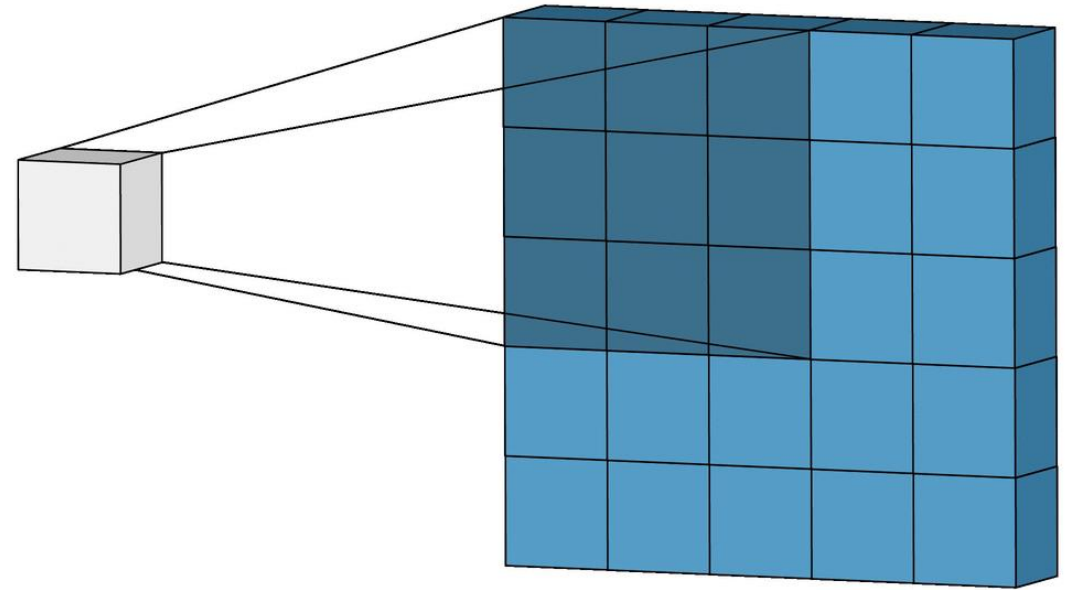
```
1 @template(N)
2 def LaplaceSharpening(img: Tensor[N, Secret[f64]]):
3   img_out = img.copy()
4   w = [[1, 1, 1], [1, -8, 1], [1, 1, 1]]
5   for x in range(N): # loop over pixels
6     for y in range(N): t = 0
7
8
9
10
11
12
13
14
15     t += w * temp
16     img_out[(x*N+y)%N] = 2*img[(x*N+y)%N] - t
17   return img_out
```





# Example: Simple Image Processing

```
1 @template(N)
2 def LaplaceSharpening(img: Tensor[N, Secret[f64]]):
3   img_out = img.copy()
4   w = [[1, 1, 1], [1, -8, 1], [1, 1, 1]]
5   for x in range(N): # loop over pixels
6     for y in range(N): t = 0
7       temp[0] = img[((x-1)*N+(y-1))%N]
8       temp[1] = img[(x*N+(y-1))%N]
9       temp[2] = img[((x+1)*N+(y-1))%N]
10      temp[3] = img[((x-1)*N+y)%N]
11      temp[4] = img[(x*N+y)%N]
12      temp[5] = img[((x+1)*N+y)%N]
13      temp[6] = img[((x-1)*N+(y+1))%N]
14      temp[7] = img[(x*N+(y+1))%N]
15      temp[8] = img[((x+1)*N+(y+1))%N]
16      t += w * temp
17      img_out[(x*N+y)%N] = 2*img[(x*N+y)%N] - t
return img_out
```

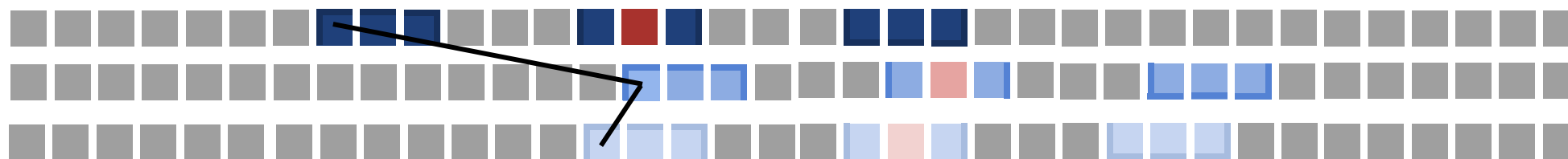
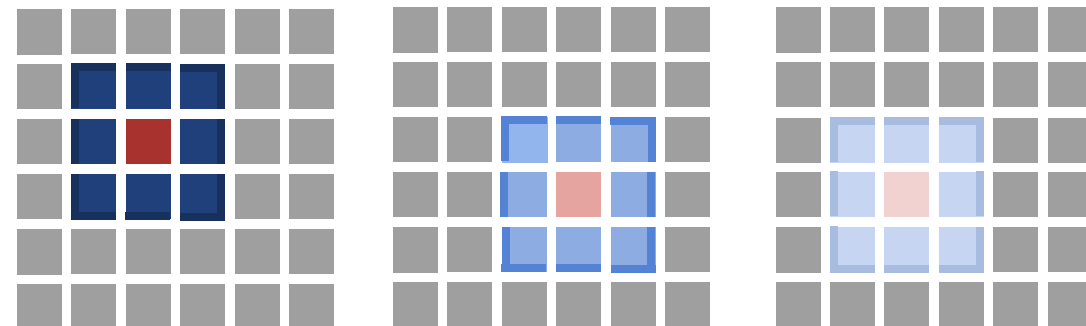


$N^2$  Homomorphic Multiplications +  $9N^2$  Homomorphic Rotations



# Example: Simple Image Processing

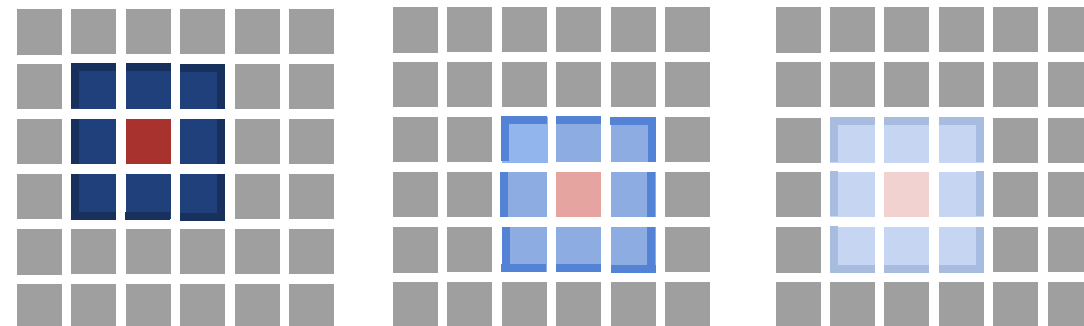
```
1 @template(N)
2 def LaplaceSharpening(img: Tensor[N, Secret[f64]]):
3   img_out = img.copy()
4   w = [[1, 1, 1], [1, -8, 1], [1, 1, 1]]
5   for x in range(N): # loop over pixels
6     for y in range(N): t = 0
7       for j in range(-1, 2): # apply kernel
8         for i in range(-1, 2):
9           t += w[i+1][j+1] * img[((x+i)*n+(y+j))%N]
10          img_out[(x*n+y)%N] = 2*img[(x*n+y)%N] - t
11   return img_out
```



⋮  
N<sup>2</sup>

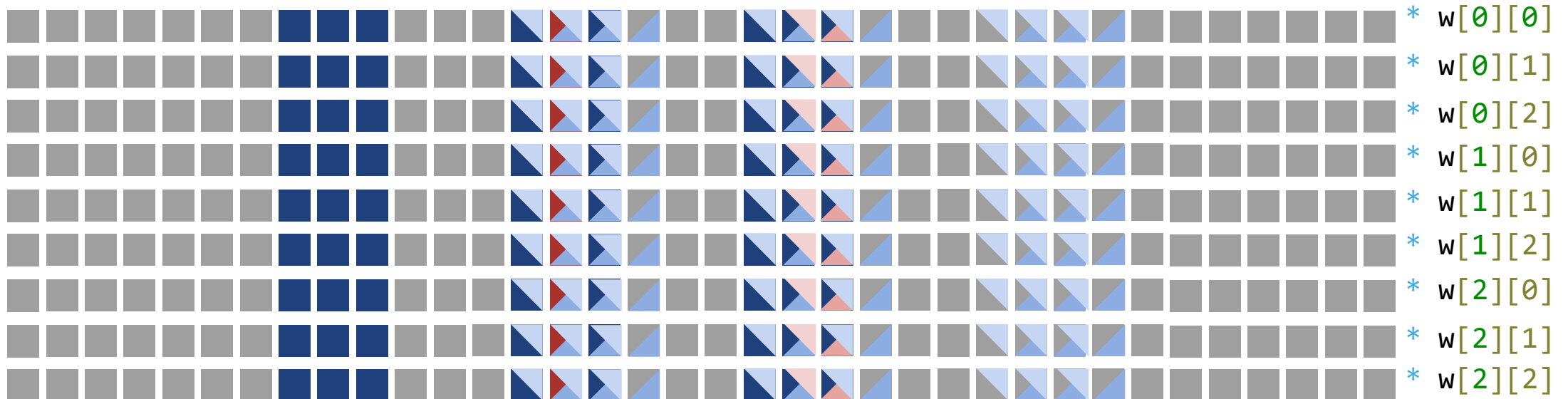
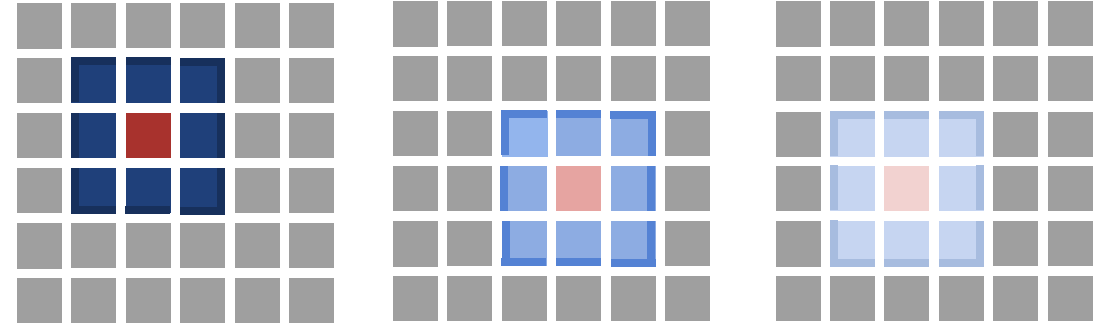
# Example: Simple Image Processing

```
1 @template(N)
2 def LaplaceSharpening(img: Tensor[N, Secret[f64]]):
3   img_out = img.copy()
4   w = [[1, 1, 1], [1, -8, 1], [1, 1, 1]]
5   for x in range(N): # loop over pixels
6     for y in range(N): t = 0
7       for j in range(-1, 2): # apply kernel
8         for i in range(-1, 2):
9           t += w[i+1][j+1] * img[((x+i)*n+(y+j))%N]
10          img_out[(x*n+y)%N] = 2*img[(x*n+y)%N] - t
11   return img_out
```



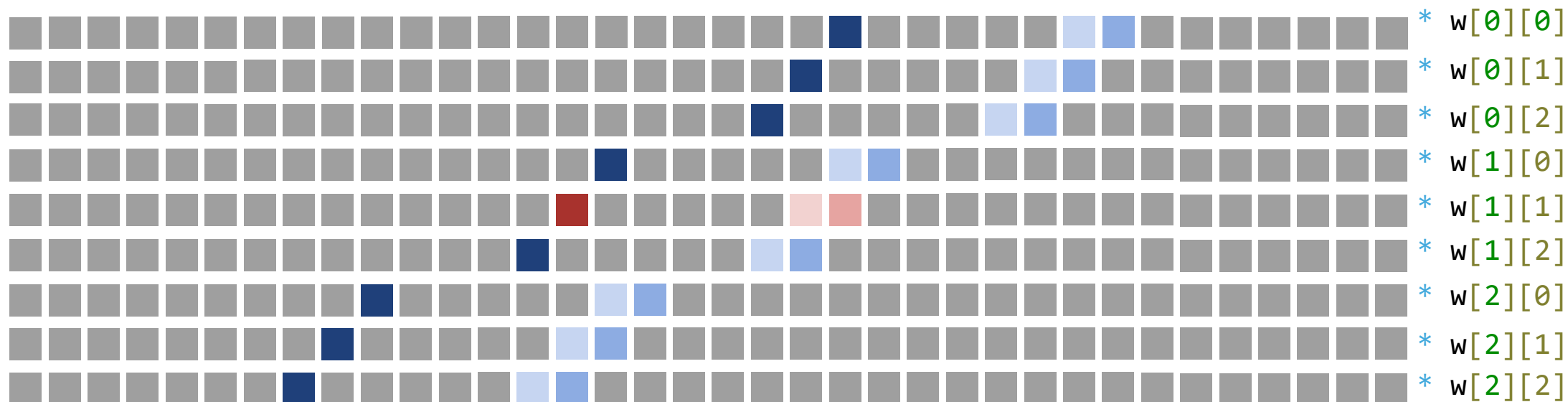
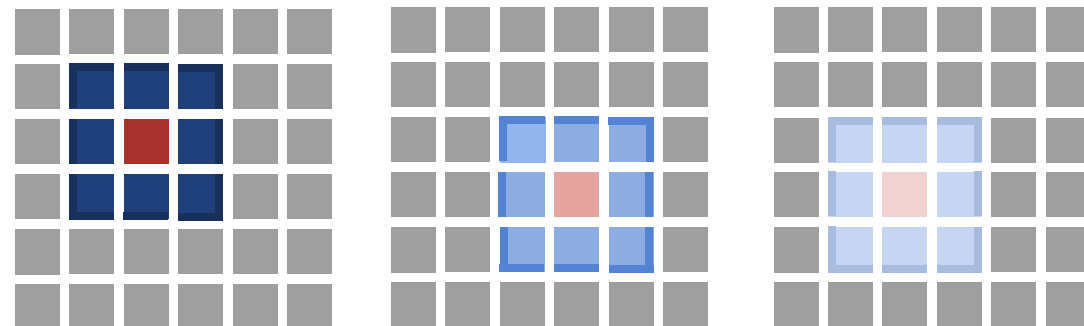
# Example: Simple Image Processing

```
1 @template(N)
2 def LaplaceSharpening(img: Tensor[N, Secret[f64]]):
3   img_out = img.copy()
4   w = [[1, 1, 1], [1, -8, 1], [1, 1, 1]]
5   for x in range(N): # loop over pixels
6     for y in range(N): t = 0
7       for j in range(-1, 2): # apply kernel
8         for i in range(-1, 2):
9           t += w[i+1][j+1] * img[((x+i)*n+(y+j))%N]
10          img_out[(x*n+y)%N] = 2*img[(x*n+y)%N] - t
11   return img_out
```



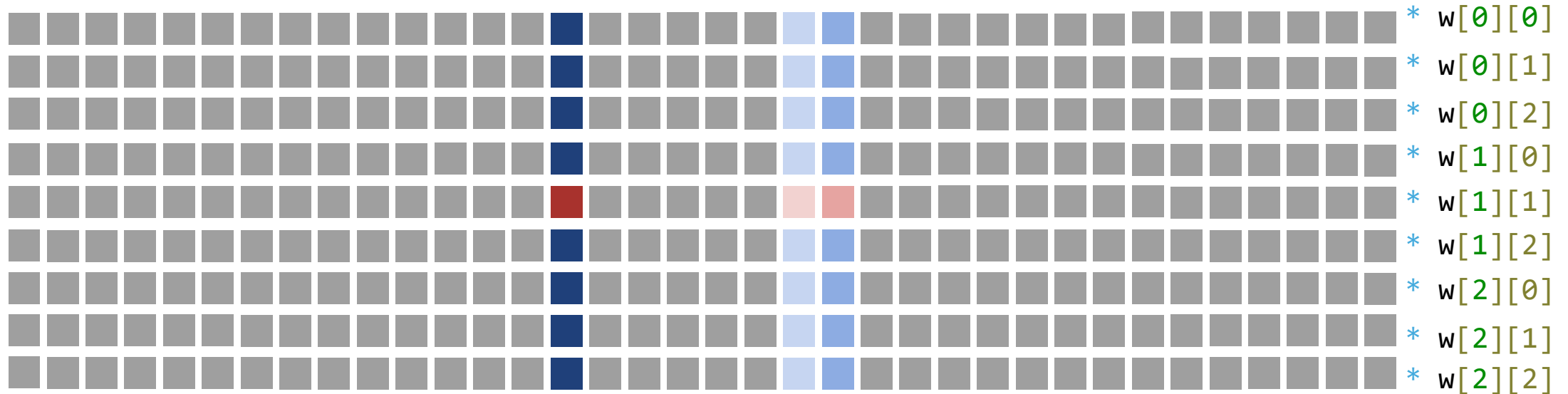
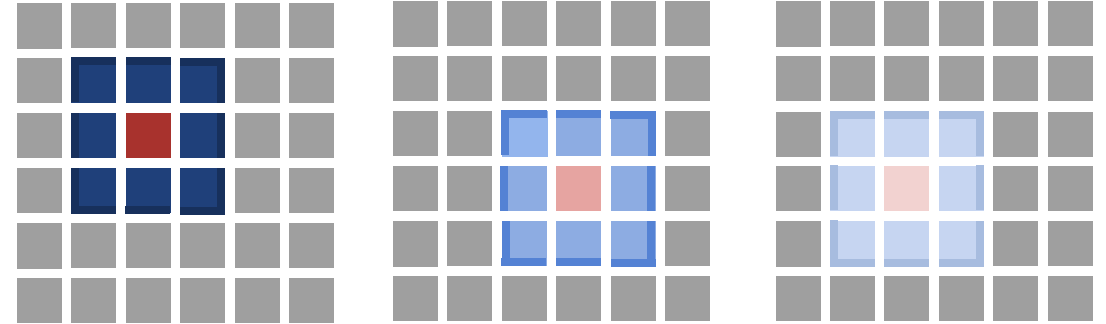
# Example: Simple Image Processing

```
1 @template(N)
2 def LaplaceSharpening(img: Tensor[N, Secret[f64]]):
3   img_out = img.copy()
4   w = [[1, 1, 1], [1, -8, 1], [1, 1, 1]]
5   for x in range(N): # loop over pixels
6     for y in range(N): t = 0
7       for j in range(-1, 2): # apply kernel
8         for i in range(-1, 2):
9           t += w[i+1][j+1] * img[((x+i)*n+(y+j))%N]
10          img_out[(x*n+y)%N] = 2*img[(x*n+y)%N] - t
11   return img_out
```



# Example: Simple Image Processing

```
1 @template(N)
2 def LaplaceSharpening(img: Tensor[N, Secret[f64]]):
3   img_out = img.copy()
4   w = [[1, 1, 1], [1, -8, 1], [1, 1, 1]]
5   for x in range(N): # loop over pixels
6     for y in range(N): t = 0
7       for j in range(-1, 2): # apply kernel
8         for i in range(-1, 2):
9           t += w[i+1][j+1] * img[((x+i)*n+(y+j))%N]
10          img_out[(x*n+y)%N] = 2*img[(x*n+y)%N] - t
11   return img_out
```



# Generalizing Batching

## Algorithm 3 Batching Pass

```
1: Algorithm BATCHPASS( $\mathcal{G}$ )
2:    $\mathcal{V}, \mathcal{E} \leftarrow \mathcal{G}$ 
3:   foreach  $op \in \mathcal{V} \wedge \text{type}(op) \equiv \text{the:secret}$ :
4:      $is \leftarrow \text{SELECTTARGETSLOT}(op, \mathcal{V}, \mathcal{E})$ 
5:     OPERANDCONVERSION( $op, is, \mathcal{V}, \mathcal{E}$ )
6:     foreach  $v \in \mathcal{V} \wedge (op, v) \in \mathcal{E}$ :
7:        $u \leftarrow \text{the:extract}(v, is)$ 
8:       REPLACE( $v, u, \mathcal{V}, \mathcal{E}$ )
9:   procedure SELECTTARGETSLOT( $op, \mathcal{V}, \mathcal{E}$ )
10:    foreach  $v \in \mathcal{V} \wedge (op, v) \in \mathcal{E}$ :
11:      switch  $v$ :
12:        case  $\text{the:insert}(\_, i)$ : return  $i$ 
13:        case  $\text{func:return}$ : return 0
14:    foreach  $v \in \mathcal{V} \wedge (v, op) \in \mathcal{E}$ :
15:      switch  $op$ :
16:        case  $\text{the:extract}(\_, i)$ :
```

# Generalizing Batching

```
10:   foreach  $v \in \mathcal{V} \wedge (op, v) \in \mathcal{E}$ :
11:     switch  $v$ :
12:       case the.insert(,  $i$ ): return  $i$ 
13:       case the.return: return 0
14:   foreach  $v \in \mathcal{V} \wedge (v, op) \in \mathcal{E}$ :
15:     switch  $op$ :
16:       case the.extract(,  $i$ ):
17:         return  $i$ 
18:   return  $\perp$ 
19: procedure OPERANDCONVERSION( $op, is, \mathcal{V}, \mathcal{E}$ )
20:   foreach  $v \in \mathcal{V} \wedge (v, op) \in \mathcal{E} \wedge \text{type}(v) \equiv \text{the.set}$ :
21:     switch  $v$ :
22:       case the.extract(,  $i$ ):
23:          $u \leftarrow \text{the.rotate}(x, i \equiv is)$ 
24:         REPLACE( $v, u, \mathcal{V}, \mathcal{E}$ )
25:       case the.ptxt[,  $p$ ):
26:          $p' \leftarrow \text{REPEAT}(p)$ 
27:          $u \leftarrow \text{the.ptxt}(p')$ 
```

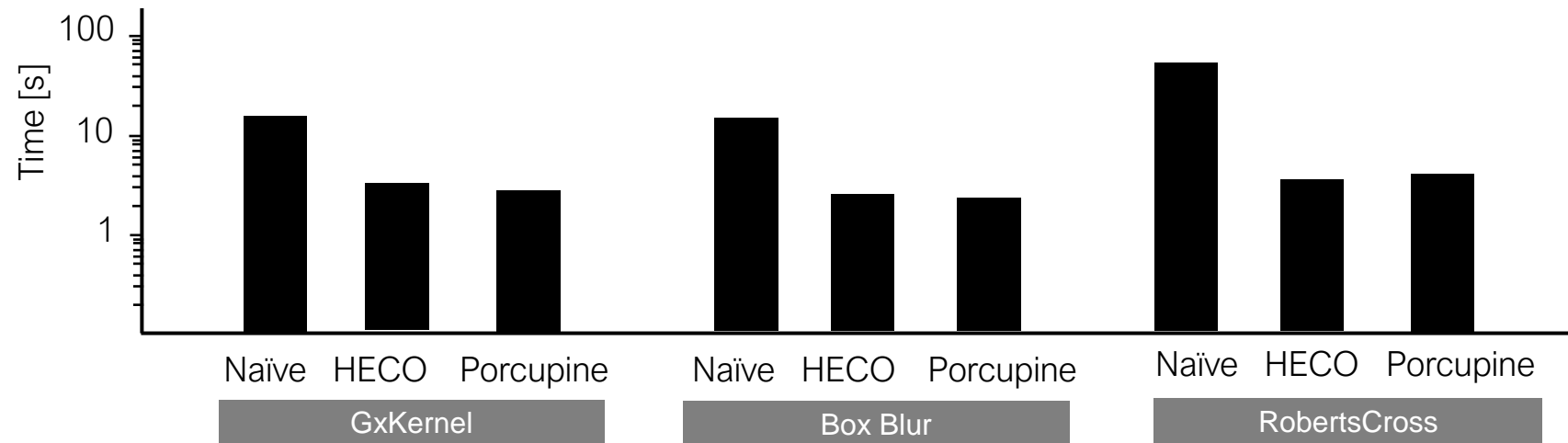


# Generalizing Batching

```
10: procedure OPERANDCONVERSION( $op, is, \mathcal{V}, \mathbb{E}$ )
11:   foreach  $v \in \mathcal{V} \wedge (v, op) \in \mathbb{E} \wedge \text{type}(v) \equiv \text{the:root}$ 
12:     switch  $v$ :
13:       case the:extract( $x, i$ ):
14:          $u \leftarrow \text{the:rotate}(x, i = is)$ 
15:         REPLACE( $v, u, \mathcal{V}, \mathbb{E}$ )
16:       case the:ptxt( $p$ ):
17:          $p' \leftarrow \text{REPEAT}(p)$ 
18:          $u \leftarrow \text{the:ptxt}(p')$ 
19:         REPLACE( $v, u, \mathcal{V}, \mathbb{E}$ )
20: procedure REPLACE( $v, u, \mathcal{V}, \mathbb{E}$ )
21:    $\mathcal{V} \leftarrow (\mathcal{V} \setminus \{v\}) \cup \{u\}$ 
22:   foreach  $w \in \mathcal{V} \wedge (v, w) \in \mathbb{E}$ :
23:      $\mathbb{E} \leftarrow (\mathbb{E} \setminus \{(v, w)\}) \cup \{(u, w)\}$ 
24:   foreach  $w \in \mathcal{V} \wedge (w, v) \in \mathbb{E}$ :
25:      $\mathbb{E} \leftarrow (\mathbb{E} \setminus \{(w, v)\}) \cup \{(w, u)\}$ 
```

# Evaluation: Effect of Batching

Comparing against “Naïve” (non-batched) implementation and “Optimal” synthesis-based solution [CD+21]



## This work:

- End-to-End FHE Toolchain Architecture
- High-Level Batching Optimization

## Ongoing FHE Abstraction Standardization Effort:



## Future work:

Limited Expressiveness

Noise Management

SIMD Batching



marbleHE/HECO



[arxiv.org/abs/2202.01649](https://arxiv.org/abs/2202.01649)

