BasedAccel

Background

Motivation

What if we wanted to compute on sensitive data, without having the resources to do it ourselves?

- Can offload computation to a server
  - Requires trusting the cloud provider
  - Do we really want amazon, microsoft, etc. to have access to our data?
- What if we could send encrypted data to the cloud provider, and get back encrypted results?
  - We can!

What is Homomorphic Encryption?

- A homomorphism is a function $\varphi : A \rightarrow B$ such that $\forall\ x, y \in A, \varphi(x)\varphi(y) = \varphi(xy)$
  - Now consider $\exists\ \varphi^{-1} : B \rightarrow A$
- A homomorphic encryption scheme is one in which the encrypting function is a homomorphism
  - Allows us to compute on two encrypted numbers and have the decrypted result be correct $(+, \times)$

$$D(E(a) \times E(b)) = a \times b$$

Why is this hard?

- Homomorphic encryption schemes are approximate
  - Loss of precision is expected in intermediate stages, leads to the divergence of intermediate values based on precision of implementation
  - Very difficult to debug
- Custom Data types
  - The data types that are needed in HE are not optimal for current CPU/GPU.
- Computationally intensive (this is where we come in)
  - Multiplying 2 integers requires ≈ 12 polynomial multiplications

What HE Scheme?

BFV VS. CKKS

- Application Specific
  - BFV is preserves computation exactly (i.e. 6 X 7 = 42)
  - CKKS is approximate (i.e. 6 X 7 ≈ 42)

- BFV
  - Simpler to debug and implement as it does not rely on FP operations

The BFV Scheme

The Computation

- Goal
  - Multiply a = 6, b =7 in the ciphertext domain to get 42 when decrypted.
- Steps
  - Encode **a** and **b** to $\mathbf{m}_1$ and $\mathbf{m}_2$
  - Encrypt $\mathbf{m}_1$ and $\mathbf{m}_2$ to $\mathbf{e}_1$ and $\mathbf{e}_2$
  - HE multiply $\mathbf{e}_1$ and $\mathbf{e}_2$
  - Relinearization
  - Decode
  - Decrypt to get 42!

Encode

- 6 is encoded as a polynomial
  - $m_1 = 0 + 1*x^1 + 1*x^2 + 0*x^3 + 0*x^4 + 0*x^5 + 0*x^6 + 0*x^7$

- 7 is encoded as a polynomial
  - $m_2 = 1 + 1*x^1 + 1*x^2 + 0*x^3 + 0*x^4 + 0*x^5 + 0*x^6 + 0*x^7$

Encrypt

- m_1 is encrypted using the **learning with errors scheme**
  - $ct_1[0] = 62950 + 61266*x^1 + 36292*x^2 + 34996*x^3 + 4259*x^4 + 17810*x^5 + 35974*x^6 + 3035*x^7$
  - $ct_1[1] = 29702 + 2268*x^1 + 45163*x^2 + 49460*x^3 + 61948*x^4 + 64792*x^5 + 15858*x^6 + 59854*x^7$

- m_2 is encrypted using the **learning with errors scheme**
  - $ct_2[0] = 5968 + 22829*x^1 + 3671*x^2 + 6355*x^3 + 24003*x^4 + 60410*x^5 + 33967*x^6 + 44192*x^7$
  - $ct_2[1] = 12239 + 7763*x^1 + 61736*x^2 + 42164*x^3 + 45609*x^4 + 2485*x^5 + 42318*x^6 + 37811*x^7$

Homomorphic Addition

- Simply add the encrypted Polynomials
  - $ct_{add}[0] = [ct_1[0] + ct_2[0]] \bmod q^*$
  - $ct_{add}[1] = [ct_1[1] + ct_2[1]] \bmod q^*$

- *: using the modulus operations makes sure that the summed ciphertext is still in the scope of the Galois field.
  - For example: q == 7,
    - (8 + 9) mod q = 3

Homomorphic Multiplication

- HE multiplication is where it gets fun!
  - $\mathrm{ct}_{\mathrm{mul}}[0] = \left[\left\lfloor \frac{t \cdot (\mathtt{ct}_1[0] \cdot \mathtt{ct}_2[0])}{q} \right\rceil\right]_q$
  - $\mathrm{ct}_{\mathrm{mul}}[1] = \left[\left\lfloor \frac{t \cdot (\mathtt{ct}_1[0] \cdot \mathtt{ct}_2[1] + \mathtt{ct}_1[1] \cdot \mathtt{ct}_2[0])}{q} \right\rceil\right]_q$
  - $\mathrm{ct}_{\mathrm{mul}}[2] = \left[\left\lfloor \frac{t \cdot (\mathtt{ct}_1[1] \cdot \mathtt{ct}_2[1])}{q} \right\rceil\right]_q$

Homomorphic Multiplication $\left[\left\lfloor \frac{t \cdot (\mathtt{ct}_1[0] \cdot \mathtt{ct}_2[0])}{q} \right\rceil\right]_q$

- What Does $\left[\left\lfloor \frac{t \cdot (\mathtt{ct}_1[0] \cdot \mathtt{ct}_2[0])}{q} \right\rceil\right]_q$ mean?
- Polynomial Multiplication: $ct_1[0]$ x $ct_2[0]$
  - $ct_1[0] = 62950 + 61266*x^1 + 36292*x^2 + 34996*x^3 + 4259*x^4 + 17810*x^5 + 35974*x^6 + 3035*x^7$
  - $ct_2[0] = 5968 + 22829*x^1 + 3671*x^2 + 6355*x^3 + 24003*x^4 + 60410*x^5 + 33967*x^6 + 44192*x^7$
- $ct_{mul\_raw1}[0] = 375685600 + 1802721038*x^1 + 1846321620*x^2 + 1662320932*x^3 + 2857903608*x^4 + 5836002065*x^5 + 7569730277*x^6 + 8827127111*x^7 + 6471063692*x^8 + 3717061071*x^9 + 3649882132*x^{10} + 3039204443*x^{11} + 2192332728*x^{12} + 1692852853*x^{13} + 134122720*x^{14} + 0*x^{15}$

Homomorphic Multiplication $\left[\left\lfloor \frac{t \cdot (\mathtt{ct}_1[0] \cdot \mathtt{ct}_2[0])}{q} \right\rceil\right]_q$

- Polynomial Modulus: $ct_{mul\_raw2}[i] = ct_{mul\_raw1}[i] - ct_{mul\_raw1}[i + 8]$
  - $ct_{mul\_raw1}[0] = 375685600 + 1802721038*x^1 + 1846321620*x^2 + 1662320932*x^3 + 2857903608*x^4 + 5836002065*x^5 + 7569730277*x^6 + 8827127111*x^7 + 6471063692*x^8 + 3717061071*x^9 + 3649882132*x^{10} + 3039204443*x^{11} + 2192332728*x^{12} + 1692852853*x^{13} + 134122720*x^{14} + 0*x^{15}$
  - $ct_{mul\_raw2}[0] = -6095378092 + -1914340033*x^1 + -1803560512*x^2 + -1376883511*x^3 + 665570880*x^4 + 4143149212*x^5 + 7435607557*x^6 + 8827127111*x^7$

Caution: Overflow

Homomorphic Multiplication $\left[\left\lfloor \frac{t \cdot (\mathtt{ct}_1[0] \cdot \mathtt{ct}_2[0])}{q} \right\rceil\right]_q$

- Times t: $ct_{mul\_raw3}[i] = (t == 8) \times ct_{mul\_raw2}[i]$
  - $ct_{mul\_raw2}[0]$ = [illegible]x^1 + -1803560512*x^2 + -1376883511*x^3 + 665570880*x^4 + 4143149212*x^5 [illegible]x^7

    Caution: Overflow
  - $ct_{mul\_raw3}[0]$ = -48763024736 + -15314720264*x^1 + -14428484096*x^2 + -11015068088*x^3 + 5324567040*x^4 + 33145193696*x^5 + 59484860456*x^6 + 70617016888*x^7

Homomorphic Multiplication $\left[\left\lfloor \frac{t \cdot (\mathtt{ct}_1[0] \cdot \mathtt{ct}_2[0])}{q} \right\rceil\right]_q$

- Divide by q: $ct_{mul\_raw4}[i] = truncate(ct_{mul\_raw3}[i] / (q == 65521))$
  - $ct_{mul\_raw3}[0] =$ -48763024736 + -15314720264*x^1 + -14428484096*x^2 + -11015068088*x^3 + 5324567040*x^4 + 33145193696*x^5 + 59484860456*x^6 + 70617016888*x^7
  - $ct_{mul\_raw4}[0] =$ -744235 + -233738*x^1 + -220212*x^2 + -168115*x^3 + 81265*x^4 + 505871*x^5 + 907875*x^6 + 1077777*x^7

Homomorphic Multiplication $\left[\left\lfloor \frac{t \cdot (\mathtt{ct}_1[0] \cdot \mathtt{ct}_2[0])}{q} \right\rceil\right]_q$

- Mod by q: $ct_{mul}[i] = ct_{mul\_raw4}[i] \% (q == 65521)$
  - $ct_{mul\_raw4}[0] = -744235 + -233738*x^1 + -220212*x^2 + -168115*x^3 + 81265*x^4 + 505871*x^5 + 907875*x^6 + 1077777*x^7$
  - $ct_{mul}[0] = 42017 + 28346*x^1 + 41872*x^2 + 28448*x^3 + 15744*x^4 + 47224*x^5 + 56102*x^6 + 29441*x^7$

Fun Fact: % is different between languages

Python - True Modulus: -2 % 5 = 3 => We want this as we polynomials represent rings!

C- Remainder Operation: -2 % 5 = -2

System Verilog Non Synthesizable: Must explicitly mark -2 as signed

Homomorphic Multiplication

- Now we have $ct_{mul}[0]$, $ct_{mul}[1]$ and $ct_{mul}[2]$ can be computed the same
  - $ct_{mul}[0] = \left[\left\lfloor \frac{t \cdot (\mathtt{ct}_1[0] \cdot \mathtt{ct}_2[0])}{q} \right\rceil\right]_q$
  - $ct_{mul}[1] = \left[\left\lfloor \frac{t \cdot (\mathtt{ct}_1[0] \cdot \mathtt{ct}_2[1] + \mathtt{ct}_1[1] \cdot \mathtt{ct}_2[0])}{q} \right\rceil\right]_q$
  - $ct_{mul}[2] = \left[\left\lfloor \frac{t \cdot (\mathtt{ct}_1[1] \cdot \mathtt{ct}_2[1])}{q} \right\rceil\right]_q$

Relinearization

- We have a Problem
  - Recall that a number is represented as two polynomials?
    - $ct_1[0] = 62950 + 61266*x^1 + 36292*x^2 + 34996*x^3 + 4259*x^4 + 17810*x^5 + 35974*x^6 + 3035*x^7$
    - $ct_1[1] = 29702 + 2268*x^1 + 45163*x^2 + 49460*x^3 + 61948*x^4 + 64792*x^5 + 15858*x^6 + 59854*x^7$
  - After HE multiplication we have three!!
    - $ct_{mul}[0]$, $ct_{mul}[1]$, and $ct_{mul}[2]$

Relinearization

- Relinearization breaks $ct_{mul}[2]$ and are summed into $ct_{mul}[0]$ and $ct_{mul}[1]$
  - $ct'_{mul}[0] = \left[\mathbf{c}_0 + \sum_{i=0}^{\ell} \mathtt{rlk}[i][0] \cdot \mathbf{c}_2^{(i)}\right]_q$
  - $ct'_{mul}[1] = \left[\mathbf{c}_1 + \sum_{i=0}^{\ell} \mathtt{rlk}[i][1] \cdot \mathbf{c}_2^{(i)}\right]_q$

Relinearization

- Relinearization Keys (Galois Keys)
    - To break-up $ct_{mul}[2]$ we need to partially decrypt $ct_{mul}[2]$ using the keys
    - Relinearization Keys allow us to do this

Relinearization: Base T decomposition $\left[\mathbf{c}_0 + \sum_{i=0}^{\ell} \mathtt{rlk}[i][0] \cdot \mathbf{c}_2^{(i)}\right]_q$

- Decompose $ct_{mul}[2]$ using the base (T == 4)
  - $ct_{mul}[2]\_BaseT[i][j] = truncate(ct_{mul}[2][j] / T^i) \% T$
- $ct_{mul}[2]\_BaseT$
  - $ct_{mul}[2]\_BaseT[0] = 0\ 1\ 0\ 1\ 2\ 3\ 2\ 3$
  - $ct_{mul}[2]\_BaseT[1] = 0\ 3\ 1\ 0\ 3\ 3\ 3\ 3$
  - $ct_{mul}[2]\_BaseT[2] = 0\ 1\ 0\ 3\ 3\ 3\ 2\ 2$
  - $ct_{mul}[2]\_BaseT[3] = 1\ 0\ 1\ 2\ 3\ 3\ 0\ 2$
  - $ct_{mul}[2]\_BaseT[4] = 2\ 1\ 2\ 1\ 2\ 2\ 1\ 1$
  - $ct_{mul}[2]\_BaseT[5] = 3\ 2\ 0\ 3\ 3\ 2\ 2\ 2$
  - $ct_{mul}[2]\_BaseT[6] = 2\ 3\ 0\ 3\ 2\ 1\ 0\ 1$
  - $ct_{mul}[2]\_BaseT[7] = 2\ 1\ 3\ 2\ 3\ 0\ 1\ 2$

Relinearization: Base T decomposition

$$\left[\mathbf{c}_0 + \sum_{i=0}^{\ell} \mathtt{rlk}[i][0] \cdot \mathbf{c}_2^{(i)}\right]_q$$

- Poly Multiplication and Poly Modulus
  - Poly Multiplication: Buf = rlk[i][0] * ct_{mul}[2]_BaseT[i]
  - ct_{mul}[2]_BaseT[0] = 0 1 0 1 2 3 2 3 * rlk[0][0] = 34067, 895, 18164, 34124, 18726, 9815, 21963, 38534
- Accumulate
  - {ct_{mul}[0] += Buf } % q
  - ct_{mul}[2]_BaseT[0] = 0 1 0 1 2 3 2 3 * rlk[0][0] = 34067, 895, 18164, 34124, 18726, 9815, 21963, 38534

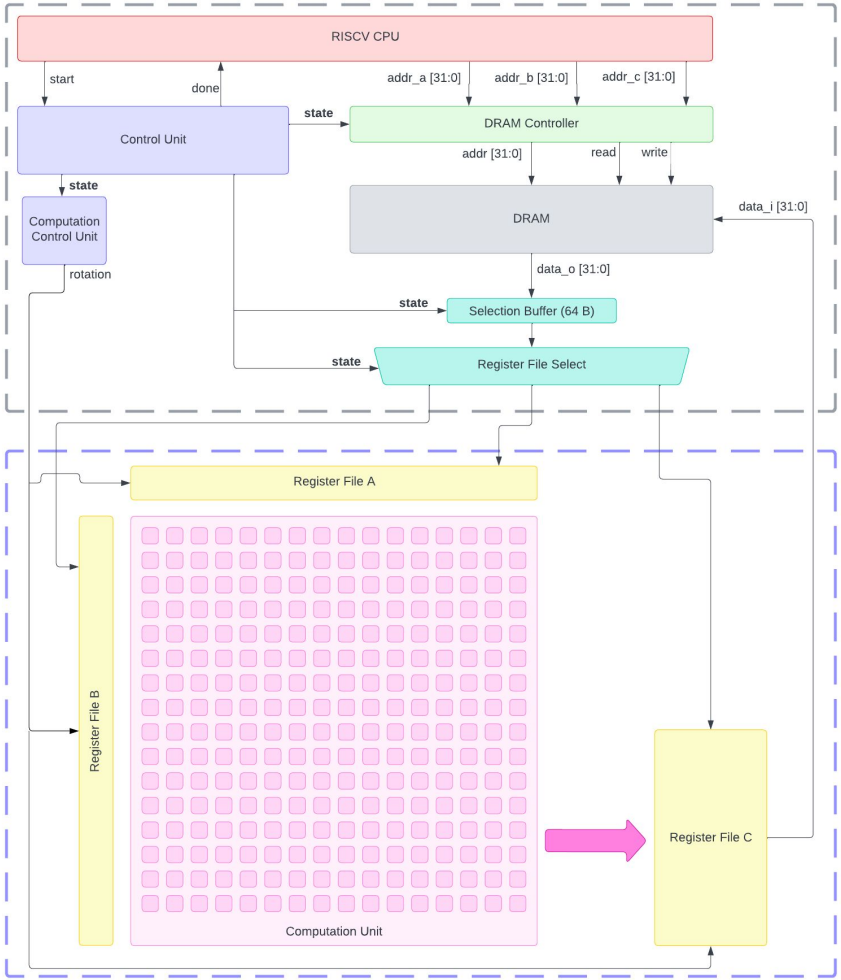
Decrypt

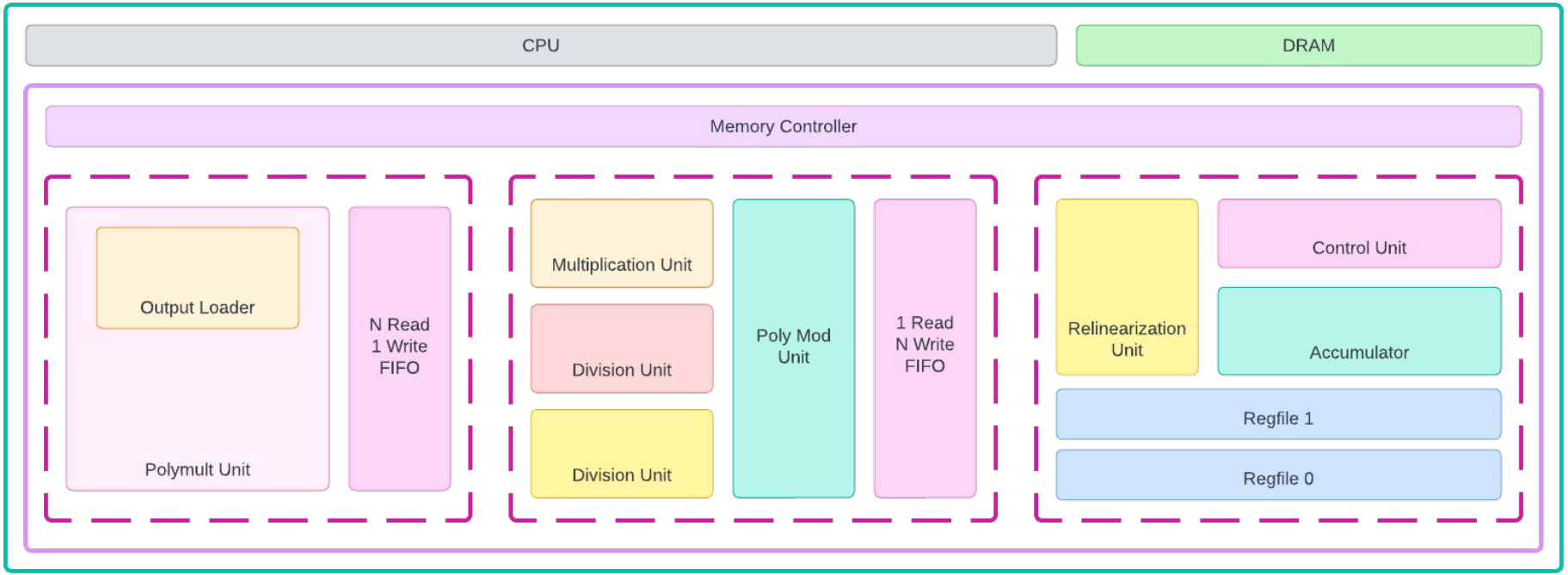
- ct_{rln} can finally be computed which represents 42
  - $ct_{rln}[0] = 6746 + 47676*x^1 + 24149*x^2 + 22056*x^3 + 33036*x^4 + 25158*x^5 + 41125*x^6 + 42683*x^7$
  - $ct_{rln}[1] = 30545 + 57750*x^1 + 42441*x^2 + 48114*x^3 + 62295*x^4 + 26764*x^5 + 37047*x^6 + 33334*x^7$

- ct_{rln} decrypted and decoded
  - $42_{encoded} = 0 + 1*x^1 + 2*x^2 + 2*x^3 + 1*x^4 + 0*x^5 + 0*x^6 + 0*x^7$

All this for 6 x 7!!

Design


RISCV CPU
start
done
addr_a [31:0]
addr_b [31:0]
addr_c [31:0]
Control Unit
state
DRAM Controller
addr [31:0]
read
write
state
Computation Control Unit
DRAM
data_i [31:0]
rotation
data_o [31:0]
state
Selection Buffer (64 B)
state
Register File Select
Register File A
Register File B
Computation Unit
Register File C


CPU
DRAM
Memory Controller
Output Loader
Polymult Unit
N Read
1 Write
FIFO
Multiplication Unit
Division Unit
Division Unit
Poly Mod
Unit
1 Read
N Write
FIFO
Relinearization
Unit
Control Unit
Accumulator
Regfile 1
Regfile 0

Pipeline

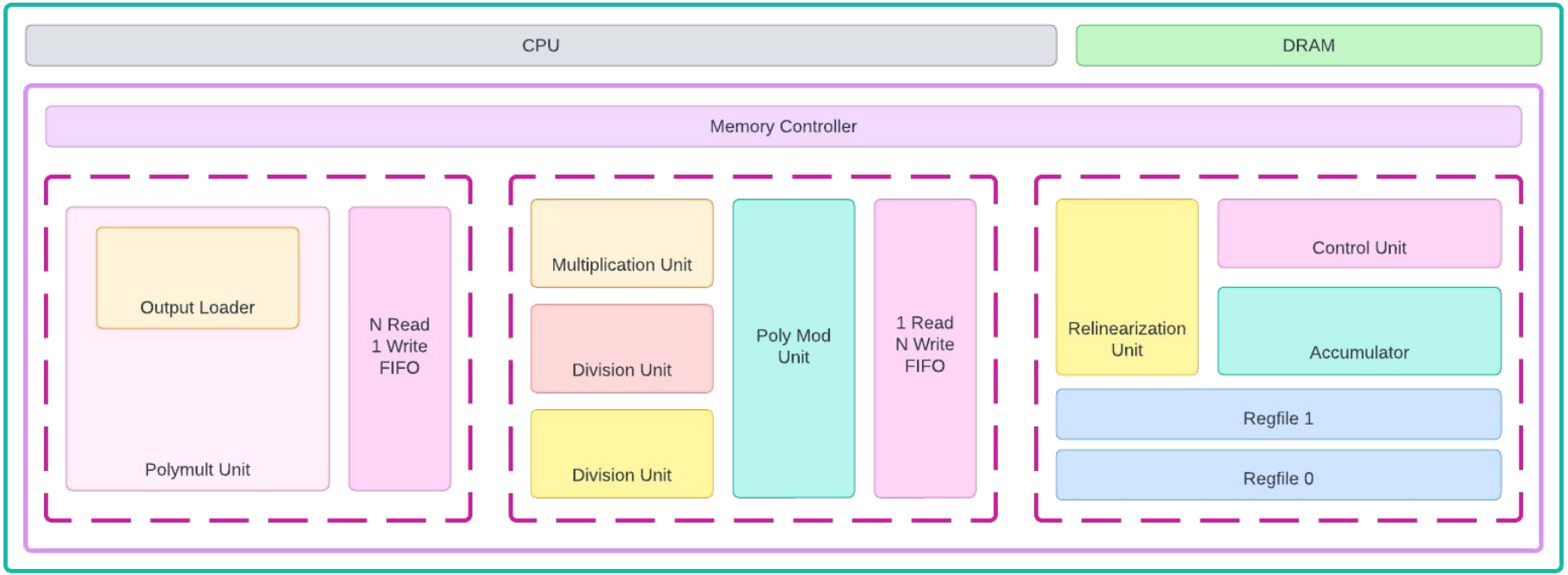
1. Polynomial Multiplication

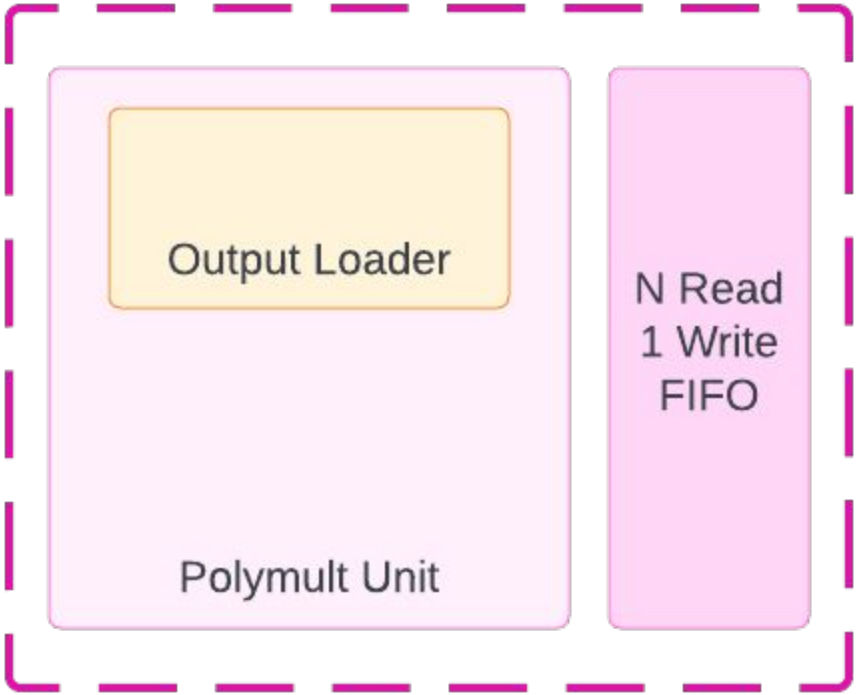
2. Re-contextualization

3. Re-linearization

Pipeline

- Computation pipelined between 3 computational segments
  - Imbalances managed by FIFO queueing
- Bottleneck occurs in first stage of the pipeline
  - Unavoidable -> computing **c2** first alleviates issues but does fix them entirely


CPU
DRAM
Memory Controller
Output Loader
Polymult Unit
N Read
1 Write
FIFO
Multiplication Unit
Division Unit
Division Unit
Poly Mod
Unit
1 Read
N Write
FIFO
Relinearization
Unit
Control Unit
Accumulator
Regfile 1
Regfile 0


Output Loader
Polymult Unit
N Read
1 Write
FIFO

Polynomial Multiplication

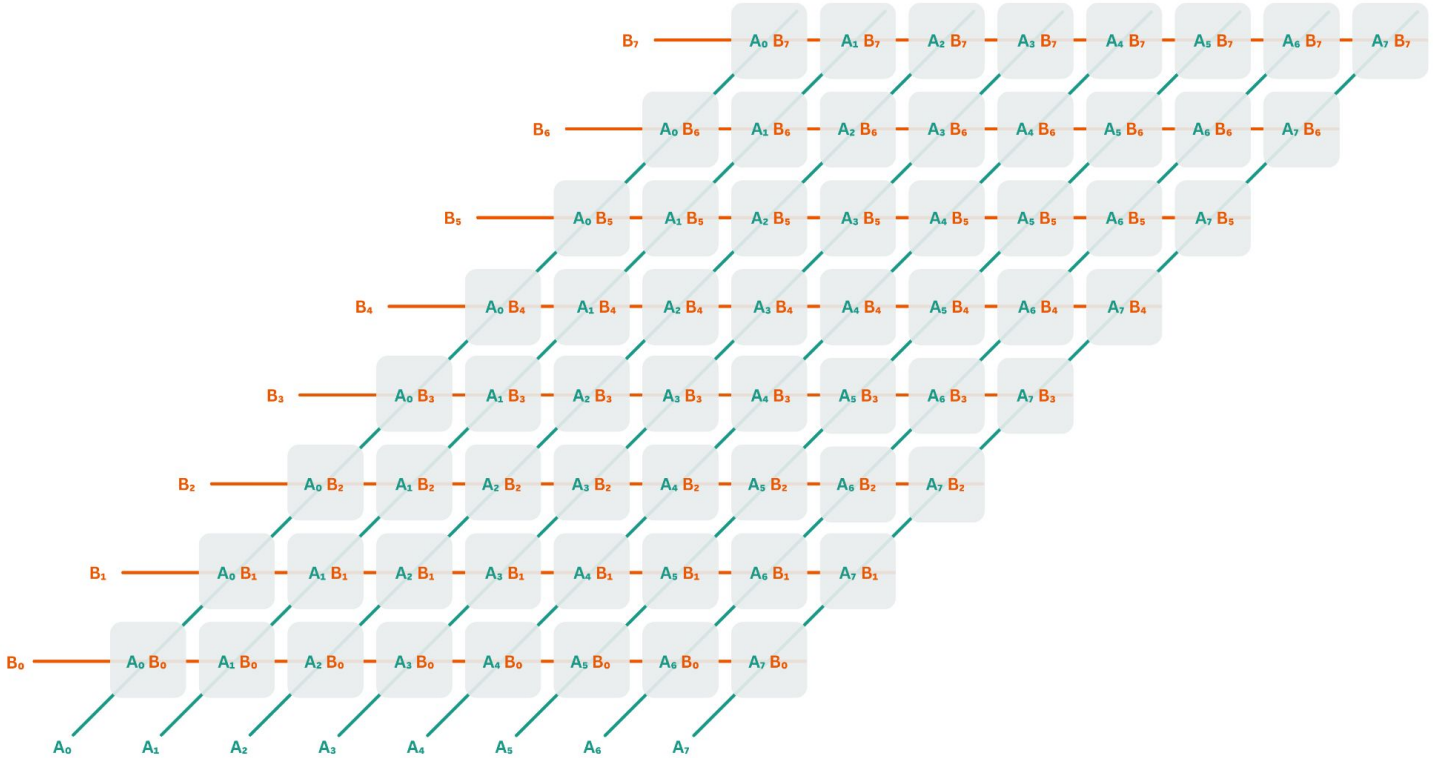
1. Computes tiles of c_0, $c_{1,0}$, $c_{1,1}$, c_2
2. Outputs results to FIFO

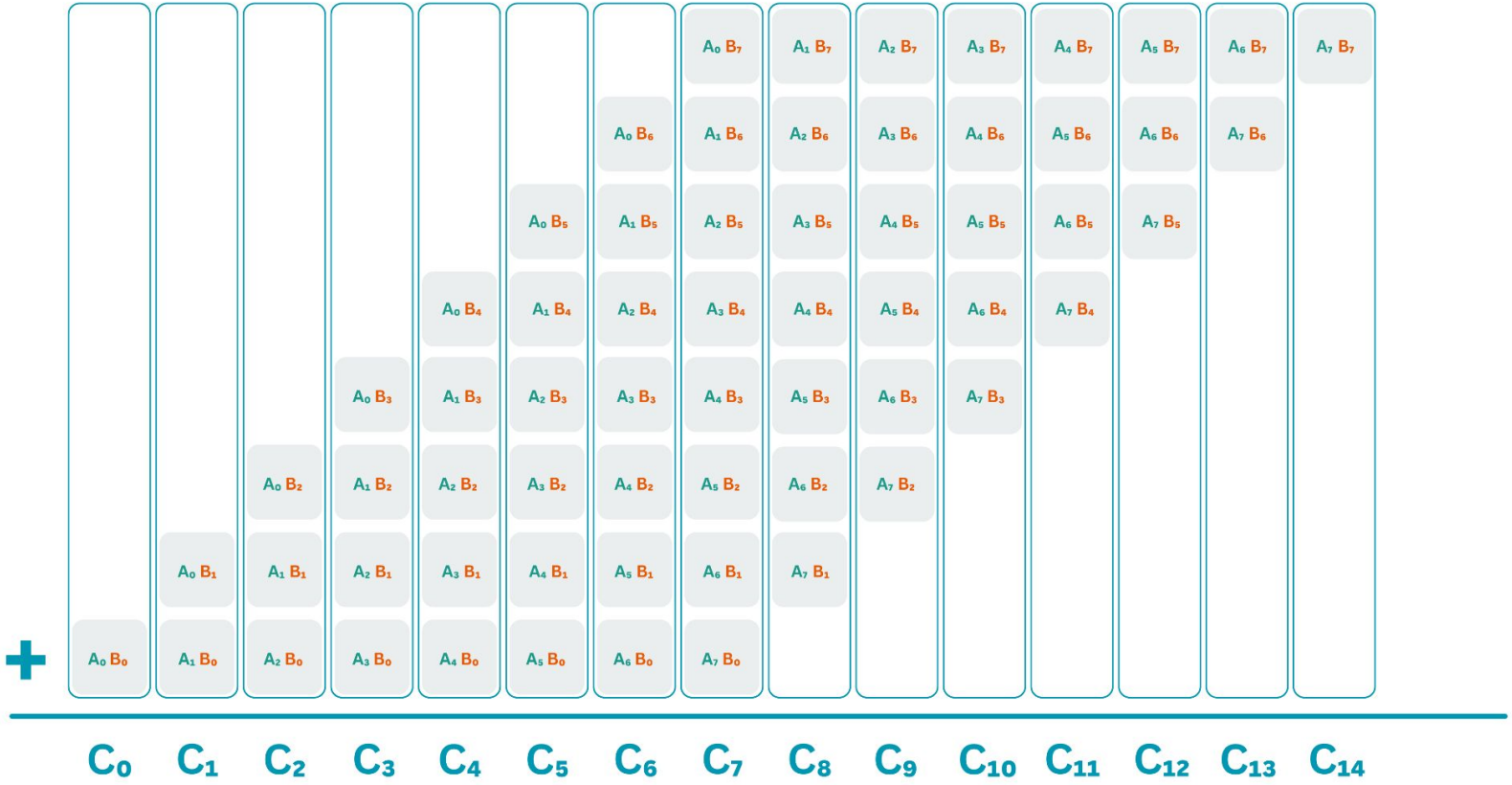
- We compute c_2 -> $c_{1,1}$ -> $c_{1,0}$ -> c_0
  - $c_{1,1} + c_{1,0} = c_1$
- Each result is of length *2*n*

$$A(x) = A_0x^0 + A_1x^1 + A_2x^2 + A_3x^3 + A_4x^4 + A_5x^5 + A_6x^6 + A_7x^7$$

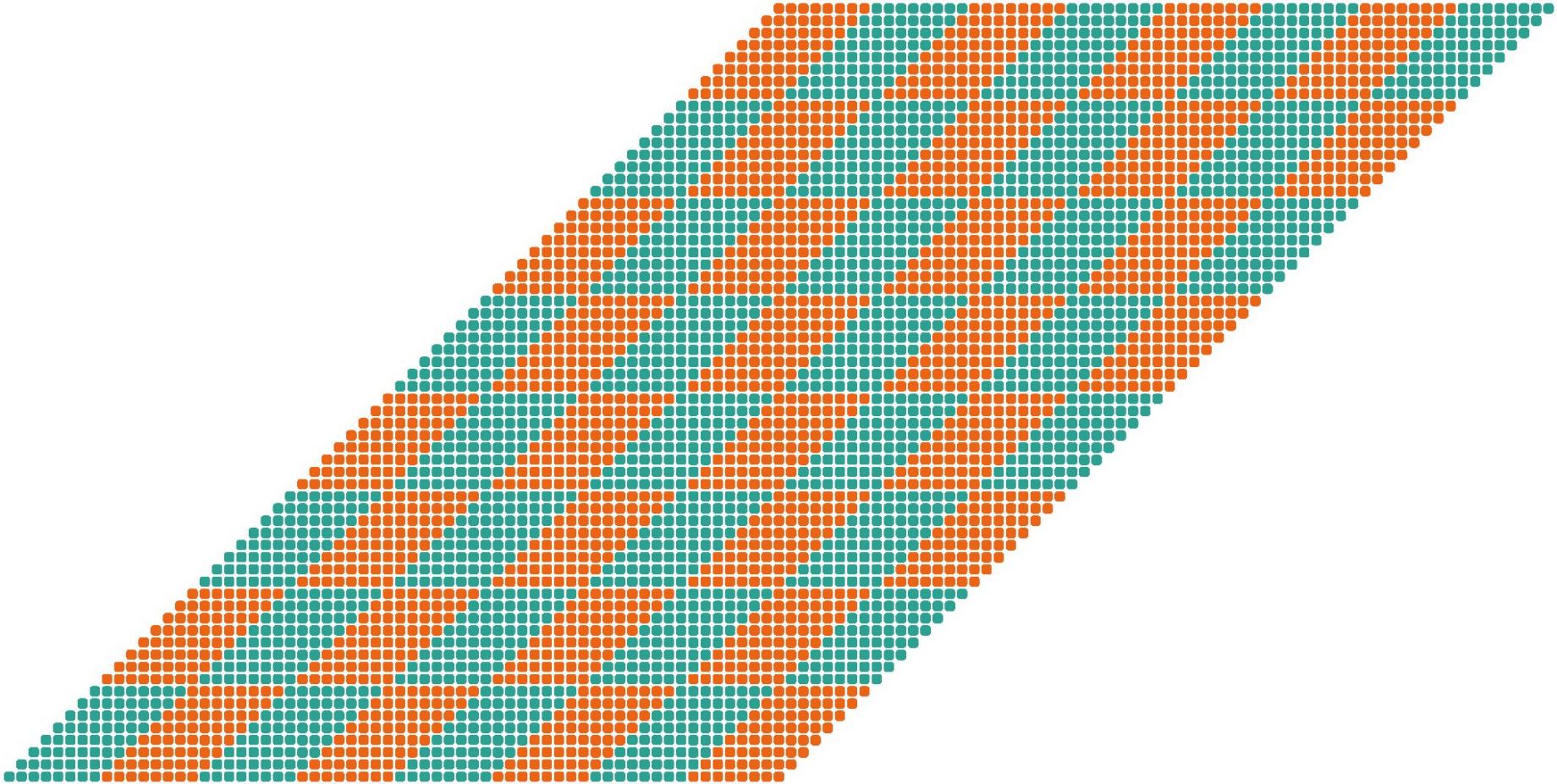
$$B(x) = B_0x^0 + B_1x^1 + B_2x^2 + B_3x^3 + B_4x^4 + B_5x^5 + B_6x^6 + B_7x^7$$

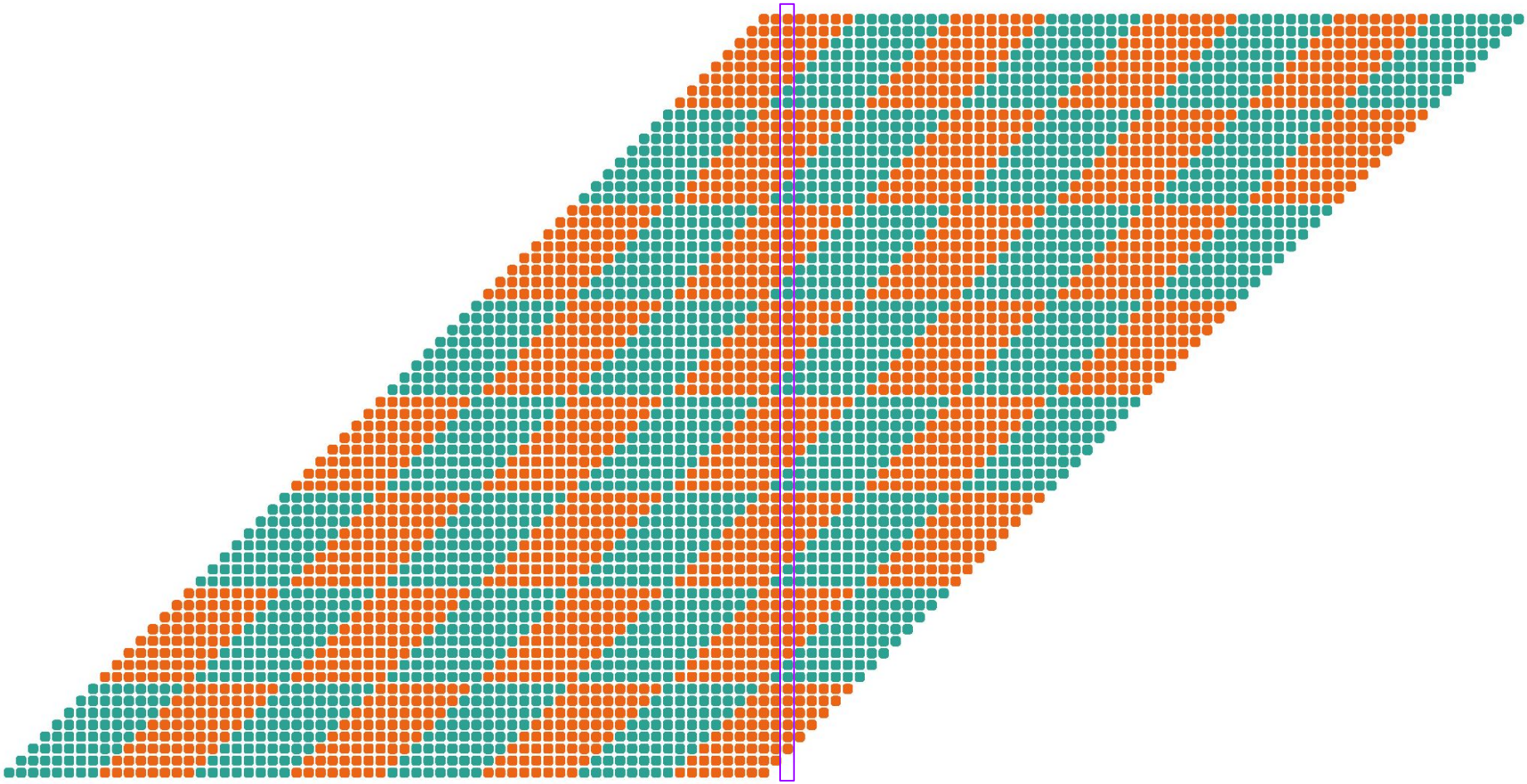
$$C(x) = C_0x^0 + C_1x^1 + C_2x^2 + C_3x^3 + C_4x^4 + C_5x^5 + C_6x^6 + C_7x^7 + C_8x^8 + C_9x^9 + C_{10}x^{10} + C_{11}x^{11} + C_{12}x^{12} + C_{13}x^{13} + C_{14}x^{14}$$

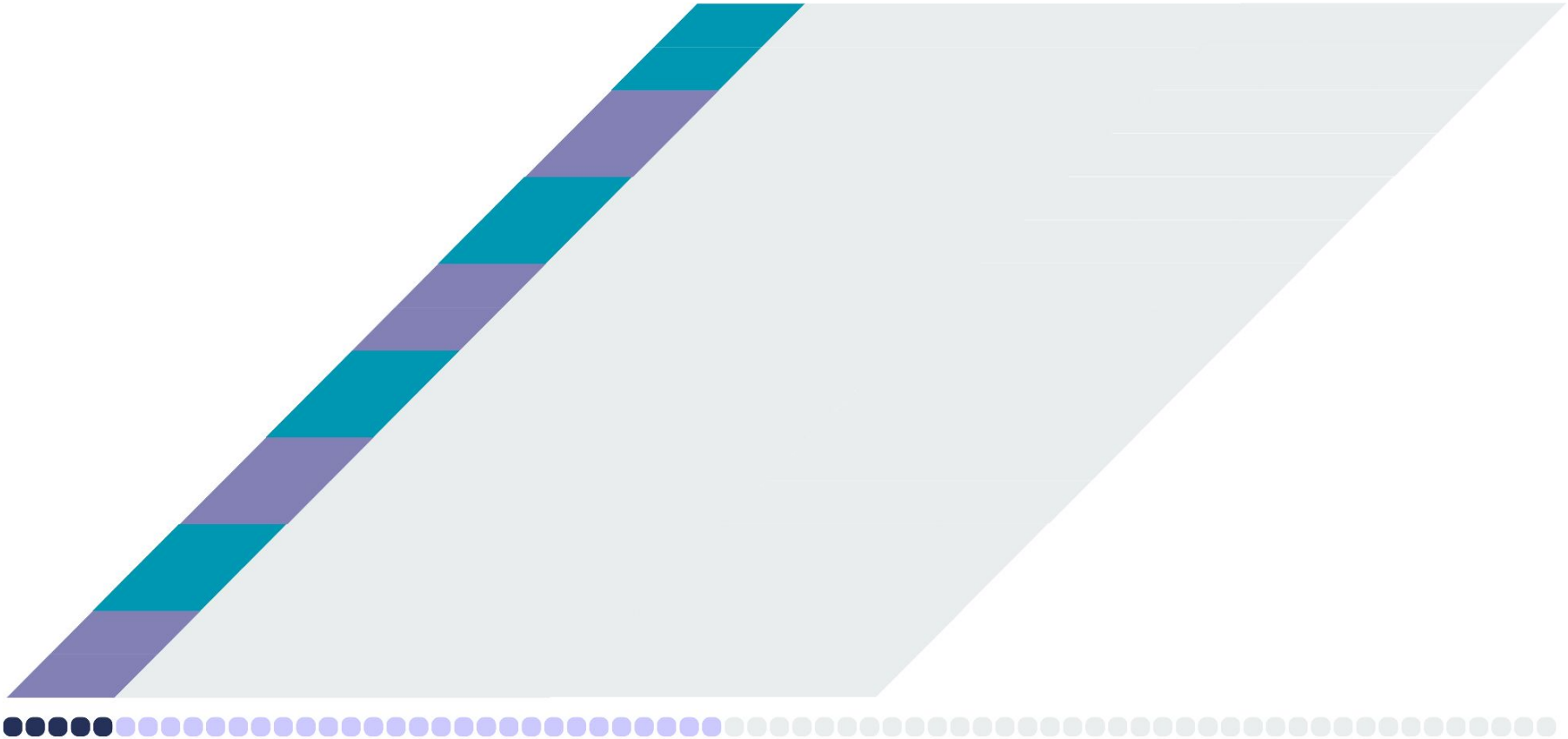


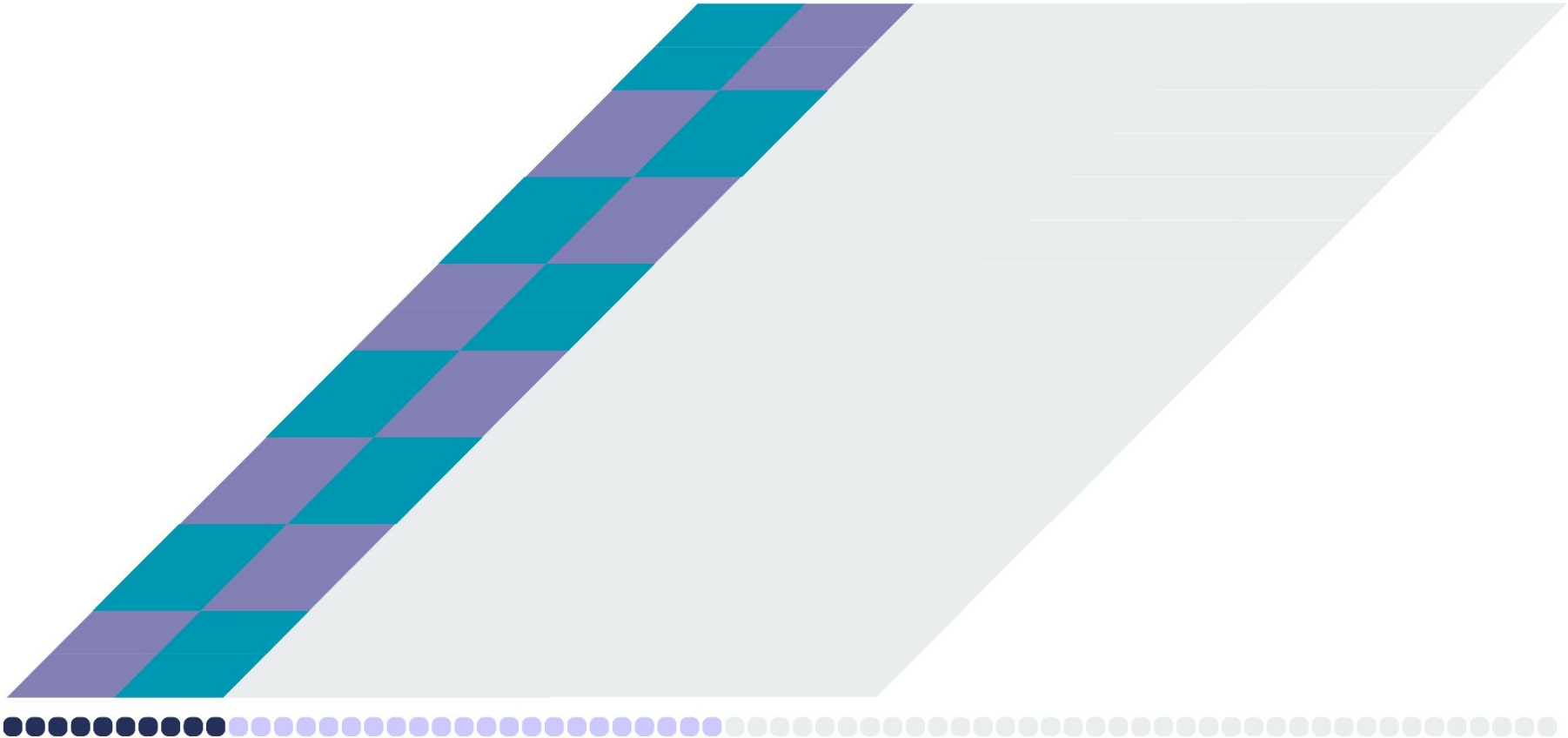


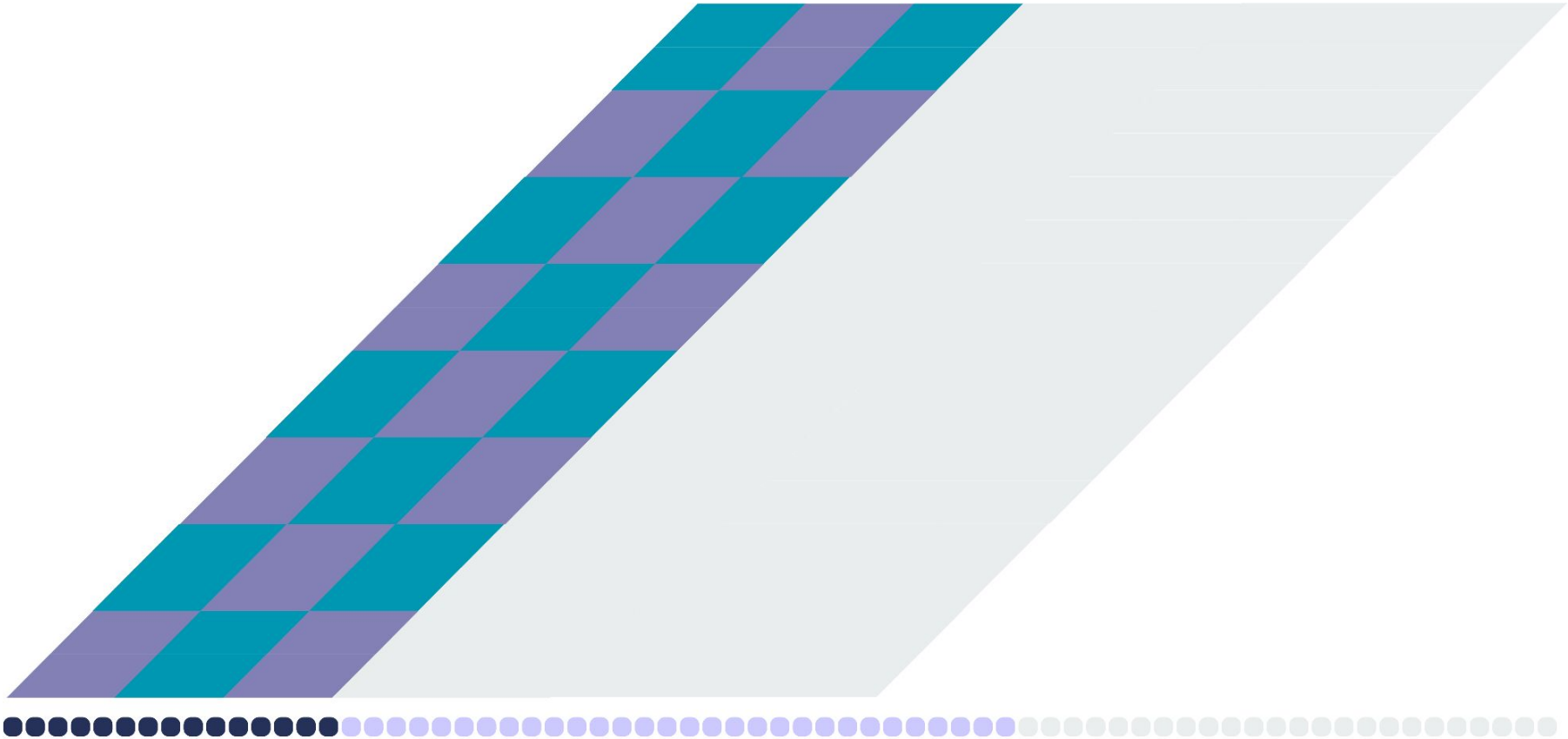
$$C(x) = C_0x^0 + C_1x^1 + C_2x^2 + C_3x^3 + C_4x^4 + C_5x^5 + C_6x^6 + C_7x^7 + C_8x^8 + C_9x^9 + C_{10}x^{10} + C_{11}x^{11} + C_{12}x^{12} + C_{13}x^{13} + C_{14}x^{14}$$

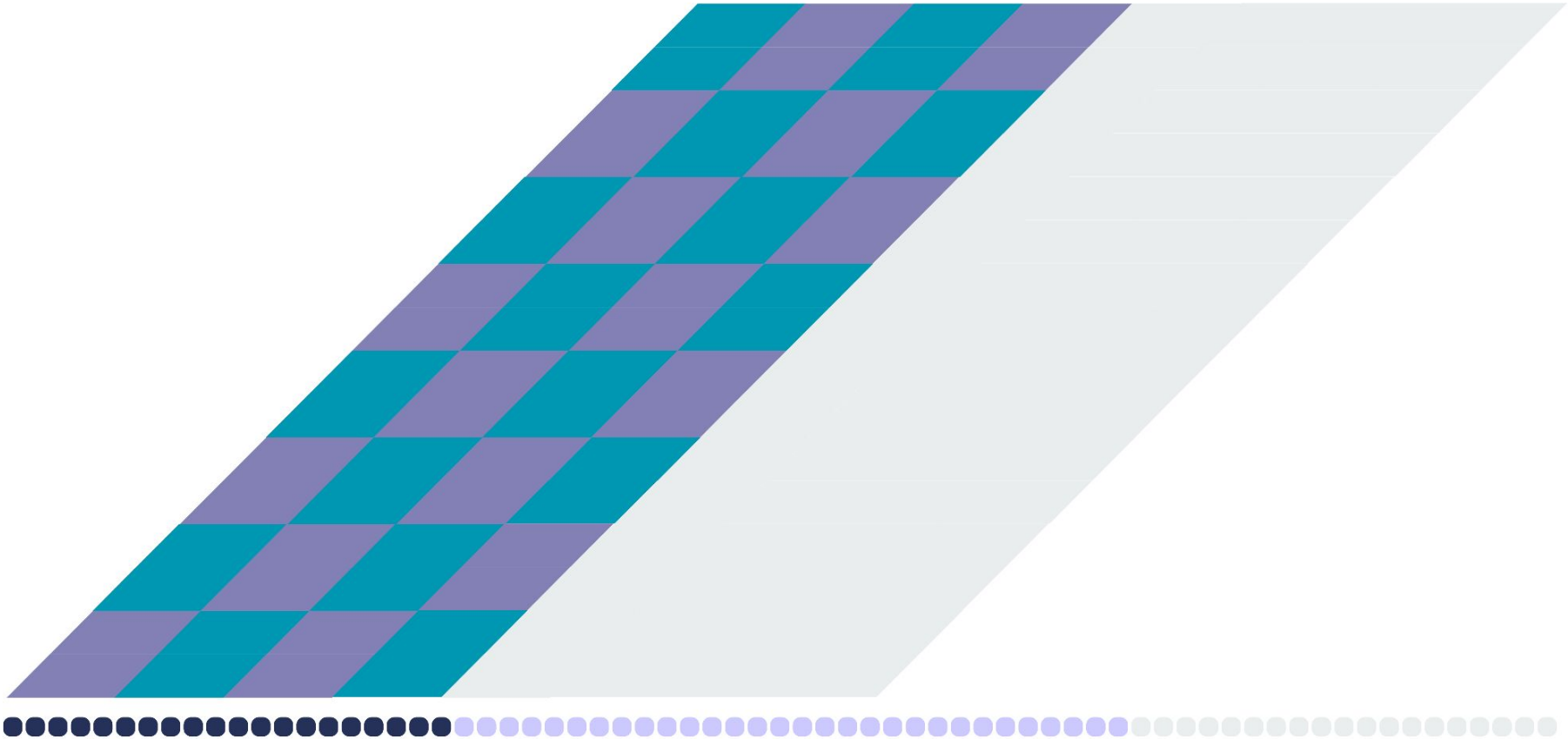


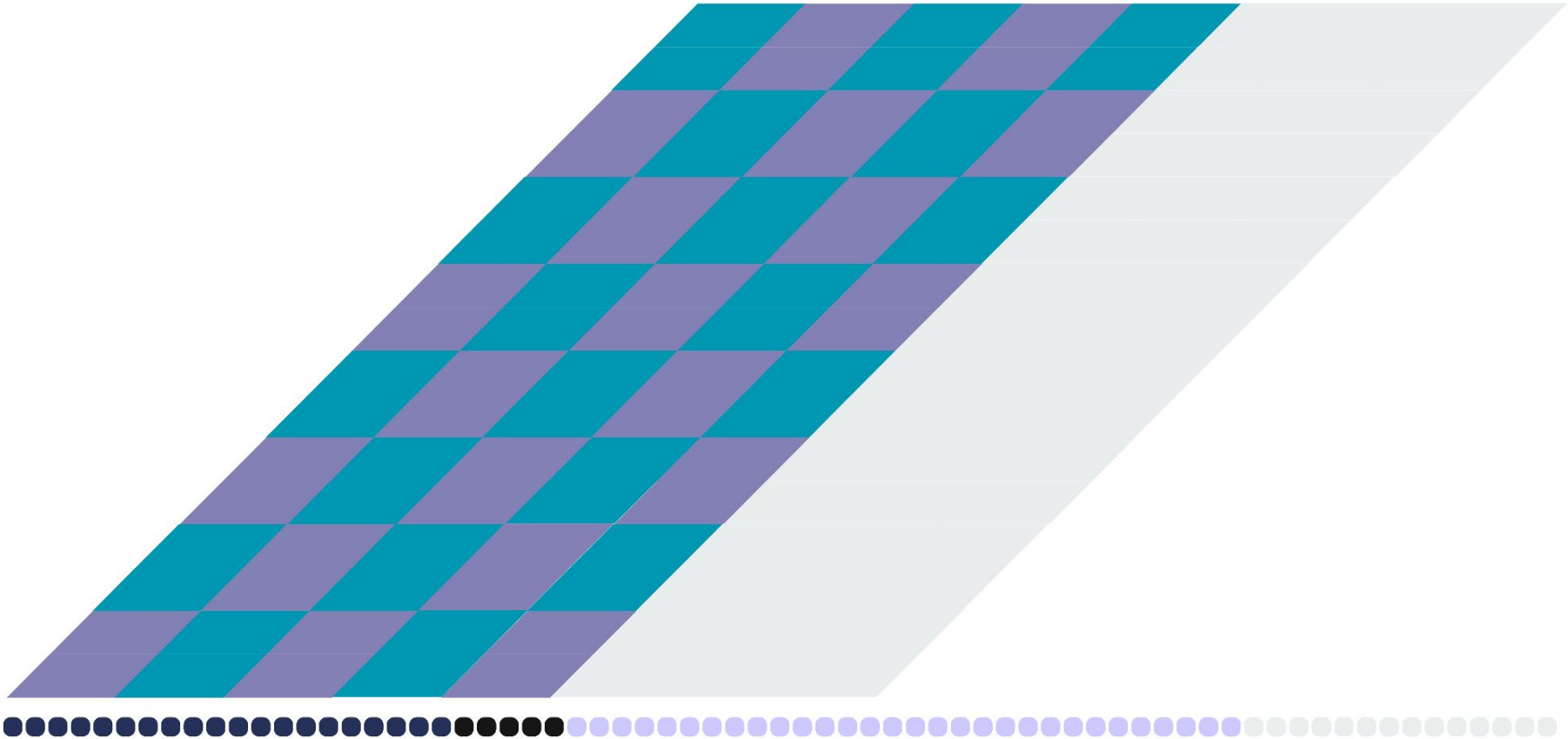


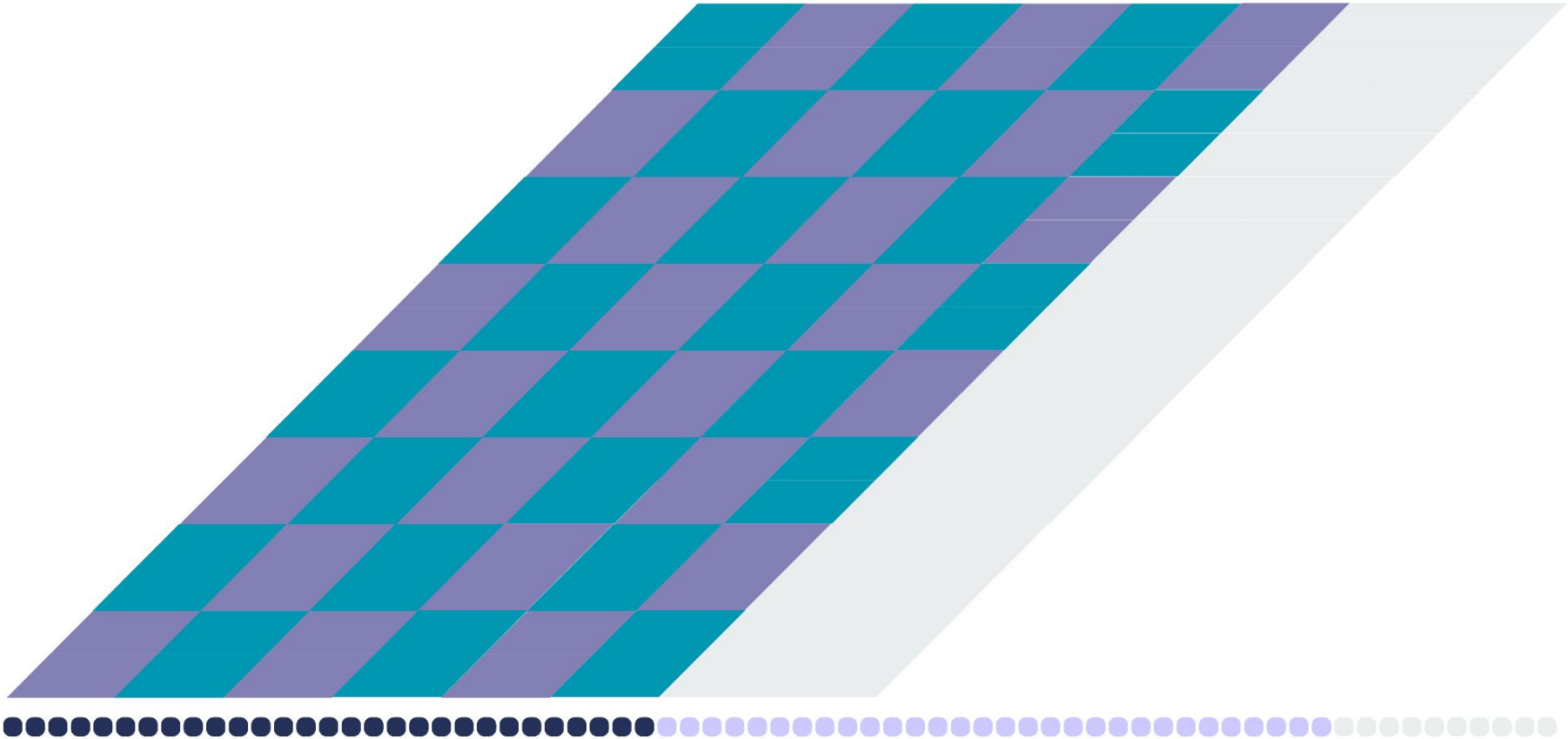


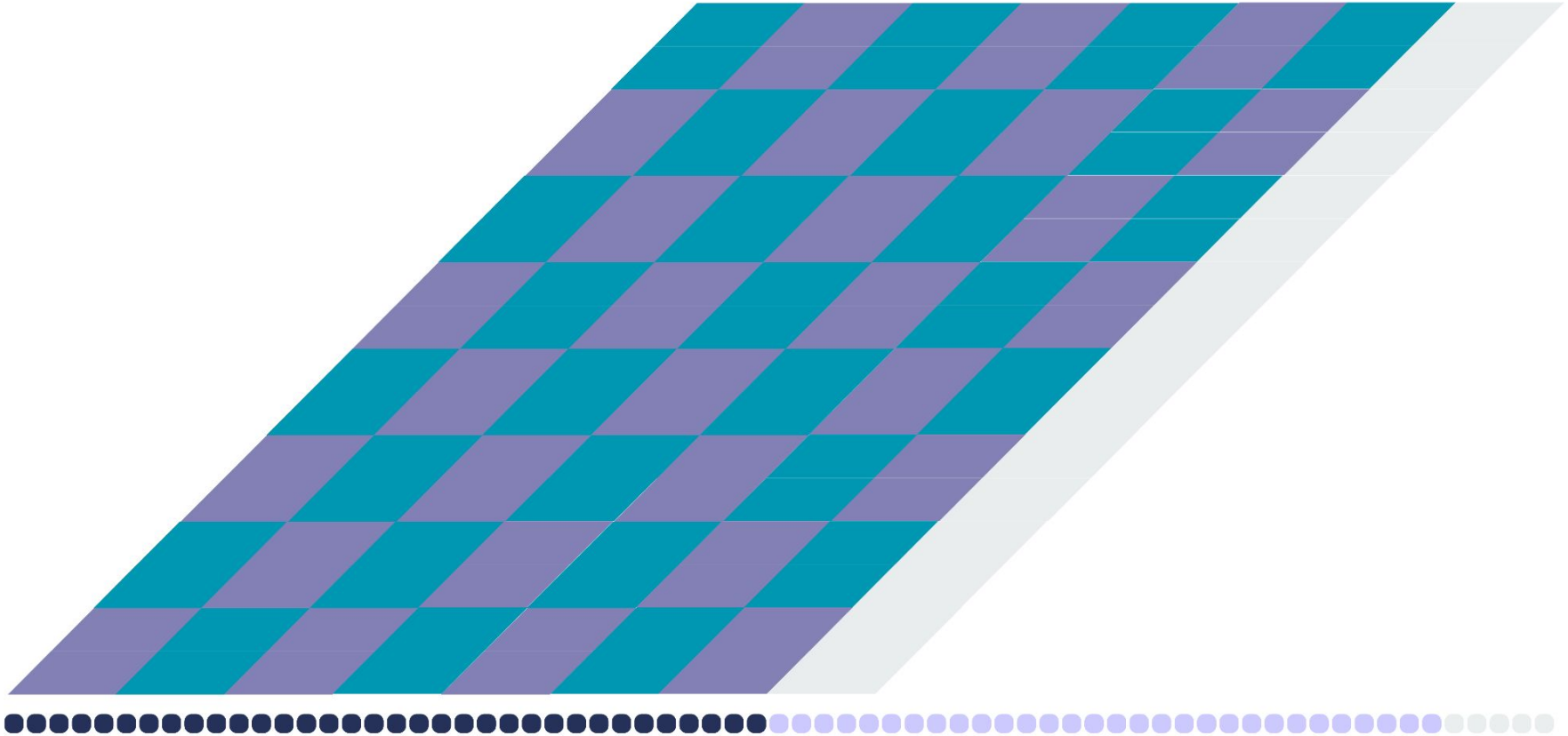


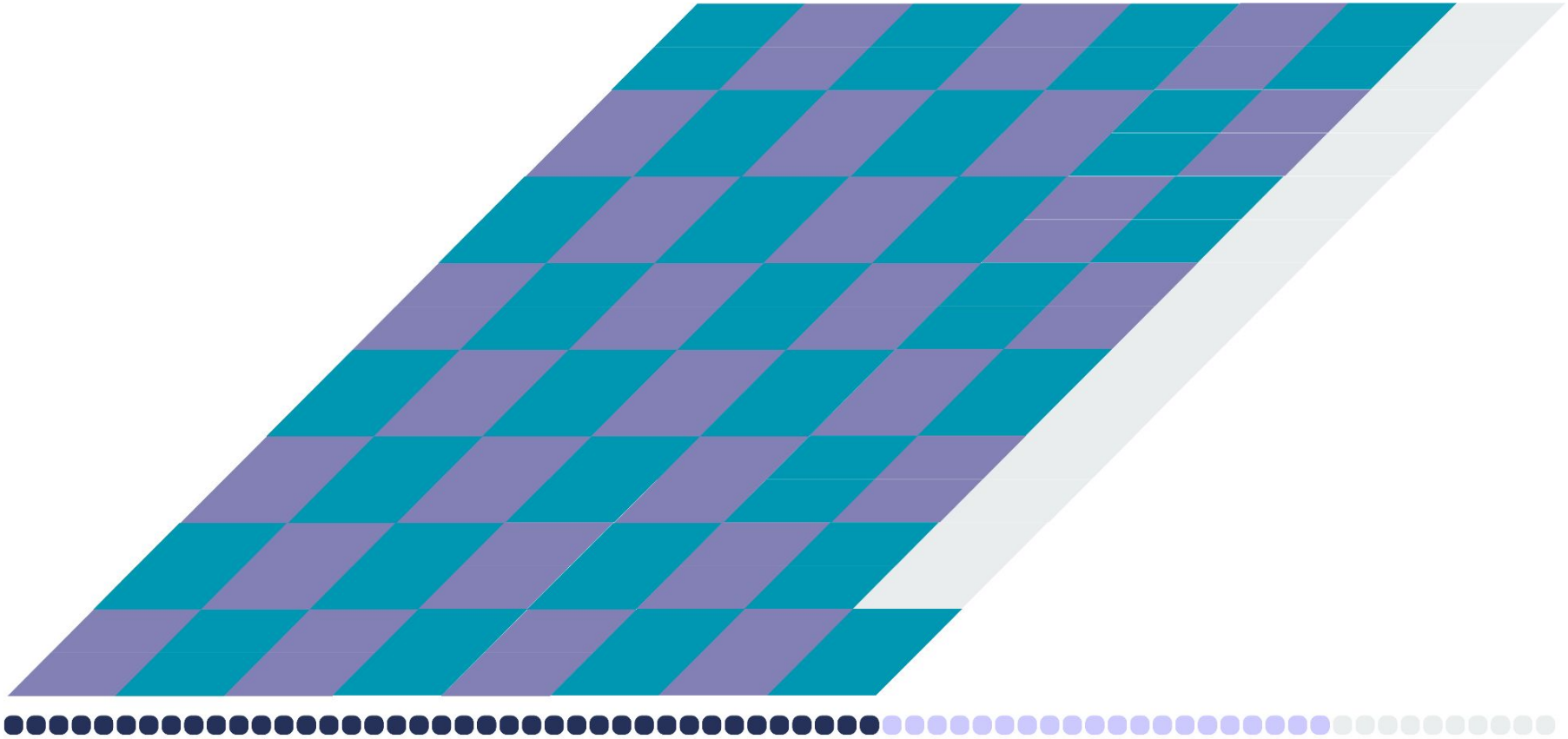


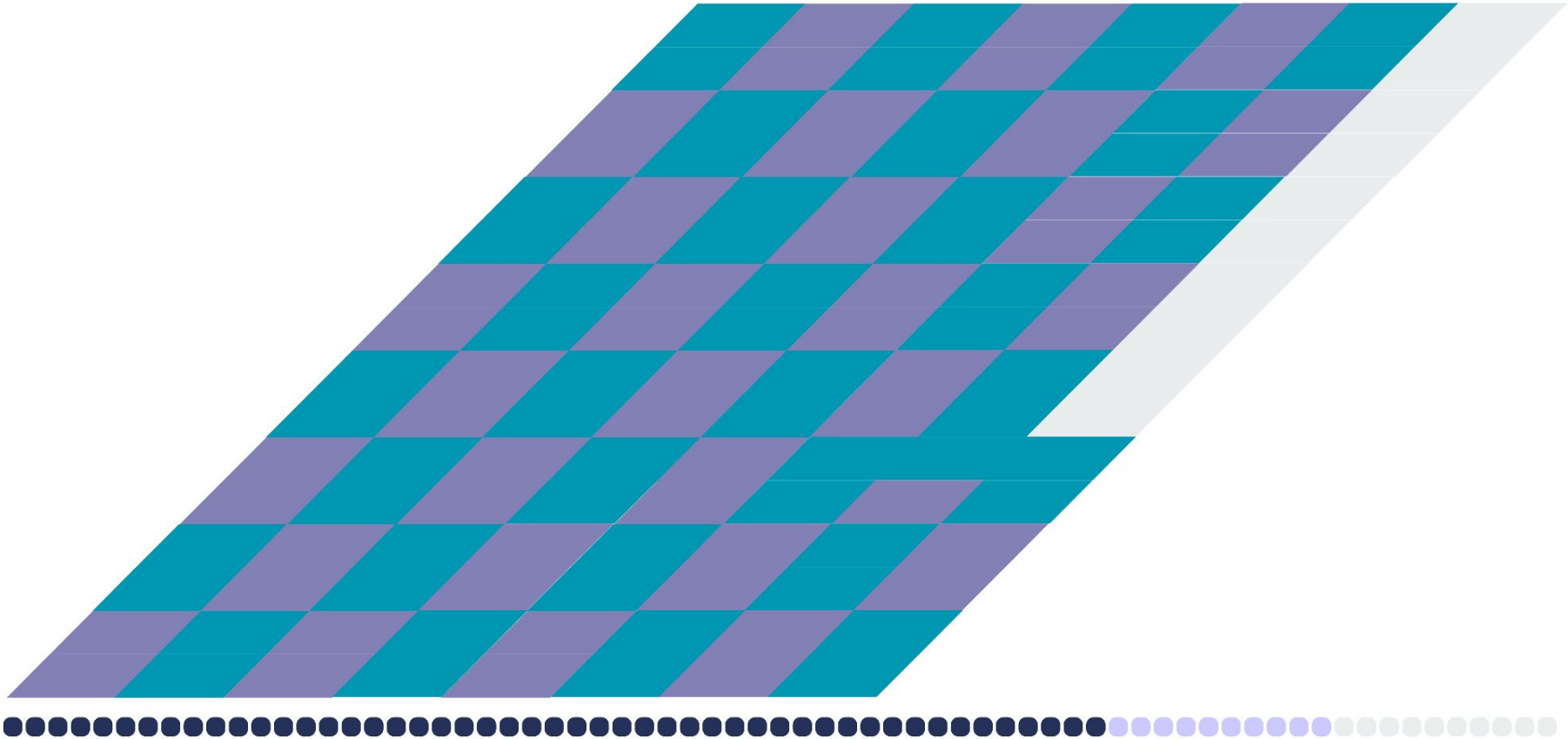


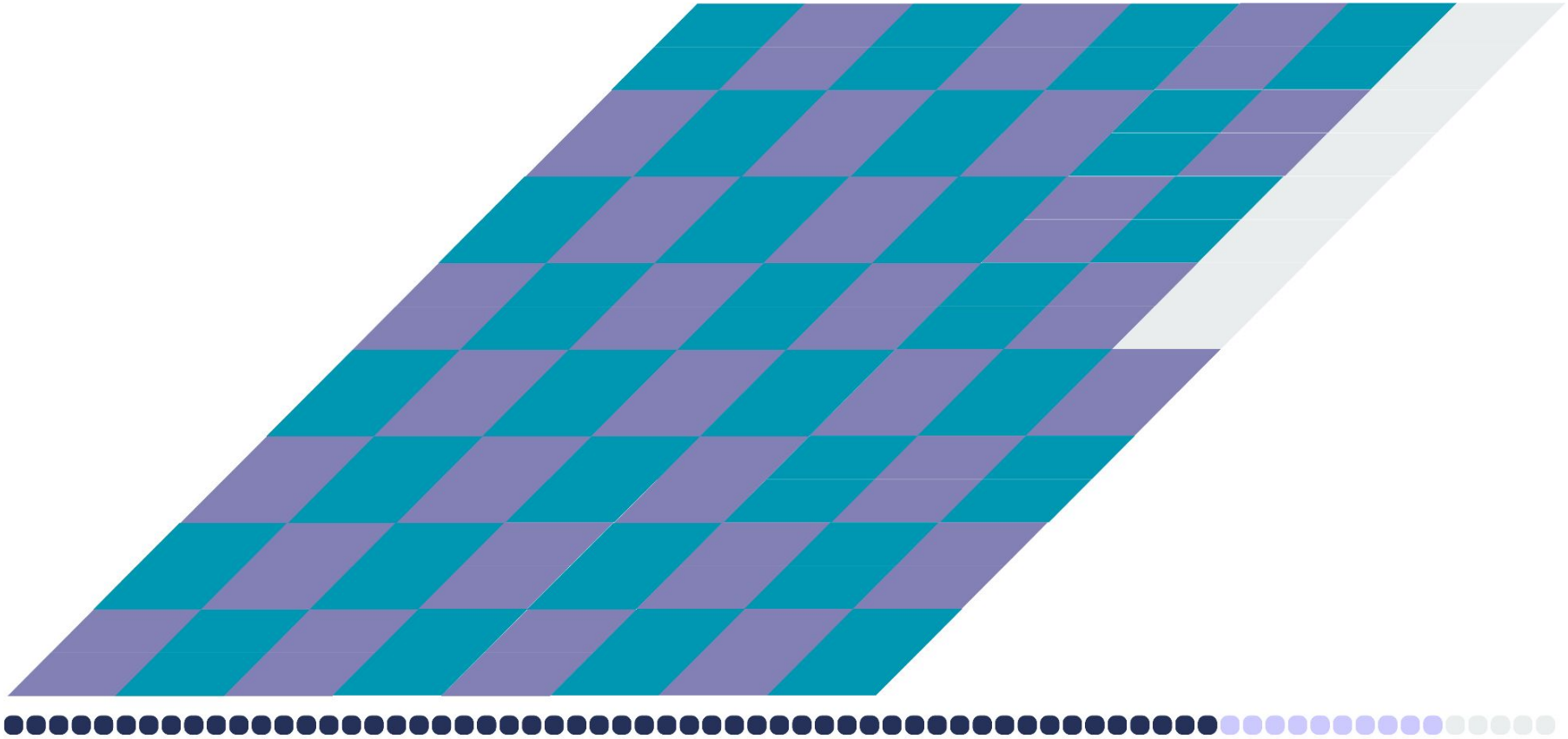


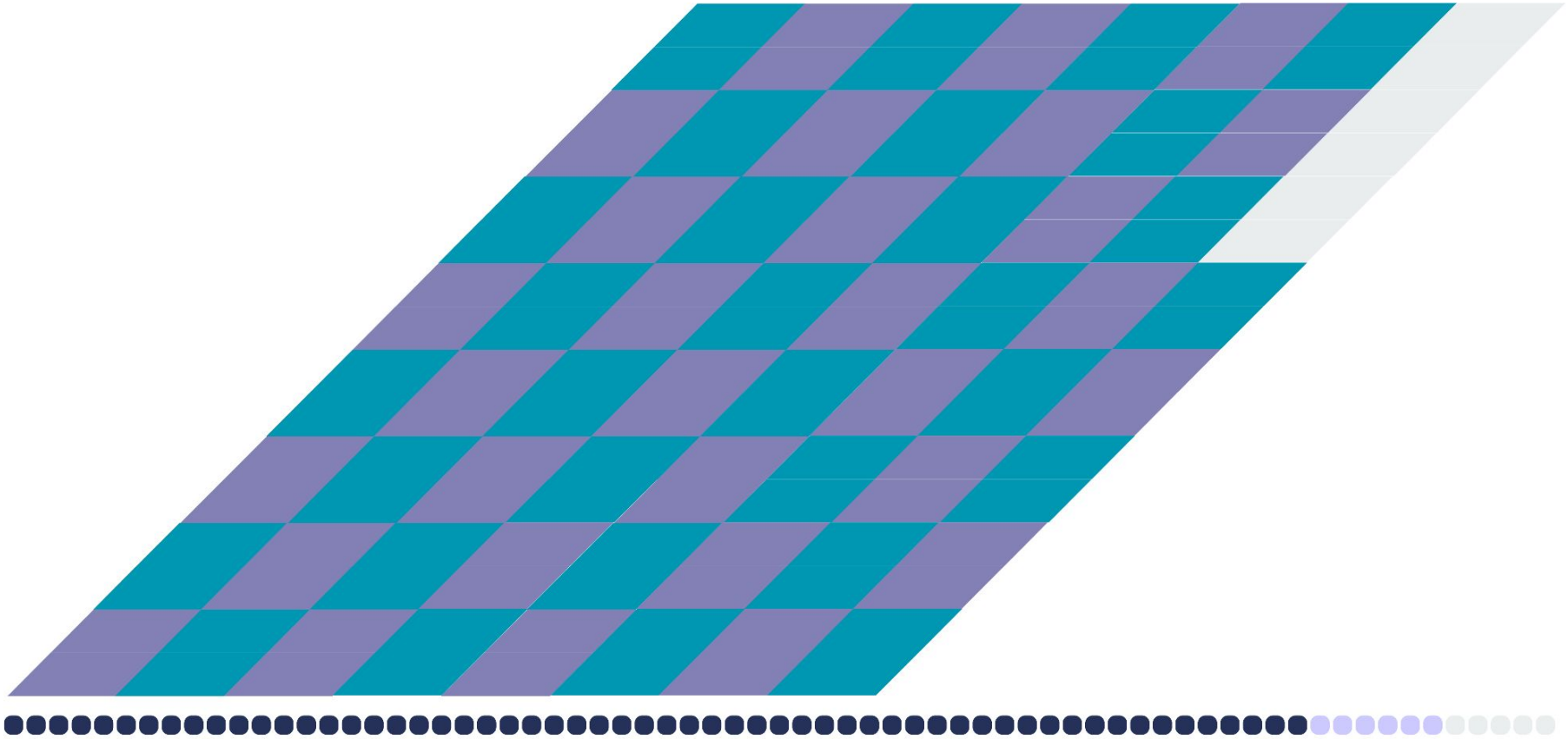


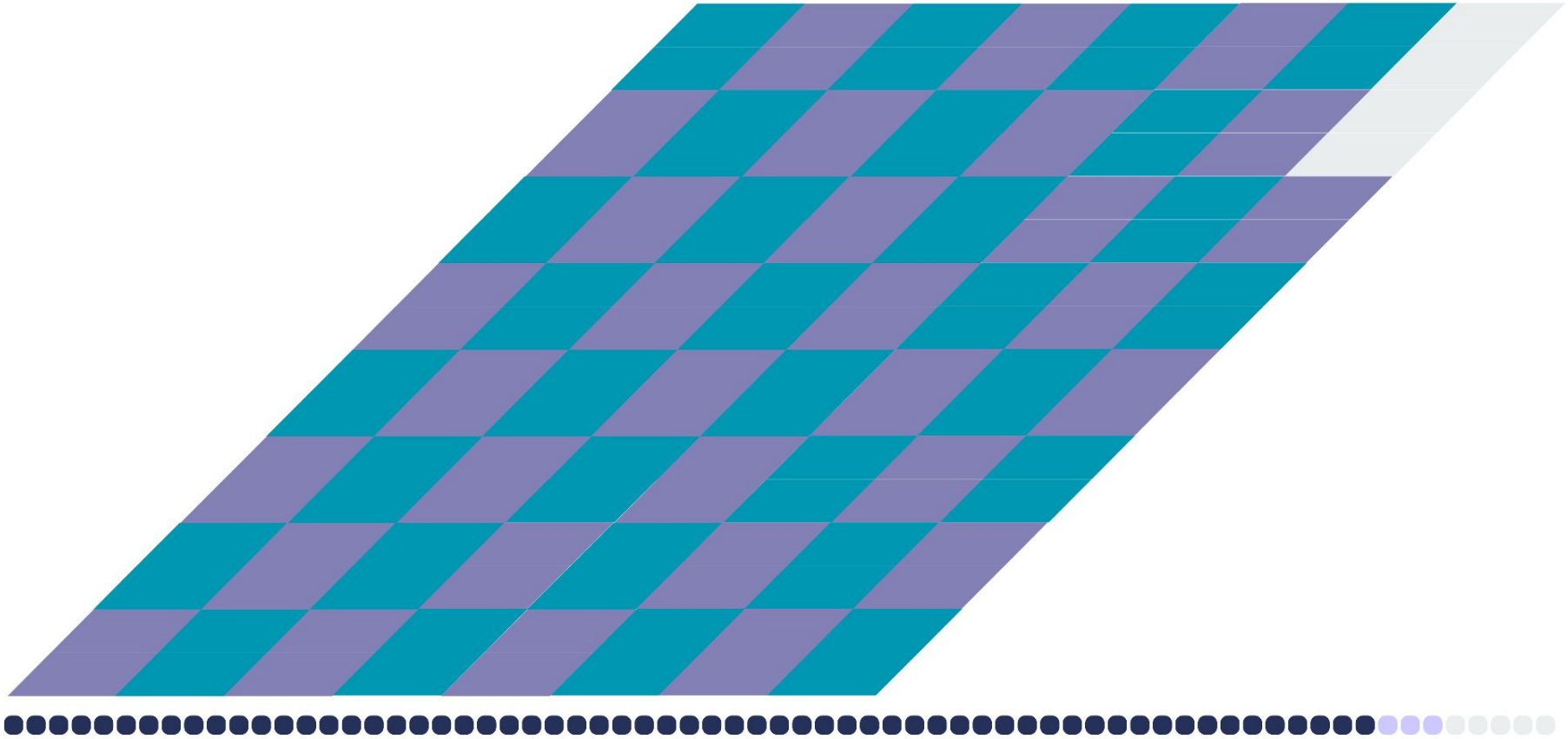


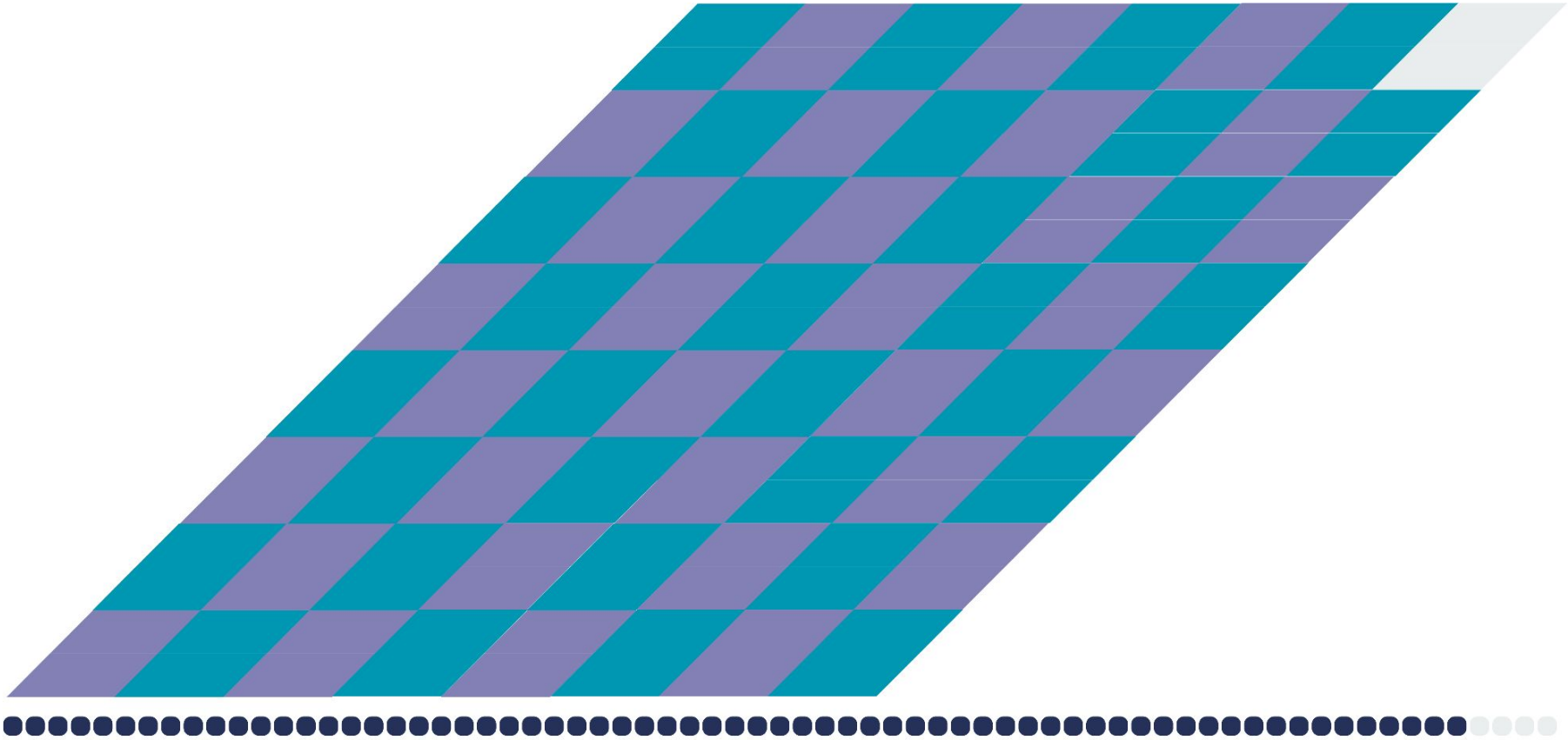


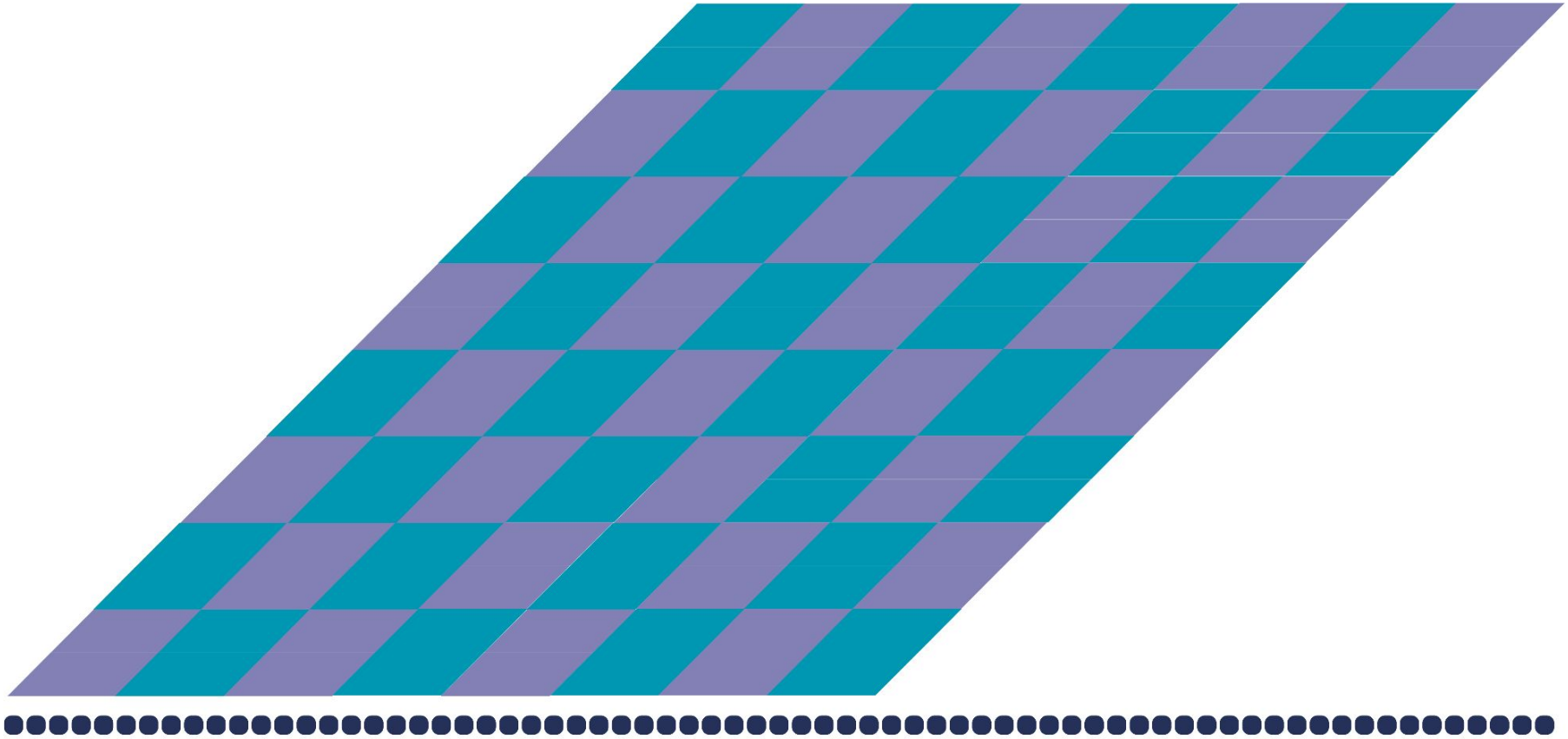








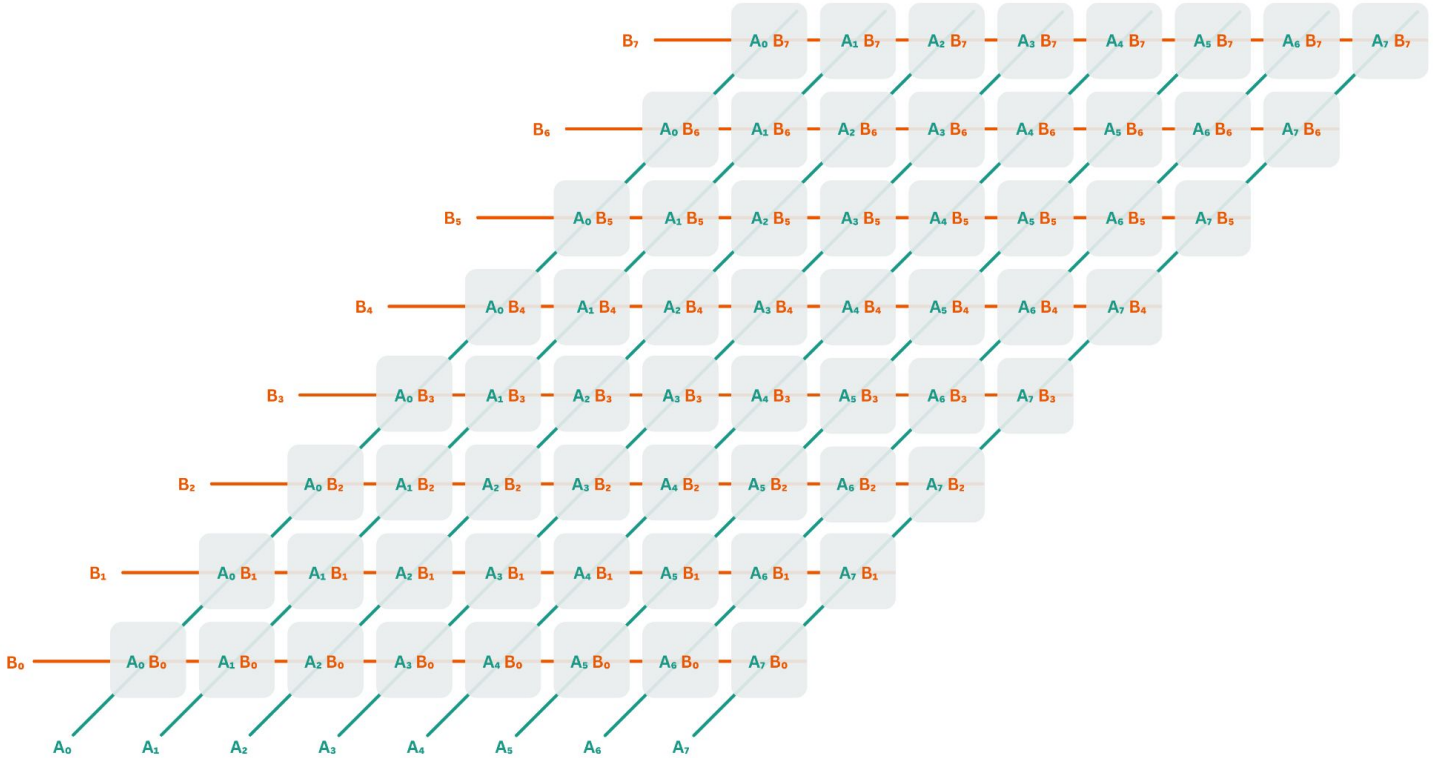




$$A(x) = A_0x^0 + A_1x^1 + A_2x^2 + A_3x^3 + A_4x^4 + A_5x^5 + A_6x^6 + A_7x^7$$

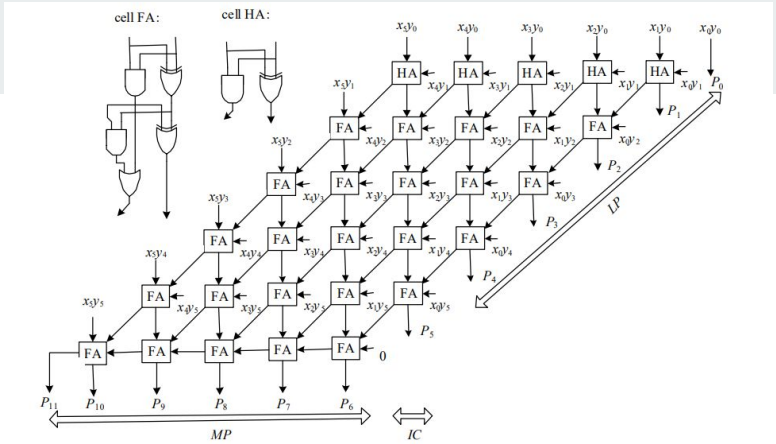
$$B(x) = B_0x^0 + B_1x^1 + B_2x^2 + B_3x^3 + B_4x^4 + B_5x^5 + B_6x^6 + B_7x^7$$

$$C(x) = C_0x^0 + C_1x^1 + C_2x^2 + C_3x^3 + C_4x^4 + C_5x^5 + C_6x^6 + C_7x^7 + C_8x^8 + C_9x^9 + C_{10}x^{10} + C_{11}x^{11} + C_{12}x^{12} + C_{13}x^{13} + C_{14}x^{14}$$

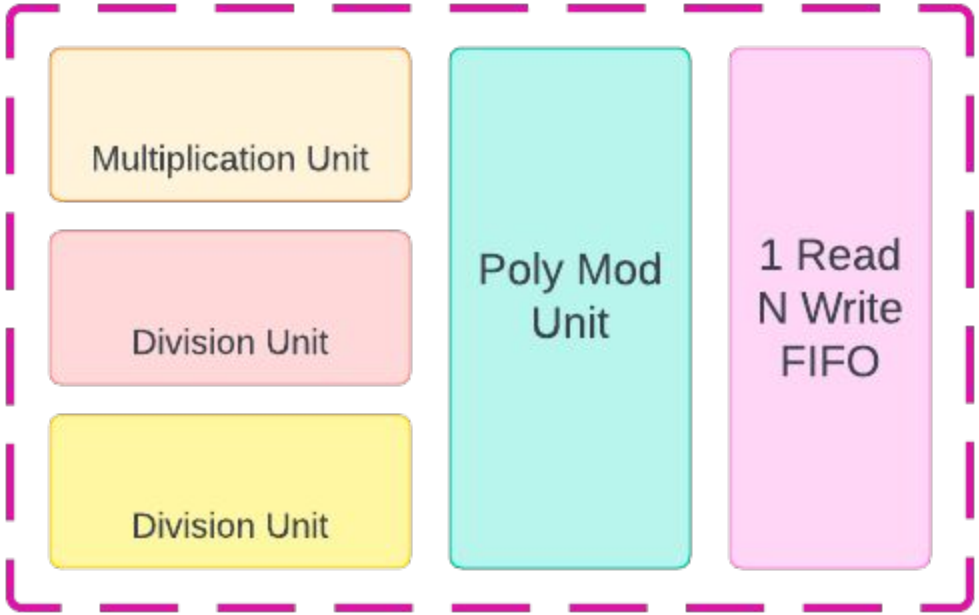


Bit Parallel Multiplier

(n x n → 2n) :: (64 x 64 → 128)



- Map add-shift multiplier technique to adder array
  - Partial products obtained in parallel → faster than add-shift
- Problem: more area needed
- Solution: we don't need all 2n bits → truncate multiplier
  - Redesign 1: output lowest n bits → halve the area
  - Redesign 2: parameterizable input bit widths → reduce area to only what's needed


Multiplication Unit
Division Unit
Division Unit
Poly Mod Unit
1 Read N Write FIFO

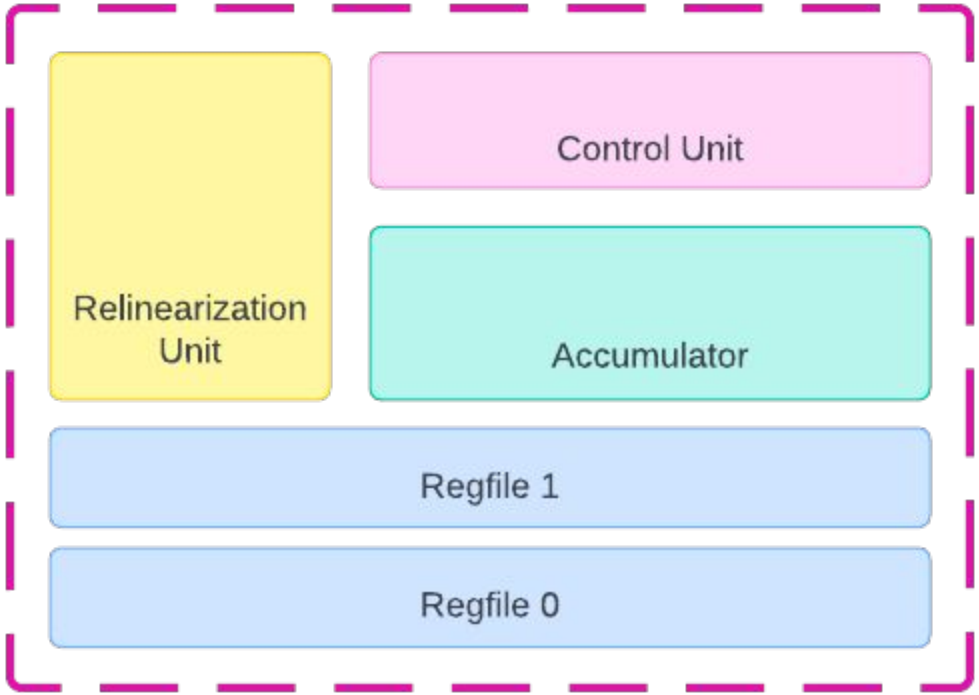
Recontextualization

$$\left[\left\lfloor \frac{t \cdot (\mathtt{ct}_1[0] \cdot \mathtt{ct}_2[0])}{q} \right\rceil\right]_q$$

1. Reduces magnitude of polynomial coefficients through sequence of multiplication, division, and modulus operations
2. Reduces a polynomial of degree *2*n* to a polynomial of degree *n*

$$C(x) = C_0x^0 + C_1x^1 + C_2x^2 + C_3x^3 + C_4x^4 + C_5x^5 + C_6x^6 + C_7x^7 + C_8x^8 + C_9x^9 + C_{10}x^{10} + C_{11}x^{11} + C_{12}x^{12} + C_{13}x^{13} + C_{14}x^{14}$$

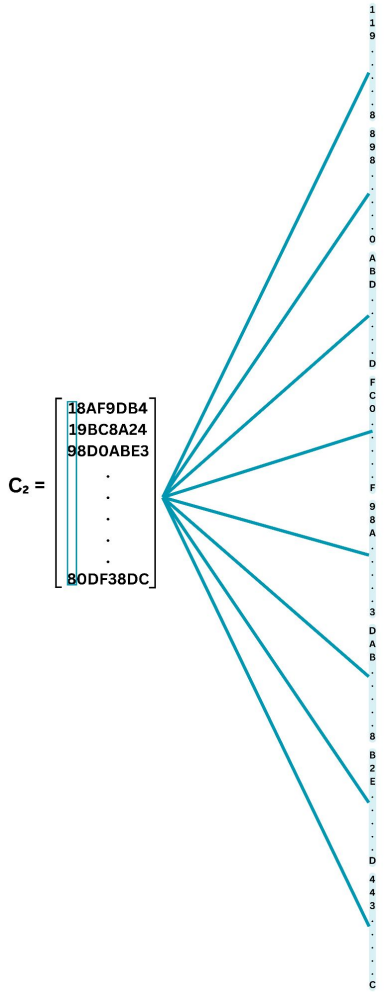
- Typically included in polynomial multiplication for code-based implementations
  - Separating this unit simplified out data flow
  - Makes tiled multiplication results easier to manage
- Polymult outputs are placed tile-by-tile into a FIFO queue
  - Removed one at a time for recontextualization

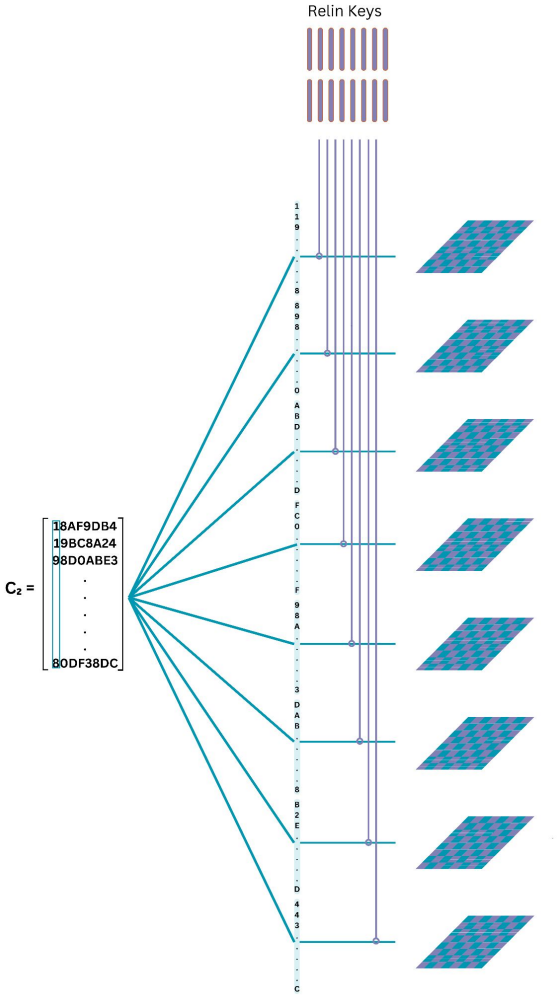

Relinearization Unit
Control Unit
Accumulator
Regfile 1
Regfile 0

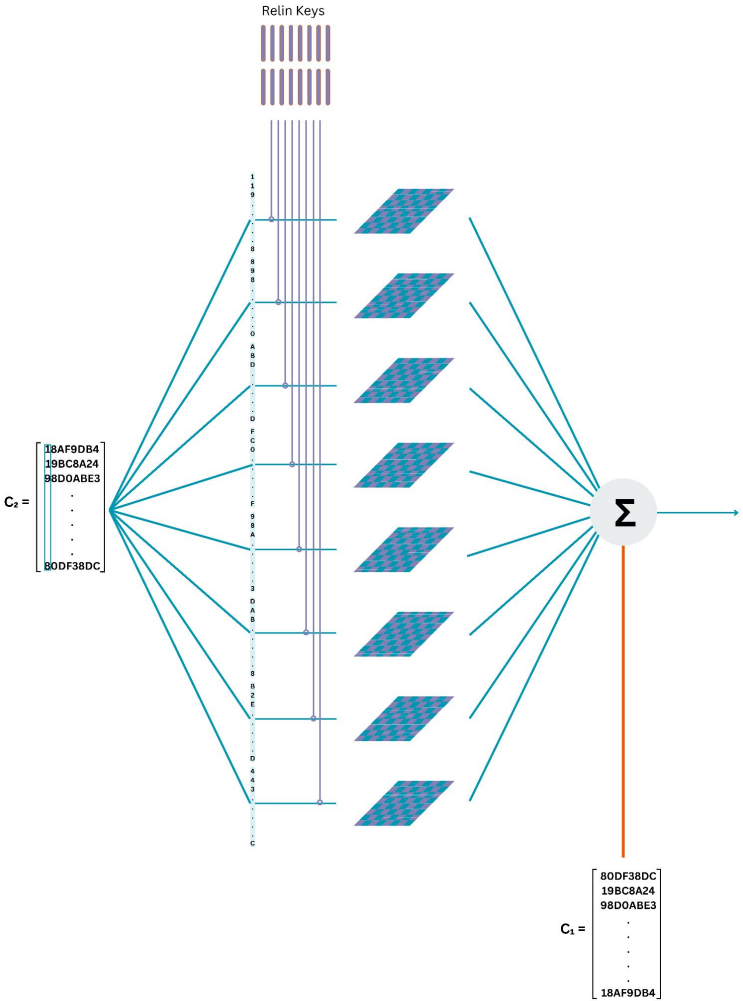
Relinearization

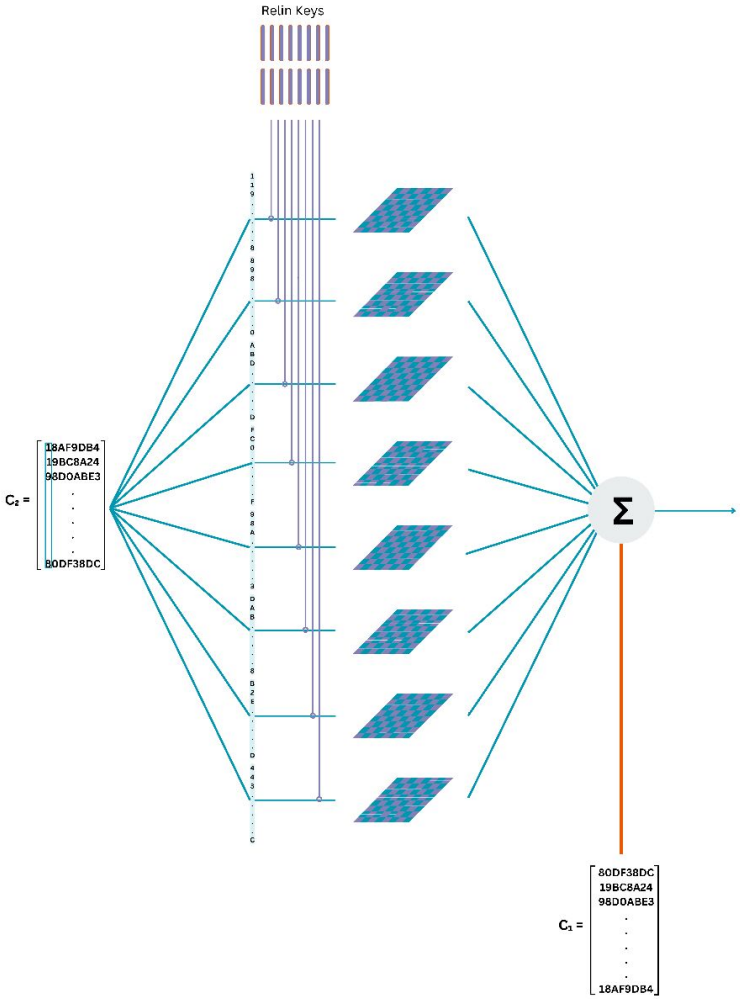
1. Calculates *base-T* decomposition of $\mathbf{c}_2$
2. Uses *relinearization keys* to compute $\mathbf{c}_{0,relin}$ and $\mathbf{c}_{1,relin}$ polynomials
3. Adds $\mathbf{c}_{0,relin}$ and $\mathbf{c}_{1,relin}$ to $\mathbf{c}_0$ and $\mathbf{c}_1$

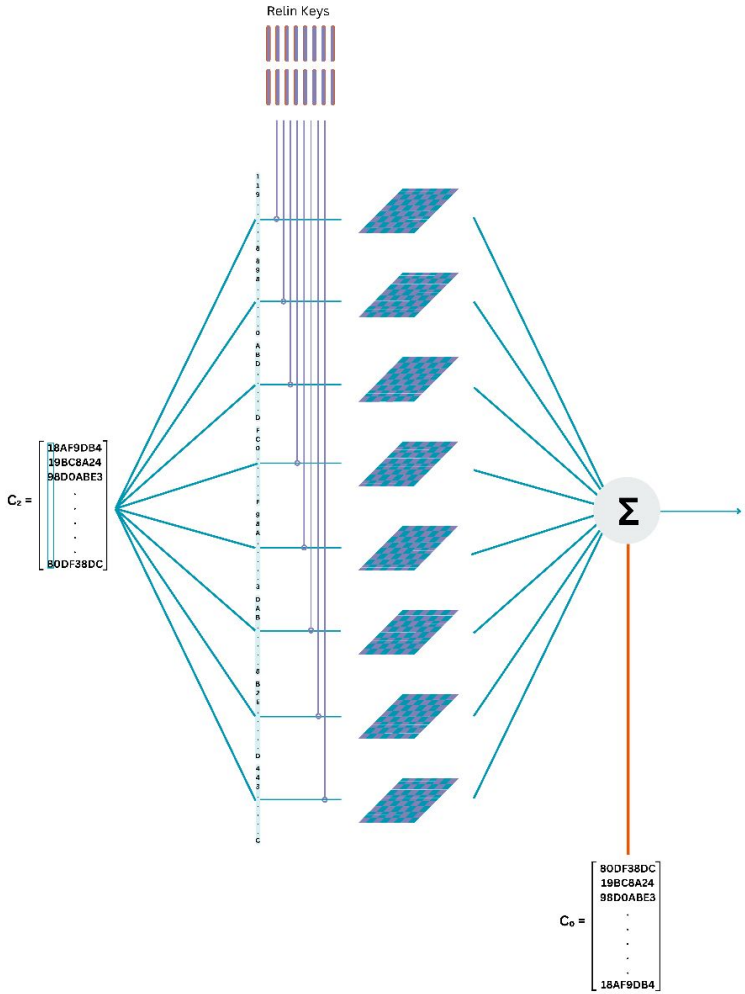
- Only $\mathbf{c}_2$ is fed into relinearization unit
  - Relinearization stage finishes before end of accelerator computation
- Outputs of relinearization unit are added to $\mathbf{c}_0$, $\mathbf{c}_1$ outputs of recontextualization stage
  - Final outputs are written back to CPU memory
- Small state based controller associated with relinearization
  - Manages output routing for recontextualization, relinearization units


C₂ =
18AF9DB4
19BC8A24
98D0ABE3
80DF38DC
119...8898...0ABD...DFC0...F98A...3DAB...8B2E...D443...C


Relin Keys
C_2 =
18AF9DB4
19BC8A24
98D0ABE3
80DF38DC


Relin Keys
C_2 =
18AF9DB4
19BC8A24
98D0ABE3
80DF38DC
Σ
C_1 =
80DF38DC
19BC8A24
98D0ABE3
18AF9DB4


Relin Keys
C_2 =
18AF9DB4
19BC8A24
98D0ABE3
80DF38DC
Σ
C_1 =
80DF38DC
19BC8A24
98D0ABE3
18AF9DB4


Relin Keys
C_2 =
18AF9DB4
19BC8A24
98D0ABE3
80DF38DC
Σ
C_0 =
80DF38DC
19BC8A24
98D0ABE3
18AF9DB4

Memory Interface

- Assume all data is pre-loaded into L1 cache
- Reads are managed by memory control wrapper
  - c11 -> c01 -> relin keys -> c10 -> c00
- Writes are queued as tiles become ready from accelerator
  - Prioritize reads over writes (finite number of reads, writes cannot create a bottleneck)

Metrics

Area

- Old: ~161k um^2
- New: ~479k um^2
- The ews machines would crash if we synthesized it all together, so we synthesized a smaller version and scaled it up
- Since we tiled our operations we managed to keep our area fairly low
- Area is just about 3x in size

Timing

- ~160Hz for both
- Our critical path was already a part of the cpu. Since we built arithmetic units we managed to meet our previous timing requirements

Speed

- Ran speed on 16x16 as ews would crash on 512x512
- Base cycles: 146,646
- New cycles: 8,780
- 16x speedup -> should scale even better with bigger

Demo

Pick 2 numbers...