Cryptographic Computations Need Compilers

Madan Musuvathi
Microsoft Research

Microsoft's Commitment to Research

~1000 People **and growing!**
ca. 650 researchers
ca. 350 engineers, testers, designers, PMs

Labs around the World
Bangalore, Beijing, Cambridge (MA), Cambridge (UK), New York, Redmond

Sponsor of Conferences and Academic Research

Biggest PhD Internship Program

22,000+
published papers

4,616
worldwide patents

3,596
worldwide
patents pending

1
Fields Medal

5
Turing Awards

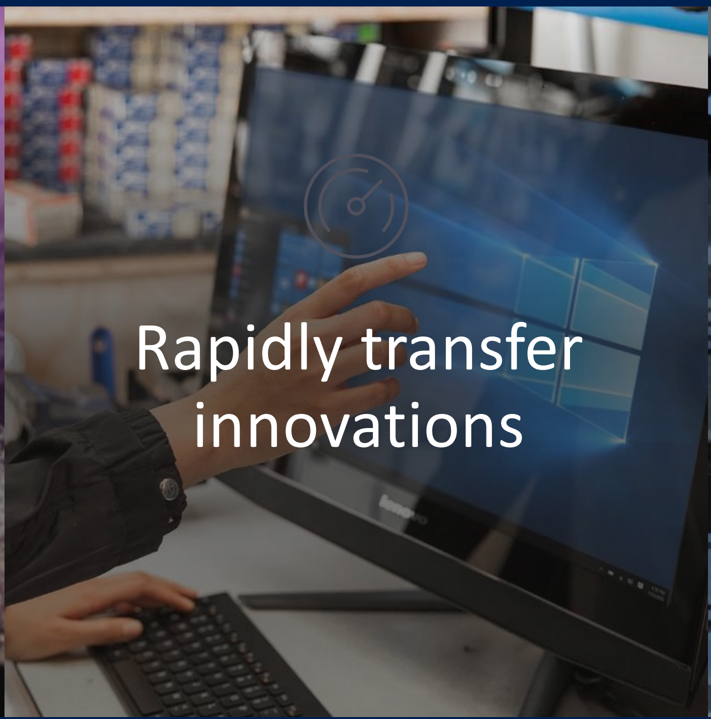
2
MacArthur
Fellowships

30+
Researchers/projects
recognized in 2017

1
Emmy for delivering
ultra-high def video

7
labs and locations
worldwide

Our Mission & Culture (est. 1991)



RiSE

Research in Software Engineering

Programming Languages

- Programming Models
- HPC, Compilers, Systems
- Verified Programming

Software Engineering

- Reliability Tools
- Productivity
- SE4AI & AI4SE

Automated Reasoning

- Foundations
- Theorem Proving

Research in Software Engineering (RiSE)


Tom Ball


Christian Bird


Nikolaj Bjorner


Ella Bounimova


Sebastian Burckhardt


Patrice Godefroid


Peli de Halleux


Markus Kuppe


Shuvendu Lahiri


Daan Leijen


Saeed Maleki


Mark Marron


Kenneth McMillan


Michal Moskal


Leonardo de Moura

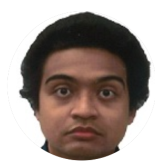

Madan Musuvathi


Todd Mytkowicz


Lev Nachmanson


Nachi Nagappan


Jonathan Protzenko


Tahina Ramananandro


Olli Saarikivi


Nikhil Swamy


Margus Veanes


Tom Zimmermann


Ben Zorn

RiSE Hires 2019

Daniel Selsam

Ph.D. Stanford University
Formal Methods & Machine Learning

Denae Ford Robinson

Ph.D. NC State University
HCI & Software Engineering

Olli Saarikivi

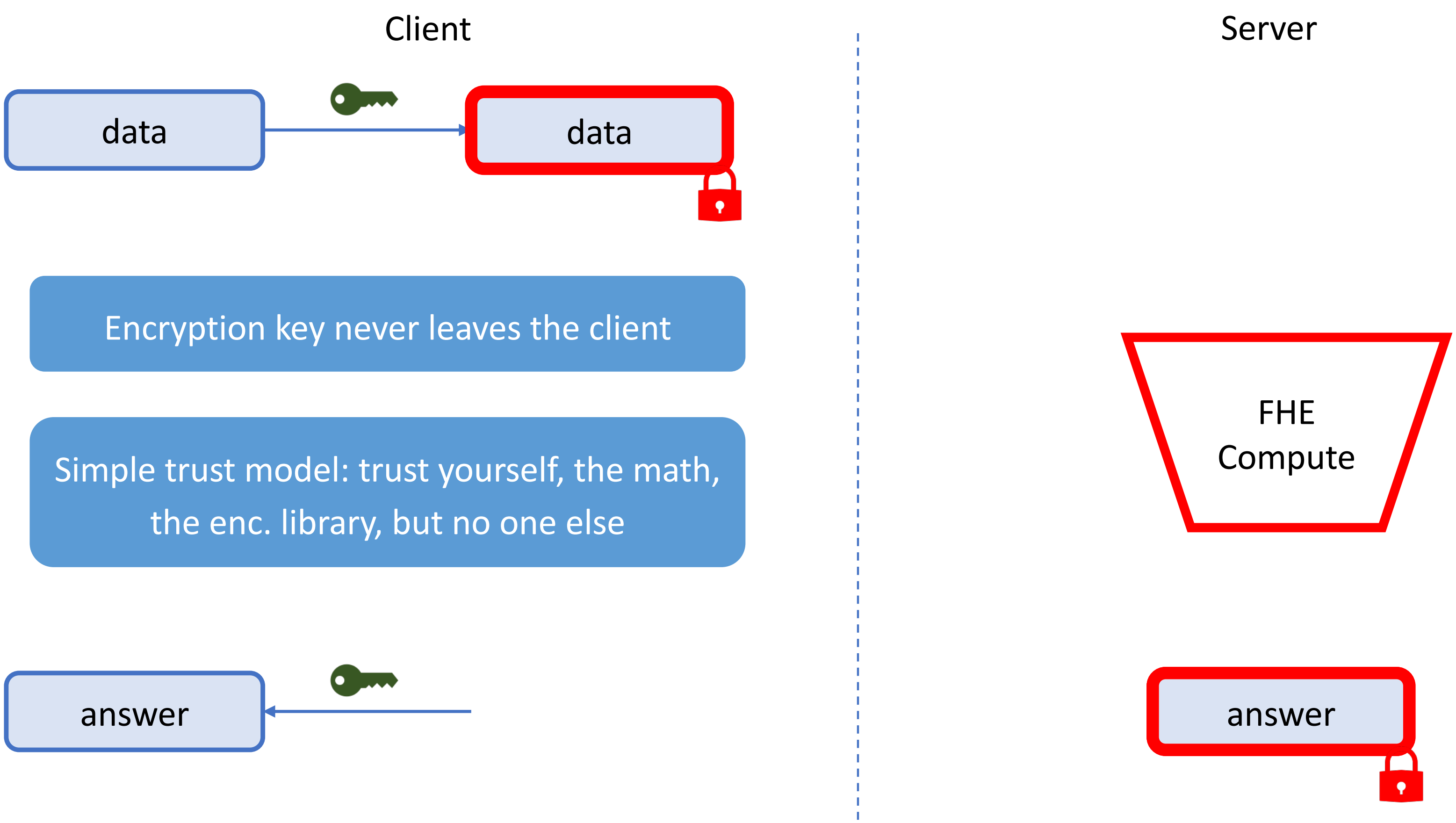
Ph.D. Aalto University
Compiler Systems

Teddy Seyed

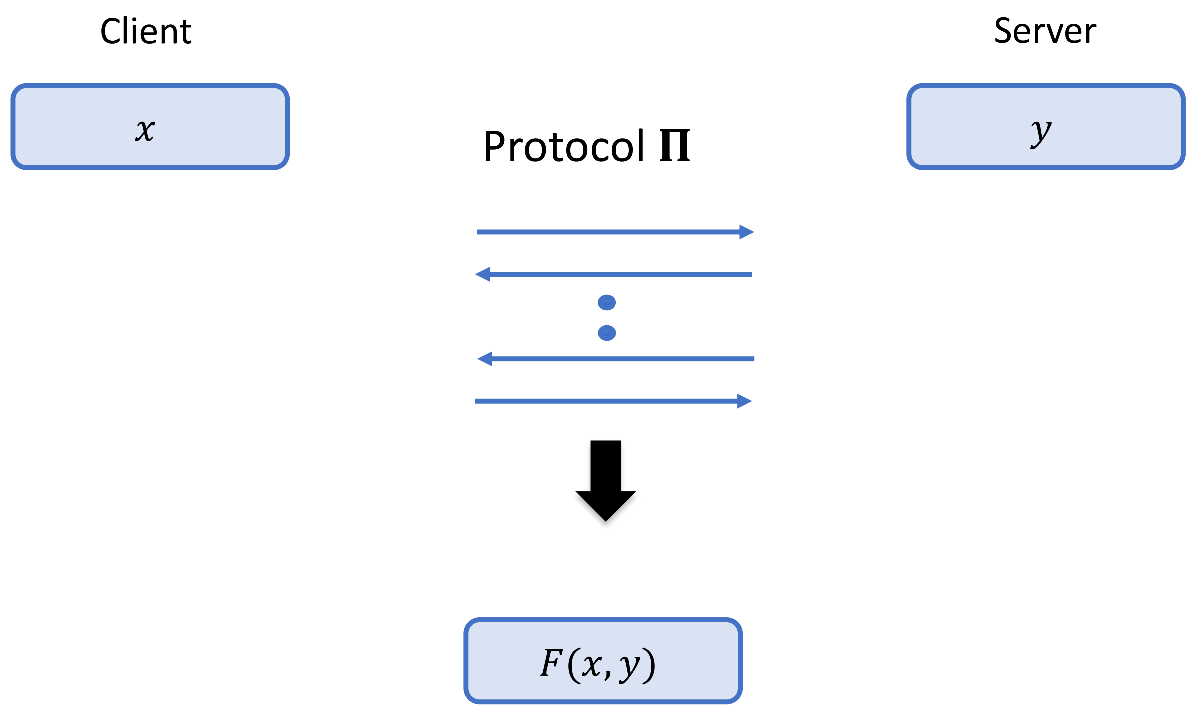
Ph.D. University of Calgary
HCI & Embedded Systems

Cryptographic Computations enable Privacy Preserving Applications

Fully Homomorphic Encryption (FHE)



Secure Multi-Party Computation (Secure MPC)



Nothing is leaked except the output

Semi-Honest Threat Model

- Hardware and software execute requested computation faithfully
- Hardware and software are curious about the data
- Client or user data must remain confidential

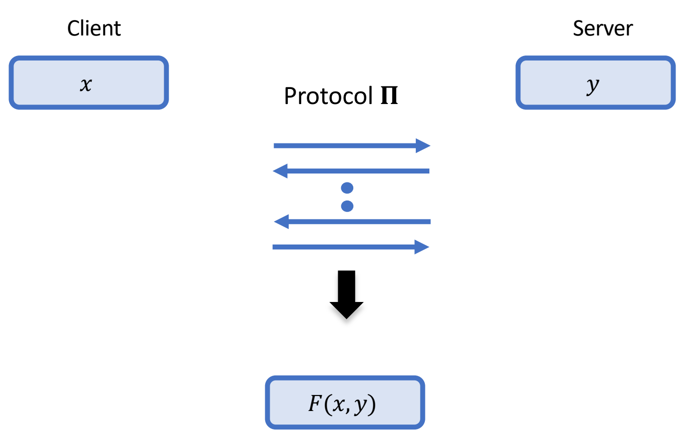
Programming Cryptographic Computations is Hard

- Involves low-level circuit programming
- Need different schemes for Boolean vs arithmetic operations
- Requires cryptographic expertise
  - To guarantee correctness, security, and efficiency

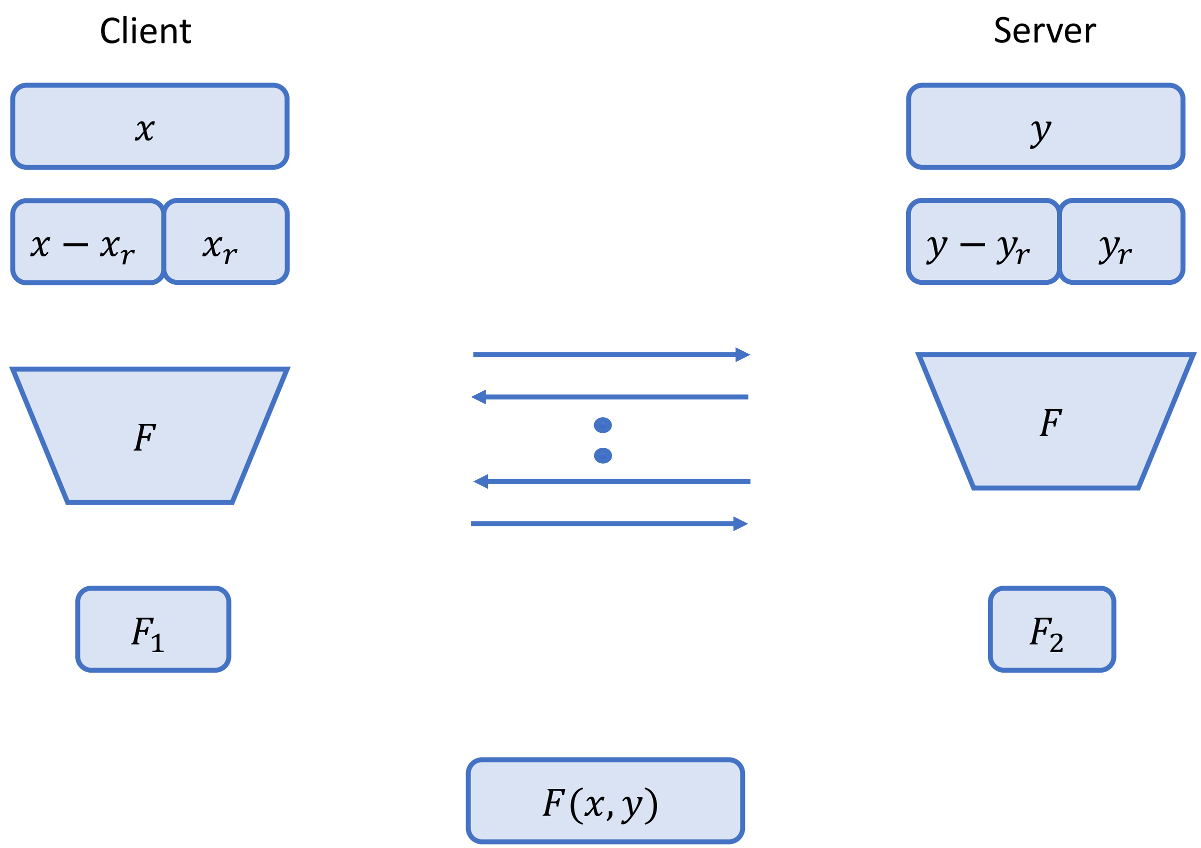
EzPC: Compiler Framework for Secure MPC

Divya Gupta, Nishanth Chandran, Aseem Rastogi, Rahul Sharma

MSR India



Secure Multi-Party Computation (Secure MPC)



EzPC Compilation

Function: $w^t x > b$

```
uint w[30] = input1();
uint x[30] = input2();
uint b = input1();

uint acc = 0;
for i in [0:30] {
acc = acc + (w[i] * x[i]); }

Output2((acc > b ? 1 : 0);
```

```
//circuit builders for arithmetic and boolean
Circuit* ycirc = s[S_YAO]->GetCircuitBuildRoutine();
Circuit* acirc = s[S_ARITH]->GetCircuitBuildRoutine();
...
if(role == SERVER) {
  //Put gates to read w and b
} else { //role == CLIENT
  //Put gates to read x
}

for(uint32_t i = 0; i < 30; i++) { //acc = w^T x
  share * a_t_0 = acirc->PutMULGate(a_w[i], a_x[i]);
  a_acc = acirc->PutADDGate(a_acc, a_t_0 );
}

//convert acc and b from arithmetic to boolean
share *y_acc = ycirc->PutA2YGate(a_acc);
share *y_b = ycirc->PutA2YGate(a_b);

share *y_pred = ycirc->PutGTGate (y_acc, y_b);
uint32_t one = 1 ;
share *y_1 = ycirc->PutCONSGate(one, bitlen);
uint32_t zero = 0 ;
share *y_0 = ycirc->PutCONSGate(zero, bitlen);
share *y_t = ycirc->PutMUXGate(y_pred, y_1, y_0);

share *y_out = ycirc->PutOUTGate(y_t, CLIENT);
party->ExecCircuit();

if(role==CLIENT) {  //only to the client
  uint32_t _o = y_out->get_clear_value<uint32_t>();
}
```

- Base types and array types
- Mathematical operators (+, *, >, &, >>, ….)
- Statements for assignments, array read/write, bounded for loops and if condition

EzPC Compilation

Function: $w^t x > b$

```
uint w[30] = input1();
uint x[30] = input2();
uint b = input1();

uint acc = 0;
for i in [0:30] {
acc = acc + (w[i] * x[i]); }

Output2((acc > b ? 1 : 0);
```

```
//circuit builders for arithmetic and boolean
Circuit* ycirc = s[S_YAO]->GetCircuitBuildRoutine();
Circuit* acirc = s[S_ARITH]->GetCircuitBuildRoutine();
...
if(role == SERVER) {
  //Put gates to read w and b
} else { //role == CLIENT
  //Put gates to read x
}

for(uint32_t i = 0; i < 30; i++) { //acc = w^Tx
  share * a_t_0 = acirc->PutMULGate(a_w[i], a_x[i]);
  a_acc = acirc->PutADDGate(a_acc, a_t_0 );
}

//convert acc and b from arithmetic to boolean
share *y_acc = ycirc->PutA2YGate(a_acc);
share *y_b = ycirc->PutA2YGate(a_b);

share *y_pred = ycirc->PutGTGate (y_acc, y_b);
uint32_t one = 1 ;
share *y_1 = ycirc->PutCONSGate(one, bitlen);
uint32_t zero = 0 ;
share *y_0 = ycirc->PutCONSGate(zero, bitlen);
share *y_t = ycirc->PutMUXGate(y_pred, y_1, y_0);

share *y_out = ycirc->PutOUTGate(y_t, CLIENT);
party->ExecCircuit();

if(role==CLIENT) {  //only to the client
  uint32_t _o = y_out->get_clear_value<uint32_t>();
}
```

- Assign variables to Boolean or Arithmetic
- Automatically insert conversion operators
- Use cryptographic cost model to optimize compilation

Fully-Homomorphic Encryption (FHE)

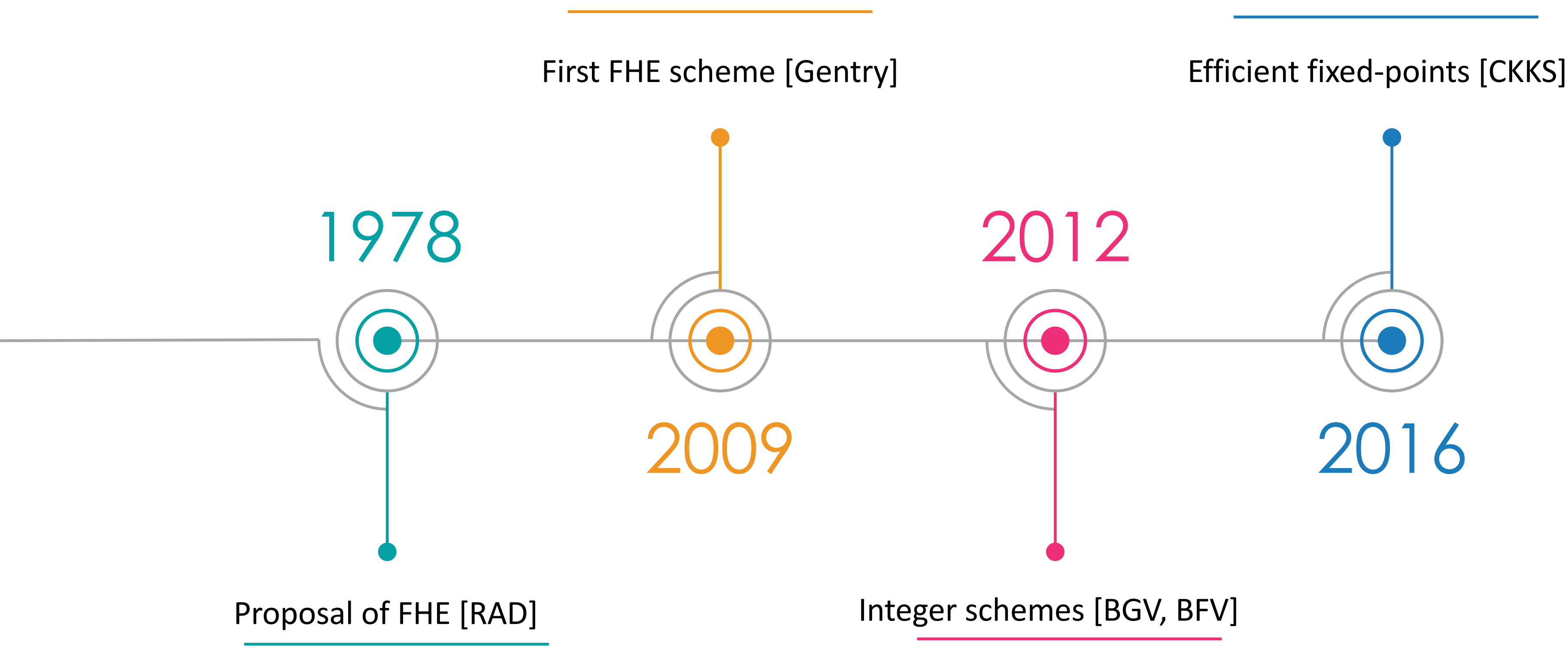
Allows computation on encrypted data

$$[\![m]\!] \triangleq Enc(m)$$

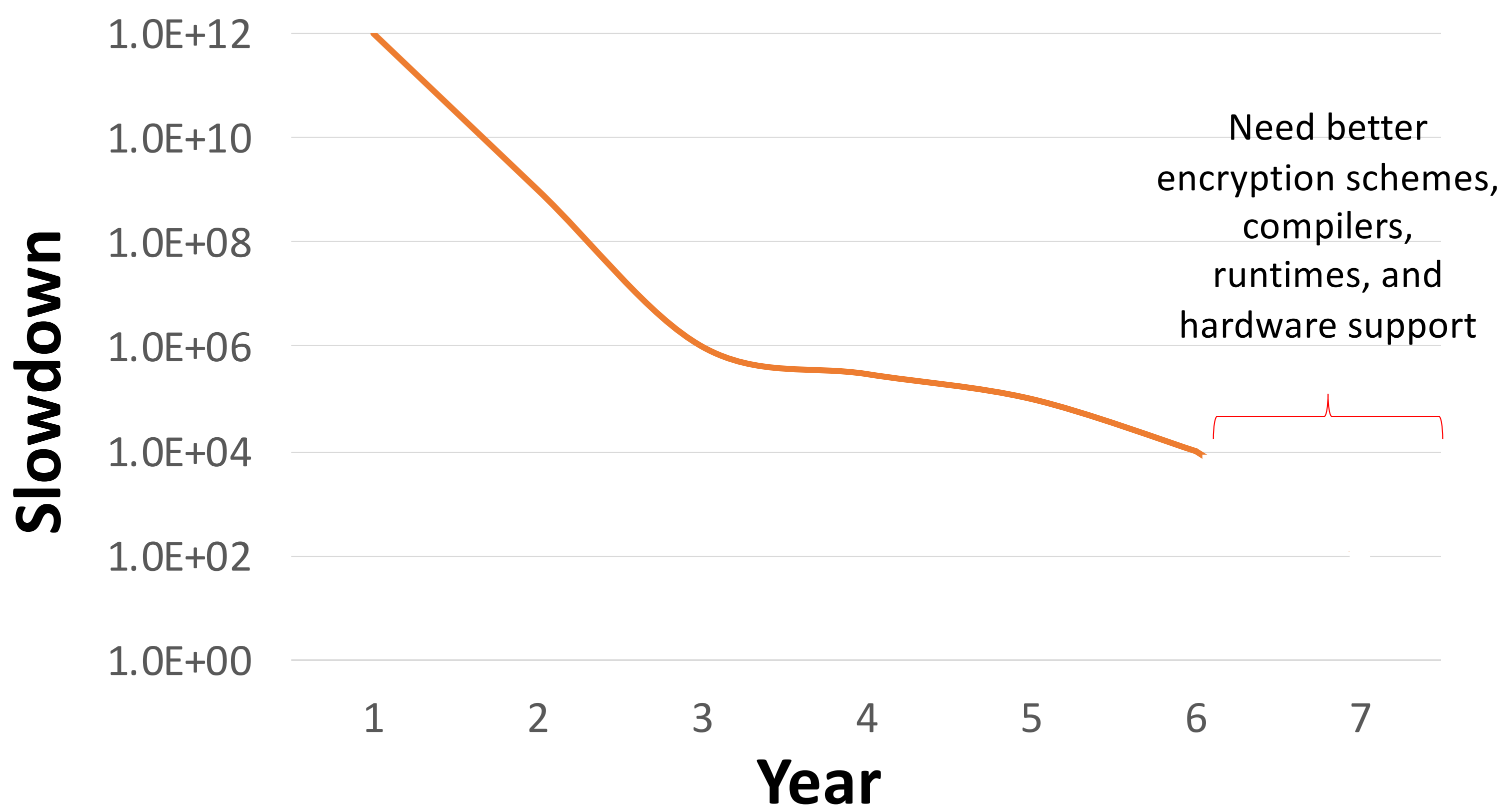
$$[\![a]\!] \oplus [\![b]\!] = [\![a+b]\!]$$

$$[\![a]\!] \otimes [\![b]\!] = [\![a \times b]\!]$$

FHE timeline



Performance overhead of FHE over unencrypted



FHE programming challenges

Computation is slow: ~50 ms per multiplication

Very large SIMD vector widths: ~16K
- can operate at 8086 speeds if you can utilize the parallelism

No branching
- a bug and a feature

Tradeoff between correctness, security, message bloat, and performance
- requires setting parameters correctly

Different encryption schemes provide different functionalities
- arithmetic, Boolean logic, table lookups, …

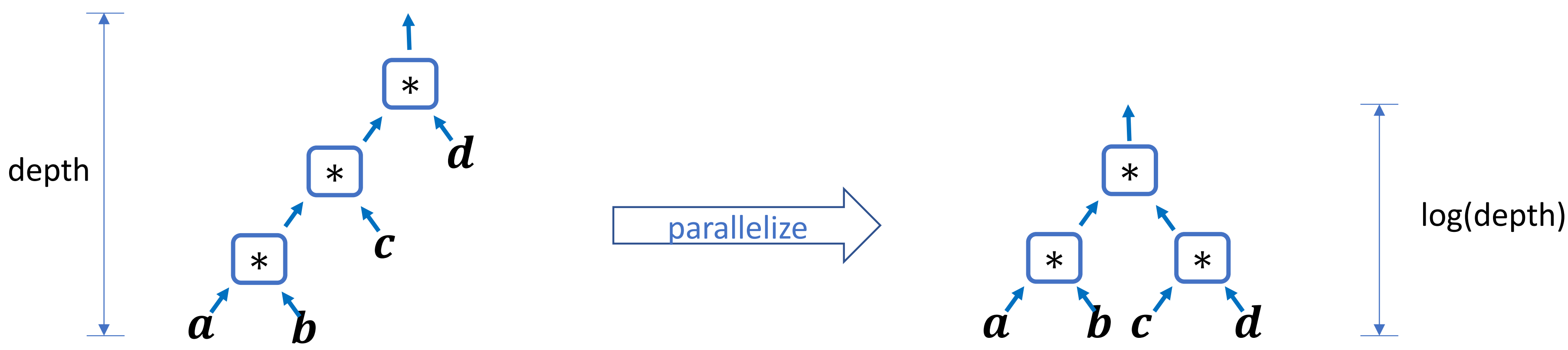
Bootstrap periodically to reduce noise

Noise growth challenge

Noise growth proportional to *multiplicative depth*

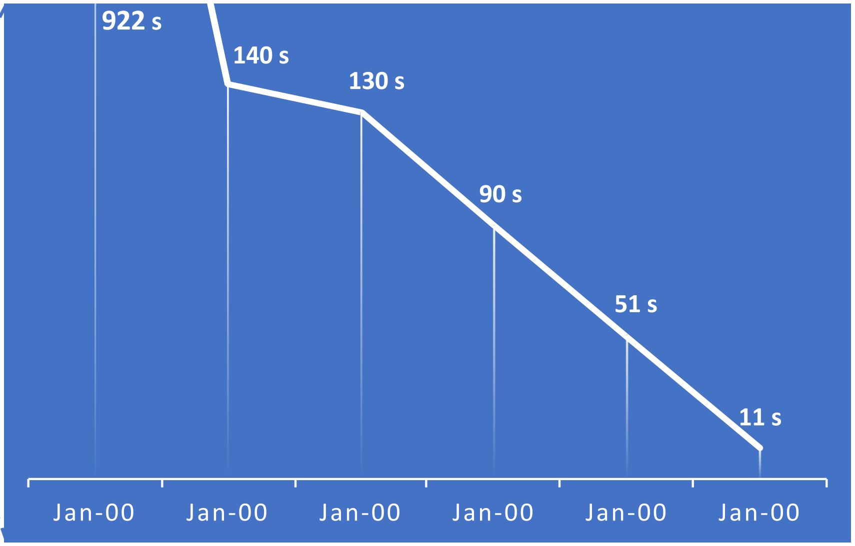
$$Noise([\![a \times b]\!]) = \max(Noise([\![b]\!]), Noise[\![a]\!]) + k$$

Need expensive *bootstrapping* after ~20 multiplications

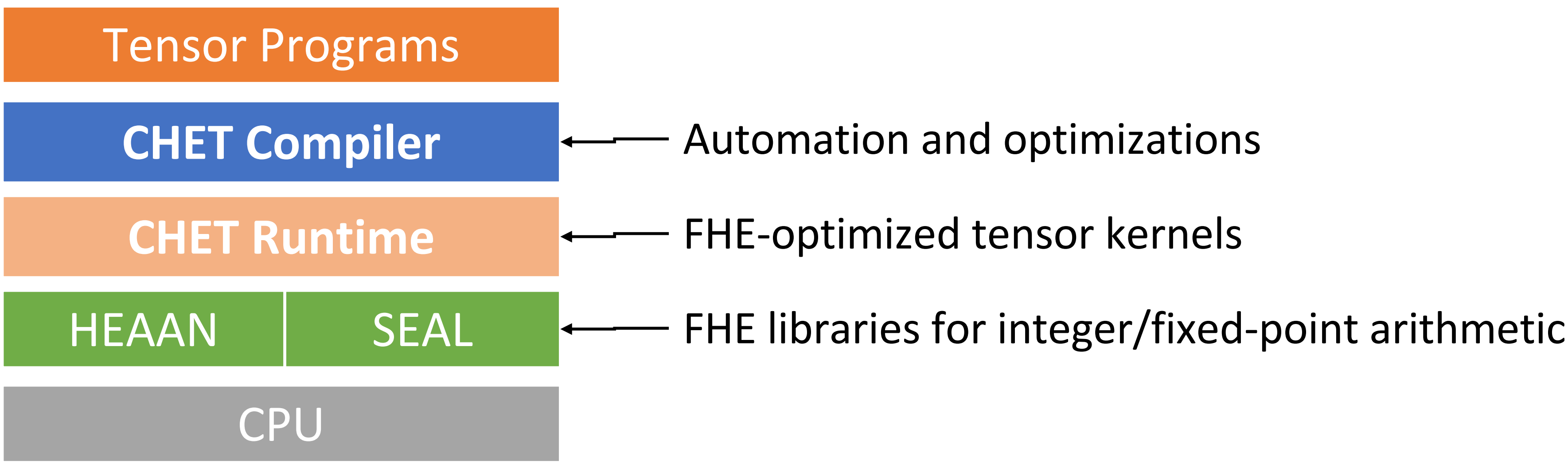


CHET - an FHE compiler [Dathathri et al. '19]

Neural Net Inference	CHET
LeNet5-small	3 s
LeNet5-medium	11 s
LeNet5-large	35 s
Median-liver-cancer	56 s
SqueezeNet-CIFAR	165 s

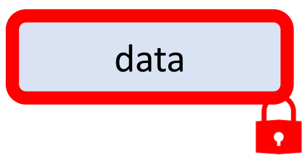


Compiling Tensor Programs for FHE libraries



Encryption Parameters

Ciphertext is a high-degree polynomial

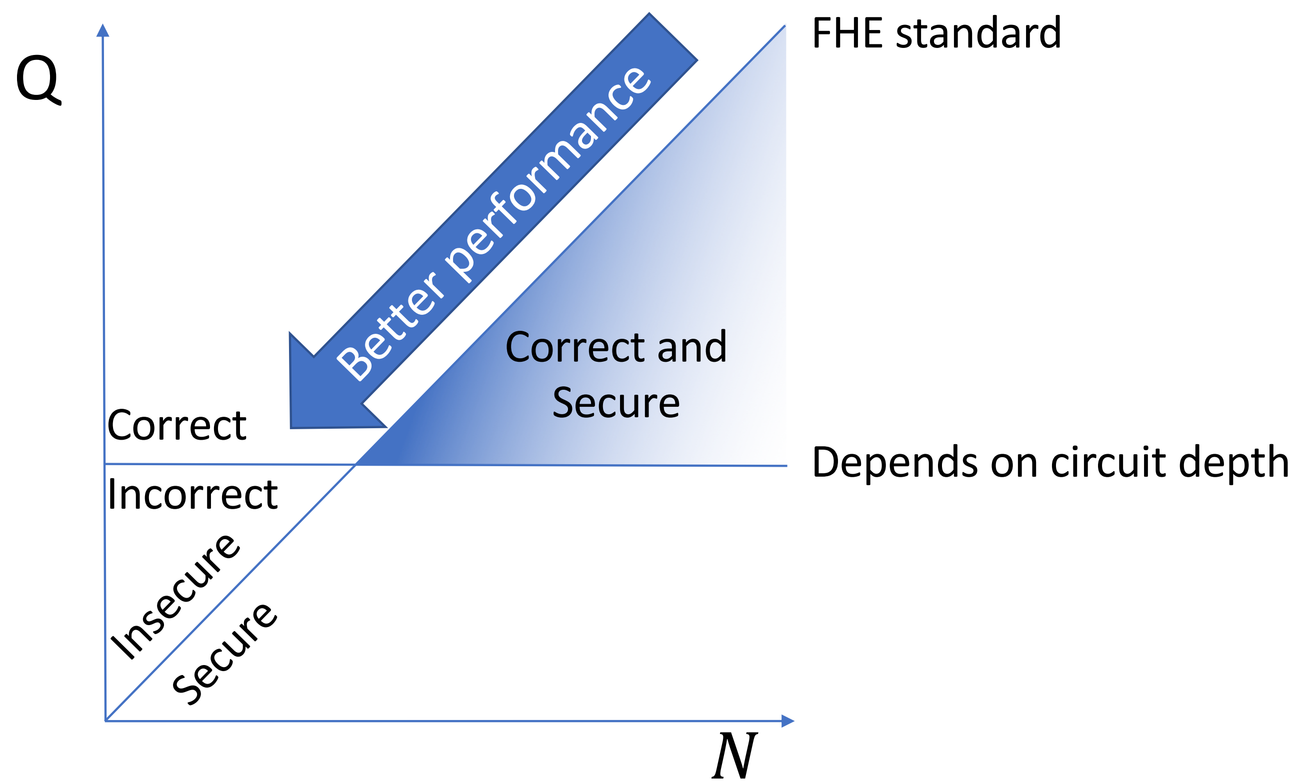


$$= a_N \cdot x^N + a_{N-1} \cdot x^{N-1} + \cdots a_1 \cdot x + a_0$$

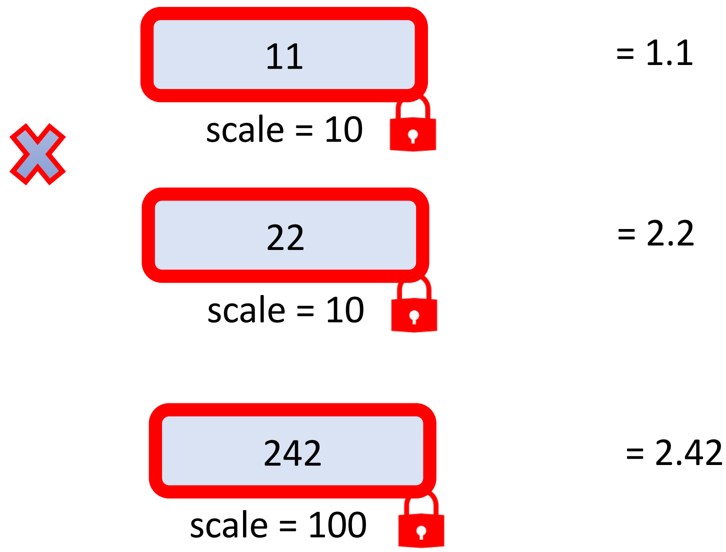
N: degree of the polynomial
Q: modulus of the coefficients

Encryption Parameters

N: degree of the polynomial
Q: modulus of the coefficients

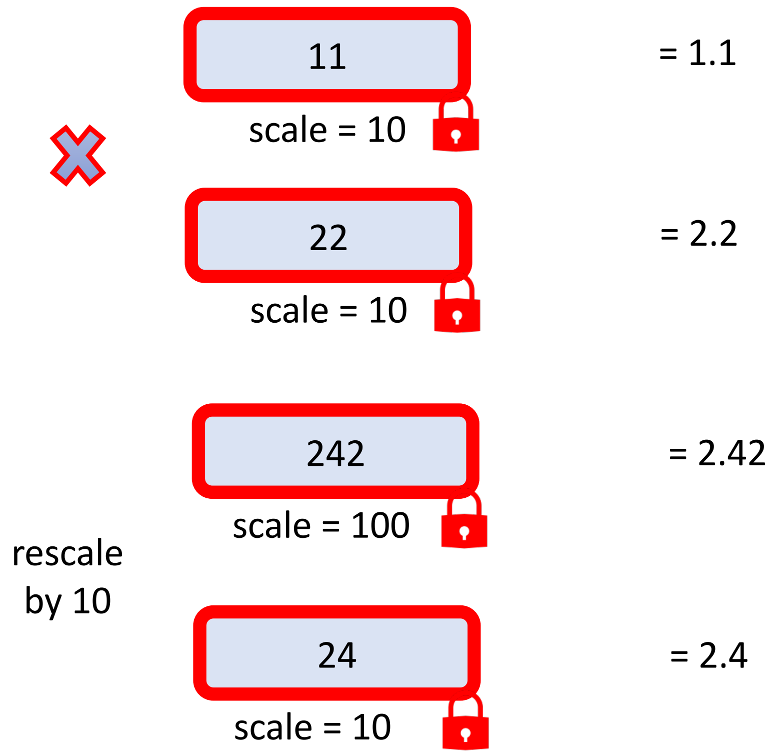


Performing Fixed Points with Scaling Factors



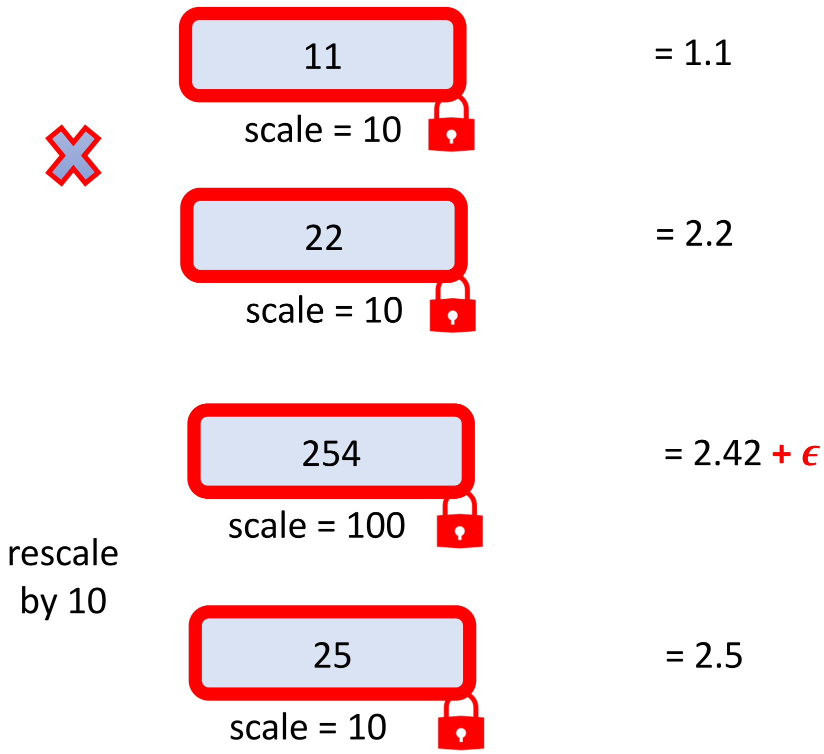
Modulus growth limits depth of circuits

Rescaling operation in CKKS ‘16

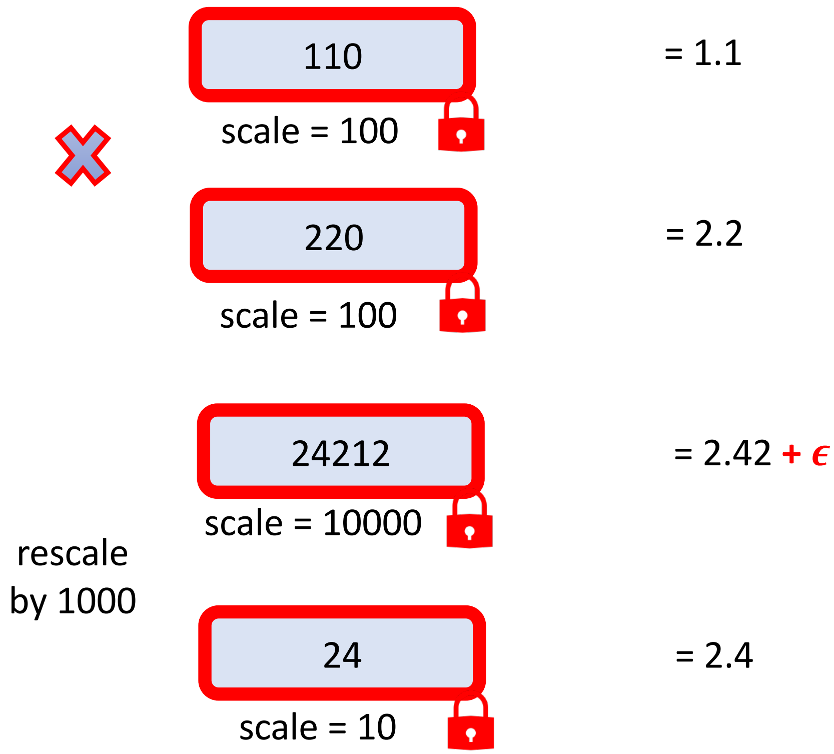


Compiler needs to insert rescale operations effectively

But, CKKS is approximate



Solution: inflate scale

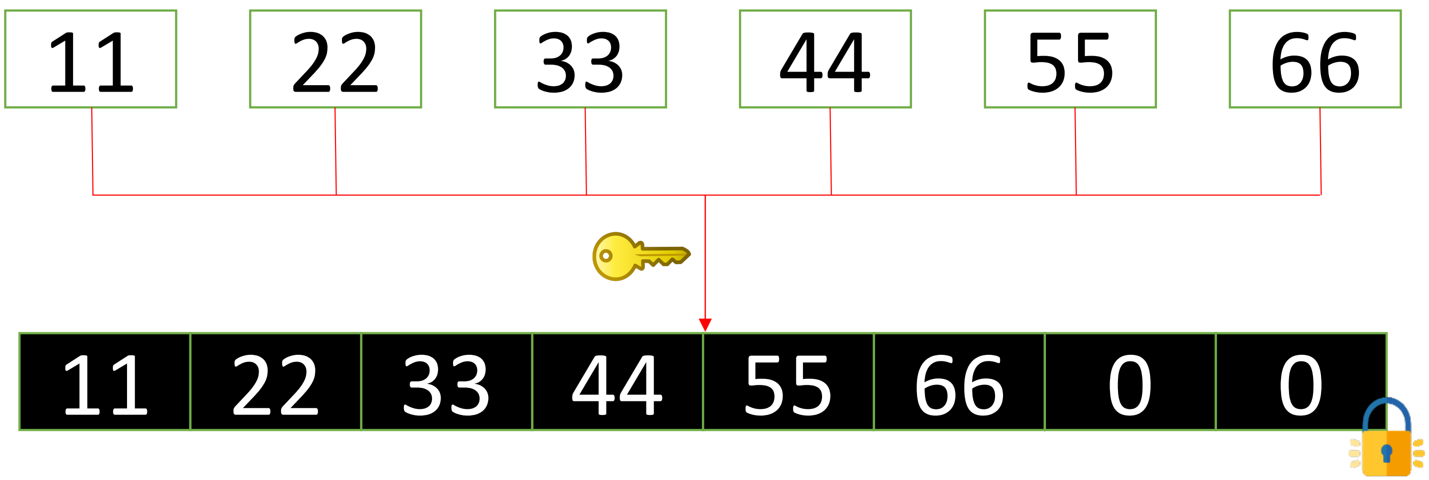


Compiler needs to manage precision and error

FHE Packing

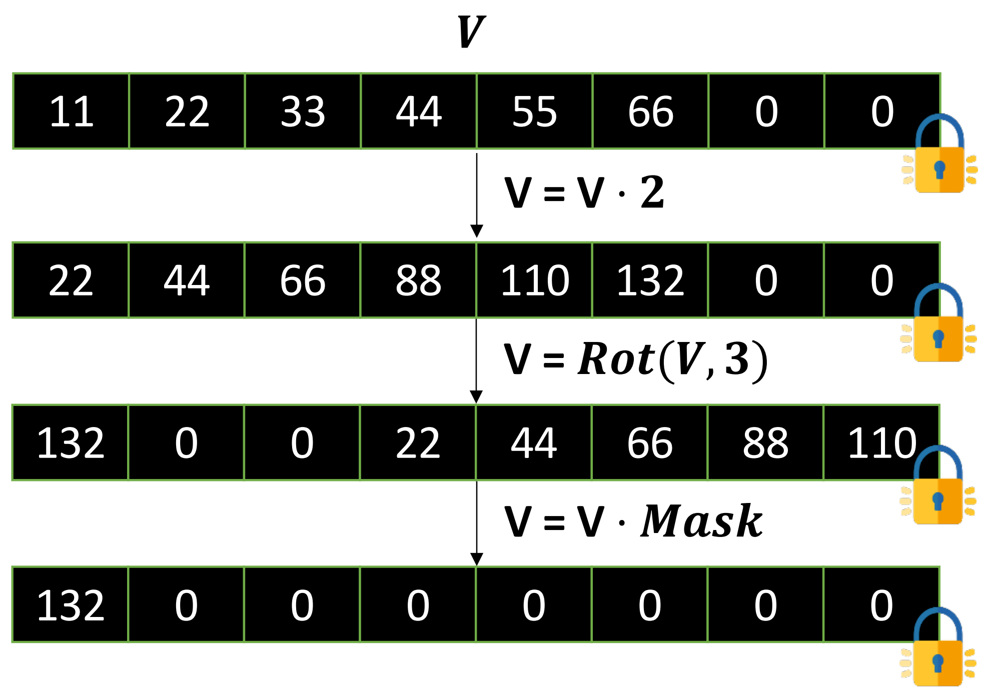
Pack many plaintext scalars into single ciphertext vector

- Fixed width of N/2 (N is 2^{10} or larger)

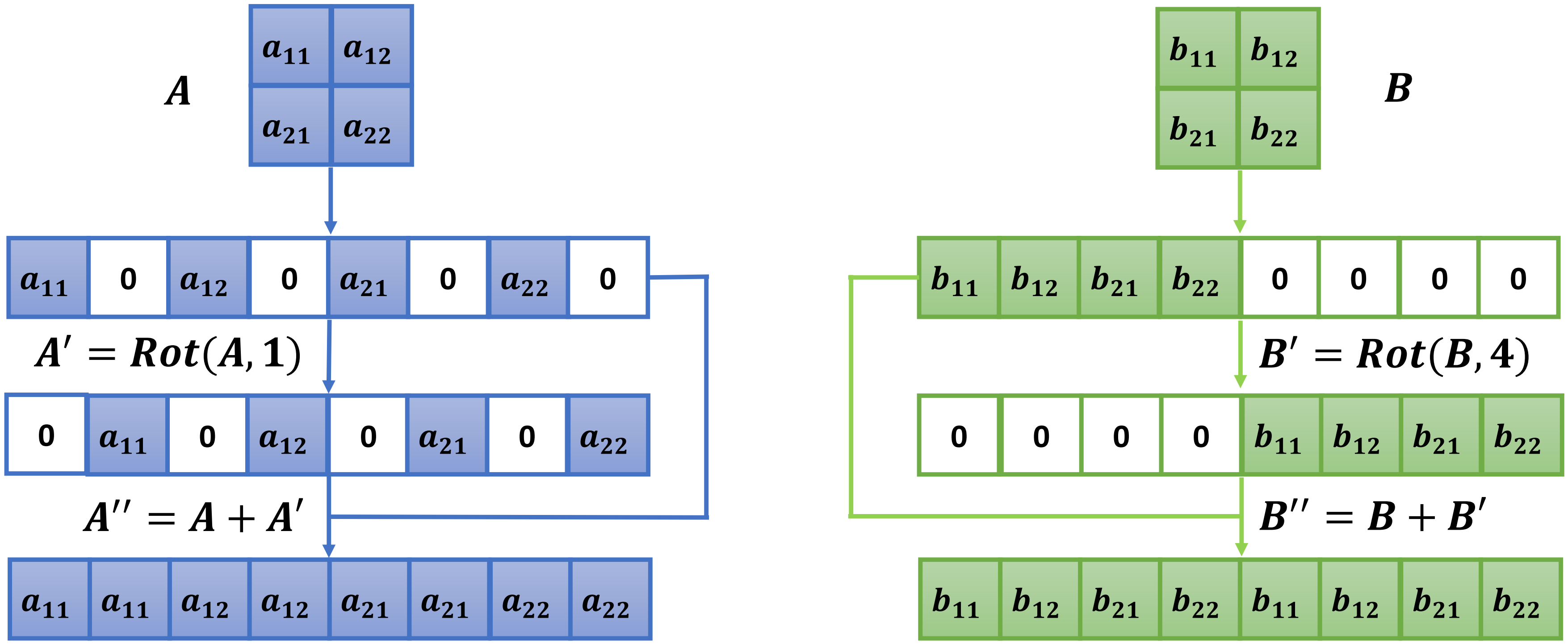


FHE Vectorization

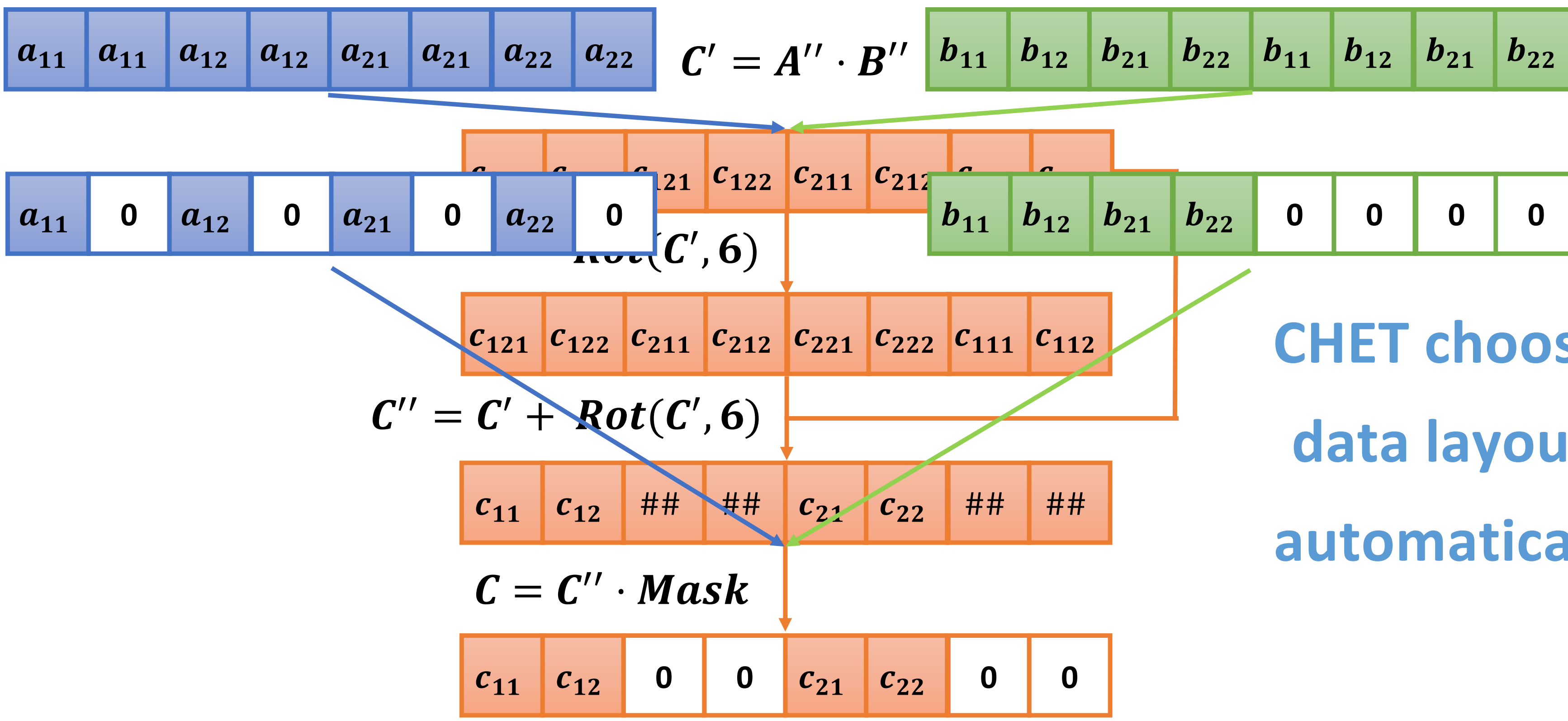
- **Limited set of SIMD instructions**:
  - Element-wise addition
  - Element-wise subtraction
  - Element-wise multiplication
  - Rescale (all elements)
  - Rotation
- Random access of a vector element not supported



Example of Matrix Multiplication: C = A x B

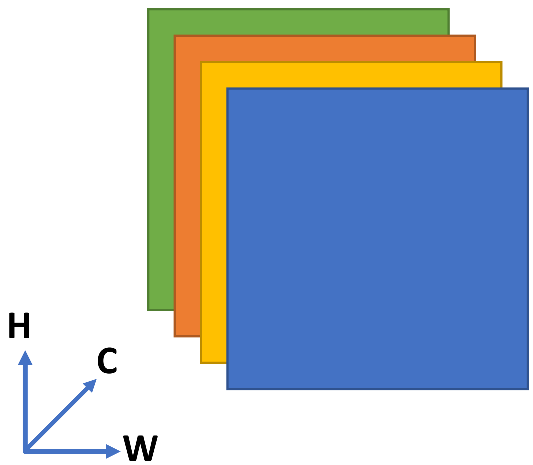


Example of Matrix Multiplication: C = A x B

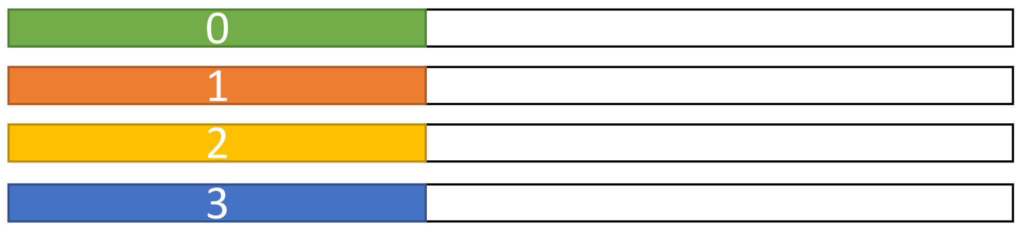


CHET chooses data layouts automatically

Mapping Tensors to Vector of Vectors

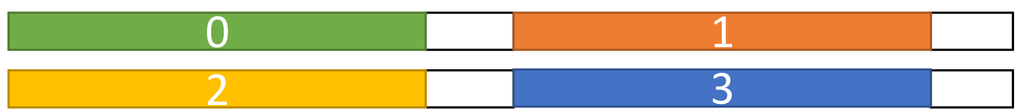


There are many ways to layout tensors into vectors, with different tradeoffs



HW (Height-Width) layout:

- Easier convolutions due to channels being aligned
- Wasted space



CHW (Channel-Height-Width) layout:

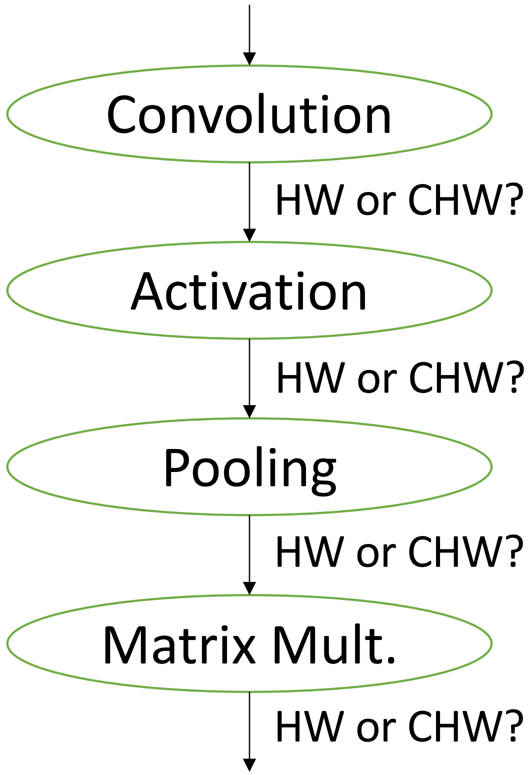
- More efficient space usage
- Convolutions require more rotations

Data Layout Selection

- Search space: explore possible data layouts
- Cost estimation: estimate cost of each search point
- Pick the best-performing one

Data Layout Selection: Search Space

- Search space is exponential
  - 2 choices per tensor operation: **HW** or **CHW**
- Prune search space: use domain knowledge -> limits to only 4 choices for the circuit
  - Convolution faster in **HW** while rest faster in **CHW**
  - Matrix multiplication faster if output is in **CHW**



Data Layout Selection: Cost Estimation

- Cost model for FHE primitives:
    - Asymptotic complexity (specific to FHE scheme)
    - Microbenchmarking to determine constants
- Cost of a circuit: sum the costs for all operations

Experimental Setup

- **Systems:**
  - Hand-written HEAAN
  - CHET with HEAAN
  - CHET with SEAL

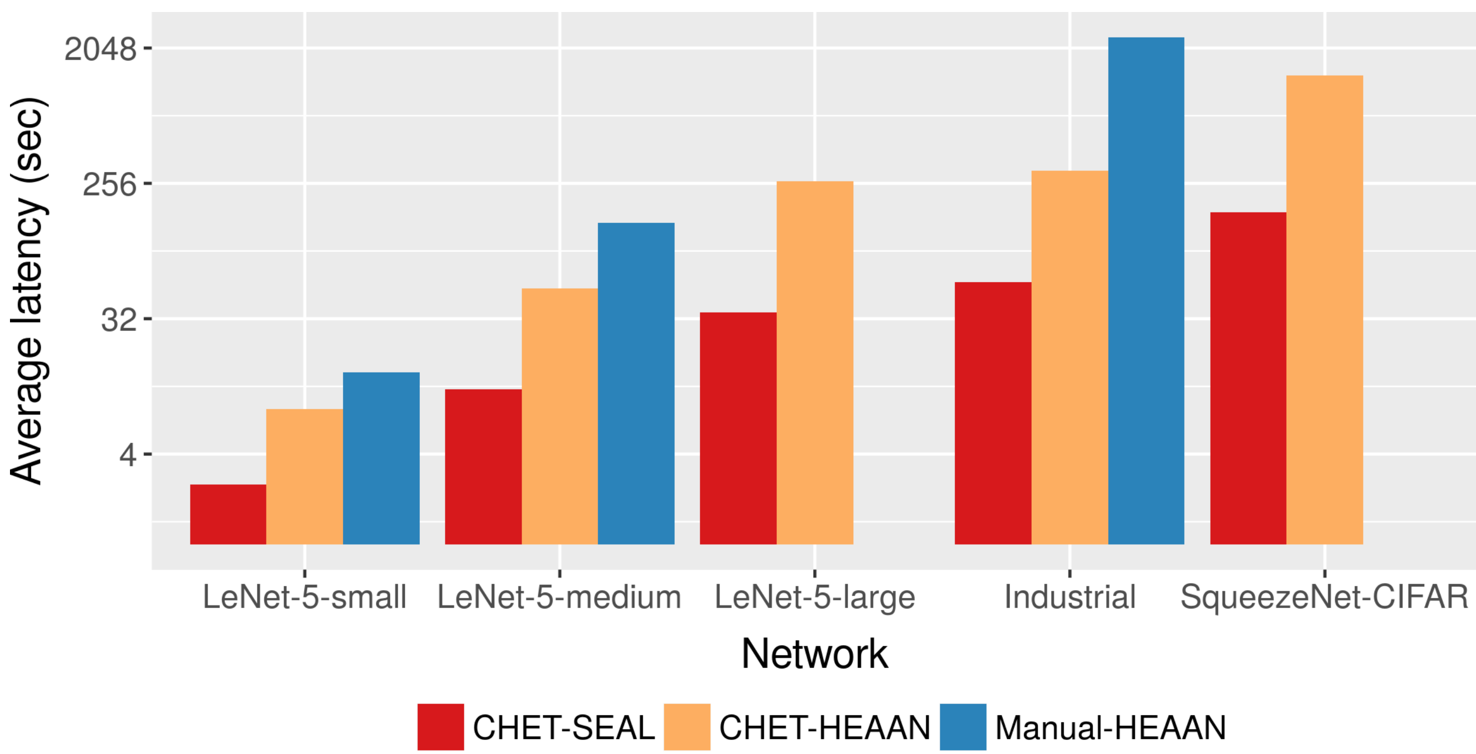
- **Machine:**
  - Dual-socket Intel Xeon E5-2667v3
  - 16 cores
  - 224 GB of memory

- **FHE-compatible Deep Neural Networks (DNN):**

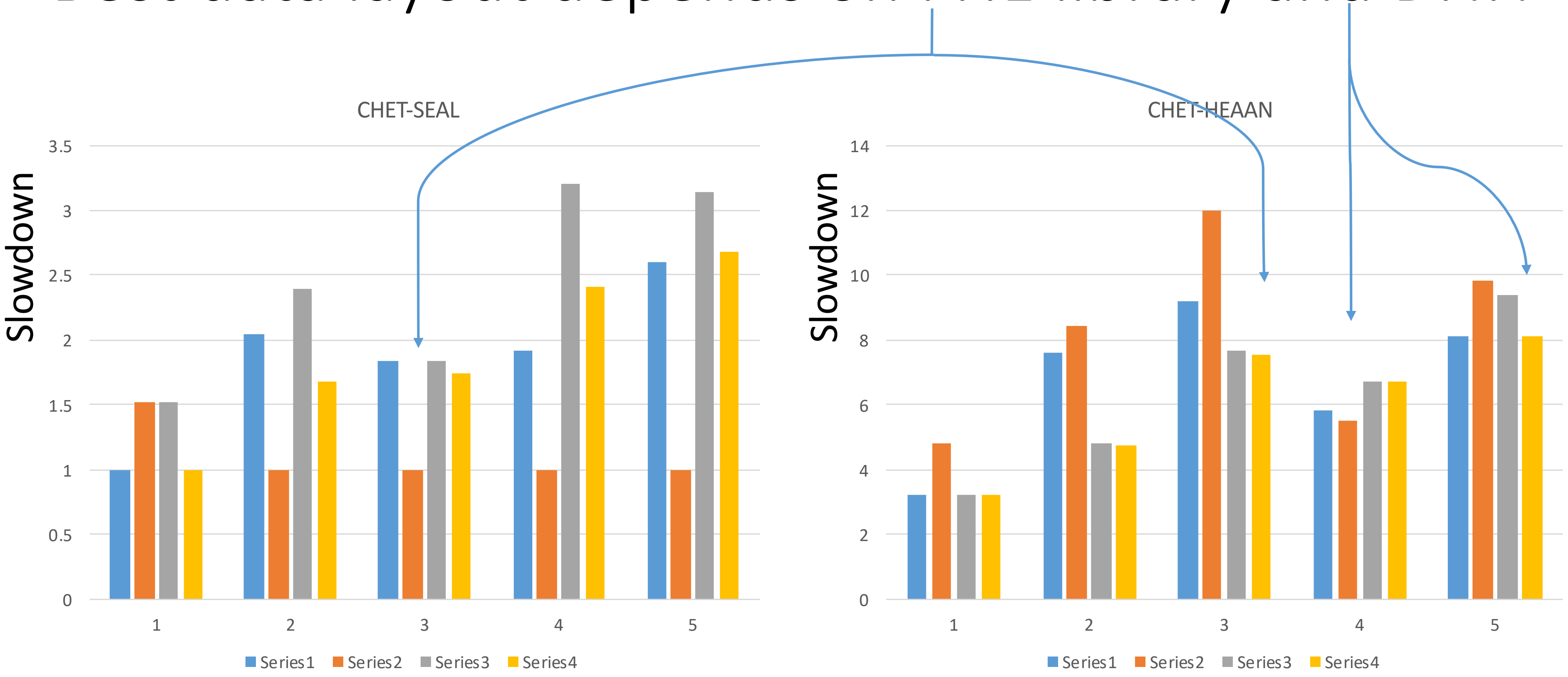
DNN	Dataset	# Layers	# FP ops (M)	Accuracy
LeNet-5-small	MNIST	8	0.2	98.5%
LeNet-5-medium	MNIST	8	5.8	99.0%
LeNet-5-large	MNIST	8	8.7	99.3%
Industrial	-	13	-	-
SqueezeNet-CIFAR	CIFAR-10	19	37.8	81.5%

- **Evaluation:**
  - Latency of image inference (batch size = 1)

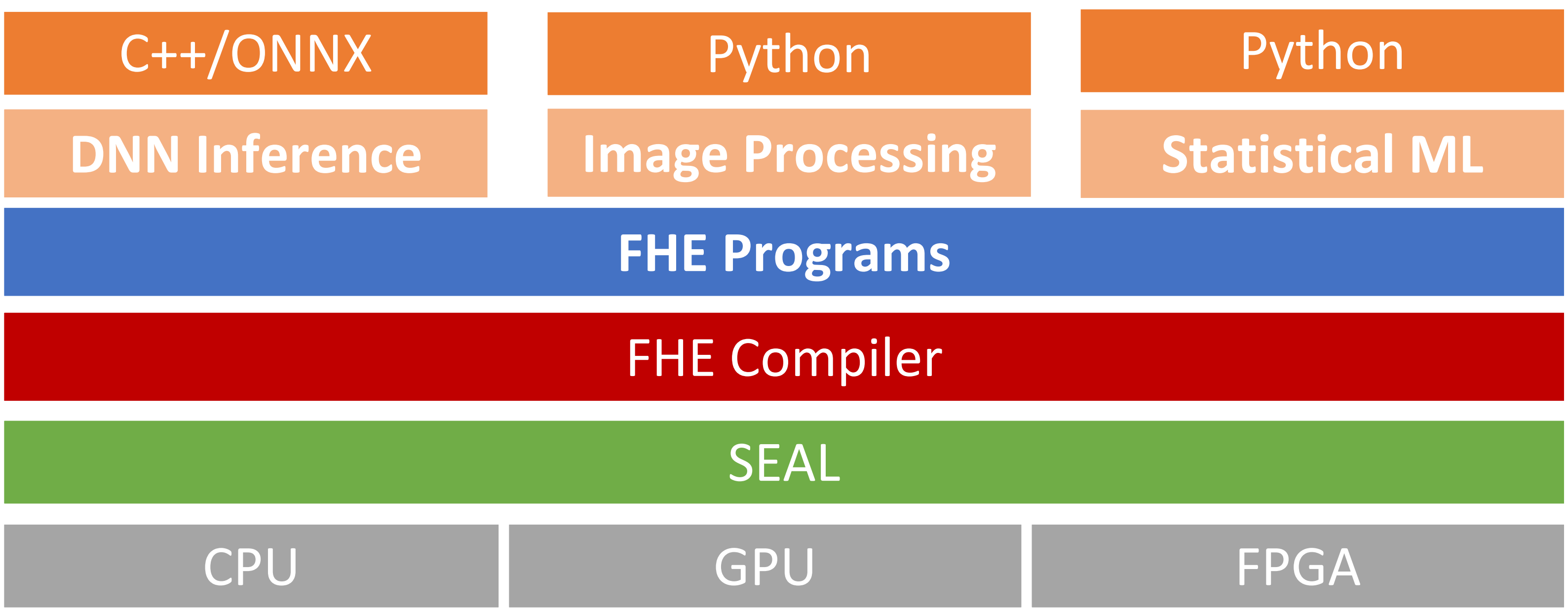
CHET outperforms hand-written implementations



Best data layout depends on FHE library and DNN



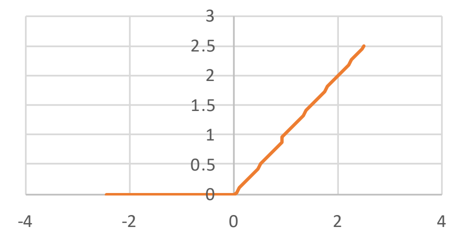
Generalizing CHET (ongoing work)



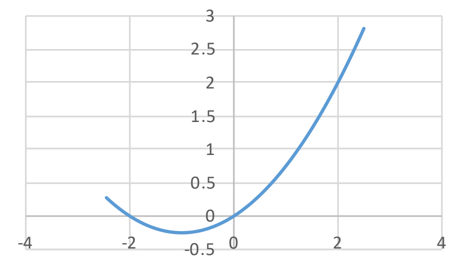
Making Computations FHE compatible

- Cannot evaluate non-polynomial operations
  - ReLU
  - Max pooling

- Options:
  1. Replace with polynomial approximation
  2. Combine Boolean and arithmetic schemes



$ReLU(x)$



$x^2 + a \cdot x$

Conclusions

- Cryptographic computation is a “PL + HPC + Systems” problem
- Currently at 8086 speeds but 2-3x already possible
  - Better encryption schemes, compilers, runtime systems, HW support
- Interesting applications possible at current speeds
- Many PL challenges still open