



Hardware Acceleration Opportunities in Homomorphic Encryption

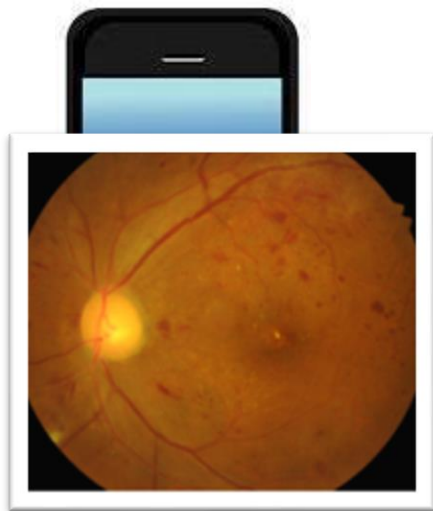
Cryptography on Hardware Platform 2023

Sujoy Sinha Roy

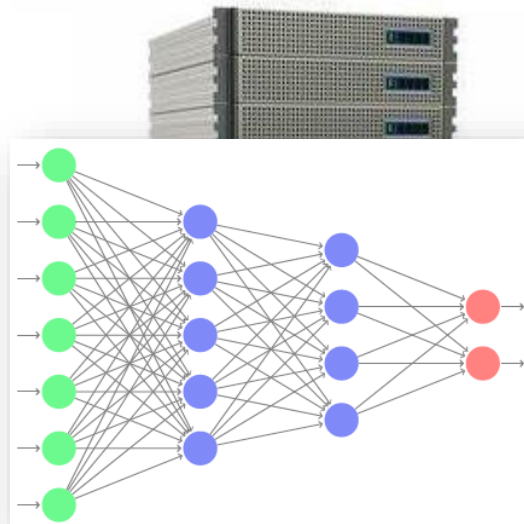
sujoy.sinharoy@iaik.tugraz.at

Privacy-Preserving Outsourcing of Computation

data



foo()

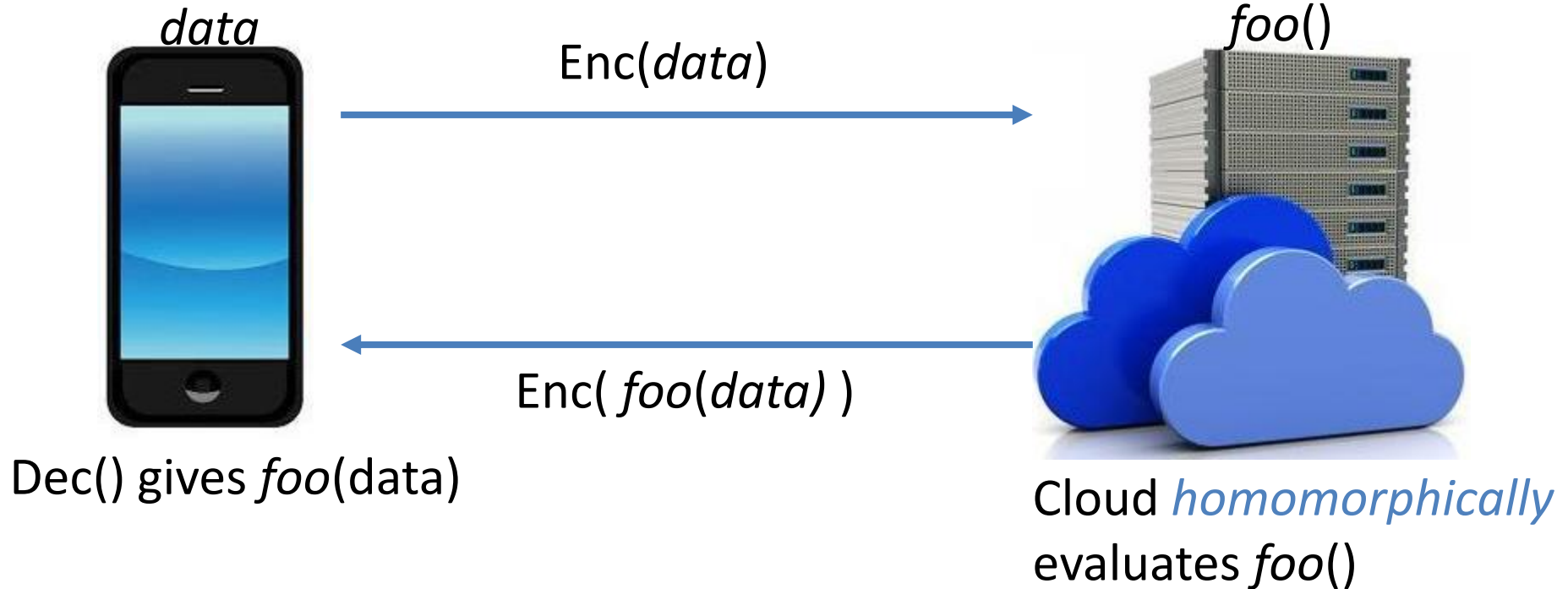


Diabetic Retinopathy [Chao et al., 2019]

User wants to compute $foo(data)$ in the cloud without losing privacy.

Fully Homomorphic Encryption (FHE)

FHE enables computation on encrypted data



Tutorial outline

1. FHE concepts
2. Parallel processing opportunities in FHE (from high-level)
3. Hardware architecture design challenges and methods
4. Results

Definition: Homomorphic Encryption

An encryption scheme $\text{Enc}(\cdot, \cdot)$ is homomorphic for an operation \square on the message space iff

$$\text{Enc}(m_1 \square m_2, k_E) = \text{Enc}(m_1, k_E) \circ \text{Enc}(m_2, k_E)$$

with \circ operation on the ciphertext.

- If $\square = +$ then $\text{Enc}(\cdot, \cdot)$ is additively homomorphic.
- If $\square = \times$ then $\text{Enc}(\cdot, \cdot)$ is multiplicatively homomorphic.

Example: Textbook RSA is multiplicatively homomorphic

- You have encryption of two messages m_1 and m_2 where

$$c_1 = m_1^e \bmod N$$

$$c_2 = m_2^e \bmod N$$

- By multiplying c_1 and c_2 you get

$$c_3 = c_1 \cdot c_2 = (m_1 \cdot m_2)^e \bmod N$$

- Hence, c_3 is encryption of $m_1 \cdot m_2$

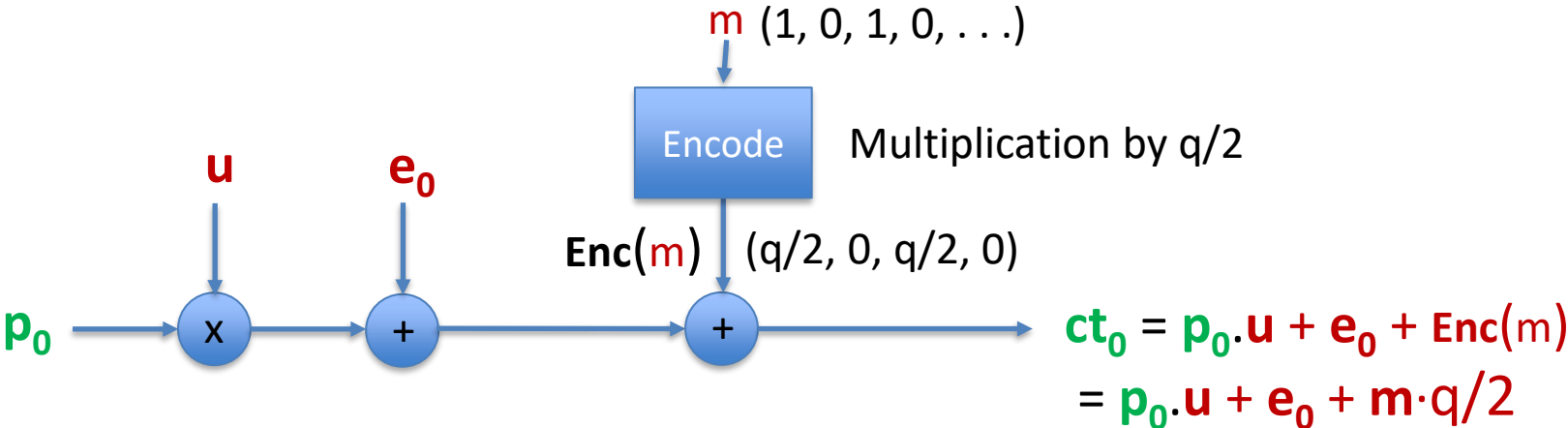
Can we get 'Additive & Multiplicative' Homomorphic Encryption?

Popular constructions of FHE use augmented Ring-LWE public-key encryption

Recap -- Ring LWE Public-Key Encryption (PKE)

Encryption:

- Input: $pk = (p_0, p_1)$, message m
- Output: $ct = (ct_0, ct_1)$

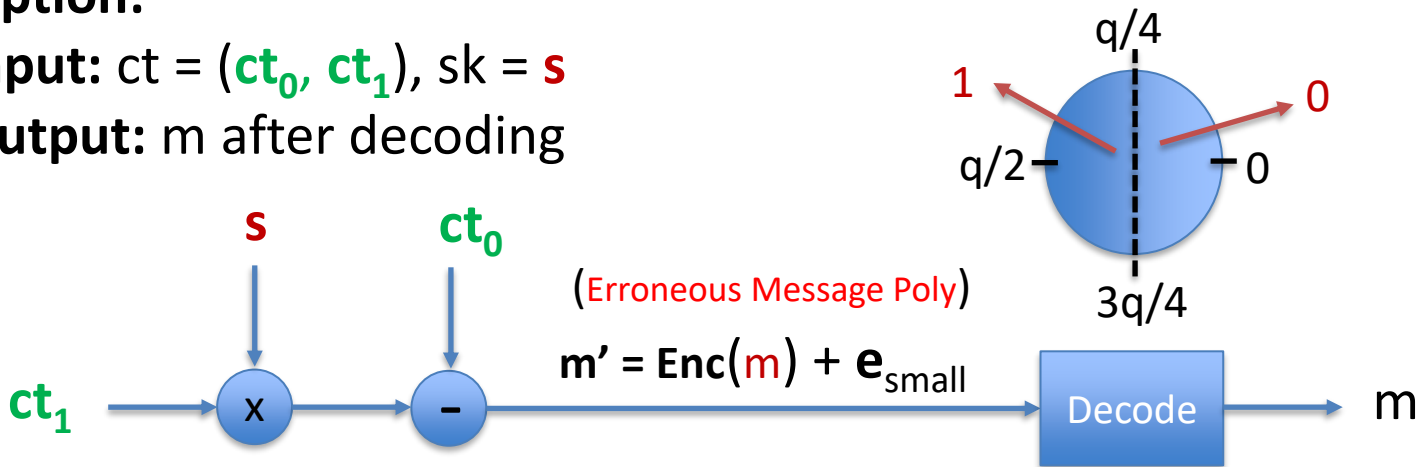


Recap -- Ring LWE Public-Key Encryption (PKE)

Decryption:

Input: $ct = (ct_0, ct_1)$, $sk = s$

Output: m after decoding



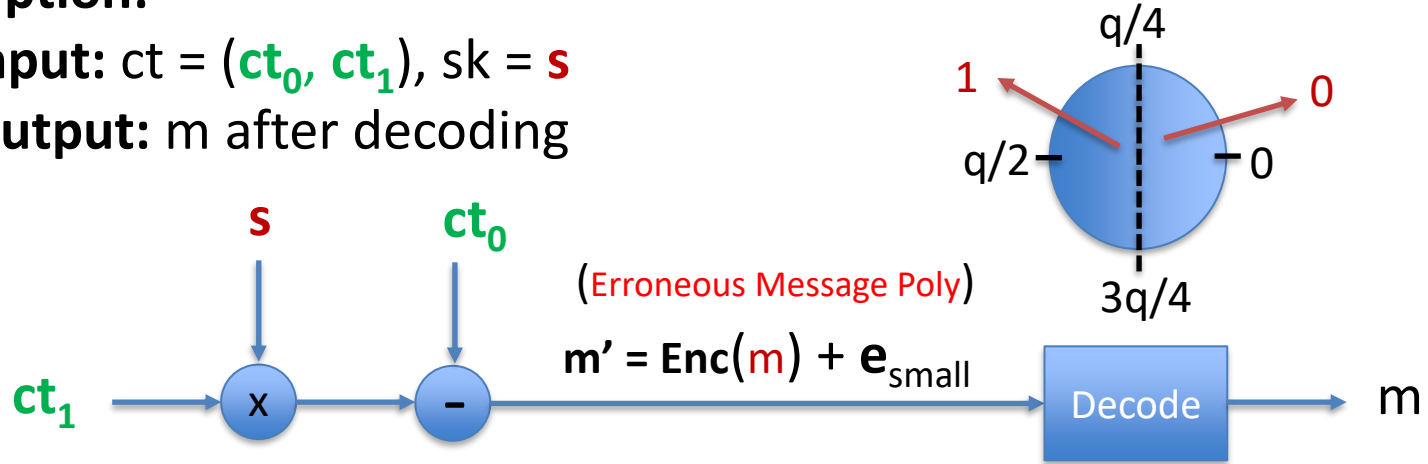
$$\begin{aligned} ct_0 + ct_1 \cdot s &= m' = \text{Enc}(m) + (e \cdot s' + e'' + e' \cdot s) \\ &= \text{Enc}(m) + e_{\text{small}} \end{aligned}$$

Recap -- Ring LWE Public-Key Encryption (PKE)

Decryption:

Input: $ct = (ct_0, ct_1)$, $sk = s$

Output: m after decoding



$$\begin{aligned} ct_0 + ct_1 \cdot s &= m' = \text{Enc}(m) + (e \cdot s' + e'' + e' \cdot s) \\ &= \text{Enc}(m) + e_{\text{small}} \end{aligned}$$

Equivalently,

$$\left\lfloor \frac{ct_0 + ct_1 \cdot s}{q/2} \right\rfloor \text{ mod } 2 = m$$

Ring-LWE PKE – Written with different symbols

Let scale factor $\Delta = q/t$ and t be plaintext modulus, e.g., $t = 2$.

Scalars are in red.

All polynomials are in blue.

Encryption

$$\begin{aligned}e_0, e_1, u &\leftarrow \text{error}(); \\ ct_0 &= p_0 \cdot u + e_0 + \Delta \cdot m \\ ct_1 &= p_1 \cdot u + e_1\end{aligned}$$



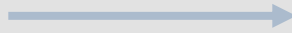
Decryption

$$m = \left\lfloor \frac{ct_0 + ct_1 \cdot s}{\Delta} \right\rfloor \text{ mod } t$$

Ring-LWE PKE shows Homomorphism

Encryption

$$\begin{aligned}e_0, e_1, u &\leftarrow \text{error}(); \\ ct_0 &= p_0 \cdot u + e_0 + \Delta \cdot m \\ ct_1 &= p_1 \cdot u + e_1\end{aligned}$$



Decryption

$$\left\lfloor \frac{ct_0 + ct_1 \cdot s}{\Delta} \right\rfloor \text{ mod } t$$

Now consider two ciphertexts $Ct_A = \{ct_{A0}, ct_{A1}\}$ and $Ct_B = \{ct_{B0}, ct_{B1}\}$

$$\begin{aligned}e_{A0}, e_{A1}, u_A &\leftarrow \text{error}(); \\ ct_{A0} &= p_0 \cdot u_A + e_{A0} + \Delta \cdot m_A \\ ct_{A1} &= p_1 \cdot u_A + e_{A1}\end{aligned}$$

$$\begin{aligned}e_{B0}, e_{B1}, u_B &\leftarrow \text{error}(); \\ ct_{B0} &= p_0 \cdot u_B + e_{B0} + \Delta \cdot m_B \\ ct_{B1} &= p_1 \cdot u_B + e_{B1}\end{aligned}$$

Ring-LWE PKE: Additive Homomorphism

Encryption

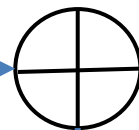
$$\begin{aligned} e_0, e_1, u &\leftarrow \text{error}(); \\ ct_0 &= p_0 \cdot u + e_0 + \Delta \cdot m \\ ct_1 &= p_1 \cdot u + e_1 \end{aligned}$$

Decryption

$$\left\lfloor \frac{ct_0 + ct_1 \cdot s}{\Delta} \right\rfloor \bmod t$$

Now consider two ciphertexts $Ct_A = \{ct_{A0}, ct_{A1}\}$ and $Ct_B = \{ct_{B0}, ct_{B1}\}$

$$\begin{aligned} e_{A0}, e_{A1}, u_A &\leftarrow \text{error}(); \\ ct_{A0} &= p_0 \cdot u_A + e_{A0} + \Delta \cdot m_A \\ ct_{A1} &= p_1 \cdot u_A + e_{A1} \end{aligned}$$



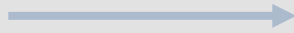
$$\begin{aligned} e_{B0}, e_{B1}, u_B &\leftarrow \text{error}(); \\ ct_{B0} &= p_0 \cdot u_B + e_{B0} + \Delta \cdot m_B \\ ct_{B1} &= p_1 \cdot u_B + e_{B1} \end{aligned}$$

$$\begin{aligned} ct_{C0} &= p_0 \cdot (u_A + u_B) + (e_{A0} + e_{B0}) + \Delta \cdot (m_A + m_B) \\ ct_{C1} &= p_1 \cdot (u_A + u_B) + (e_{A1} + e_{B1}) \end{aligned}$$

Ring-LWE PKE: Multiplicative Homomorphism

Encryption

$$\begin{aligned} e_0, e_1, u &\leftarrow \text{error}(); \\ ct_0 &= p_0 \cdot u + e_0 + \Delta \cdot m \\ ct_1 &= p_1 \cdot u + e_1 \end{aligned}$$

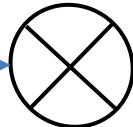


Decryption

$$\left\lfloor \frac{ct_0 + ct_1 \cdot s}{\Delta} \right\rfloor \text{ mod } t$$

Now consider two ciphertexts $Ct_A = \{ct_{A0}, ct_{A1}\}$ and $Ct_B = \{ct_{B0}, ct_{B1}\}$

$$\begin{aligned} e_{A0}, e_{A1}, u_A &\leftarrow \text{error}(); \\ ct_{A0} &= p_0 \cdot u_A + e_{A0} + \Delta \cdot m_A \\ ct_{A1} &= p_1 \cdot u_A + e_{A1} \end{aligned}$$



$$\begin{aligned} e_{B0}, e_{B1}, u_B &\leftarrow \text{error}(); \\ ct_{B0} &= p_0 \cdot u_B + e_{B0} + \Delta \cdot m_B \\ ct_{B1} &= p_1 \cdot u_B + e_{B1} \end{aligned}$$

Polynomial multiplication

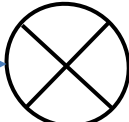
$$ct_{A0} * ct_{B0} \rightarrow (\text{noisy crap}) + \Delta^2 \cdot (m_A \times m_B)$$

Ring-LWE PKE: Multiplicative Homomorphism

| Encryption | Decryption |
|---|--|
| $e_0, e_1, u \leftarrow \text{error}();$ $ct_0 = p_0 \cdot u + e_0 + \Delta \cdot m$ $ct_1 = p_1 \cdot u + e_1$ | $\left[\frac{ct_0 + ct_1 \cdot s}{\Delta} \right] \text{ mod } t$ |

Now consider two ciphertexts $Ct_A = \{ct_{A0}, ct_{A1}\}$ and $Ct_B = \{ct_{B0}, ct_{B1}\}$

$$e_{A0}, e_{A1}, u_A \leftarrow \text{error}();$$
$$ct_{A0} = p_0 \cdot u_A + e_{A0} + \Delta \cdot m_A$$
$$ct_{A1} = p_1 \cdot u_A + e_{A1}$$

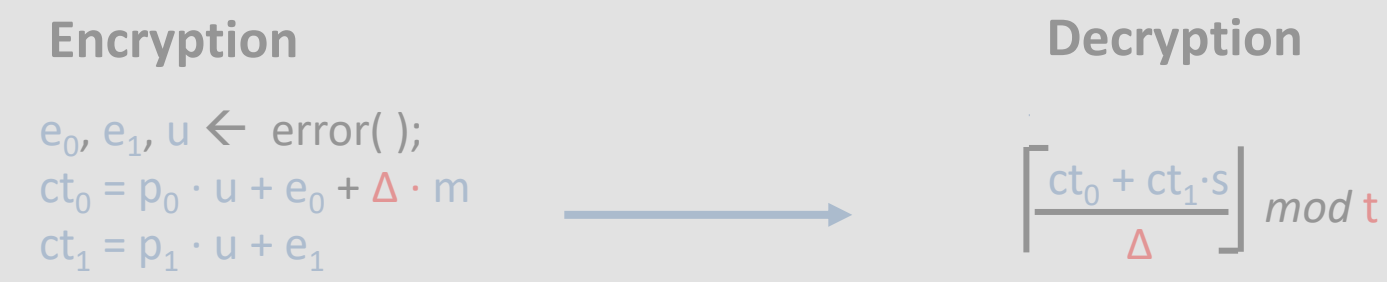


$$e_{B0}, e_{B1}, u_B \leftarrow \text{error}();$$
$$ct_{B0} = p_0 \cdot u_B + e_{B0} + \Delta \cdot m_B$$
$$ct_{B1} = p_1 \cdot u_B + e_{B1}$$

Intuition →

Polynomial multiplication
 $ct_{A0} * ct_{B0} \rightarrow (\text{noisy crap}) + \Delta^2 \cdot (m_A \times m_B)$
After dividing the expression by Δ we get:
 $(\text{noisy crap})/\Delta + \Delta \cdot (m_A \times m_B)$

Ring-LWE PKE: Multiplicative Homomorphism



Now consider two ciphertexts $Ct_A = \{ct_{A0}, ct_{A1}\}$ and $Ct_B = \{ct_{B0}, ct_{B1}\}$

$$\begin{aligned} e_{A0}, e_{A1}, u_A &\leftarrow \text{error}(\); \\ ct_{A0} &= p_0 \cdot u_A + e_{A0} + \Delta \cdot m_A \\ ct_{A1} &= p_1 \cdot u_A + e_{A1} \end{aligned}$$



$$\begin{aligned} e_{B0}, e_{B1}, u_B &\leftarrow \text{error}(\); \\ ct_{B0} &= p_0 \cdot u_B + e_{B0} + \Delta \cdot m_B \\ ct_{B1} &= p_1 \cdot u_B + e_{B1} \end{aligned}$$

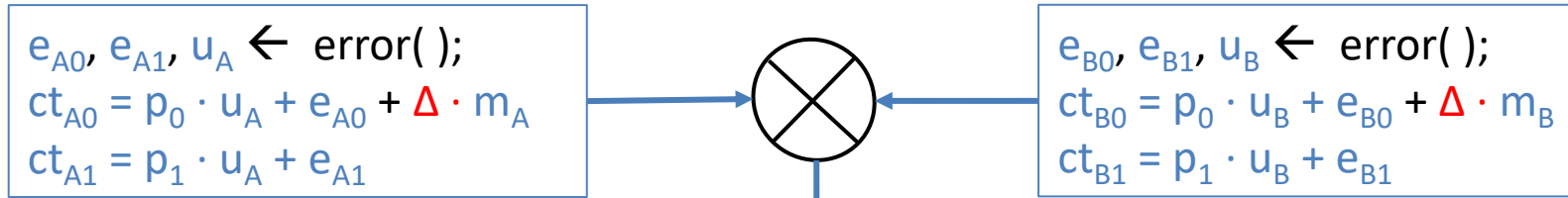
This looks like an encryption of $(m_A \times m_B)$

Polynomial multiplication
 $ct_{A0} * ct_{B0} \rightarrow (\text{noisy crap}) + \Delta^2 \cdot (m_A \times m_B)$
After dividing the expression by Δ we get:
 $(\text{noisy crap})/\Delta + \Delta \cdot (m_A \times m_B)$

Ring-LWE PKE: Multiplicative Homomorphism



Now consider two ciphertexts $Ct_A = \{ct_{A0}, ct_{A1}\}$ and $Ct_B = \{ct_{B0}, ct_{B1}\}$

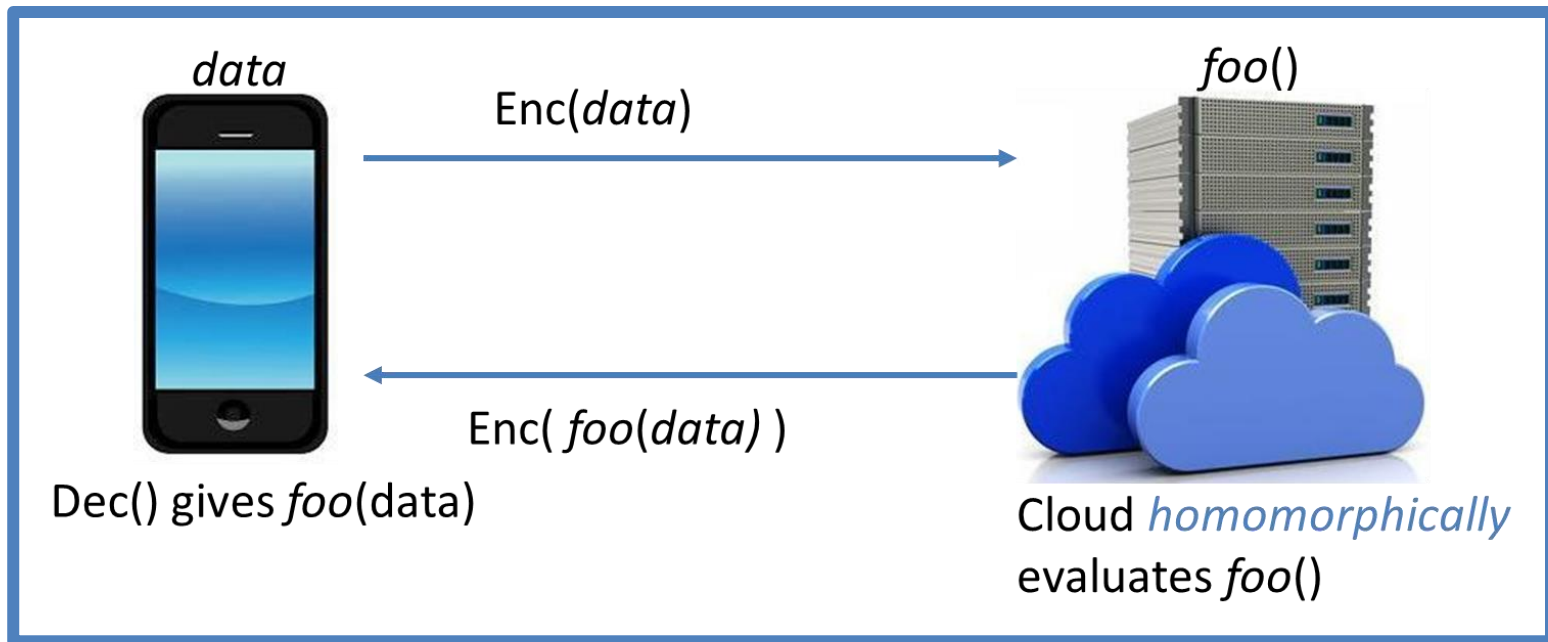


Polynomial multiplication
 $ct_{A0} * ct_{B0} \rightarrow (\text{noisy crap}) + \Delta^2 \cdot (m_A \times m_B)$
After dividing the expression by Δ we get:
 $(\text{noisy crap})/\Delta + \Delta \cdot (m_A \times m_B)$

That is the basic idea only.

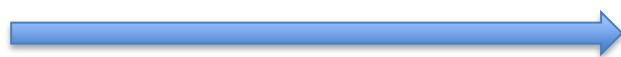
Actual Mult is a lot more complex!

The Biggest Problem in FHE



$foo(data)$

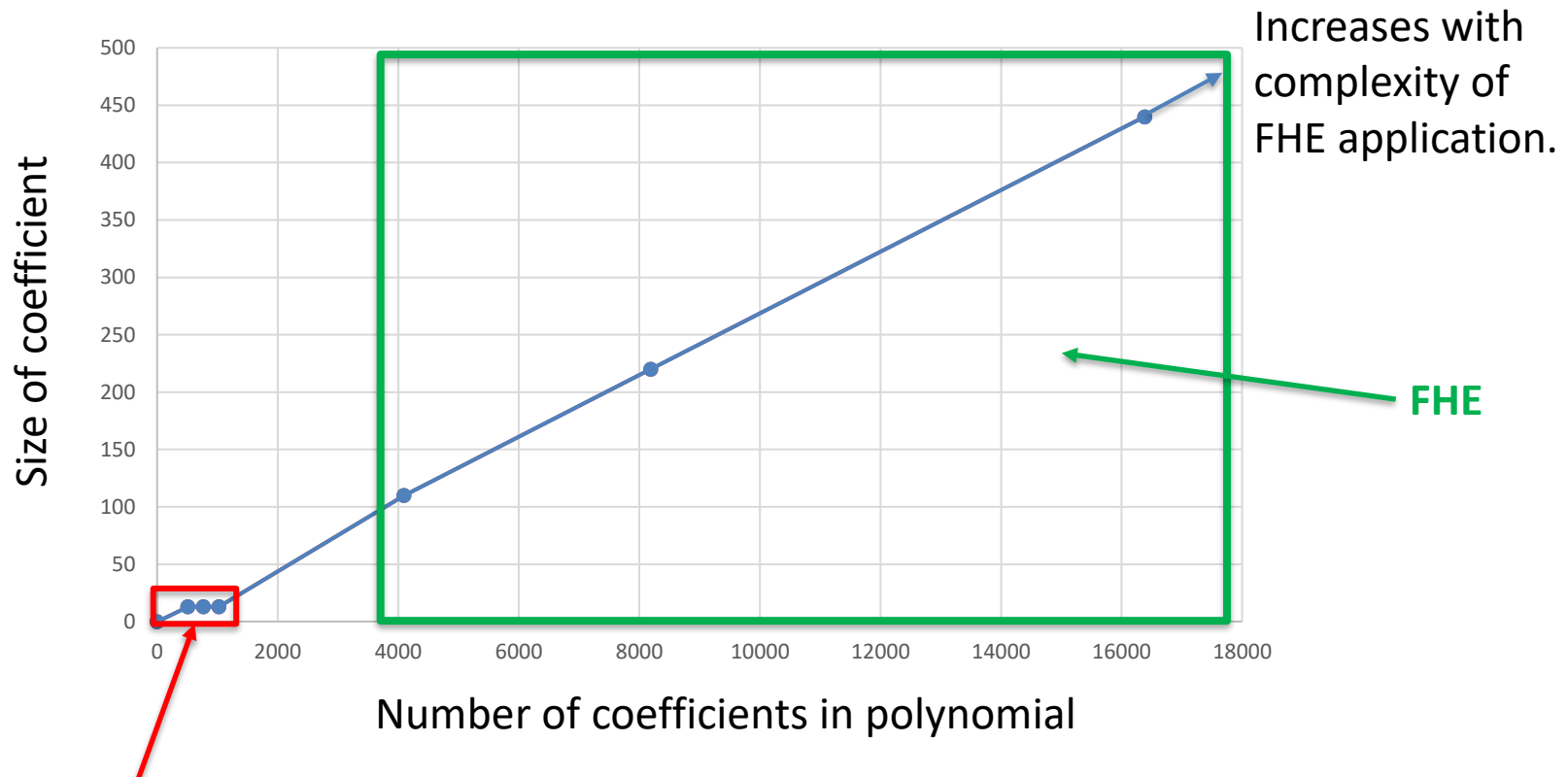
Takes 1s



$foo(Enc(data))$

Takes 10^4 to 10^5 s

Polynomial size



Post-quantum crypto

FHE

FHE does lots of (large) polynomial arithmetic.

How to accelerate FHE?

Tutorial outline

1. FHE concepts
- 2. Parallel processing opportunities in FHE (from high-level)**
3. Hardware architecture design challenges and methods
4. Results

What makes acceleration of FHE very challenging?

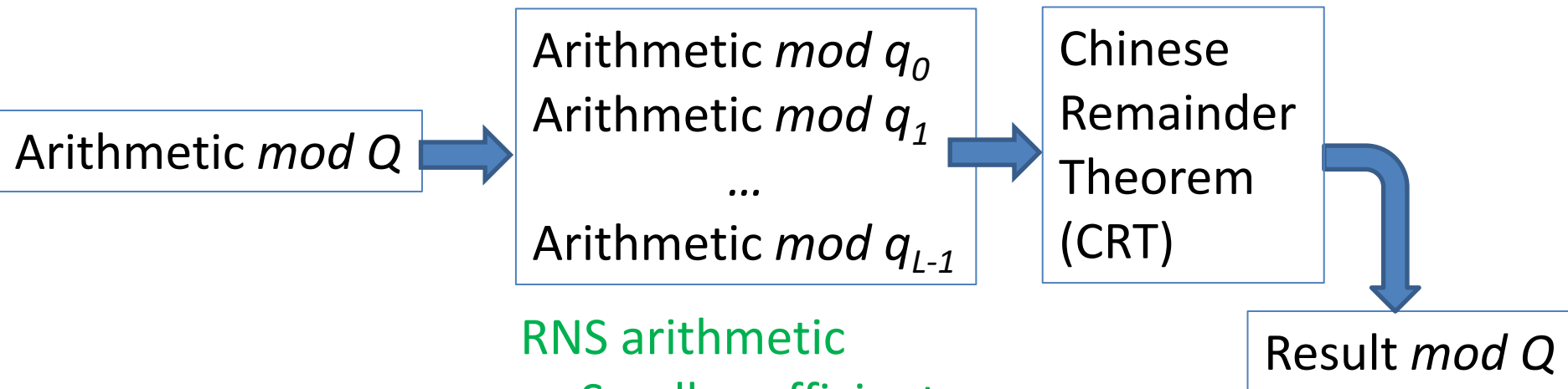
- Lots of polynomial arithmetic operations
 - Large degree polynomial arithmetic
 - Long integer arithmetic
- Memory management
 - Ciphertexts could be several MBs
 - On-Chip memory is limited
 - Off-Chip data transfer is very slow

What makes acceleration of FHE very challenging?

- Lots of polynomial arithmetic operations
 - Large degree polynomial arithmetic
 - **Long integer arithmetic** This problem is solved using CRT
- Memory management
 - Ciphertexts could be several MBs
 - On-Chip memory is limited
 - Off-Chip data transfer is very slow

Dealing with long-int coefficients using RNS

1. Take a modulus $Q = \prod_0^{L-1} q_i$ where q_i are coprime.
2. Use Residue Number System (RNS).



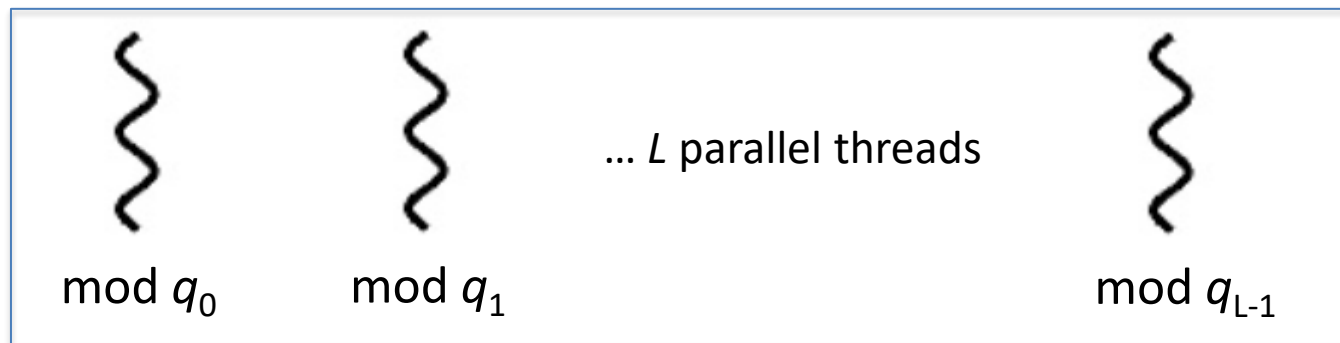
RNS arithmetic

- Small coefficients
- Parallel computation

E.g., Parallel computation flow with CRT

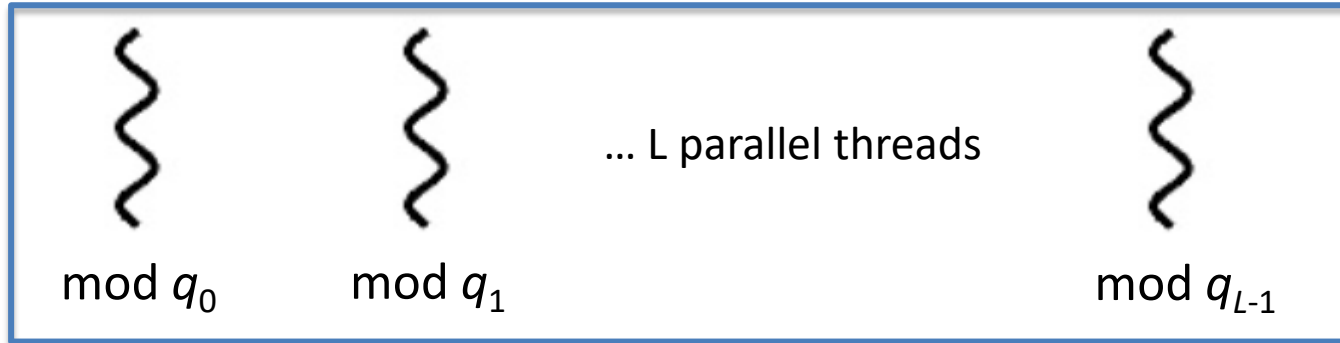
Ciphertexts are polynomials in $R_Q = \mathbb{Z}_Q / \langle X^n + 1 \rangle$
E.g., $\log(Q) = 500$, $n = 2^{15}$

Let $Q = \prod q_i$ where q_i are NTT primes.
Apply Residue Number System (RNS)



Chinese Remainder Theorem (CRT) to obtain R_Q
(Used during modulus switching steps)

... (1) Residue polynomial arithmetic layer



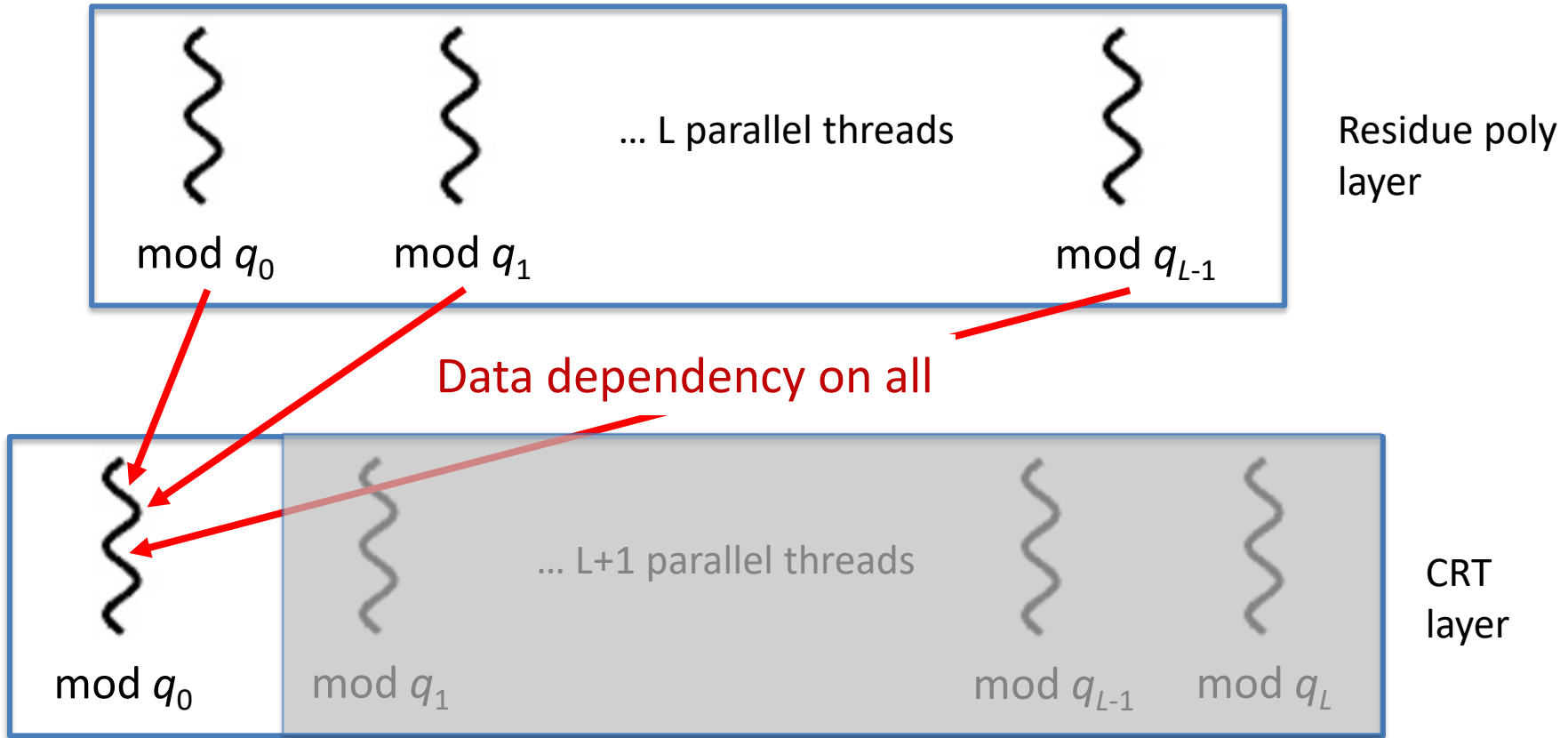
Data flow diagram



Hardware acceleration

*RPAU stands for 'Residue Polynomial Arithmetic Unit'

... (2) Residue polynomial layer ↔ CRT layer

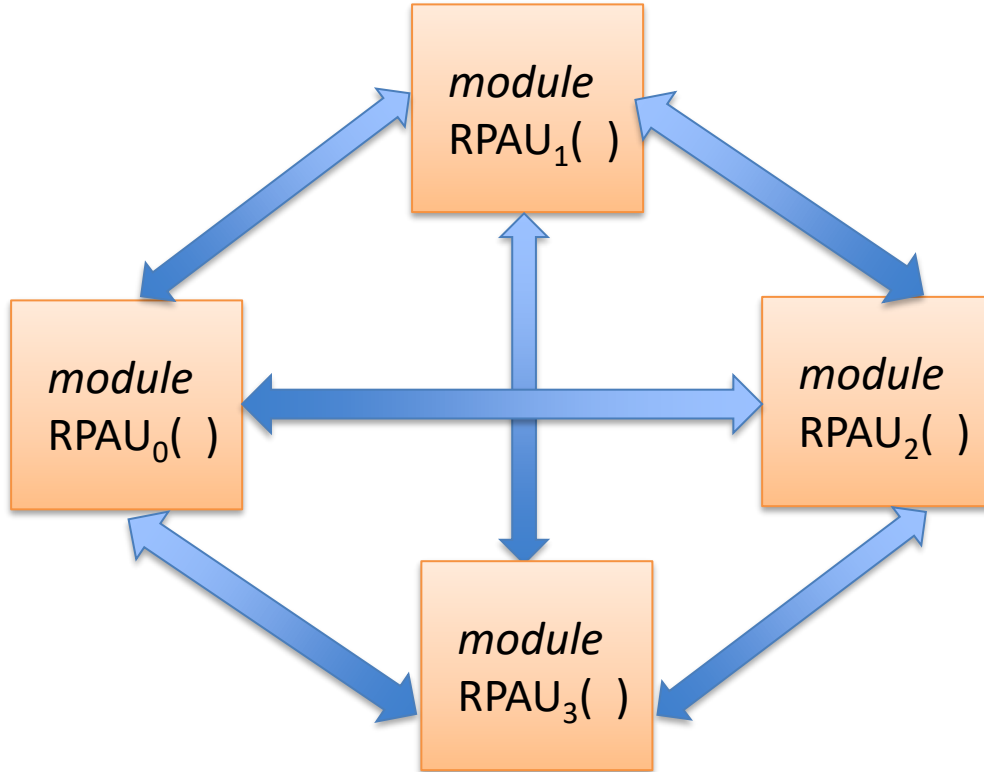


Each thread in CRT layer combines ***all threads*** from previous layer.

... (3) Residue polynomial layer ↔ CRT layer

Therefore, threads or RPAUs need to exchange data with each other.

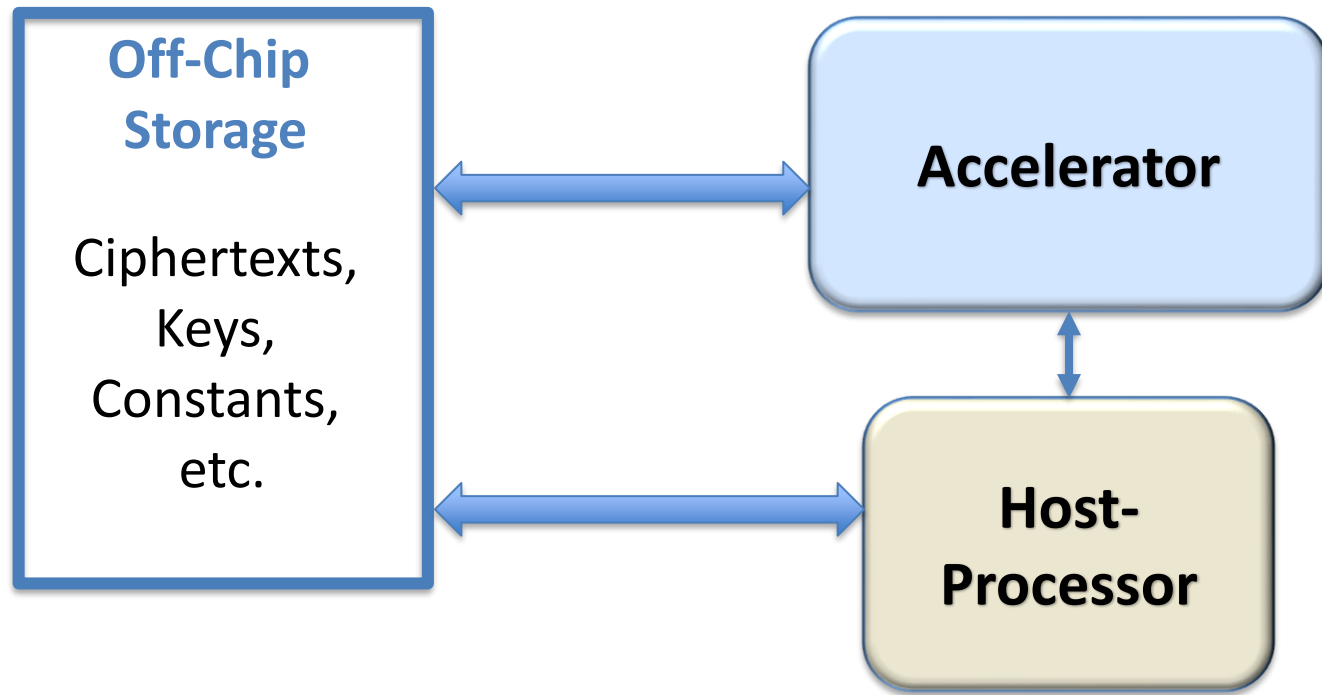
Example



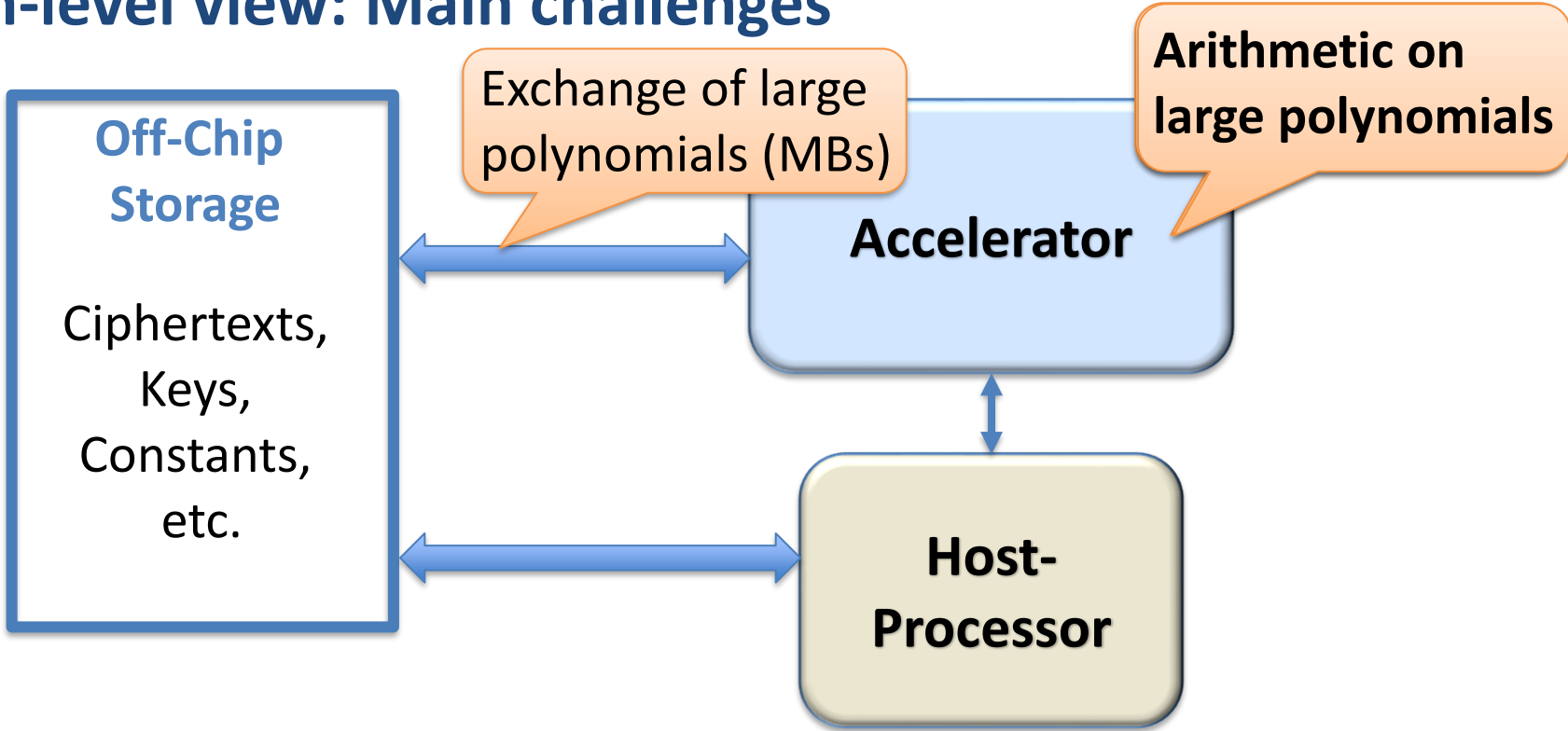
Tutorial outline

1. FHE concepts
2. Parallel processing opportunities in FHE (from high-level)
- 3. Hardware architecture design challenges and methods**
4. Results

System-level view



System-level view: Main challenges



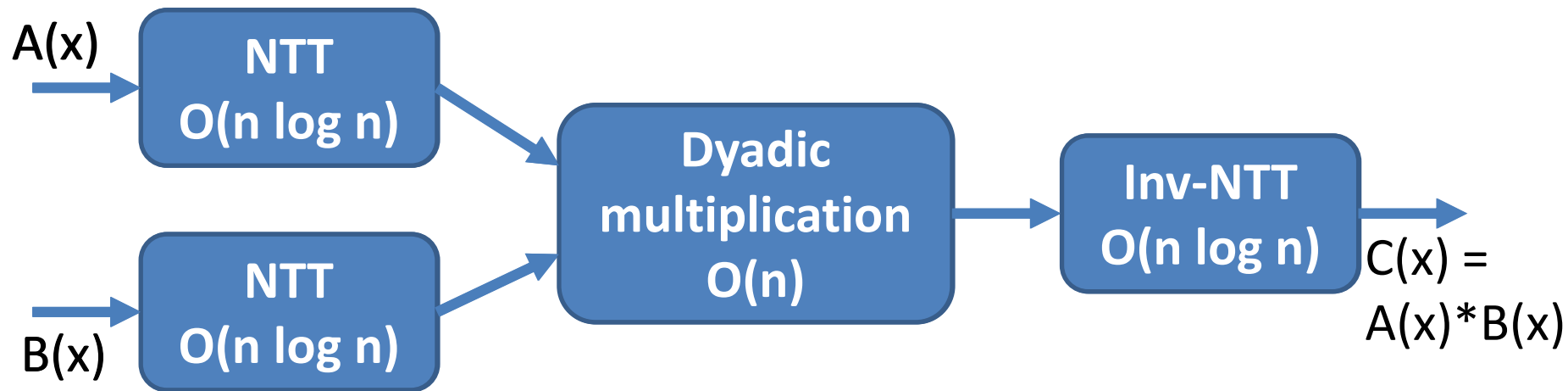
How to multiply two **very large** polynomials?

- Schoolbook multiplication: $O(n^2)$
- Karatsuba multiplication: $O(n^{1.585})$
- Toom-Cook (generalization of Karatsuba)
- **Fast Fourier Transform (FFT) multiplication: $O(n \log n)$**

FFT is the best choice

Asymptotic complexity plays its role.

NTT-based Polynomial Multiplication



NTT or Number Theoretic Transform is special FFT with integers.

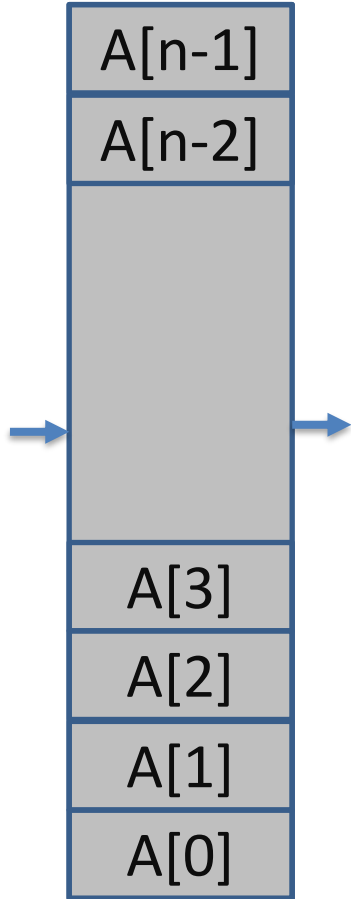
Let's consider an application example.

Polynomial size $n = 2^{15}$

And $\log(q_i) = 60$

NTT and of a polynomial $A[]$

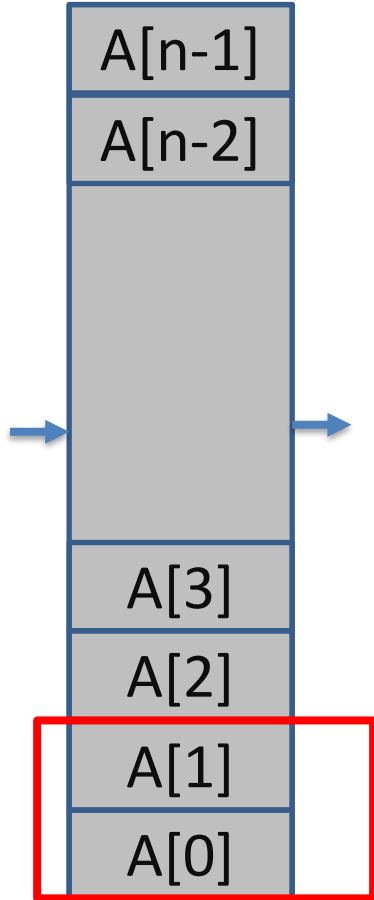
Simplified NTT loops



```
for (m=2; m<=n; m=2m) {  
    for (j=0; j<=m/2-1; j++) {  
        for (k=0; j<n; k=k+m) {  
            index = f(m, j, k);  
            Butterfly(A[index], A[index+m/2]);  
        }  
    }  
}
```

NTT and Memory access

Simplified NTT loops

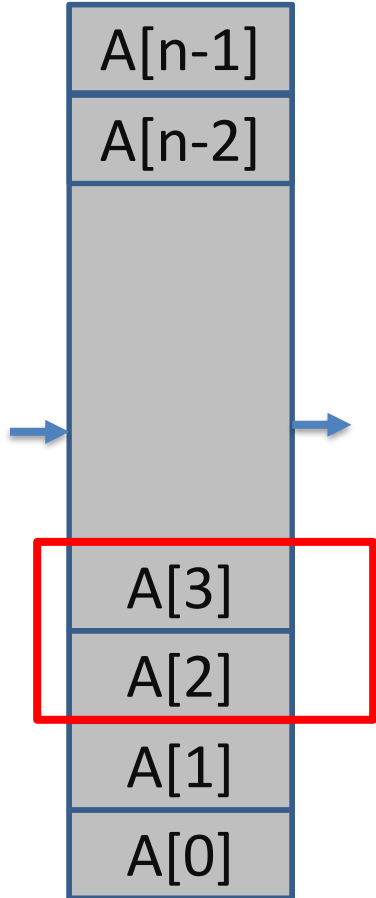


```
for (m=2; m<=n; m=2*m) {  
    for (j=0; j<=m/2-1; j++) {  
        for (k=0; k<n; k=k+m) {  
            index = f(m, j, k);  
            Butterfly(A[index], A[index+m/2]);  
        }  
    }  
}
```

NTT starts with $m=2$
Butterfly($A[0]$, $A[1]$)

NTT and Memory access

Simplified NTT loops

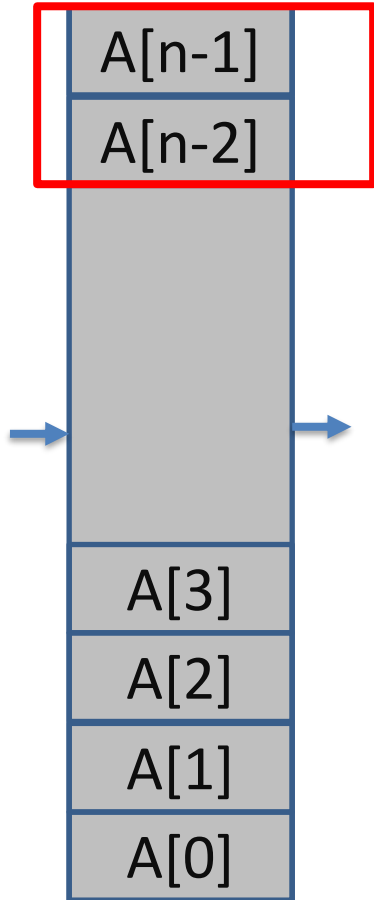


```
for (m=2; m<=n; m=2*m) {  
    for (j=0; j<=m/2-1; j++) {  
        for (k=0; k<n; k=k+m) {  
            index = f(m, j, k);  
            Butterfly(A[index], A[index+m/2]);  
        }  
    }  
}
```

... with $m=2$
Butterfly(A[2], A[3])

NTT and Memory access

Simplified NTT loops

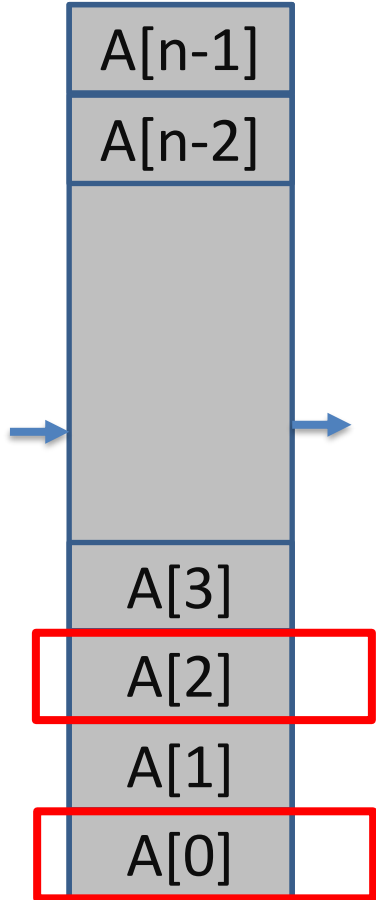


```
for (m=2; m<=n; m=2m) {  
    for (j=0; j<=m/2-1; j++) {  
        for (k=0; j<n; k=k+m) {  
            index = f(m, j, k);  
            Butterfly(A[index], A[index+m/2]);  
        }  
    }  
}
```

... with $m=2$, finally
 $\text{Butterfly}(A[n-2], A[n-1])$

NTT and Memory access

Simplified NTT loops



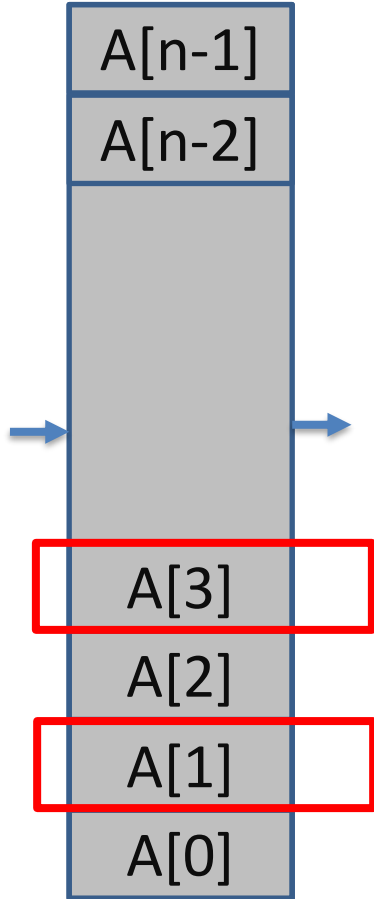
```
for (m=2; m<=n; m=2*m) {  
    for (j=0; j<=m/2-1; j++) {  
        for (k=0; k<n; k=k+m) {  
            index = f(m, j, k);  
            Butterfly(A[index], A[index+m/2]);  
        }  
    }  
}
```

Next, m increments to $m=4$.

Butterfly($A[0]$, $A[2]$), Butterfly($A[4]$, $A[6]$) ...

NTT and Memory access

Simplified NTT loops



```
for (m=2; m<=n; m=2*m) {  
    for (j=0; j<=m/2-1; j++) {  
        for (k=0; k<n; k=k+m) {  
            index = f(m, j, k);  
            Butterfly(A[index], A[index+m/2]);  
        }  
    }  
}
```

Next, m increments to $m=4$.

Butterfly($A[1]$, $A[3]$), Butterfly($A[5]$, $A[7]$) ...

Can we speedup polynomial multiplication using several NTT cores in parallel?

Answer: Yes

Can we speedup polynomial multiplication using several NTT cores in parallel?

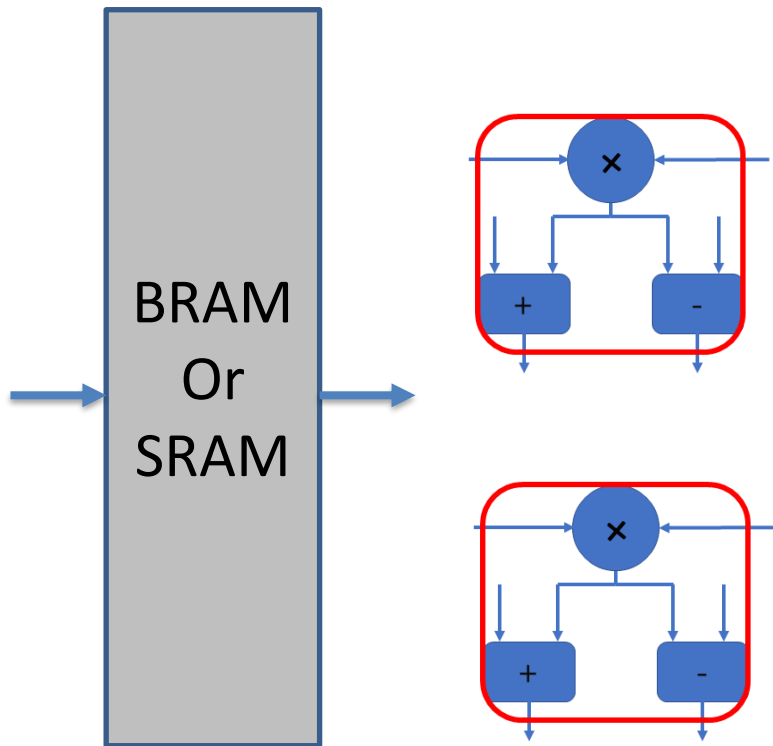
Answer: Yes

Is parallel NTT easy to implement?

Answer: Complexity of implementation increases with number of cores

Parallel NTT

Challenge 1: Port limitation in BRAM or SRAM

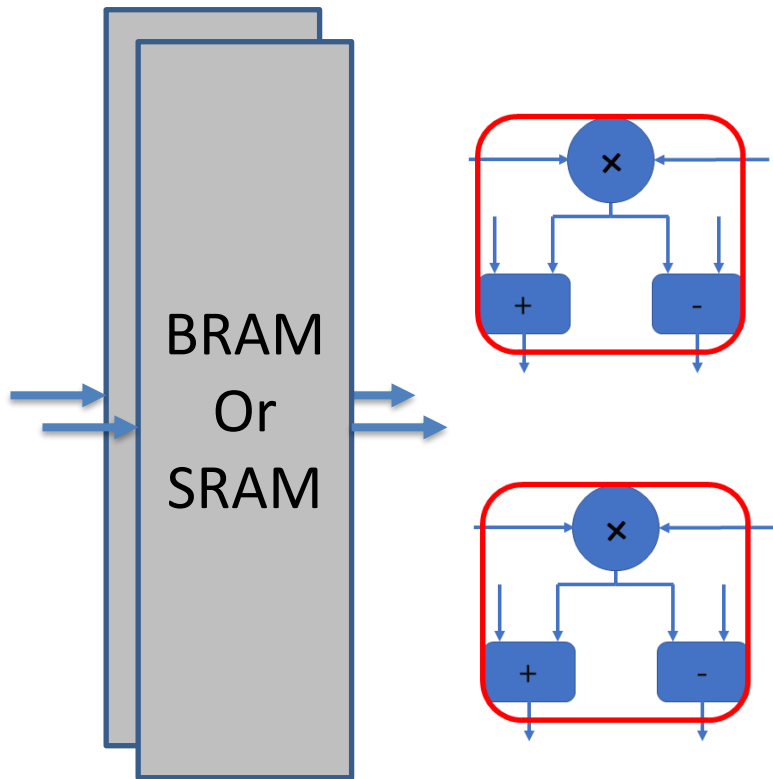


Problem:

- One BRAM has only two ports.
- Each NTT core needs two ports

Parallel NTT

Challenge 1: Port limitation in BRAM or SRAM



Problem:

- One BRAM has only two ports.
- Each NTT core needs two ports

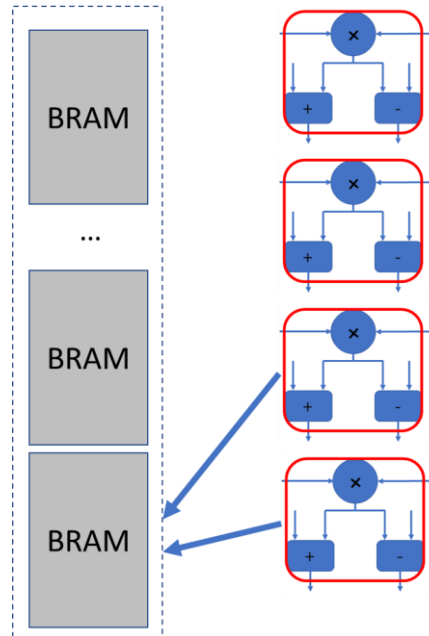
Solution: Use BRAMs in parallel.

New problem: How to distribute data?

Parallel NTT

Challenge 2: Memory access conflicts

Two or more cores try to read/write the same BRAM element.
But BRAM has a limited number of ports to satisfy one core.

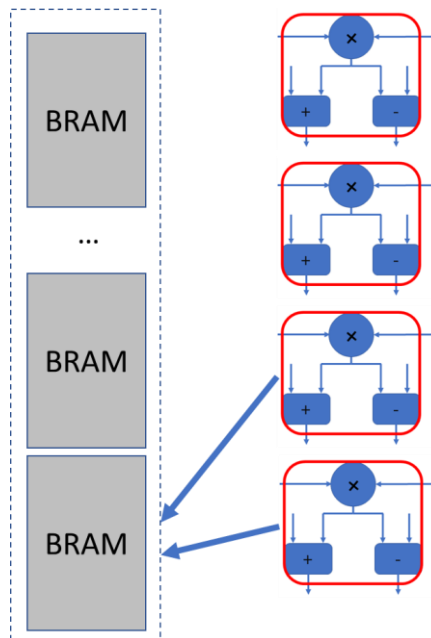


Problem:
Two cores are trying to access
the same BRAM.

Parallel NTT

Challenge 2: Memory access conflicts

Two or more cores try to read/write the same BRAM element.
But BRAM has a limited number of ports to satisfy one core.



Problem:

Two cores are trying to access the same BRAM.

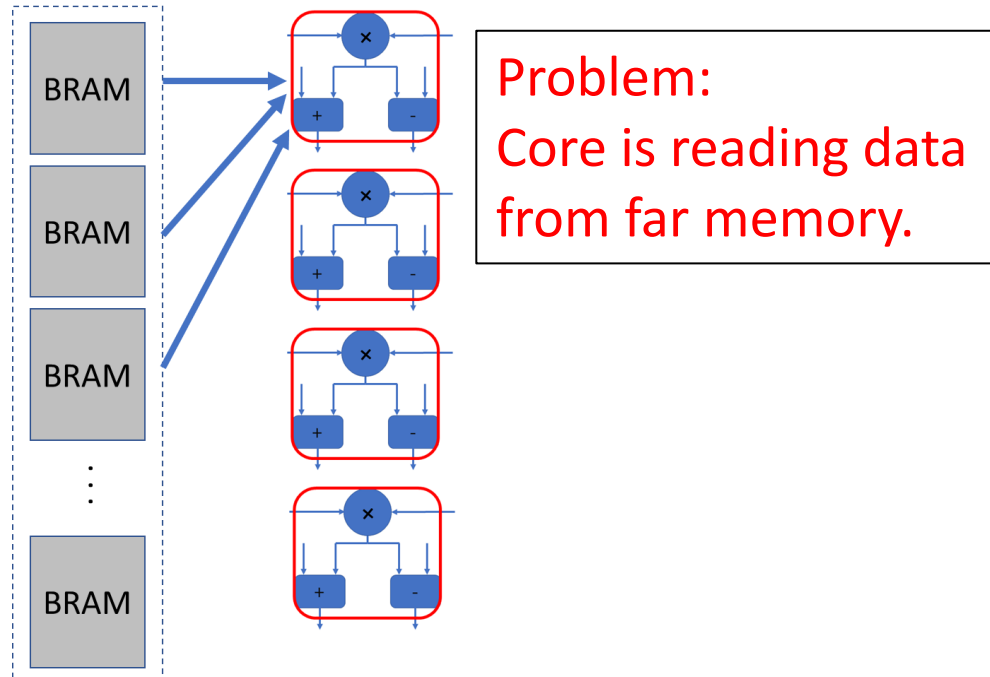
Solution: Make BRAM accesses mutually exclusive.

Parallel NTT

Challenge 3: Data routing

Core requires data from distant BRAM memory

- Long routing of data wires \rightarrow slow clock frequency

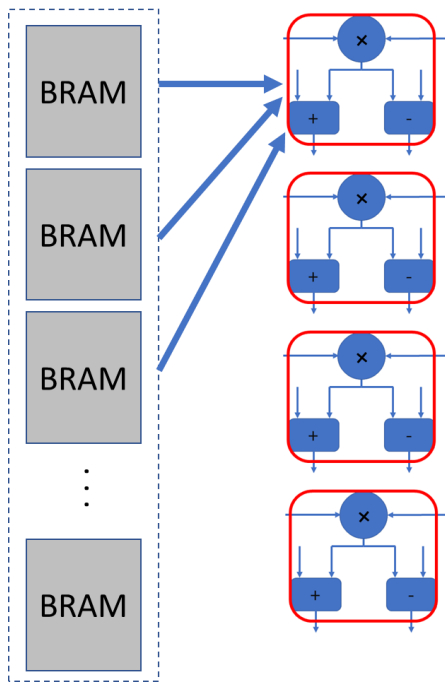


Parallel NTT

Challenge 3: Data routing

Core requires data from distant BRAM memory

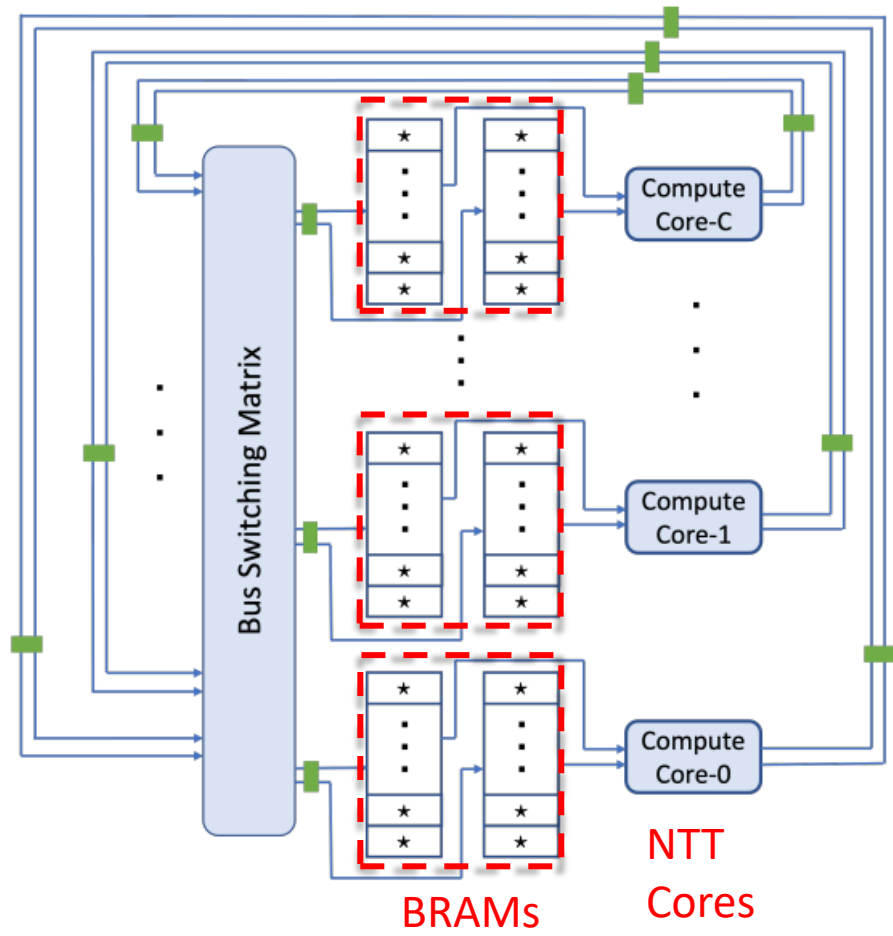
- Long routing of data wires \rightarrow slow clock frequency



Problem:
Core is reading data
from far memory.

Solution: There is no solution
to this problem.
Localizing read or write (not both)
is possible.

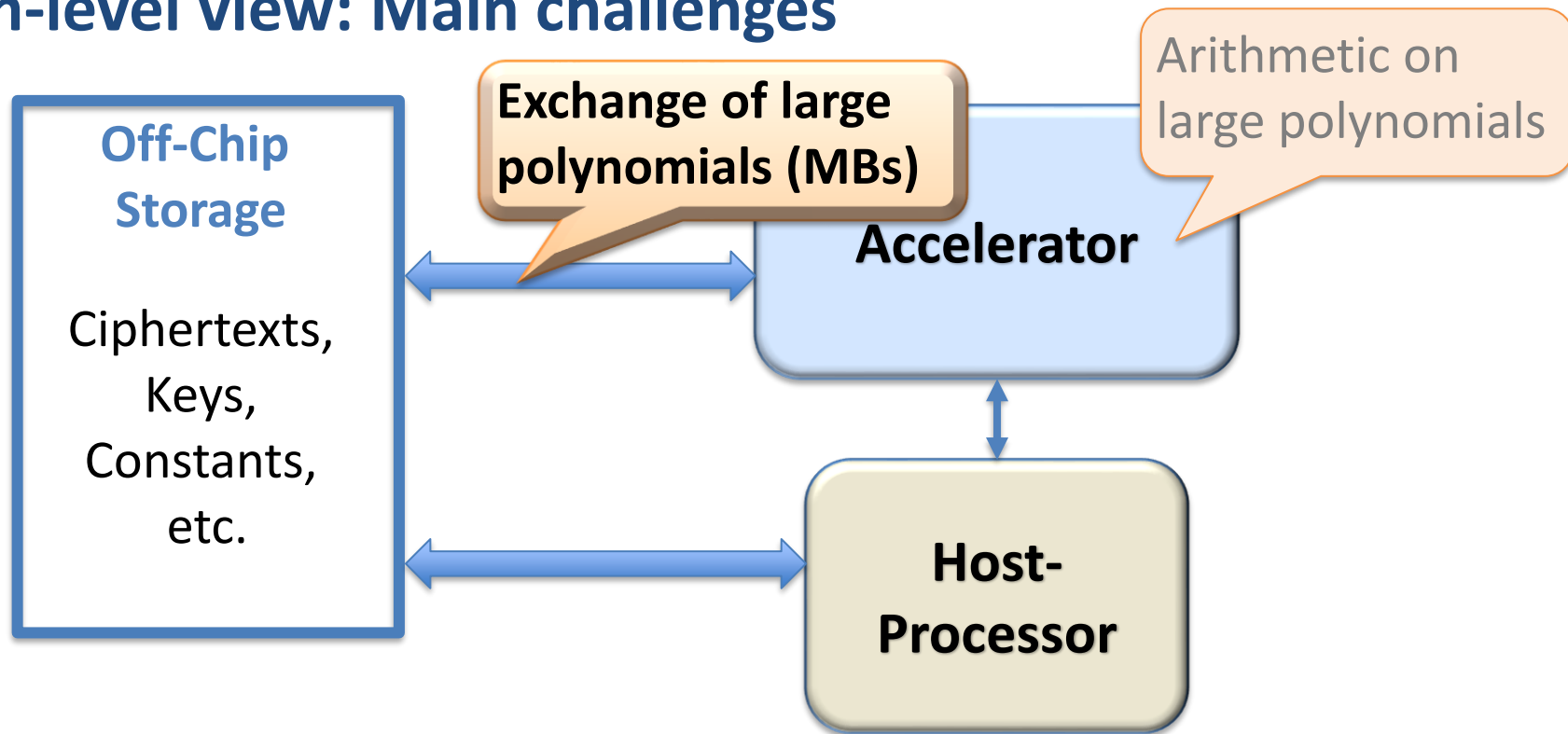
This paper localizes the read operation.
BRAM is exclusively read by only one core.



Data-write paths are heavily pipelined.

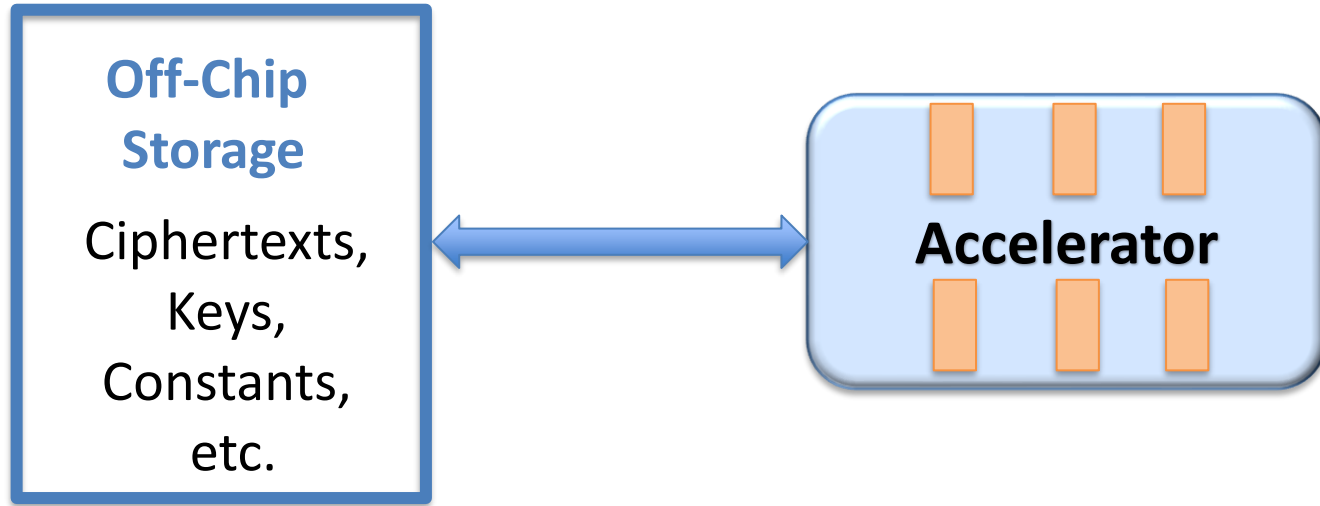
- Pipeline register
- ★ Coefficient of a polynomial

System-level view: Main challenges



Next topic: Memory management

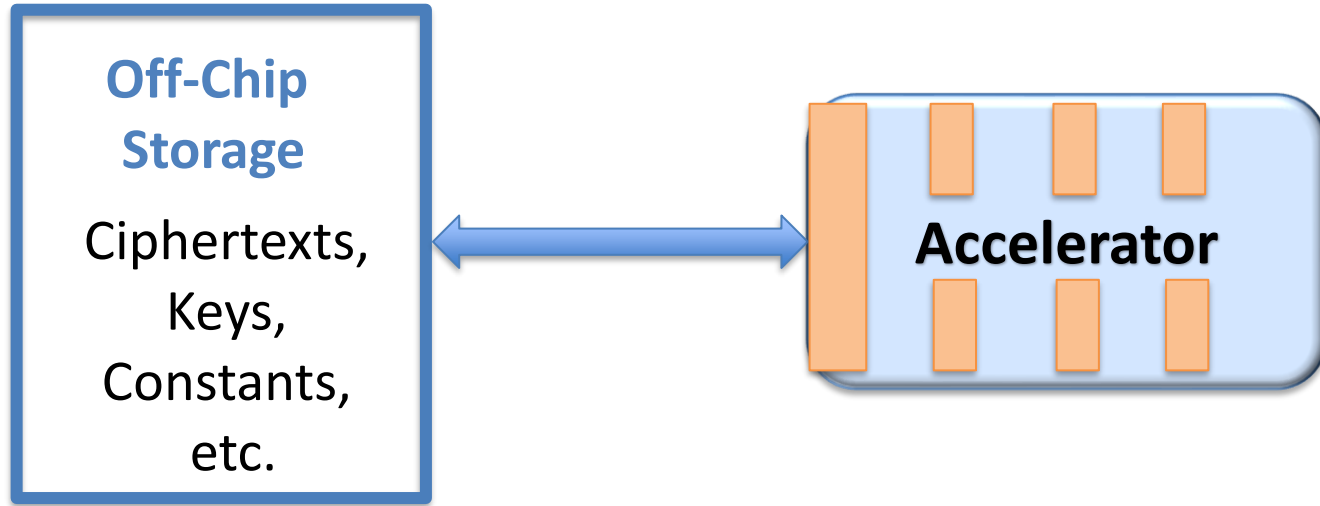
Memory organization and management



Common techniques

- Lots of on-chip memory (BRAM/SRAM) for storing operands

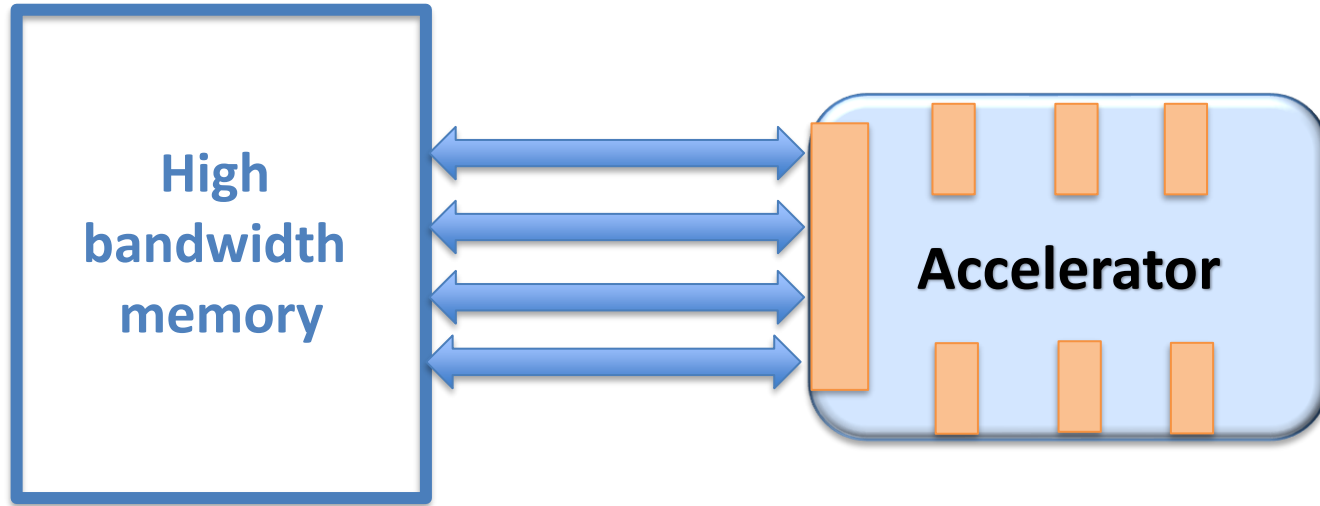
Memory organization and management



Common techniques

- Lots of on-chip memory (BRAM/SRAM) for storing operands
- Perform communication-computation parallelism using cache

Memory organization and management



Common techniques

- Lots of on-chip memory (BRAM/SRAM) for storing operands
- Perform communication-computation parallelism using cache
- High-bandwidth off-chip memory and with multiple channels

Tutorial outline

1. FHE concepts
2. Parallel processing opportunities in FHE (from high-level)
3. Hardware architecture design challenges and methods
❖ **Implementation**
4. Results

Implementations

There are two main tracks

1. True accelerator prototype in ASIC/FPGA
2. Simulation-based modelling of accelerator

Real HW prototypes:
HEAWS^[1], HEAX^[2], CoFHEE^[3], Medha^[4]

Simulation-based works:
F1^[5], BTS^[6], CraterLake^[7], ...

[1] Furkan Turan et al. HEAWS: an accelerator for homomorphic encryption on the amazon AWS FPGA. IEEE ToC, 2020.

[2] M. Sadegh Riazi et al. HEAX: an architecture for computing on encrypted data. ASPLOS 2020.

[3] Mohammed Nabeel et al. CoFHEE: A Co-processor for Fully Homomorphic Encryption Execution. DATE 2023.

[4] Mert et al. Medha: Microcoded Hardware Accelerator for computing on Encrypted Data. CHES 2023.

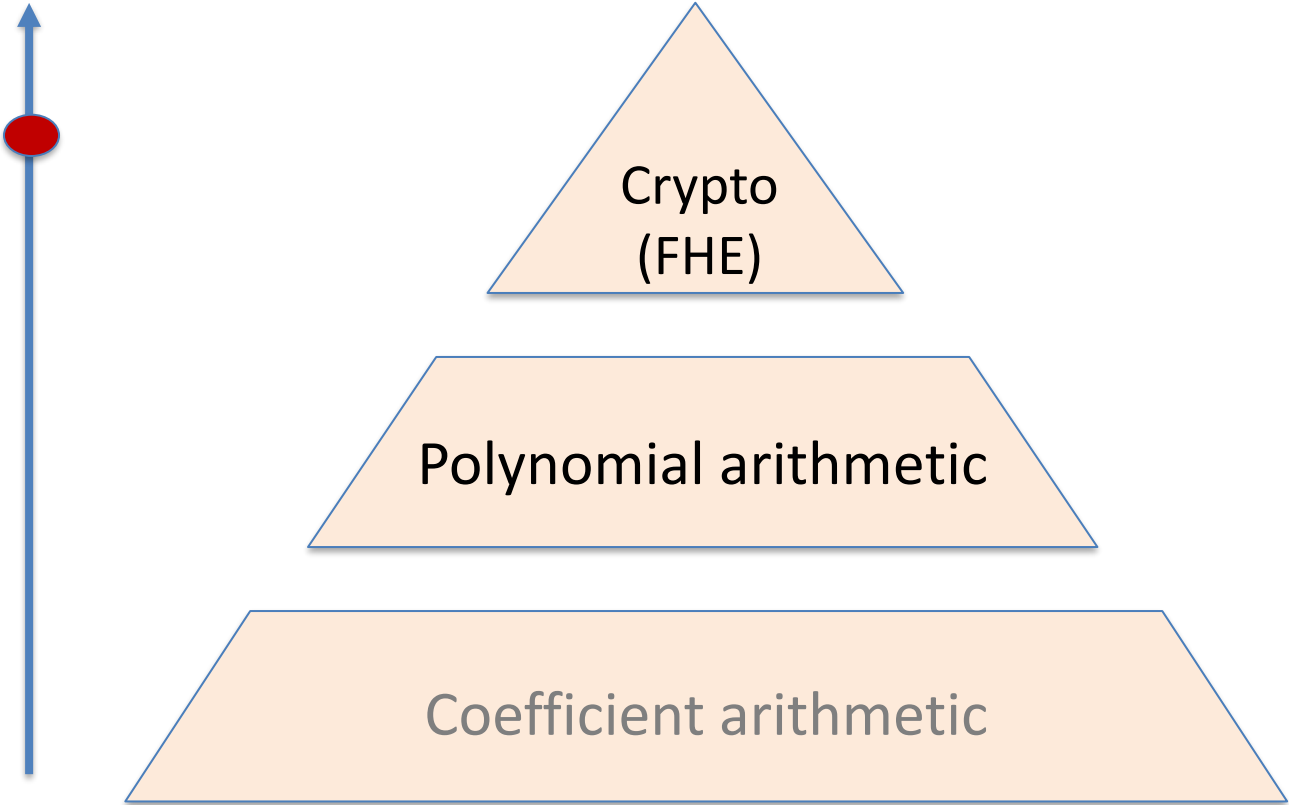
[5] Axel Feldmann et al. F1: A fast and programmable accelerator for fully homomorphic encryption. MICRO 2021.

[6] Sangpyo Kim et al. BTS: An Accelerator for Bootstrappable Fully Homomorphic Encryption. ISCA 2022.

[7] Samardzic et al. CraterLake: A Hardware Accelerator for Efficient Unbounded Computation on Encrypted Data. ISCA 2022.

Briefly talk about

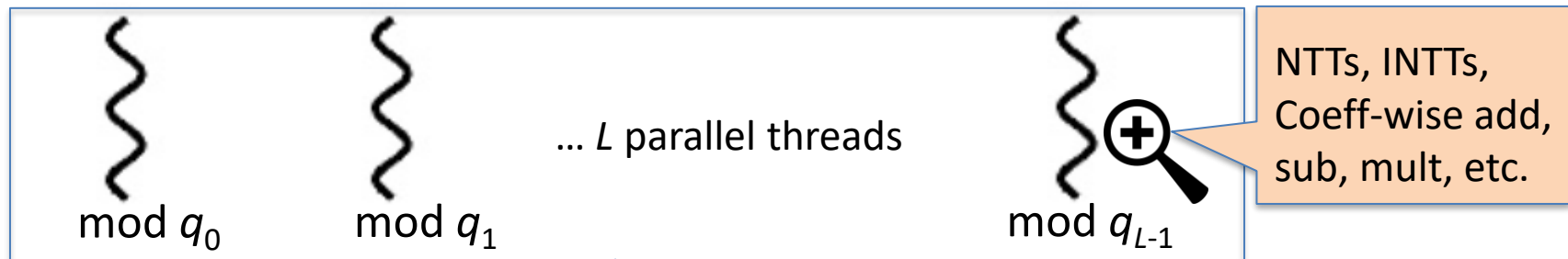
Next, FHE accelerator



High level computation flow

Ciphertexts are polynomials in $R_Q = \mathbb{Z}_Q / \langle X^n + 1 \rangle$
E.g., $\log(Q) = 500$, $n = 2^{15}$

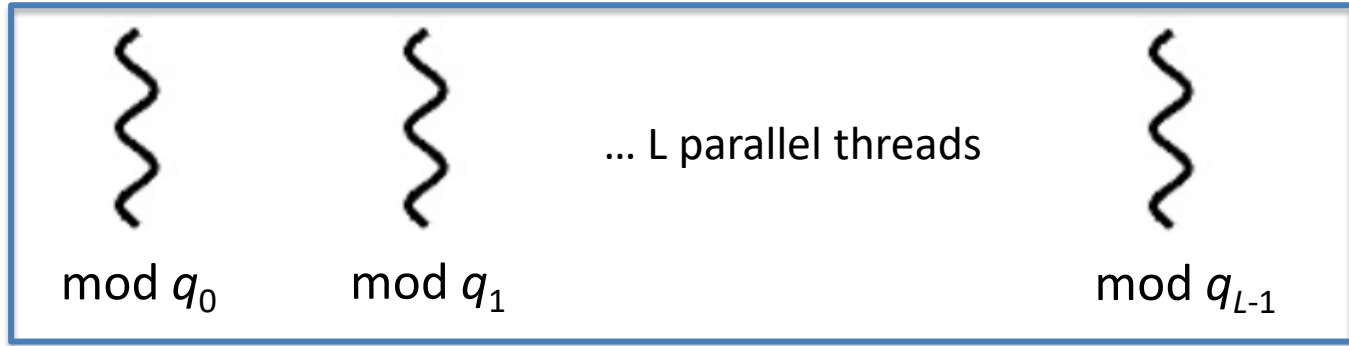
Let $Q = \prod q_i$ where q_i are NTT primes.
Apply Residue Number System (RNS)



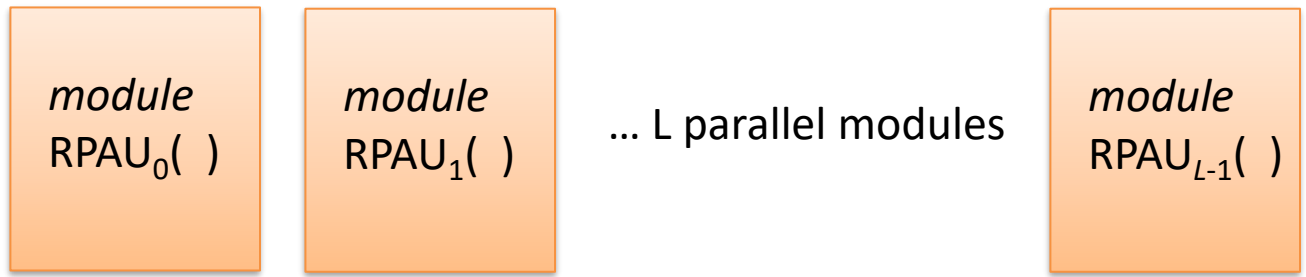
Residue polynomial arithmetic in parallel

Chinese Remainder Theorem (CRT) to obtain R_Q
(Used during modulus switching steps)

... Residue polynomial arithmetic layer



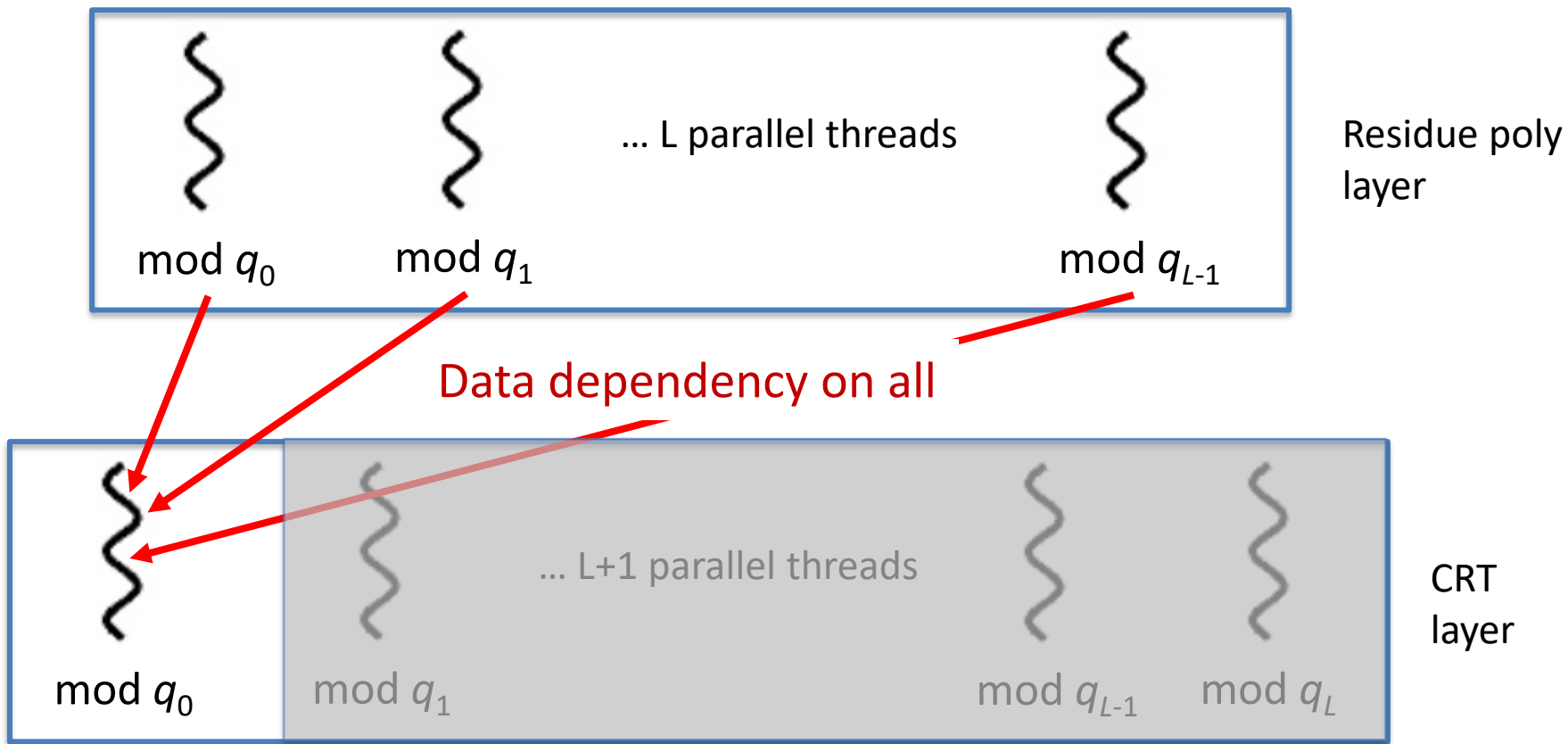
Data flow diagram



Arch. block diagram

*RPAU stands for Residue Polynomial Arithmetic Unit

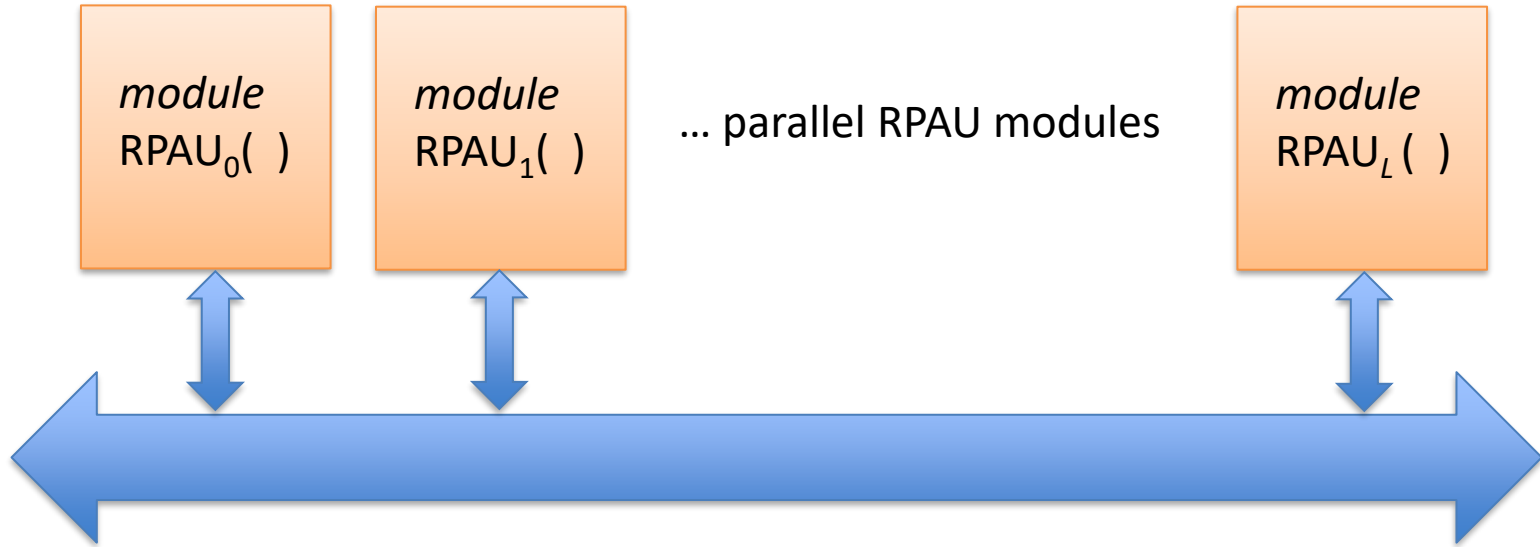
... Residue polynomial layer ↔ CRT layer



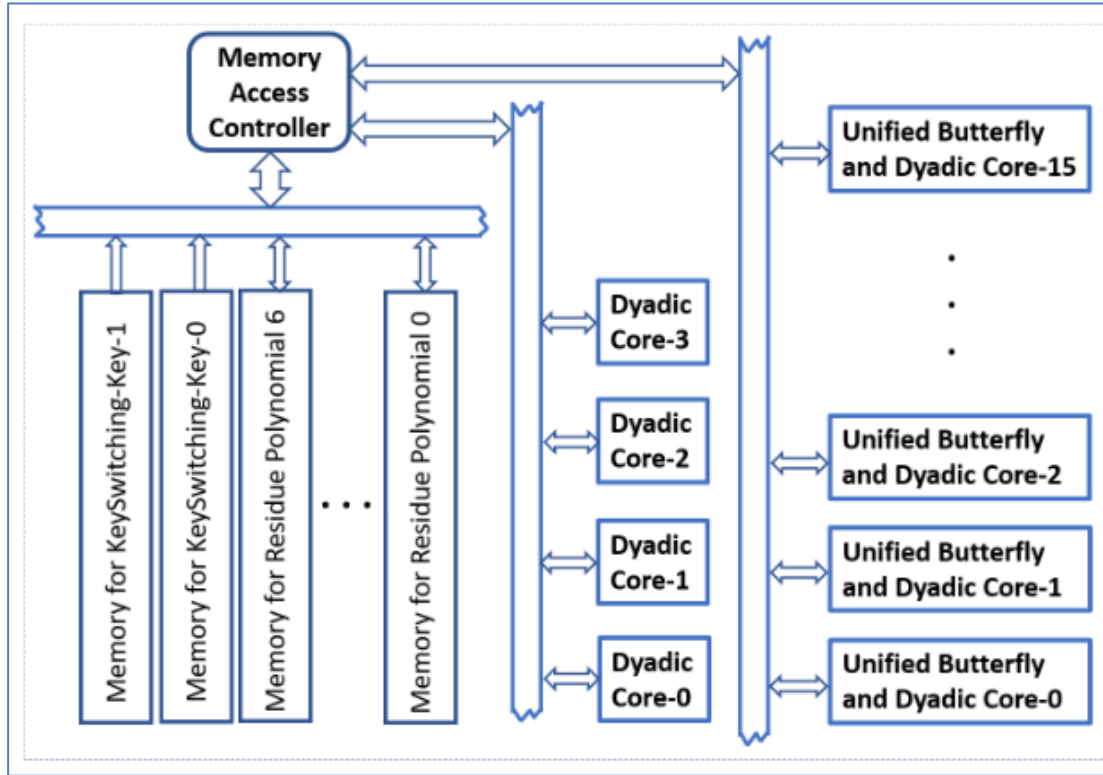
Each thread in CRT layer combines ***all threads*** from previous layer.

... Residue polynomial layer ↔ CRT layer

Therefore, RPAUs need to exchange data with each other.



RPAU ()



Example RPAU. It uses 16 NTT butterfly cores and 4 coefficient-wise (dyadic) arithmetic cores. Polynomials are stored in 'Memory' made of BRAMs.

Instruction Parallelism in RPAU ()

Parallel execution of instructions

HE. Mult

$$\begin{aligned} \bar{d}_{0,j} &\leftarrow \bar{c}_{0,j} \star \bar{c}'_{0,j} \\ \bar{d}_{1,j} &\leftarrow \bar{c}_{0,j} \star \bar{c}'_{1,j} + \bar{c}_{1,j} \star \bar{c}'_{0,j} \\ \bar{d}_{2,j} &\leftarrow \bar{c}_{1,j} \star \bar{c}'_{1,j} \end{aligned}$$

HE. Relin

$$\{c''_{0,j}, c''_{1,j}\} \leftarrow 0$$

$$d_{2,j} \leftarrow \text{INTT}(\bar{d}_{2,j})$$

for $i = 0$ to $L - 1$ do

Obtain $d_{2,i}$ from RPAU _{i}

$$r_{2,i} \leftarrow \text{Coeff. Reduce}(d_{2,i}, q_j)$$

$$\tilde{t} \leftarrow \text{NTT}(r_{2,i})$$

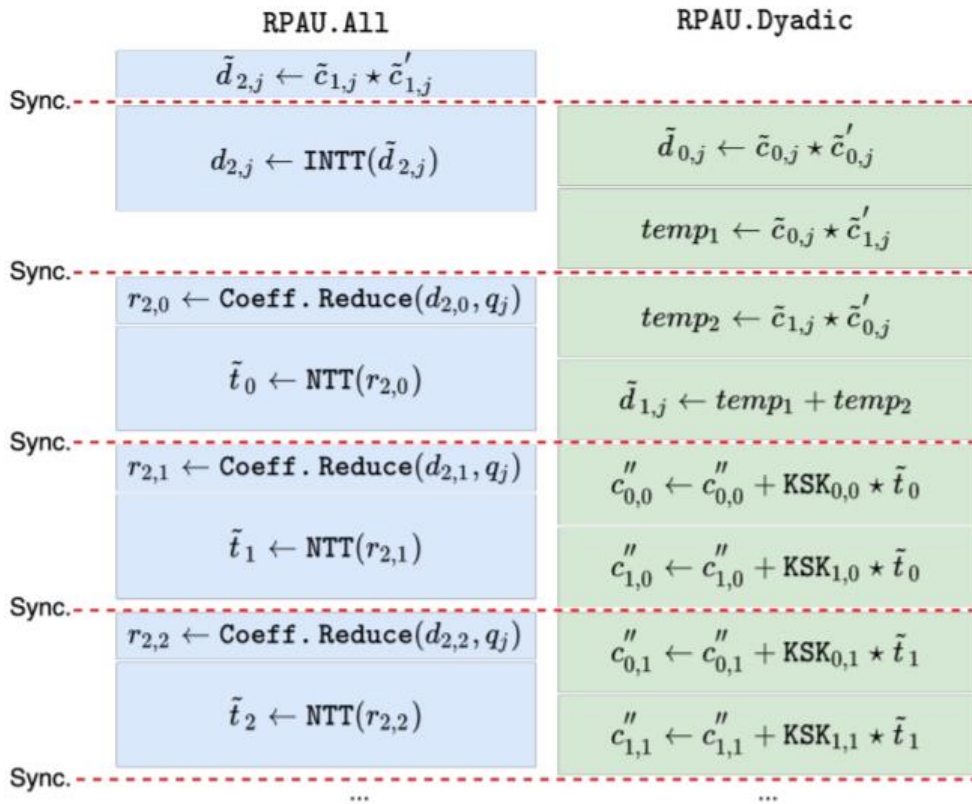
$$c''_{0,i} \leftarrow c''_{0,i} + \text{KSK}_{0,i} \star \tilde{t}$$

$$c''_{1,i} \leftarrow c''_{1,i} + \text{KSK}_{1,i} \star \tilde{t}$$

end for

$$(d_{0,j}, d_{1,j}) \leftarrow [c'' \cdot p^{-1}]$$

Homomorphic multiplication & key-switching.
(The most expensive operation)



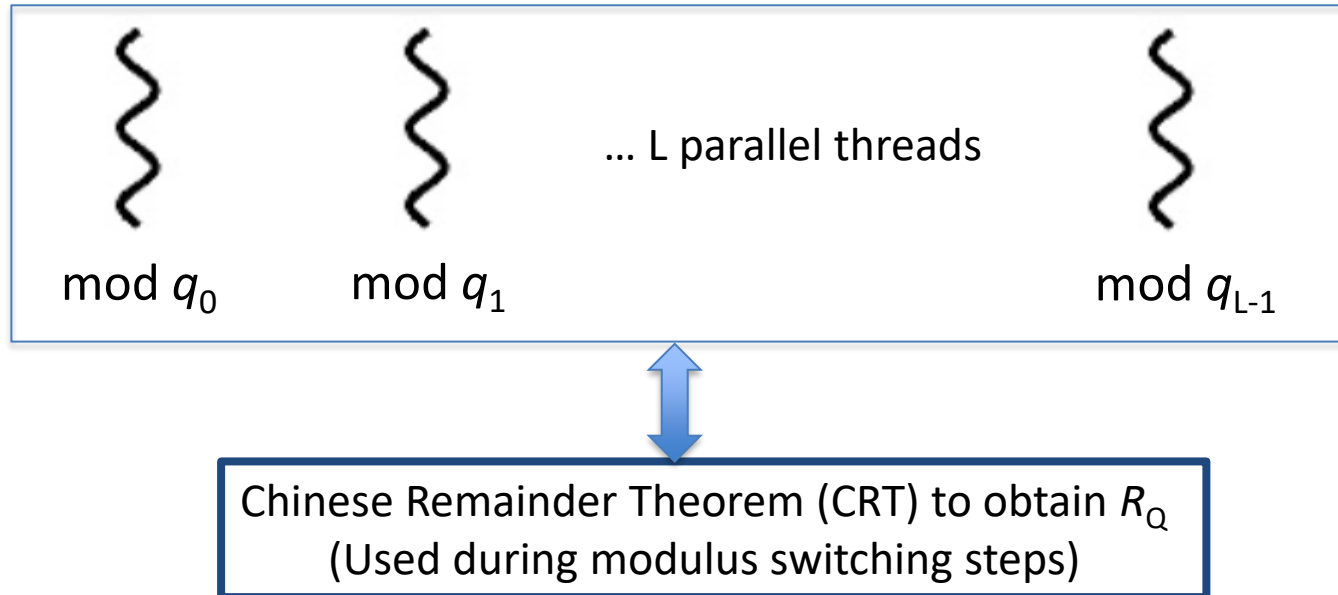
This reduces 40% cycle count

Placement of RPAUs

CRT requires combining the residues.

→ Therefore, RPAUs need to communicate with each other

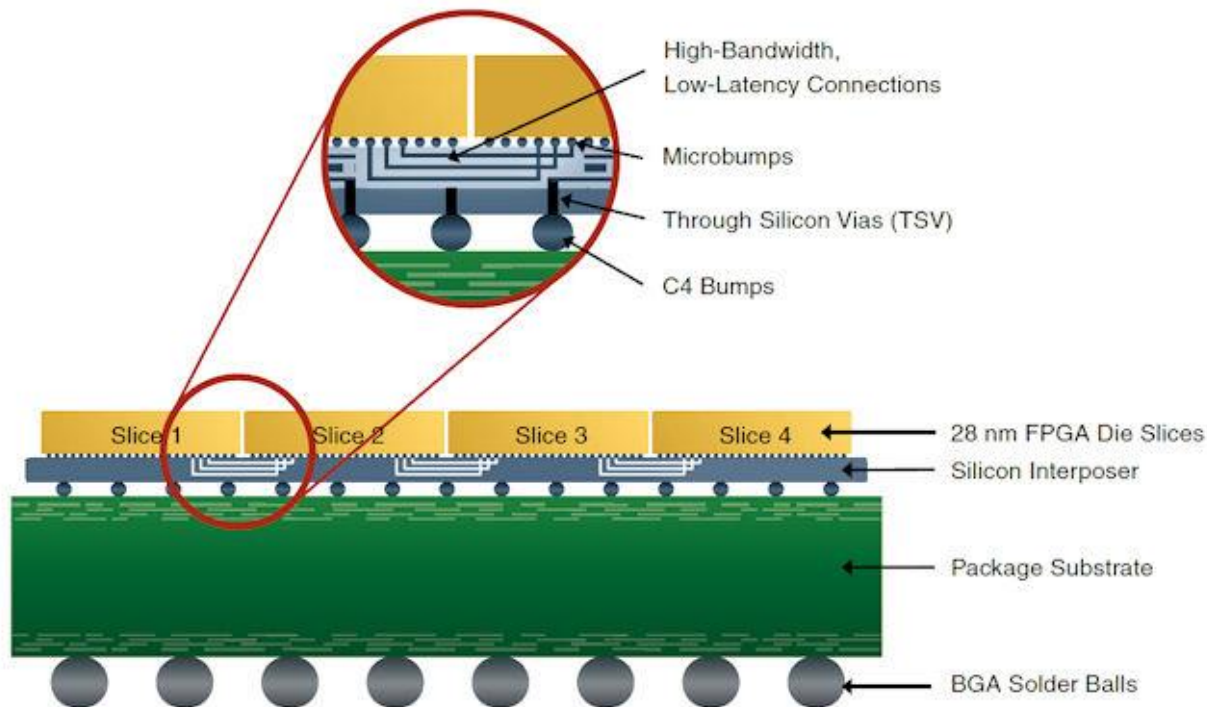
How to interconnect the RPAUs in large 3D FPGAs?



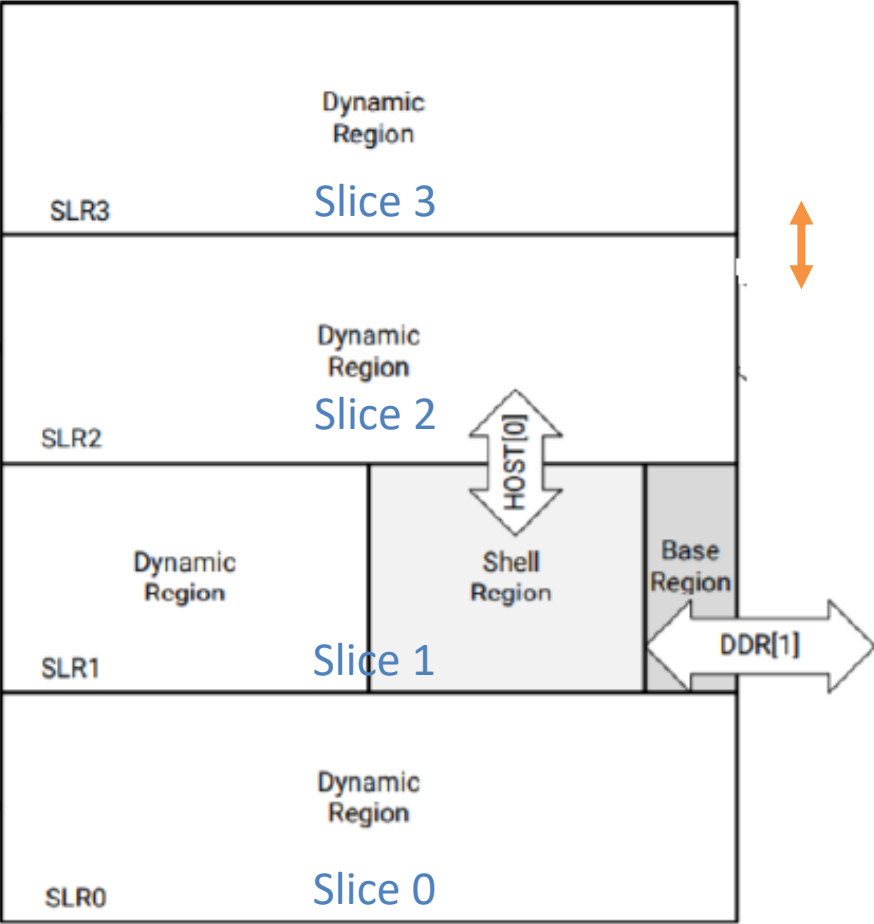
Large SLR FPGA

Large FPGAs are multi-die

- The FPGA is split into four SLRs.
- Connected by a limited number of wires.



Large SLR FPGA – top view



There are a limited number of interconnects.

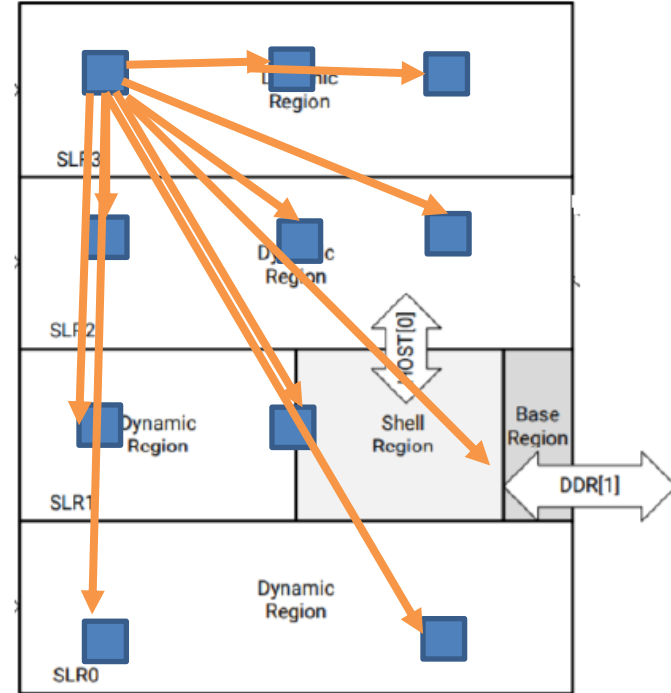
Large design cannot be spread arbitrarily across SLRs.


Xilinx Alveo U250 FPGA. This FPGA is 1000x larger than the FPGA used in this course.

Placement-friendly interconnection of RPAUs

- FPGA Constraints
 - The FPGA is split into four SLRs.
 - Connected by a limited number of wires.
- Some operations require exchanging the residue polynomials between RPAUs
- Naïve solution: A "star-like" network

Each RPAU has its own connections

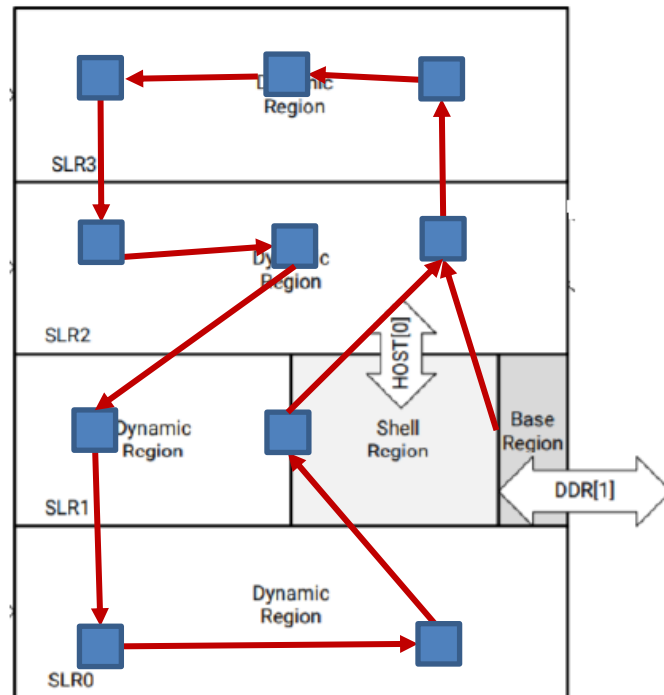


 One RPAU

Placement-friendly interconnection of RPAUs

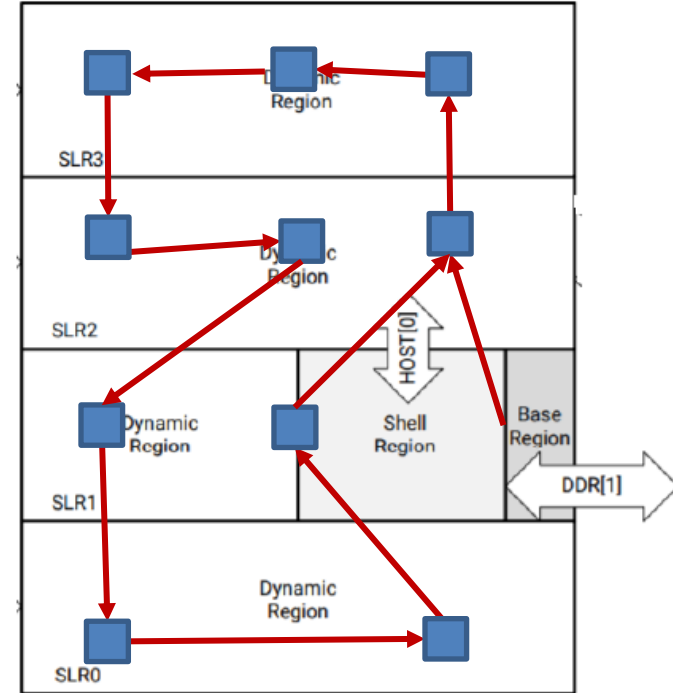
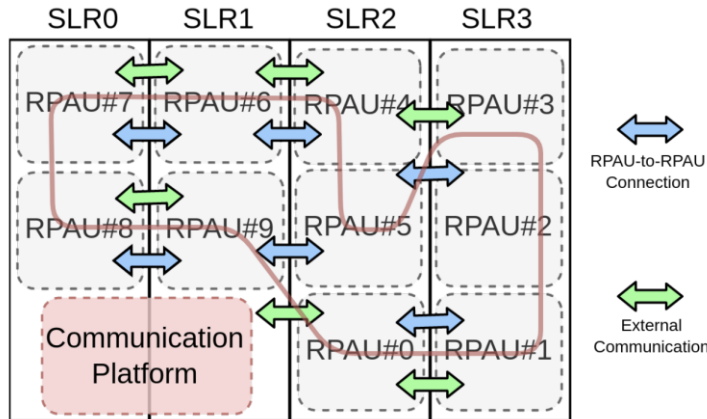
- FPGA Constraints
 - The FPGA is split into four SLRs.
 - Connected by a limited number of wires.
- Some operations require exchanging the residue polynomials between RPAUs
- **Solution:** A "ring" interconnection of RPAUs

- Only two neighbour RPAUs are connected.
- Data sent to an RPAU through a chain of RPAUs.
- No additional computation overhead

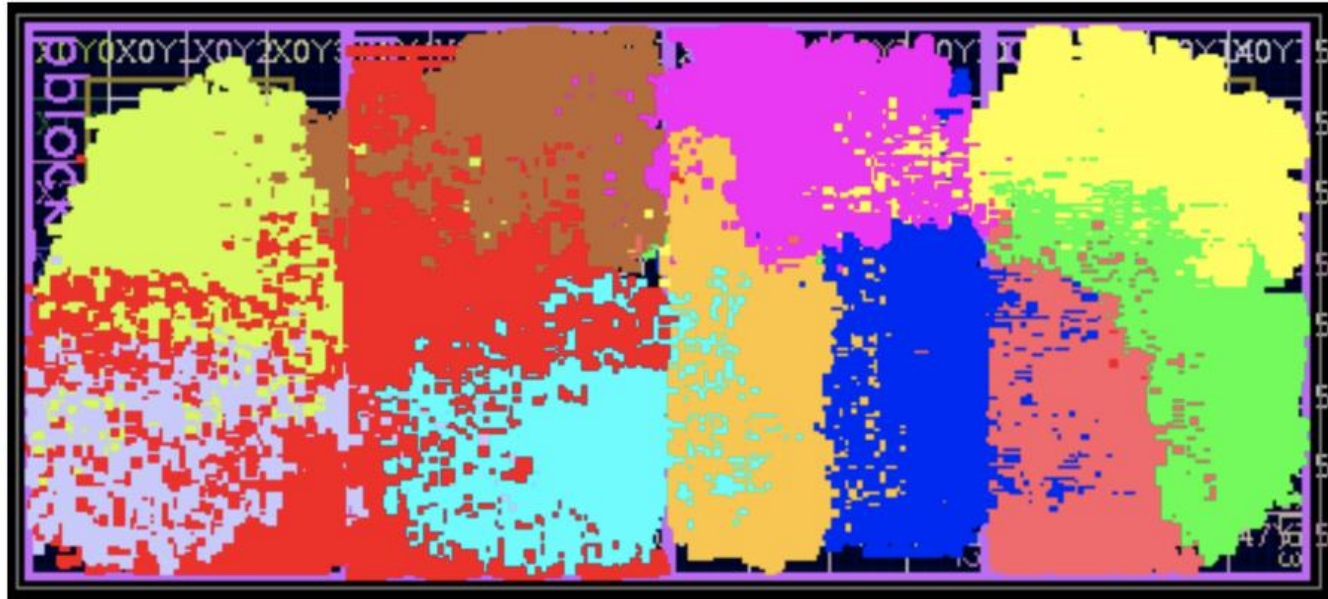


Placement-friendly interconnection of RPAUs

- FPGA Constraints
 - The FPGA is split into four SLRs.
 - Connected by a limited number of wires.
- Some operations require exchanging the residue polynomials between RPAUs
- Placement of 10 RPAUs using “ring” interconnect



Floorplan of the design



- RPAU0
- RPAU1
- RPAU2
- RPAU3
- RPAU4
- RPAU5
- RPAU6
- RPAU7
- RPAU8
- RPAUp
- platform_i

Full system overview

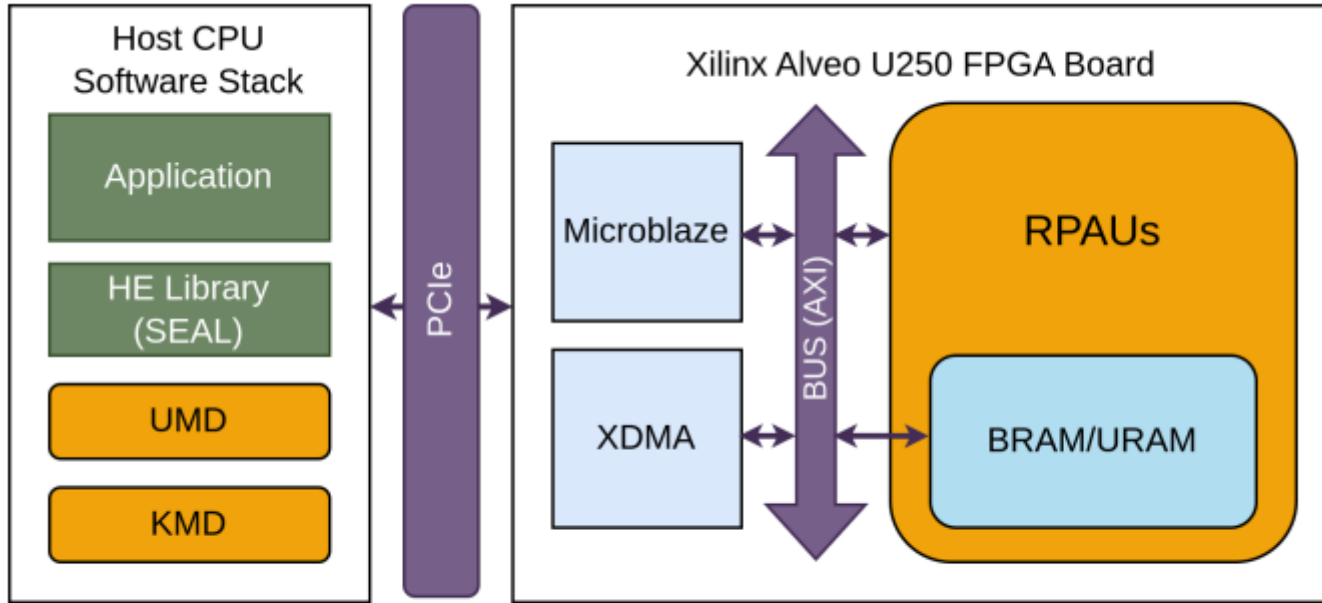


Figure 8: CPU-FPGA interface and software stack

FPGA is used as an accelerator card of a server. HW/SW codesign is used to run applications.

FPGA Acceleration results



Our Group's research: Open Problems in FHE

1. How to make hardware accelerators for larger parameter sets?
2. How to support different parameters?
3. How to support different FHE schemes?
4. How to implement FHE Bootstrapping?
5. From FPGA to ASIC accelerators
 - More parallel processing
 - Custom memory
 - Higher clock frequency and lower power consumption