FULLY HOMOMORPHIC ENCRYPTION WITH POLYLOG OVERHEAD

Craig Gentry and Shai Halevi
IBM Watson

Nigel Smart
Univ. Of Bristol

Homomorphic Encryption

- Usual procedures (KeyGen,Enc,Dec)
  - Say, encrypting bits
- Usual semantic-security requirement
  - $(pk, Enc_{pk}(0)) \sim (pk, Enc_{pk}(1))$
- Additional Eval procedure
  - Evaluate arithmetic circuits on ciphertexts
  - Result decrypted to the evaluation of the same circuit on the underlying plaintext bits
  - Ciphertext does not grow with circuit complexity
- This work: asymptotically efficient Eval

Contemporary HE Schemes

- The [Gentry'09] approach
  - Ciphertext is noisy (to get security)
  - Noise grow with homomorphic evaluation
  - Until ciphertext is too noisy to decrypt
- Ciphertext is inherently large
  - Need to leave lots of room for noise to grow
  - It takes $\tilde{\Omega}(\lambda)$-bit ciphertext to encrypt a single bit
    - λ is the security parameter
- Implementing each binary arithmetic gate takes at least $\tilde{\Omega}(\lambda)$ time
  - $\tilde{\Omega}(\lambda)$ time just to read the input ciphertexts

Our Result

- Homomorphic evaluation of T-gate binary arithmetic circuits of average width $\tilde{\Omega}(\lambda)$ in time **T·polylog(λ)**
- More Generally, a T-gate, W-average-width circuit can be evaluated homomorphically in time $\boldsymbol{\tilde{O}}(\underbrace{\lceil \boldsymbol{W}/\boldsymbol{\lambda} \rceil \cdot \boldsymbol{\lambda}}_{\text{time per level}} \cdot \underbrace{\boldsymbol{T}/\boldsymbol{W}}_{\text{\# of levels}})$

Our Approach

- Use HE over polynomial rings
- Pack an array of bits in each ciphertext
- Use ring-automorphisms to move bits around in the arrays
- Efficient data-movement schemes
  - Using Beneš/Waksman networks and extensions

BACKGROUND

- Homomorphic Encryption over Polynomials Rings
- Polynomial-CRT representation, plaintext slots
- Homomorphic SIMD operations

Hom.Enc. Over Polynomial Rings

- Used, e.g., in [BGV'12], [LTV'12], [B'12]
- Native plaintext space is $R_2 = Z_2[X]/\Phi_m(X)$
  - Binary polynomials modulo $\Phi_m(X)$ (m odd)
  - $\Phi_m(X)$ is m'th cyclotomic polynomial, deg=$\phi(m)$
- $\Phi_m(X)$ irreducible over Z, but not mod 2
  - $\Phi_m(X) = \prod_{j=1}^{\ell} F_j(X)$ (mod 2)
  - F_j's are irreducible, all have the same degree d
    - degree d is the order of 2 in Z_m^*
  - For some m's we can get $\ell = \frac{\phi(m)}{d} = \Omega(\frac{\mathrm{m}}{\log\ \mathrm{m}})$

Plaintext Slots

- Plaintext element $a \in R_2$ encodes ℓ values
  - $a \cong [\alpha_1, \dots, \alpha_\ell],\ \alpha_j = (a \, mod \, F_j)$
  - Polynomial Chinese Remainders
- Can use a's for which each α_j is a bit
- Ops +,× work independently on the slots
  - ℓ-ADD: $a + a' \cong [\alpha_1 + \alpha'_1, \dots, \alpha_\ell + \alpha'_\ell]$
  - ℓ-MUL: $a \times a' \cong [\alpha_1 \times \alpha'_1, \dots, \alpha_\ell \times \alpha'_\ell]$

Homomorphic SIMD [SV'11]

SIMD = **S**ingle **I**nstruction **M**ultiple **D**ata

- Computing the same function on ℓ inputs at the price of one computation
- Pack the inputs into the slots
  - Bit-slice, inputs to *j*'th instance go in *j*'th slots
- Compute the function once
- After decryption, decode the ℓ output bits from the output plaintext polynomial

Aside: an ℓ-SELECT Operation

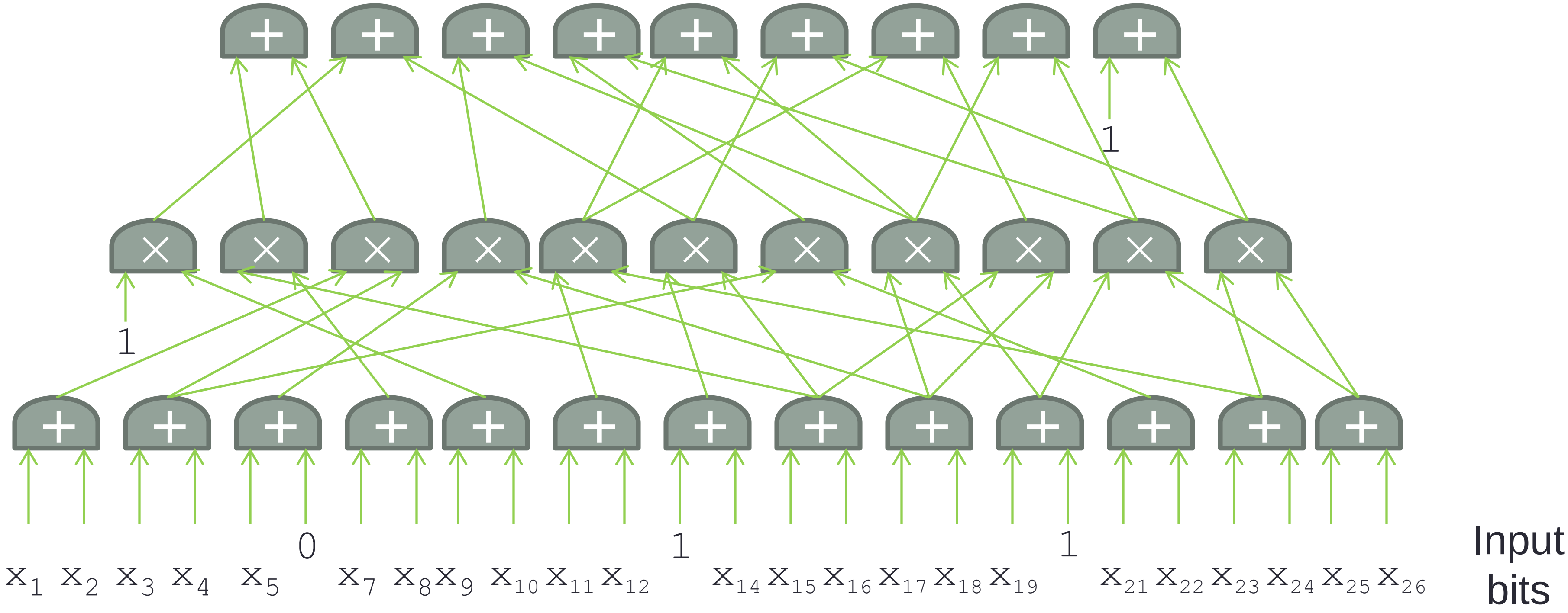
	x_1	x_2	x_3	x_4	x_5	x_6	x_7	=	x_1	0	0	x_4	0	x_6	0
x	1	0	0	1	0	1	0								
												+			
	x_8	x_9	x_{10}	x_{11}	x_{12}	x_{13}	x_{14}	=	0	x_9	x_{10}	0	x_{12}	0	x_{14}
x	0	1	1	0	1	0	1								
									x_1	x_9	x_{10}	x_4	x_{12}	x_6	x_{14}

- We will use this later

COMPUTING ON DATA ARRAYS

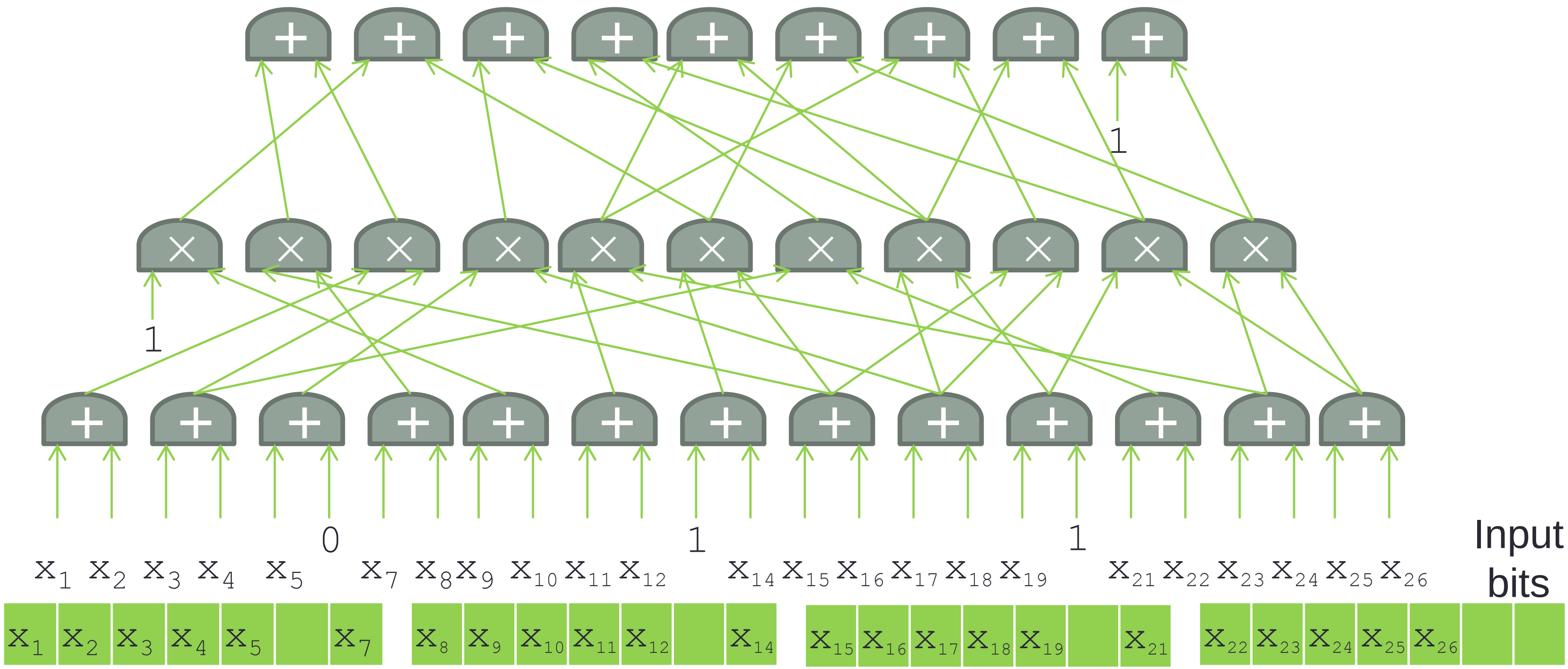
Forget about encryption for the moment…

So you want to compute some function…



ADD and MUL are a *complete* set of operations.

So you want to compute some function <u>using SIMD</u>…

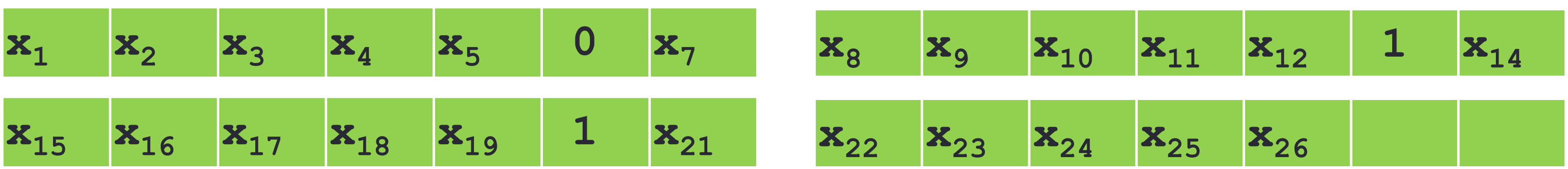


ℓ-ADD and ℓ-MUL are not a complete set of operations!!!

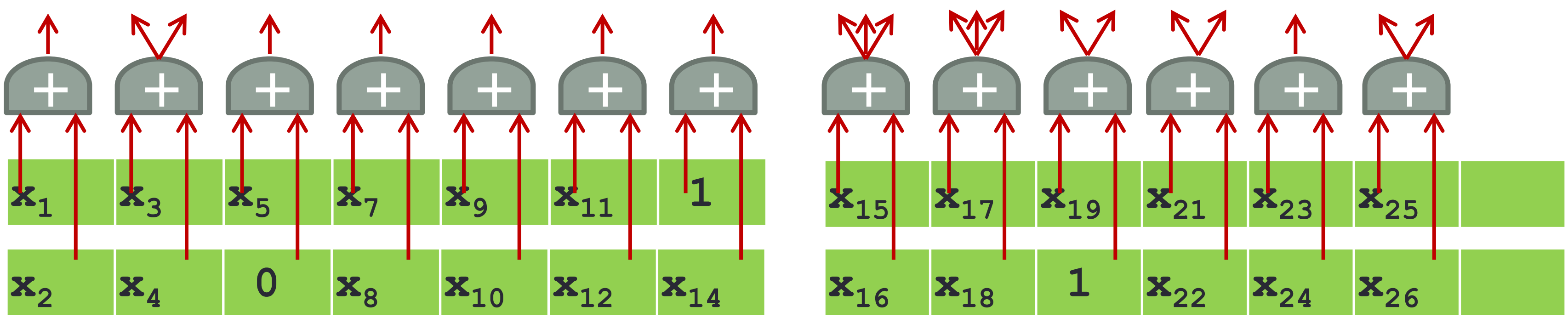
… unless, of course, we use ℓ=1… ☹

Routing Values Between Levels

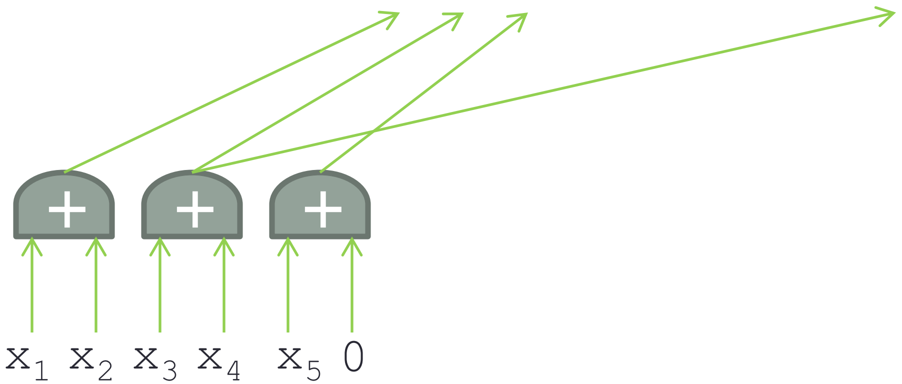
- We need to map this

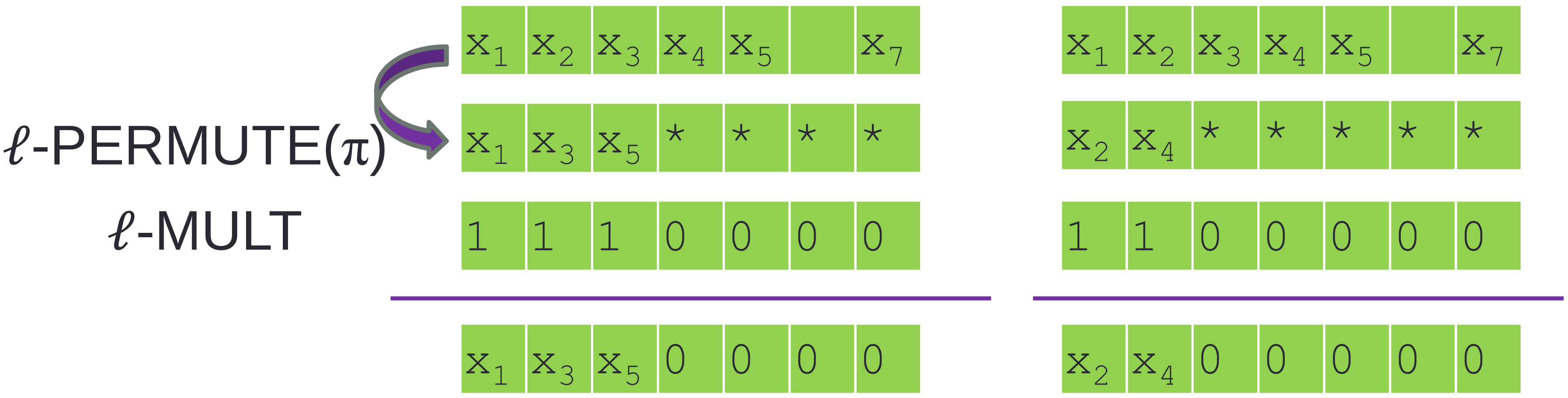


- Into that … so we can use ℓ-add

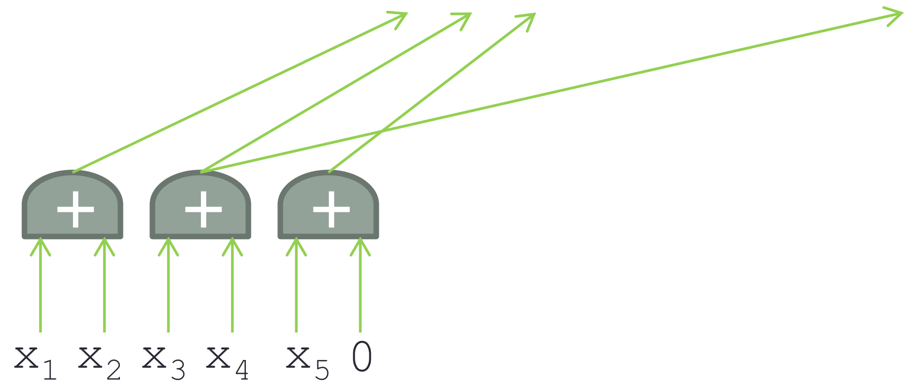


ℓ-ADD, ℓ-MUL, ℓ-PERMUTE: a complete set of SIMD ops

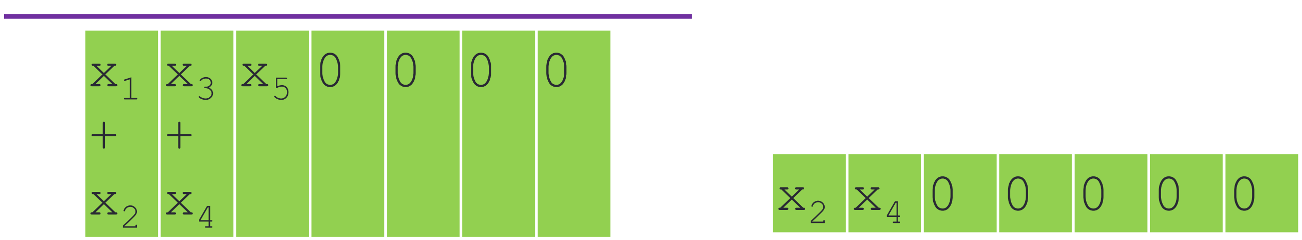




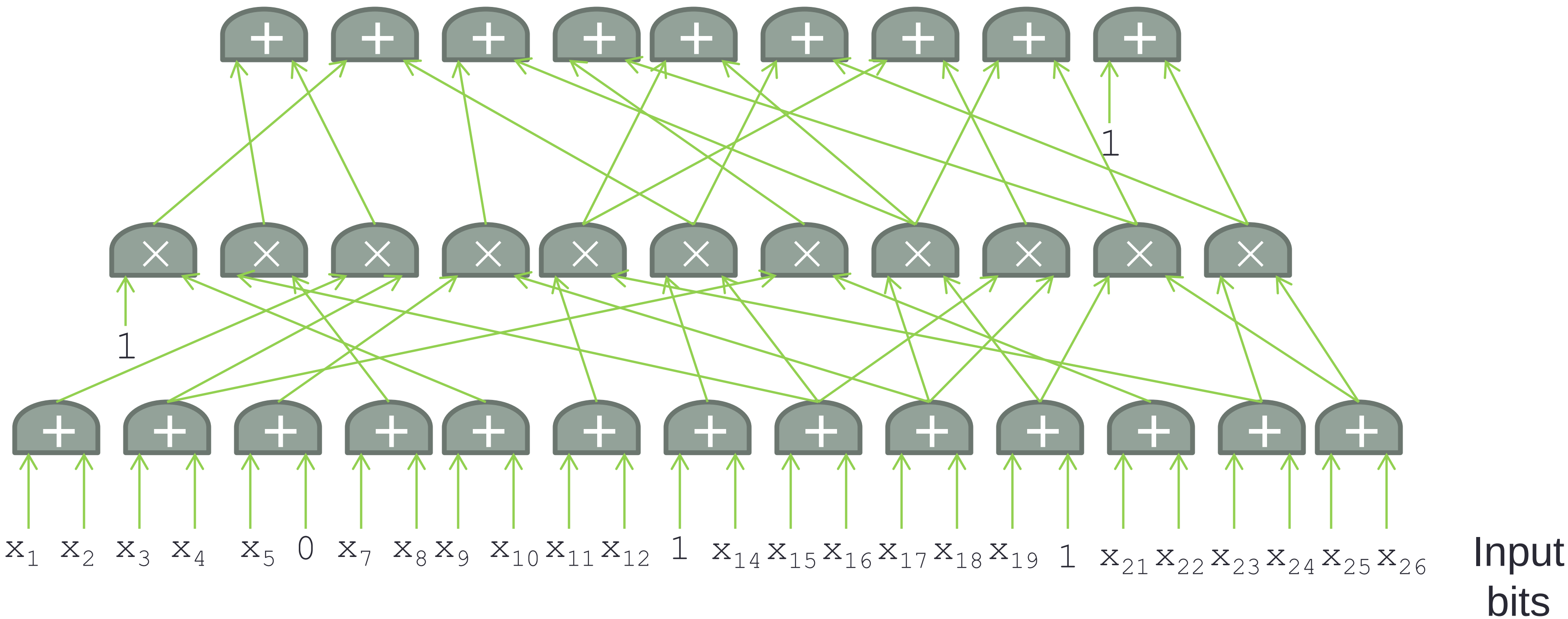
ℓ-ADD, ℓ-MUL, ℓ-PERMUTE: a complete set of SIMD ops



ℓ-ADD



ℓ-ADD, ℓ-MUL, ℓ-PERMUTE: a complete set of SIMD ops



Use ℓ-PERMUTE for routing between circuit levels

- Not quite obvious

Routing Values Between Levels: Three Problems to Solve

1. How to implement ℓ-permute?
   - $a \in R_2$ encodes ℓ-array using polynomial-CRT
   - We are given an encryption of a
2. Fan-out: need to **clone** values from high fan-out gates before routing to next level
3. **Big permutation**: For a width-W level, we need a permutation over 2W values
   - Implemented using ℓ-permute on ℓ-arrays
   - Even when $W \gg \ell$

Implementing ℓ-Permute

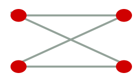
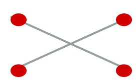
- Recall: native plaintext is binary polynomial modulo $\Phi_m(X)$, $a \in R_2 = Z_2[X]/\Phi_m(X)$
  - $a \cong [\alpha_1, \dots, \alpha_\ell]$, $\alpha_j = (a \bmod F_j)$
  - $a + a' \cong [\alpha_1 + \alpha'_1, \dots, \alpha_\ell + \alpha'_\ell]$
  - $a \times a' \cong [\alpha_1 \times \alpha'_1, \dots, \alpha_\ell \times \alpha'_\ell]$
- Is there a natural operation on polynomials that moves values between slots?

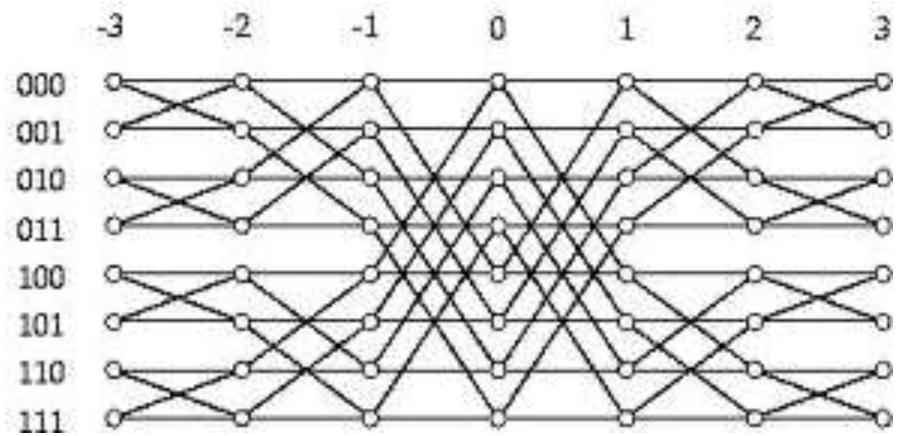
Moving Values Between Slots

- [BGV12] use automorphisms $a(X) \rightarrow a(X^j)$
  - Similar technique in [LPR'10]
- Very roughly, yields cyclic shifts
  - E.g., if $a(\mathrm{X}) \cong [\alpha_1, \alpha_2, \ldots, \alpha_\ell]$
    then $a(\mathrm{X}^5) \cong [\alpha_\ell, \alpha_1, \ldots, \alpha_{\ell-1}]$
  - Can be used to shift by any amount
- Can be implemented homomorphically
- This gives us shifts
  - But we want arbitrary permutations, efficiently

From Shifts to Arbitrary Permutations

Use Beneš/Walksman Permutation Networks:

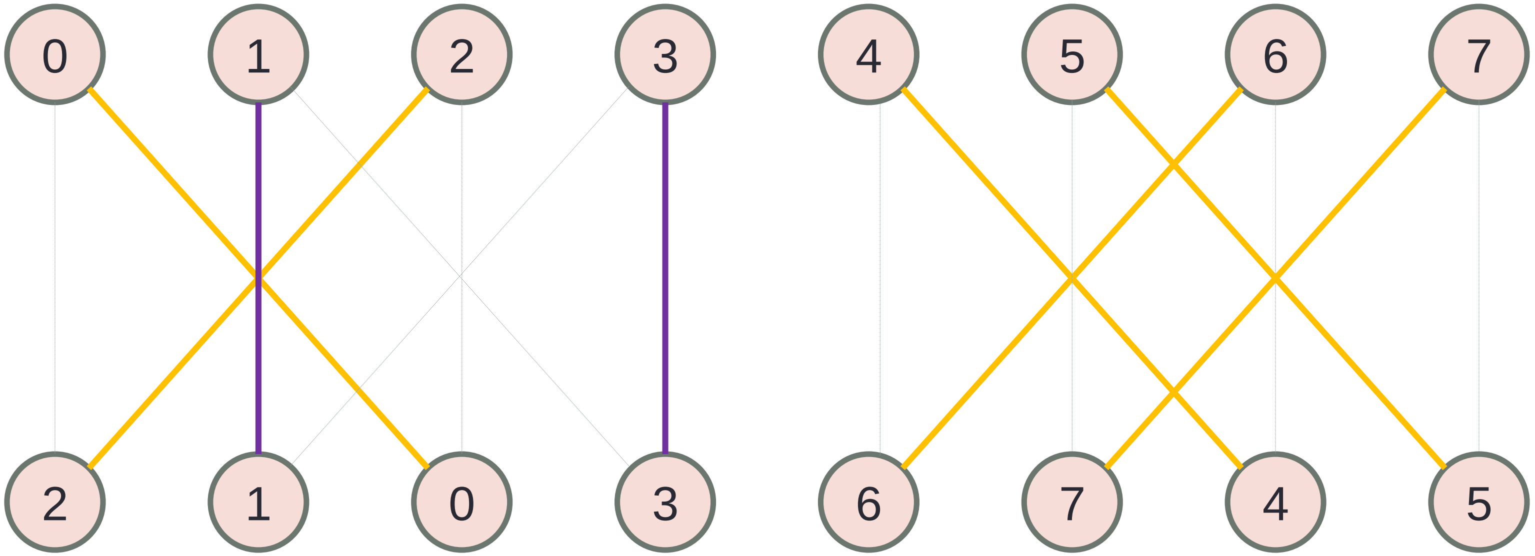
- Two back-to-back butterflies
  - Every exchange  is controlled by a bit
  - Values sent on either straight edges  or cross edges 



- Every permutation can be realized by appropriate setting of the control bits

Realizing Permutation Networks

- <u>Claim:</u> every butterfly level can be realized by two shifts and two SELECTs
- Example:



Control bits: 1 0 1 1

Realizing Permutation Networks

a_1	0	1	2	3	4	5	6	7	input
a_2	0	1	2	3	4	5	6	7	shift(-2)
a_3	0	1	2	3	4	5	6	7	shift(2)

2	1	0	3	6	7	4	5	output

Realizing Permutation Networks

a_1	0	**1**	2	**3**	4	5	6	7	input
a_2	**2**	3	4	5	**6**	**7**	0	1	shift(-2)
a_3	6	7	**0**	1	2	3	**4**	**5**	shift(2)
a_4	**2**	**1**	4	**3**	**6**	**7**	0	1	SELECT(a_1,a_2)
a_5	**2**	**1**	**0**	**3**	**6**	**7**	**4**	**5**	SELECT(a_3,a_4)
	2	1	0	3	6	7	4	5	output

Realizing Permutation Networks

Claim: every level of the Benes network can be realized by two shifts and two SELECTs

Proof : In every level, all the exchanges are between nodes at the same distance

- Distance 2^i for some i

Can implement all these exchanges using shift(2^i), shift(-2^i), and two SELECTs ∎

Realizing Permutation Networks

- Every level takes 2 shifts and 2 SELECTs
- There are 2log(ℓ) levels

$\Rightarrow$ Any permutation on ℓ-arrays can be realized using 4log(ℓ) shifts and 4log(ℓ) SELECTs

- Some more complications when ℓ is not a power of two
  - But still only O(log ℓ) operations

Routing Values Between Levels

✓Implementing ℓ-permute

- Using $X \mapsto X^j$ to get simple shifts
- Benes network to get arbitrary permutation
- Takes O(log ℓ) operations

- Cloning values from high fan-out gates
- Permutations over $W \gg \ell$ elements

} not today

- Both can be done in O(log W) operations

➔ **Intra-level routing takes** $O(\frac{W}{\ell}\log(W))$ **ops**

- For a width-W level

Low Overhead Homomorphic Encryption

- Pack inputs into ℓ-arrays
  - ℓ can made as large as $\widetilde{\Omega}(\lambda)$
- SIMD operations to implement each level
- Route values to their place for next level
- Each level takes $\widetilde{\boldsymbol{O}}(\lceil \boldsymbol{W}/\boldsymbol{\lambda} \rceil \cdot \boldsymbol{\lambda})$ work
- Total work for size-T width-W circuit is $\widetilde{\boldsymbol{O}}(\lceil \boldsymbol{W}/\boldsymbol{\lambda} \rceil \cdot \boldsymbol{\lambda} \cdot \boldsymbol{T}/\boldsymbol{W})$

QUESTIONS?

Handling Large Permutations

- Can we arbitrarily permute $m \times \ell$ items, given in m arrays of size ℓ, using ℓ-ADD, ℓ-MUL, ℓ-PERMUTE?

- **Theorem** (Lev, Pippenger, Valiant '84): A permutation π over $m \times \ell$ addresses (viewed as a rectangle) can be decomposed as $\pi = \pi_3 \circ \pi_2 \circ \pi_1$, where:
  - π_1 only permutes within the columns
  - π_2 only permutes within the rows
  - π_3 only permutes within the columns

- Within rows: Use ℓ-PERMUTE on each row (array).
- Within columns: swap elements with same index using ℓ-SELECT.

Decomposing Permutations

13	18	14	16	12
15	3	4	5	6
7	8	9	20	19
2	17	1	11	10

?

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20

Decomposing Permutations

13	18	14	16	12
15	3	4	5	6
7	8	9	20	19
2	17	1	11	10

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20

13	17	14	5	6
15	3	4	16	12
7	8	9	11	10
2	18	1	20	19

Decomposing Permutations

13	18	14	16	12
15	3	4	5	6
7	8	9	20	19
2	17	1	11	10

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20

13	17	14	5	6
15	3	4	16	12
7	8	9	11	10
2	18	1	20	19

6	17	13	14	5
16	12	3	4	15
11	7	8	9	10
1	2	18	19	20

Decomposing Permutations

13	18	14	16	12
15	3	4	5	6
7	8	9	20	19
2	17	1	11	10

?

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20

13	17	14	5	6
15	3	4	16	12
7	8	9	11	10
2	18	1	20	19

6	17	13	14	5
16	12	3	4	15
11	7	8	9	10
1	2	18	19	20

Decomposing Permutations

13	18	14	16	12
15	3	4	5	6
7	8	9	20	19
2	17	1	11	10

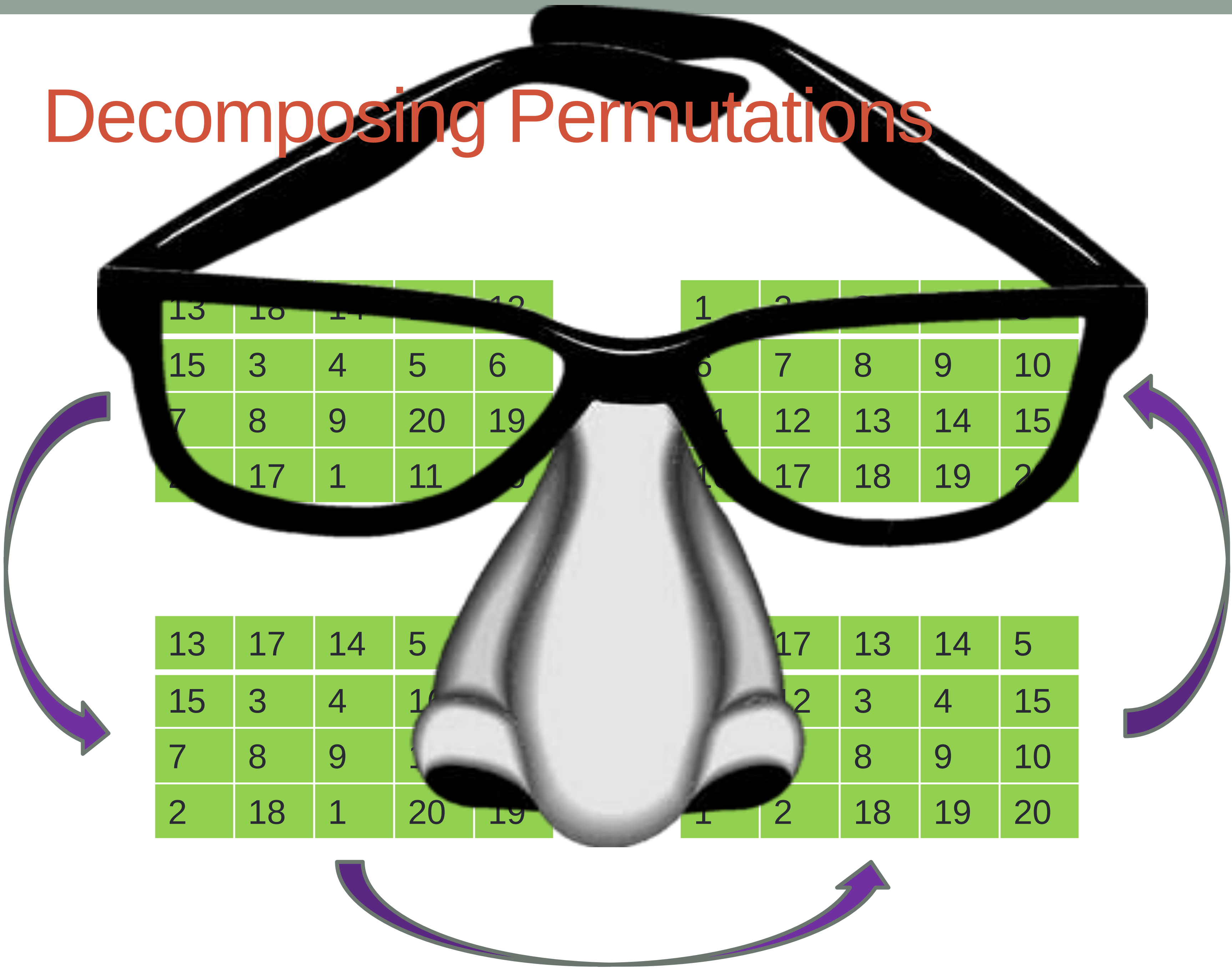
?

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20

13	17	14	5	6
15	3	4	16	12
7	8	9	11	10
2	18	1	20	19

6	17	13	14	5
16	12	3	4	15
11	7	8	9	10
1	2	18	19	20

Decomposing Permutations



Fan-Out and Cloning

intended multiplicity

2	3	1	1	2	2	1
x_1	x_2	x_3	x_4	x_5	x_6	x_7

1	2	2	4	1	2	1
x_8	x_9	x_{10}	x_{11}	x_{12}	x_{13}	x_{14}

2	1	3	1	2	1	2
x_{15}	x_{16}	x_{17}	x_{18}	x_{19}	x_{20}	x_{21}

variables

Fan-Out and Cloning

2	3	1	1	2	2	1
x_1	x_2	x_3	x_4	x_5	x_6	x_7

1	2	2	4	1	2	1
x_8	x_9	x_{10}	x_{11}	x_{12}	x_{13}	x_{14}

2	1	3	1	2	1	2
x_{15}	x_{16}	x_{17}	x_{18}	x_{19}	x_{20}	x_{21}

Sort by intended multiplicity:

4	3	3	2	2	2	2
x_{11}	x_2	x_{17}	x_1	x_5	x_6	x_9

2	2	2	2	2	1	1
x_{10}	x_{13}	x_{15}	x_{19}	x_{21}	x_3	x_4

1	1	1	1	1	1	1
x_7	x_8	x_{12}	x_{14}	x_{16}	x_{18}	x_{20}

Fan-Out and Cloning

4	3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1
x_{11}	x_2	x_{17}	x_1	x_5	x_6	x_9	x_{10}	x_{13}	x_{15}	x_{19}	x_{21}	x_3	x_4	x_7	x_8	x_{12}	x_{14}	x_{16}	x_{18}	x_{20}

Replicate

4	3	3				
x_{11}	x_2	x_{17}				

Replicate and shift

			4	3	3	
			x_{11}	x_2	x_{17}	

Fan-Out and Cloning

4	3	3	2	2	2	2
x_{11}	x_2	x_{17}	x_1	x_5	x_6	x_9

2	2	2	2	2	1	1
x_{10}	x_{13}	x_{15}	x_{19}	x_{21}	x_3	x_4

1	1	1	1	1	1	1
x_7	x_8	x_{12}	x_{14}	x_{16}	x_{18}	x_{20}

Merge

4	3	3	4	3	3	
x_{11}	x_2	x_{17}	x_{11}	x_2	x_{17}	

Fan-Out and Cloning

4	3	3	2	2	2	2
x_{11}	x_{2}	x_{17}	x_{1}	x_{5}	x_{6}	x_{9}

2	2	2	2	2	1	1
x_{10}	x_{13}	x_{15}	x_{19}	x_{21}	x_{3}	x_{4}

1	1	1	1	1	1	1
x_{7}	x_{8}	x_{12}	x_{14}	x_{16}	x_{18}	x_{20}

Replicate, shift, merge

4	3	3	4	3	3	2
x_{11}	x_{2}	x_{17}	x_{11}	x_{2}	x_{17}	x_{1}

2	2	2	2	2	2	2
x_{5}	x_{6}	x_{9}	x_{10}	x_{13}	x_{15}	x_{19}

2						
x_{21}						

Replicate, shift

4	3	3	4	3	3	2
x_{11}	x_{2}	x_{17}	x_{11}	x_{2}	x_{17}	x_{1}

2	2	2	2	2	2	2
x_{5}	x_{6}	x_{9}	x_{10}	x_{13}	x_{15}	x_{19}

	2					
	x_{21}					

Fan-Out and Cloning

4	3	3	2	2	2	2
x_{11}	x_2	x_{17}	x_1	x_5	x_6	x_9

2	2	2	2	2	1	1
x_{10}	x_{13}	x_{15}	x_{19}	x_{21}	x_3	x_4

1	1	1	1	1	1	1
x_7	x_8	x_{12}	x_{14}	x_{16}	x_{18}	x_{20}

4	3	3	4	3	3	2
x_{11}	x_2	x_{17}	x_{11}	x_2	x_{17}	x_1

2	2	2	2	2	2	2
x_5	x_6	x_9	x_{10}	x_{13}	x_{15}	x_{19}

2	2					
x_{21}	x_{21}					

Merge

4	3	3	4	3	3	2
x_{11}	x_2	x_{17}	x_{11}	x_2	x_{17}	x_1

2	2	2	2	2	2	2
x_5	x_6	x_9	x_{10}	x_{13}	x_{15}	x_{19}

Fan-Out and Cloning

4	3	3	2	2	2	2
x_{11}	x_2	x_{17}	x_1	x_5	x_6	x_9

2	2	2	2	2	1	1
x_{10}	x_{13}	x_{15}	x_{19}	x_{21}	x_3	x_4

1	1	1	1	1	1	1
x_7	x_8	x_{12}	x_{14}	x_{16}	x_{18}	x_{20}

Copy, merge

4	3	3	4	3	3	2
x_{11}	x_2	x_{17}	x_{11}	x_2	x_{17}	x_1

2	2	2	2	2	2	2
x_5	x_6	x_9	x_{10}	x_{13}	x_{15}	x_{19}

2	2	1	1			
x_{21}	x_{21}	x_3	x_4			

Copy

4	3	3	4	3	3	2
x_{11}	x_2	x_{17}	x_{11}	x_2	x_{17}	x_1

2	2	2	2	2	2	2
x_5	x_6	x_9	x_{10}	x_{13}	x_{15}	x_{19}

1	1	1	1	1	1	1
x_7	x_8	x_{12}	x_{14}	x_{16}	x_{18}	x_{20}

Each variable appears at least as much as needed