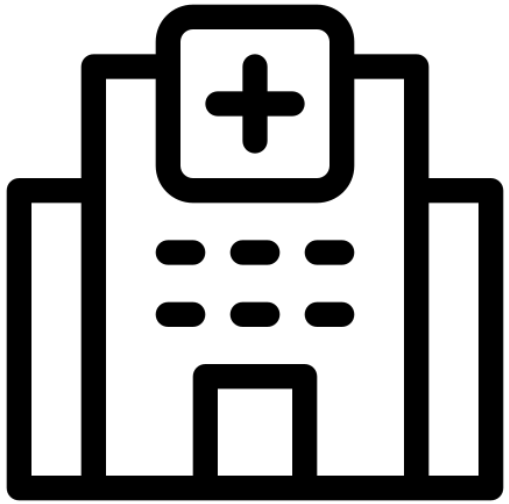
Bridging Usability and Performance: A Tensor Compiler for Autovectorizing Homomorphic Encryption

Edward Chen, Fraser Brown, Wenting Zheng



Why Privacy-Preserving Computation Matters?



Healthcare

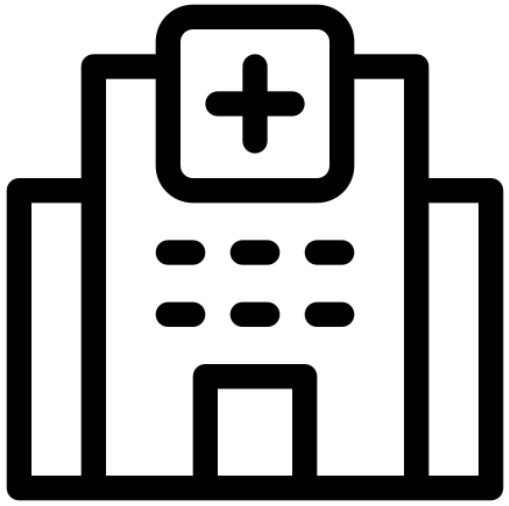


Finance



LLMs

Why Privacy-Preserving Computation Matters?



Healthcare



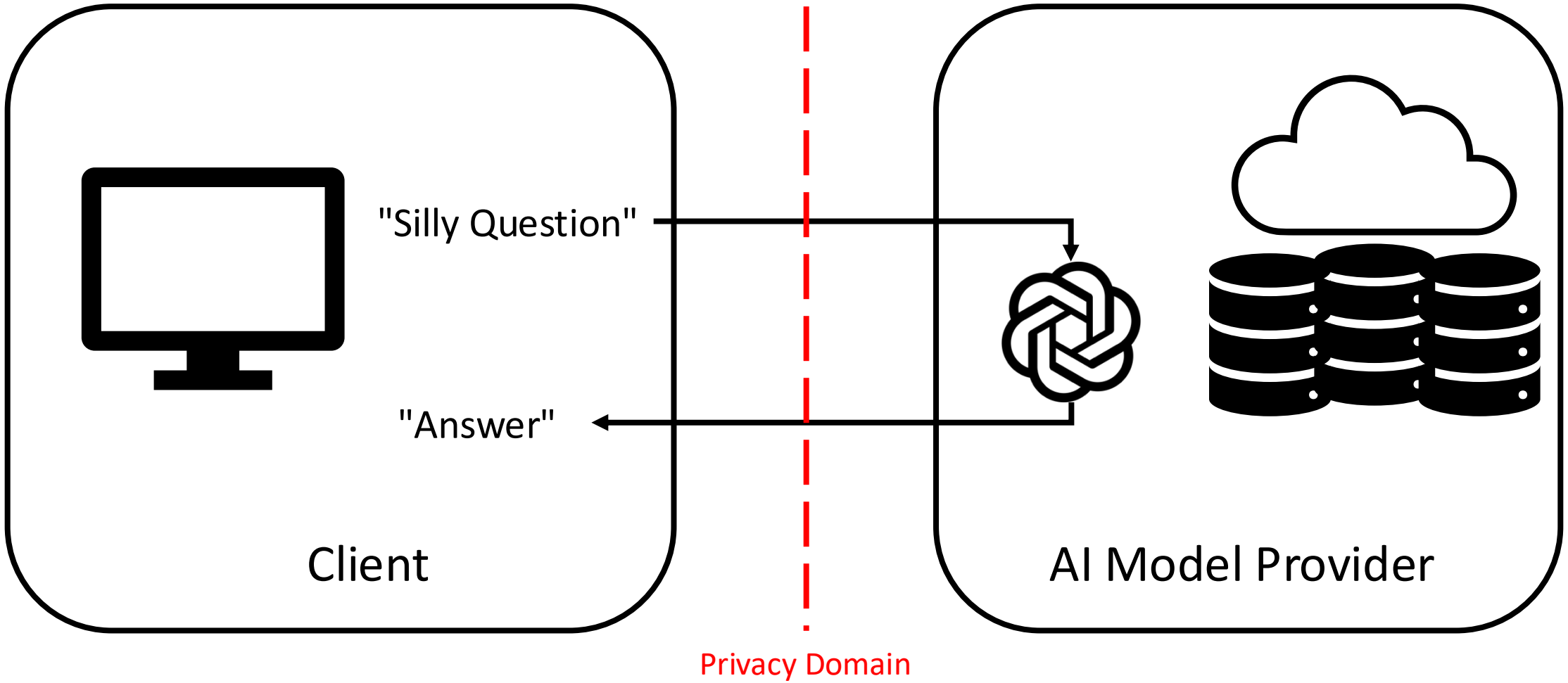
Finance



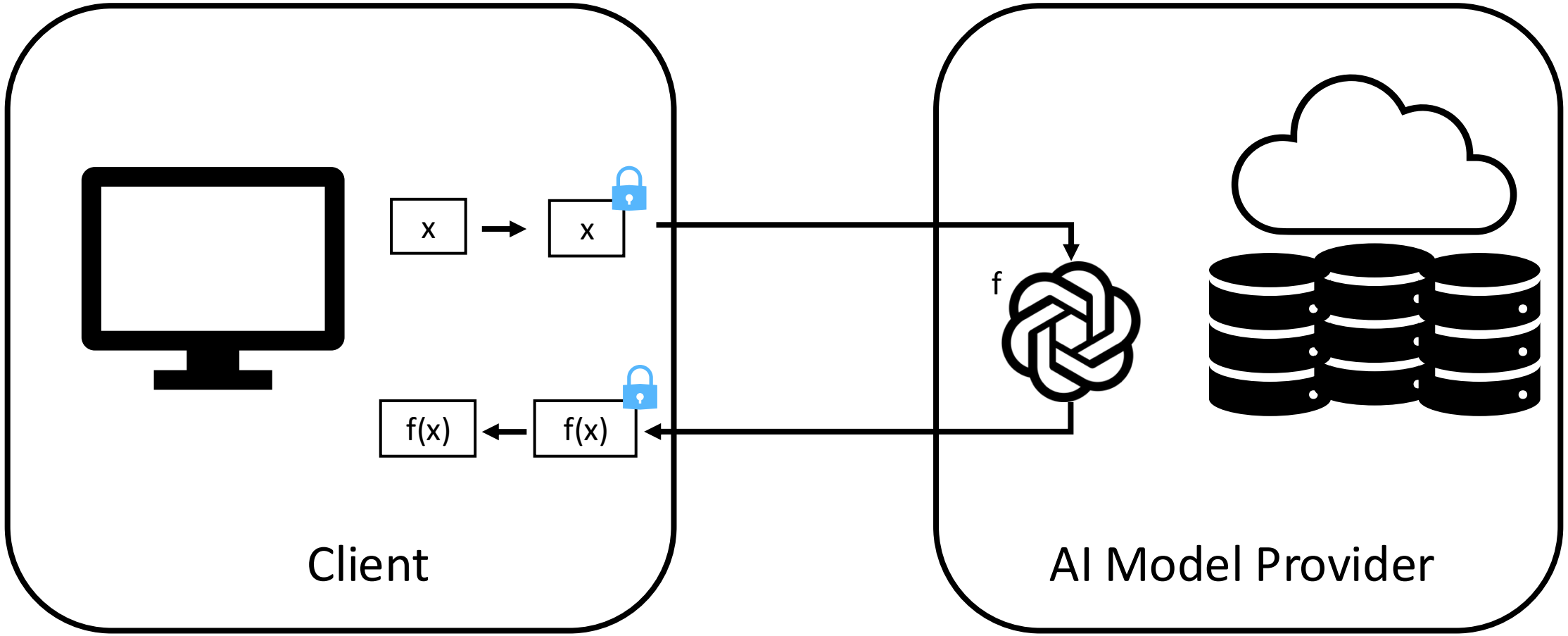
LLMs

Each domain has ***sensitive data*** that can/should ***not*** be openly shared

Threat Model: Privacy-Preserving Computation



Fully Homomorphic Encryption



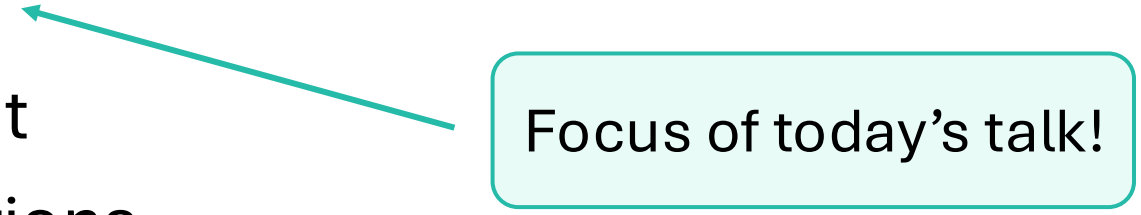
Programming in FHE is *Tedious* and *Brittle*

Programming in FHE is *Tedious* and *Brittle*

- Ciphertext packing
- Bootstrap placement
- Approximation functions
- Parameter selection

Programming in FHE is *Tedious* and *Brittle*

- Ciphertext packing
- Bootstrap placement
- Approximation functions
- Parameter selection



What is Packing?

- FHE uses a vector programming model
- Packing is organizing values into the slots of FHE vectors

Packing Maps Tensor Values Into *Layouts*

Plaintext:

00	01	02	03
10	11	12	13
20	21	22	23
30	31	32	33



Ciphertext:



Naïve Layout – 1 element per ciphertext

Plaintext:

00	01	02	03
10	11	12	13
20	21	22	23
30	31	32	33

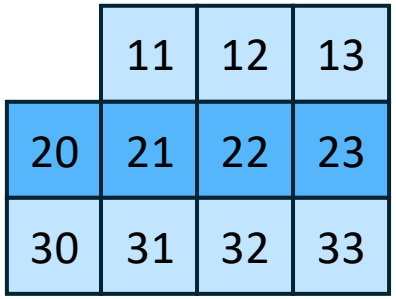


Ciphertext:

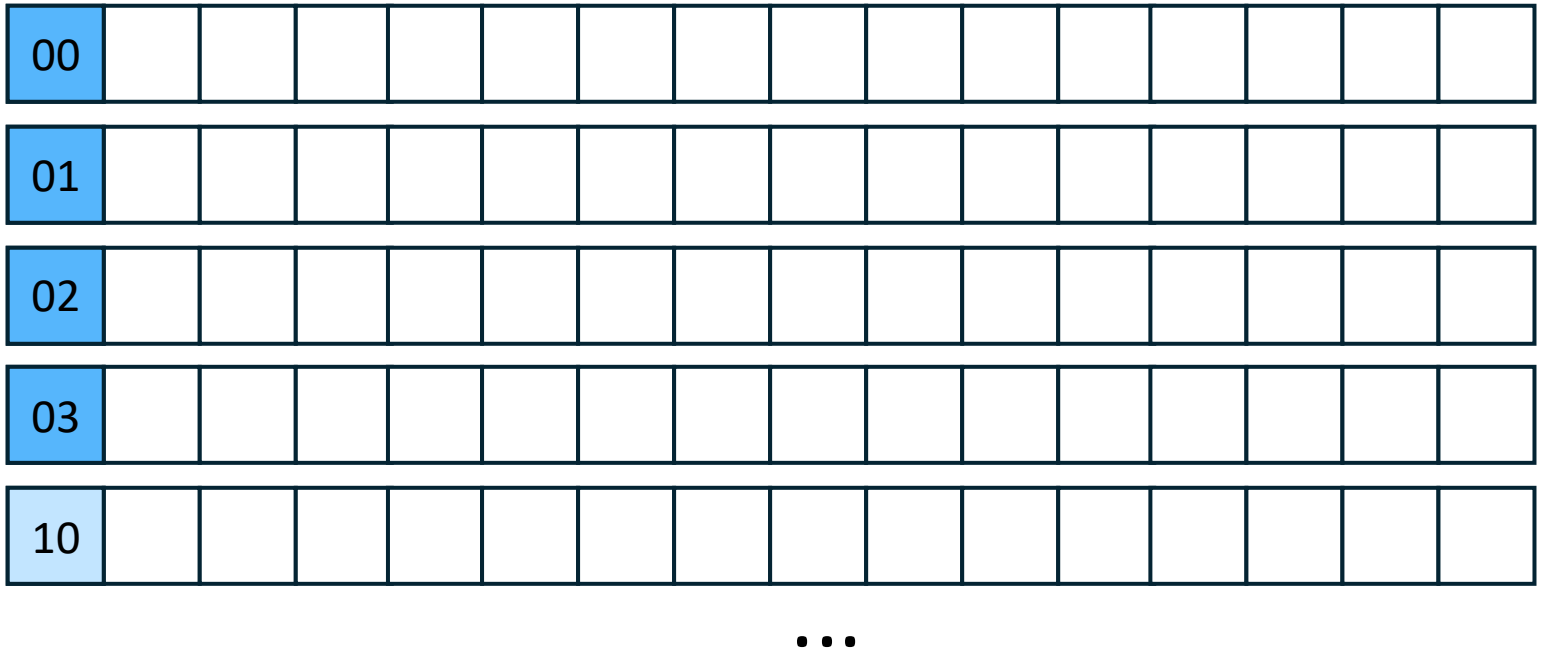


Naïve Layout – 1 element per ciphertext

Plaintext:

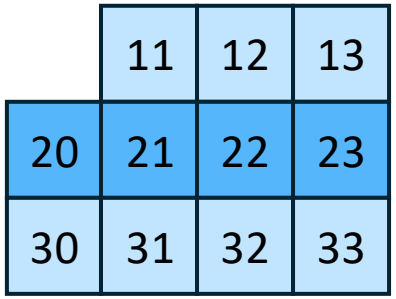


Ciphertext:

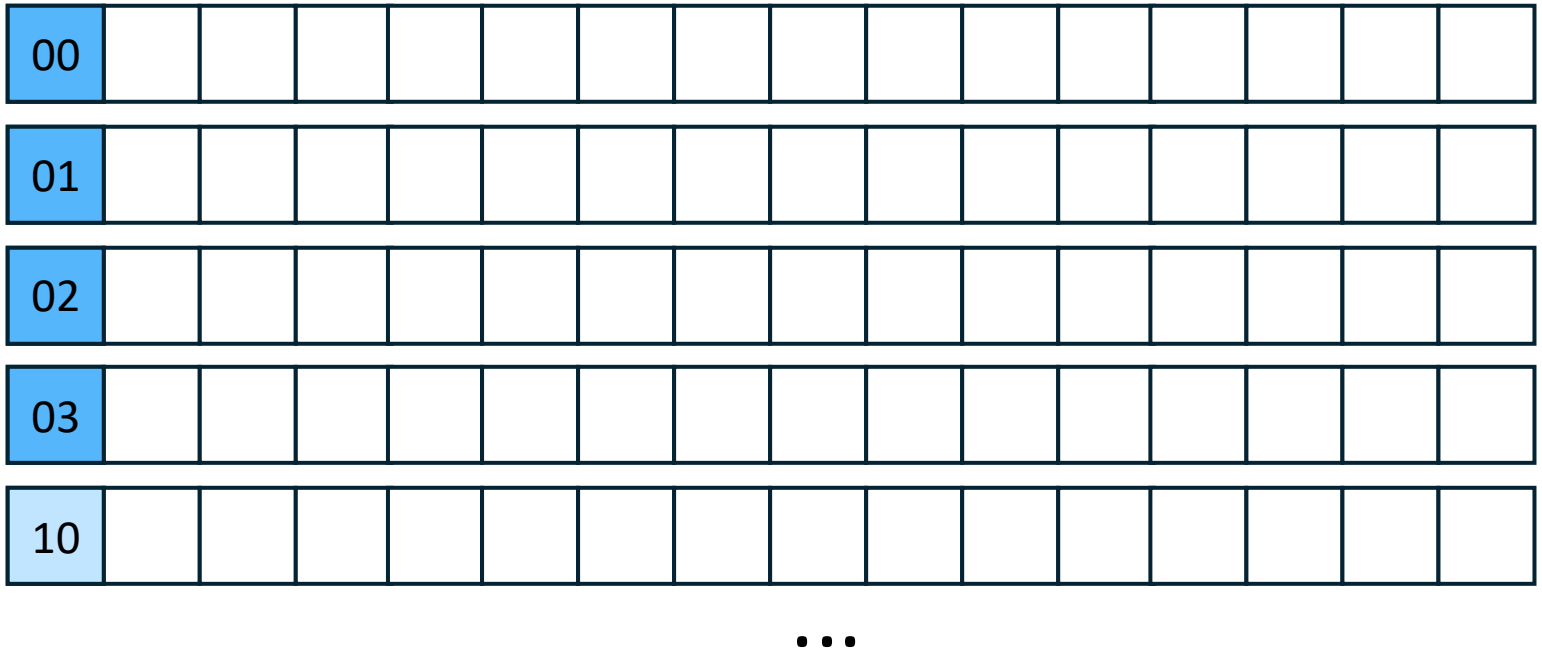


Naïve Layout – 1 element per ciphertext

Plaintext:



Ciphertext:





Low slot utilization → poor performance

Row-major Layout – Compact Packing

Plaintext:

00	01	02	03
10	11	12	13
20	21	22	23
30	31	32	33



Ciphertext:



Row-major Layout – Compact Packing

Plaintext:

10	11	12	13
20	21	22	23
30	31	32	33



Ciphertext:



Row-major Layout – Compact Packing

Plaintext:

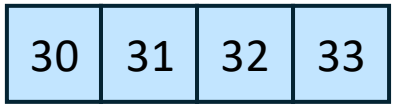
20	21	22	23
30	31	32	33

Ciphertext:

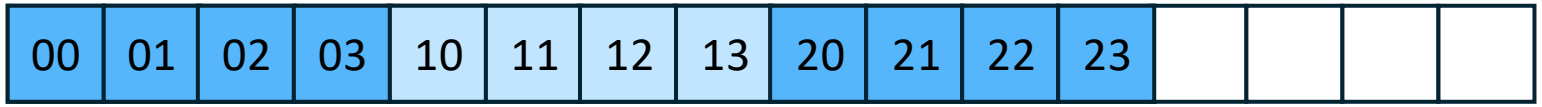
00	01	02	03	10	11	12	13								

Row-major Layout – Compact Packing

Plaintext:



Ciphertext:



Row-major Layout – Compact Packing

Plaintext:

Ciphertext:

00	01	02	03	10	11	12	13	20	21	22	23	30	31	32	33

Row-major Layout – Compact Packing

Plaintext:

00	01	02	03
10	11	12	13
20	21	22	23
30	31	32	33

Ciphertext:

00	01	02	03	10	11	12	13	20	21	22	23	30	31	32	33

Why Packing Makes Programming Difficult?

Why Packing Makes Programming Difficult?

Choosing ***bad*** layouts requires additional data movement

- e.g., layout conversions, summation within ciphertexts, ...

Why Packing Makes Programming Difficult?

Choosing *bad* layouts requires additional data movement

- e.g., layout conversions, summation within ciphertexts, ...

Data movement is *expensive* in FHE

Why Packing Makes Programming Difficult?

Choosing *bad* layouts requires additional data movement

- e.g., layout conversions, summation within ciphertexts, ...

Data movement is *expensive* in FHE

- No direct indexing in FHE vectors

Why Packing Makes Programming Difficult?

Choosing *bad* layouts requires additional data movement

- e.g., layout conversions, summation within ciphertexts, ...

Data movement is *expensive* in FHE

- No direct indexing in FHE vectors
- Limited operations:
  - SIMD addition (+)
  - SIMD multiplication (×)
  - Cyclic rotation (<<)

Why Packing Makes Programming Difficult?

Choosing *bad* layouts requires additional data movement

- e.g., layout conversions, summation within ciphertexts, ...

Data movement is *expensive* in FHE

- No direct indexing in FHE vectors
- Limited operations:
  - SIMD addition (+)
  - SIMD multiplication (×)
  - Cyclic rotation (<<)
- Rearranging values requires masking (×) & rotations (<<)

New Packing Research Each Year

[1] C. Juvekar, V. Vaikuntanathan, and A. Chandrakasan, "GAZELLE: A low latency framework for secure neural network inference," in *Proc. 27th USENIX Secur. Symp.*, Baltimore, MD, USA, 2018, pp. 1651–1669.

New Packing Research Each Year

[1] C. Juvekar, V. Vaikuntanathan, and A. Chandrakasan, "GAZELLE: A low latency framework for secure neural network inference," in *Proc. 27th USENIX Secur. Symp.*, Baltimore, MD, USA, 2018, pp. 1651–1669.

[2] X. Jiang, M. Kim, K. Lauter, and Y. Song, "Secure outsourced matrix computation and application to neural networks," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*, Toronto, ON, Canada, 2018, pp. 1209–1222, doi: 10.1145/3243734.3243837.

New Packing Research Each Year

[1] C. Juvekar, V. Vaikuntanathan, and A. Chandrakasan, "GAZELLE: A low latency framework for secure neural network inference," in *Proc. 27th USENIX Secur. Symp.*, Baltimore, MD, USA, 2018, pp. 1651–1669.

[2] X. Jiang, M. Kim, K. Lauter, and Y. Song, "Secure outsourced matrix computation and application to neural networks," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*, Toronto, ON, Canada, 2018, pp. 1209–1222, doi: 10.1145/3243734.3243837.

[3] A. Brutzkus, R. Gilad-Bachrach, and O. Elisha, "Low latency privacy preserving inference," in *Proc. 36th Int. Conf. Mach. Learn. (ICML)*, Long Beach, CA, USA, 2019, vol. 97, pp. 812–821.

New Packing Research Each Year

[1] C. Juvekar, V. Vaikuntanathan, and A. Chandrakasan, "GAZELLE: A low latency framework for secure neural network inference," in *Proc. 27th USENIX Secur. Symp.*, Baltimore, MD, USA, 2018, pp. 1651–1669.

[2] X. Jiang, M. Kim, K. Lauter, and Y. Song, "Secure outsourced matrix computation and application to neural networks," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*, Toronto, ON, Canada, 2018, pp. 1209–1222, doi: 10.1145/3243734.3243837.

[3] A. Brutzkus, R. Gilad-Bachrach, and O. Elisha, "Low latency privacy preserving inference," in *Proc. 36th Int. Conf. Mach. Learn. (ICML)*, Long Beach, CA, USA, 2019, vol. 97, pp. 812–821.

[4] E. Lee, J.-W. Lee, J. Lee, Y.-S. Kim, Y. Kim, J.-S. No, and W. Choi, "Low-complexity deep convolutional neural networks on fully homomorphic encryption using multiplexed parallel convolutions," in *Proc. 39th Int. Conf. Mach. Learn. (ICML)*, Baltimore, MD, USA, 2022, vol. 162, pp. 12403–12422.

New Packing Research Each Year

[1] C. Juvekar, V. Vaikuntanathan, and A. Chandrakasan, "GAZELLE: A low latency framework for secure neural network inference," in *Proc. 27th USENIX Secur. Symp.*, Baltimore, MD, USA, 2018, pp. 1651–1669.

[2] X. Jiang, M. Kim, K. Lauter, and Y. Song, "Secure outsourced matrix computation and application to neural networks," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*, Toronto, ON, Canada, 2018, pp. 1209–1222, doi: 10.1145/3243734.3243837.

[3] A. Brutzkus, R. Gilad-Bachrach, and O. Elisha, "Low latency privacy preserving inference," in *Proc. 36th Int. Conf. Mach. Learn. (ICML)*, Long Beach, CA, USA, 2019, vol. 97, pp. 812–821.

[4] E. Lee, J.-W. Lee, J. Lee, Y.-S. Kim, Y. Kim, J.-S. No, and W. Choi, "Low-complexity deep convolutional neural networks on fully homomorphic encryption using multiplexed parallel convolutions," in *Proc. 39th Int. Conf. Mach. Learn. (ICML)*, Baltimore, MD, USA, 2022, vol. 162, pp. 12403–12422.

[5] D. Kim, J. Park, J. Kim, S. Kim, and J. H. Ahn, "HyPHEN: A hybrid packing method and its optimizations for homomorphic encryption-based neural networks," *IEEE Access*, vol. 12, pp. 3024–3038, 2024, doi: 10.1109/ACCESS.2023.3348170.

New Packing Research Each Year

[1] C. Juvekar, V. Vaikuntanathan, and A. Chandrakasan, "GAZELLE: A low latency framework for secure neural network inference," in *Proc. 27th USENIX Secur. Symp.*, Baltimore, MD, USA, 2018, pp. 1651–1669.

[2] X. Jiang, M. Kim, K. Lauter, and Y. Song, "Secure outsourced matrix computation and application to neural networks," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*, Toronto, ON, Canada, 2018, pp. 1209–1222, doi: 10.1145/3243734.3243837.

[3] A. Brutzkus, R. Gilad-Bachrach, and O. Elisha, "Low latency privacy preserving inference," in *Proc. 36th Int. Conf. Mach. Learn. (ICML)*, Long Beach, CA, USA, 2019, vol. 97, pp. 812–821.

[4] E. Lee, J.-W. Lee, J. Lee, Y.-S. Kim, Y. Kim, J.-S. No, and W. Choi, "Low-complexity deep convolutional neural networks on fully homomorphic encryption using multiplexed parallel convolutions," in *Proc. 39th Int. Conf. Mach. Learn. (ICML)*, Baltimore, MD, USA, 2022, vol. 162, pp. 12403–12422.

[5] D. Kim, J. Park, J. Kim, S. Kim, and J. H. Ahn, "HyPHEN: A hybrid packing method and its optimizations for homomorphic encryption-based neural networks," *IEEE Access*, vol. 12, pp. 3024–3038, 2024, doi: 10.1109/ACCESS.2023.3348170.

[6] Q. Pang, J. Zhu, H. Möllering, W. Zheng, and T. Schneider, "BOLT: Privacy-preserving, accurate and efficient inference for transformers," in *Proc. IEEE Symp. Secur. Privacy (SP)*, San Francisco, CA, USA, 2024, pp. 4753–4771, doi: 10.1109/SP54263.2024.00130.

New Packing Research Each Year

[1] C. Juvekar, V. Vaikuntanathan, and A. Chandrakasan, "GAZELLE: A low latency framework for secure neural network inference," in *Proc. 27th USENIX Secur. Symp.*, Baltimore, MD, USA, 2018, pp. 1651–1669.

[2] X. Jiang, M. Kim, K. Lauter, and Y. Song, "Secure outsourced matrix computation and application to neural networks," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*, Toronto, ON, Canada, 2018, pp. 1209–1222, doi: 10.1145/3243734.3243837.

[3] A. Brutzkus, R. Gilad-Bachrach, and O. Elisha, "Low latency privacy preserving inference," in *Proc. 36th Int. Conf. Mach. Learn. (ICML)*, Long Beach, CA, USA, 2019, vol. 97, pp. 812–821.

[4] E. Lee, J.-W. Lee, J. Lee, Y.-S. Kim, Y. Kim, J.-S. No, and W. Choi, "Low-complexity deep convolutional neural networks on fully homomorphic encryption using multiplexed parallel convolutions," in *Proc. 39th Int. Conf. Mach. Learn. (ICML)*, Baltimore, MD, USA, 2022, vol. 162, pp. 12403–12422.

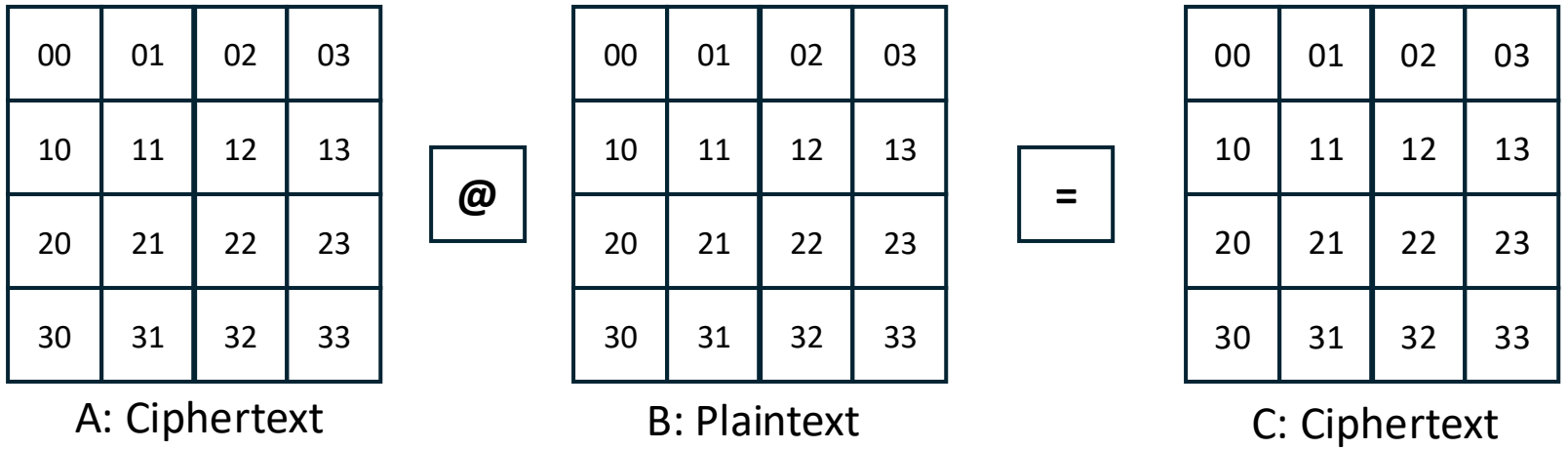
[5] D. Kim, J. Park, J. Kim, S. Kim, and J. H. Ahn, "HyPHEN: A hybrid packing method and its optimizations for homomorphic encryption-based neural networks," *IEEE Access*, vol. 12, pp. 3024–3038, 2024, doi: 10.1109/ACCESS.2023.3348170.

[6] Q. Pang, J. Zhu, H. Möllering, W. Zheng, and T. Schneider, "BOLT: Privacy-preserving, accurate and efficient inference for transformers," in *Proc. IEEE Symp. Secur. Privacy (SP)*, San Francisco, CA, USA, 2024, pp. 4753–4771, doi: 10.1109/SP54263.2024.00130.

[7] A. Ebel, K. Garimella, and B. Reagen, "Orion: A fully homomorphic encryption framework for deep learning," in *Proc. 30th ACM Int. Conf. Archit. Support Program. Lang. Oper. Syst. (ASPLOS), Volume 2*, Rotterdam, Netherlands, 2025, art. no. 16, doi: 10.1145/3676641.3716008.

…

Challenges Walkthrough: Matrix Multiplication



A: Ciphertext B: Plaintext C: Ciphertext

1. Find Layouts For Each Input Tensor

00	01	02	03
10	11	12	13
20	21	22	23
30	31	32	33

A: Ciphertext

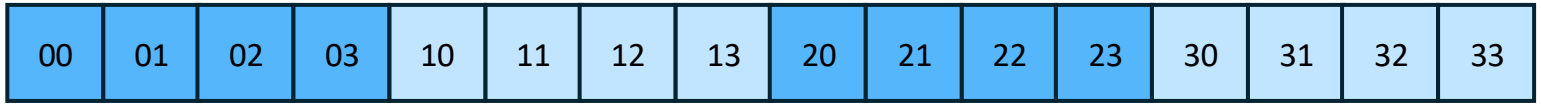
1. Find Layouts For Each Input Tensor

00	01	02	03
10	11	12	13
20	21	22	23
30	31	32	33

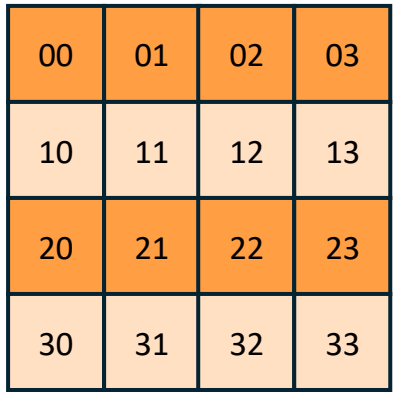
A: Ciphertext

1. Find Layouts For Each Input Tensor

A: Row-major layout

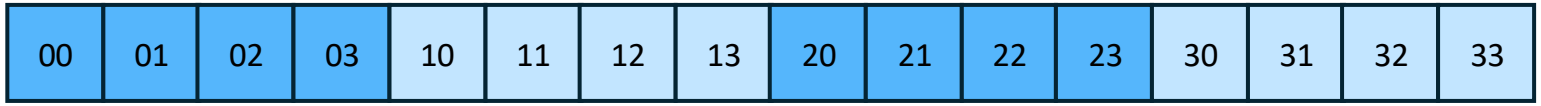


1. Find Layouts For Each Input Tensor

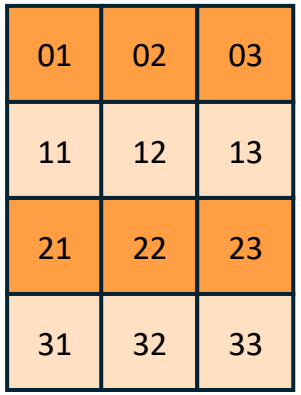


A: Ciphertext

A: Row-major layout

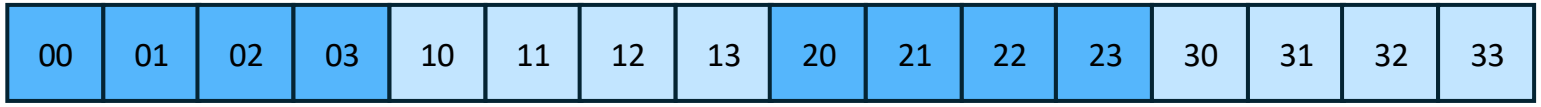


1. Find Layouts For Each Input Tensor

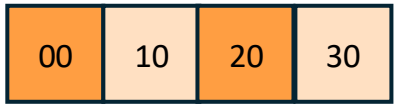


A: Ciphertext

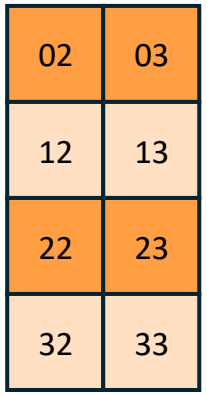
A: Row-major layout



A: Column-major layout

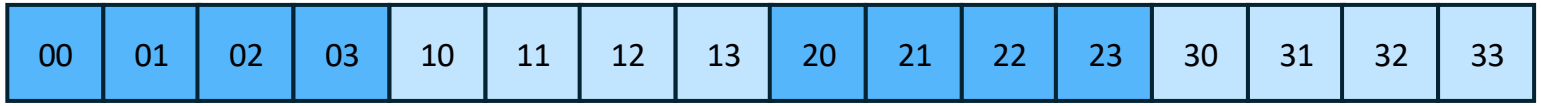


1. Find Layouts For Each Input Tensor

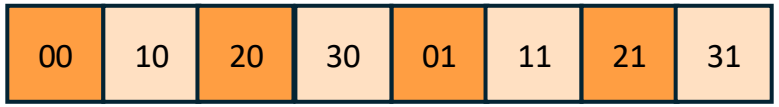


A: Ciphertext

A: Row-major layout

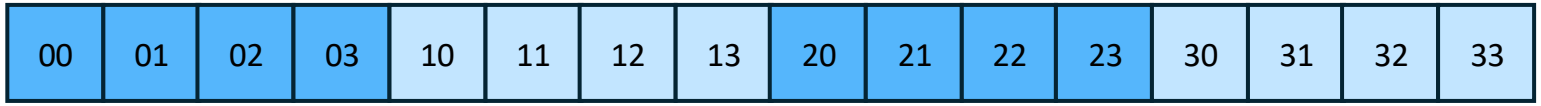


A: Column-major layout

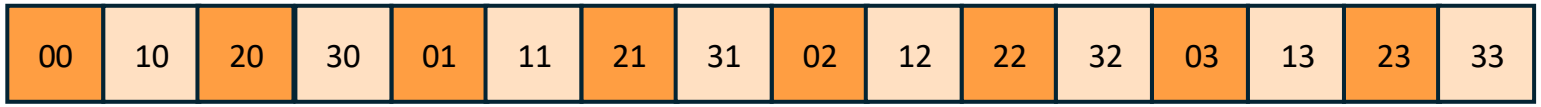


1. Find Layouts For Each Input Tensor

A: Row-major layout



A: Column-major layout



1. Find Layouts For Each Input Tensor

00	01	02	03
10	11	12	13
20	21	22	23
30	31	32	33

A: Ciphertext

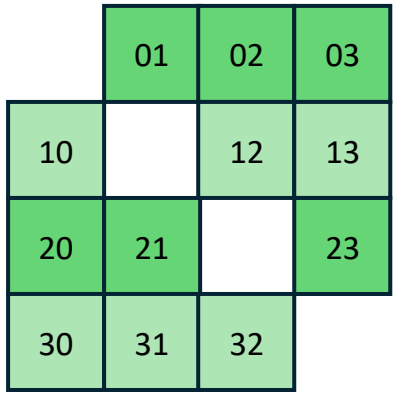
A: Row-major layout

00	01	02	03	10	11	12	13	20	21	22	23	30	31	32	33

A: Column-major layout

00	10	20	30	01	11	21	31	02	12	22	32	03	13	23	33

1. Find Layouts For Each Input Tensor



A: Ciphertext

A: Row-major layout

00	01	02	03	10	11	12	13	20	21	22	23	30	31	32	33

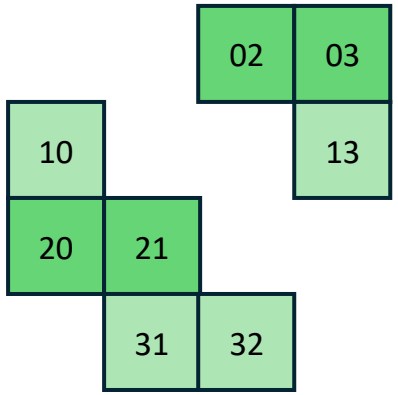
A: Column-major layout

00	10	20	30	01	11	21	31	02	12	22	32	03	13	23	33

A: Diagonal-major layout

00	11	22	33

1. Find Layouts For Each Input Tensor



A: Ciphertext

A: Row-major layout

00	01	02	03	10	11	12	13	20	21	22	23	30	31	32	33

A: Column-major layout

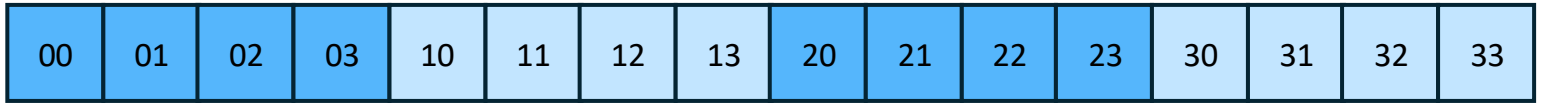
00	10	20	30	01	11	21	31	02	12	22	32	03	13	23	33

A: Diagonal-major layout

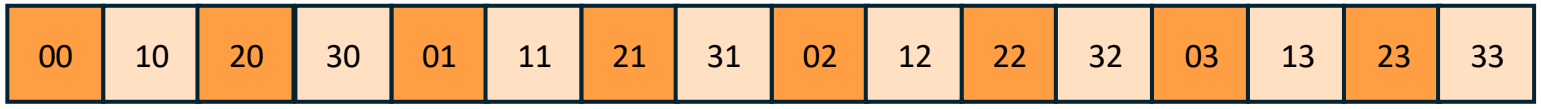
00	11	22	33	01	12	23	30

1. Find Layouts For Each Input Tensor

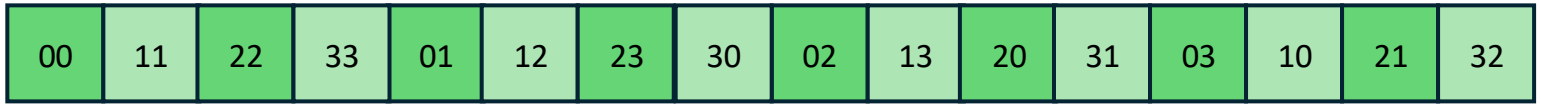
A: Row-major layout



A: Column-major layout



A: Diagonal-major layout



1. Find Layouts For Each Input Tensor

00	01	02	03
10	11	12	13
20	21	22	23
30	31	32	33

A: Ciphertext

A: Row-major layout

00	01	02	03	10	11	12	13	20	21	22	23	30	31	32	33

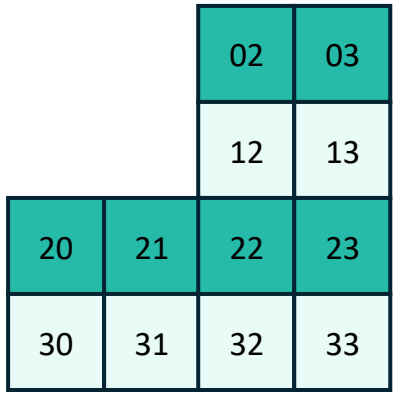
A: Column-major layout

00	10	20	30	01	11	21	31	02	12	22	32	03	13	23	33

A: Diagonal-major layout

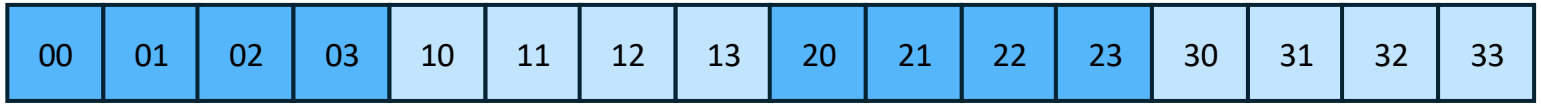
00	11	22	33	01	12	23	30	02	13	20	31	03	10	21	32

1. Find Layouts For Each Input Tensor

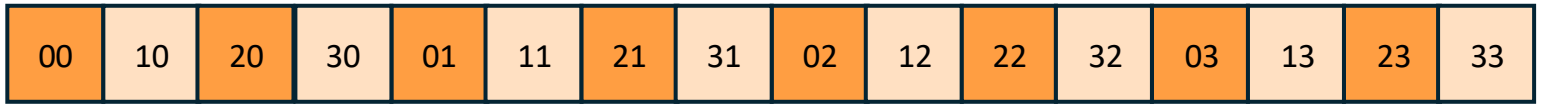


A: Ciphertext

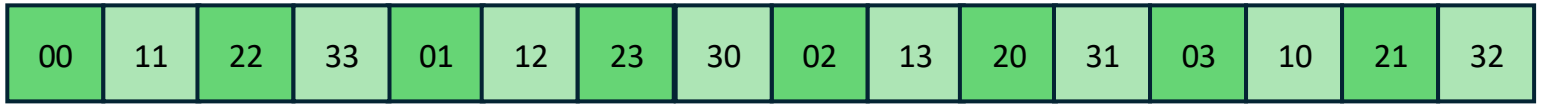
A: Row-major layout



A: Column-major layout



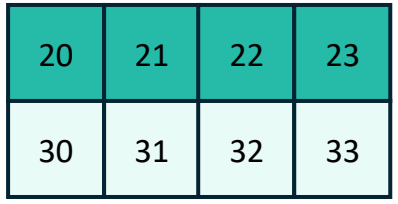
A: Diagonal-major layout



A: Tiled layout

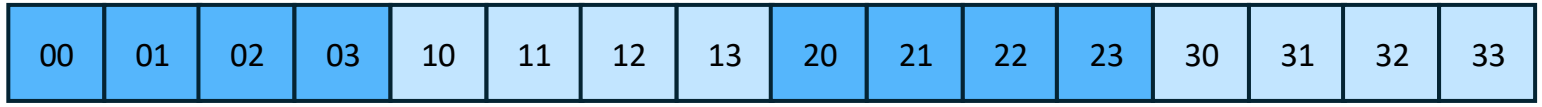


1. Find Layouts For Each Input Tensor

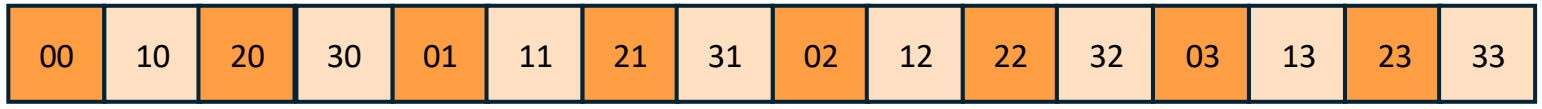


A: Ciphertext

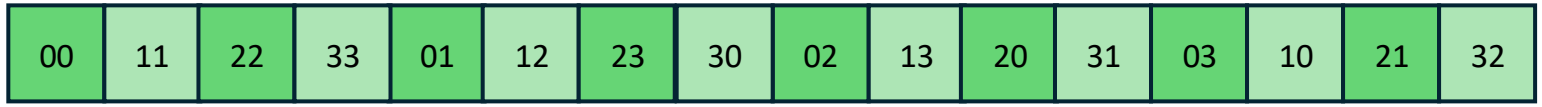
A: Row-major layout



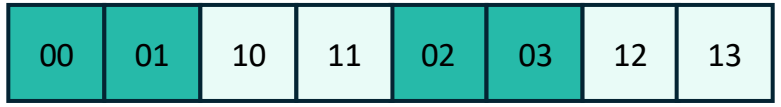
A: Column-major layout



A: Diagonal-major layout



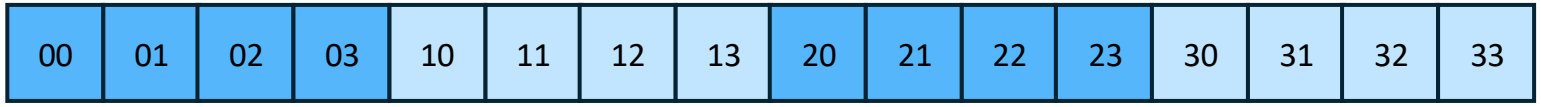
A: Tiled layout



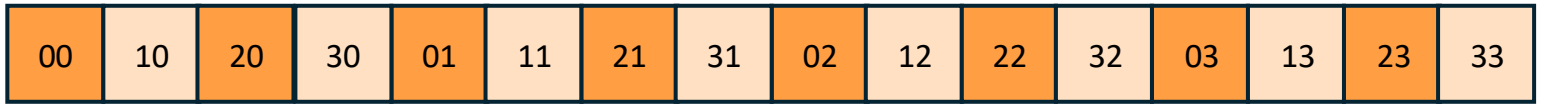
1. Find Layouts For Each Input Tensor

And many more…

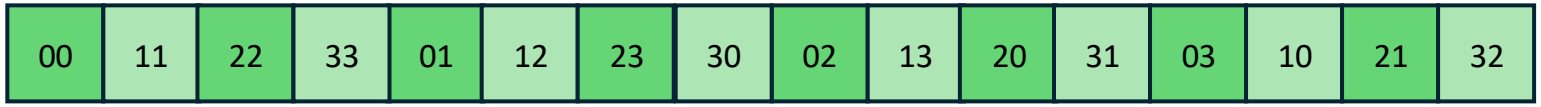
A: Row-major layout



A: Column-major layout



A: Diagonal-major layout

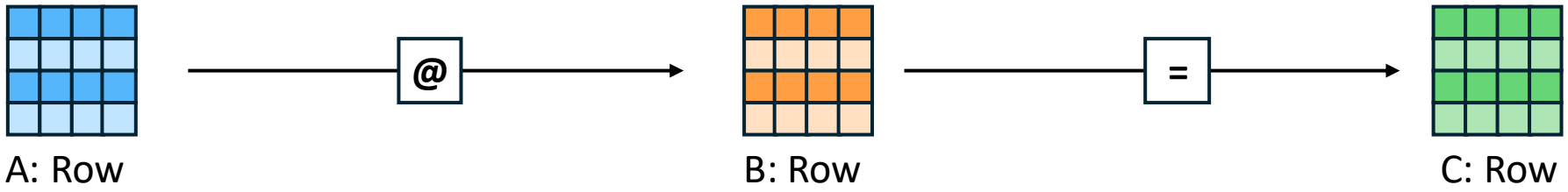


A: Tiled layout

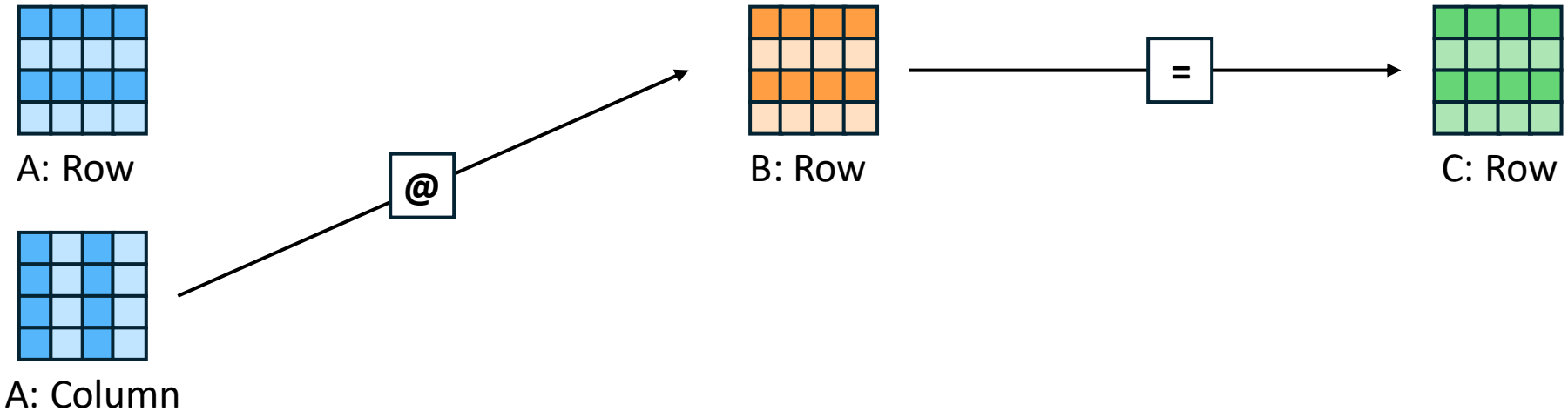


2. Find Cheapest Layouts for Tensor Operator

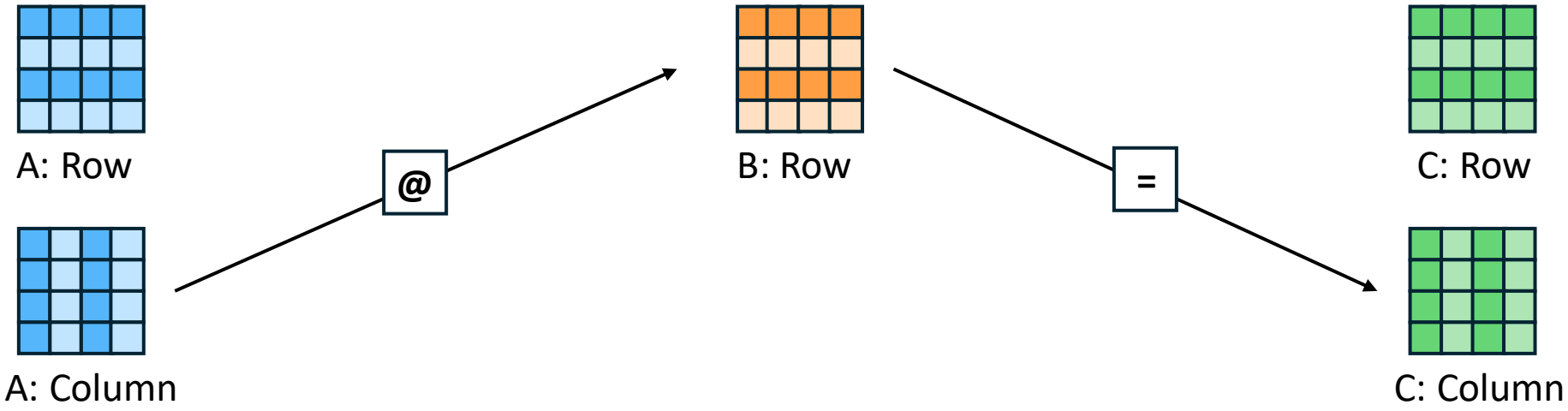
2. Find Cheapest Layouts for Tensor Operator



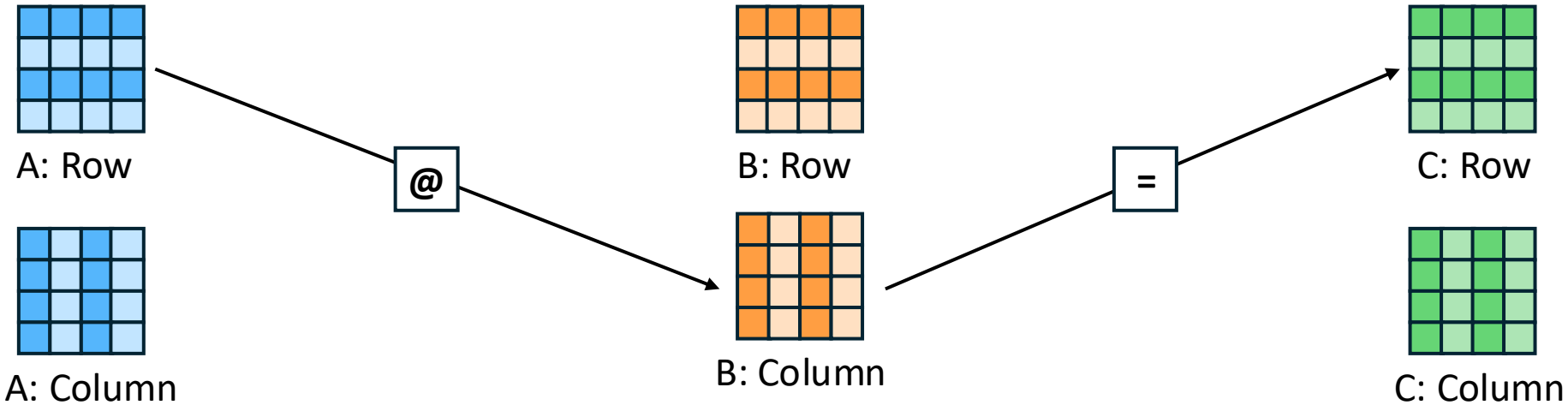
2. Find Cheapest Layouts for Tensor Operator



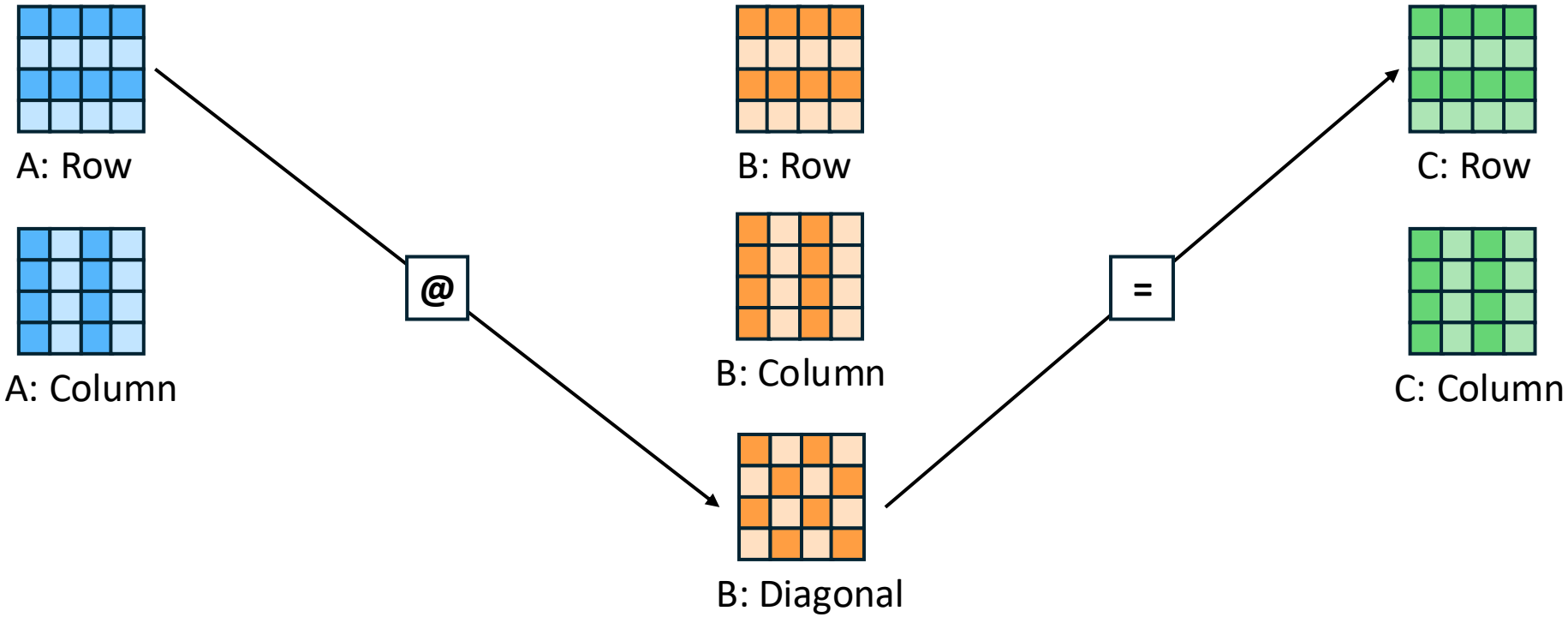
2. Find Cheapest Layouts for Tensor Operator



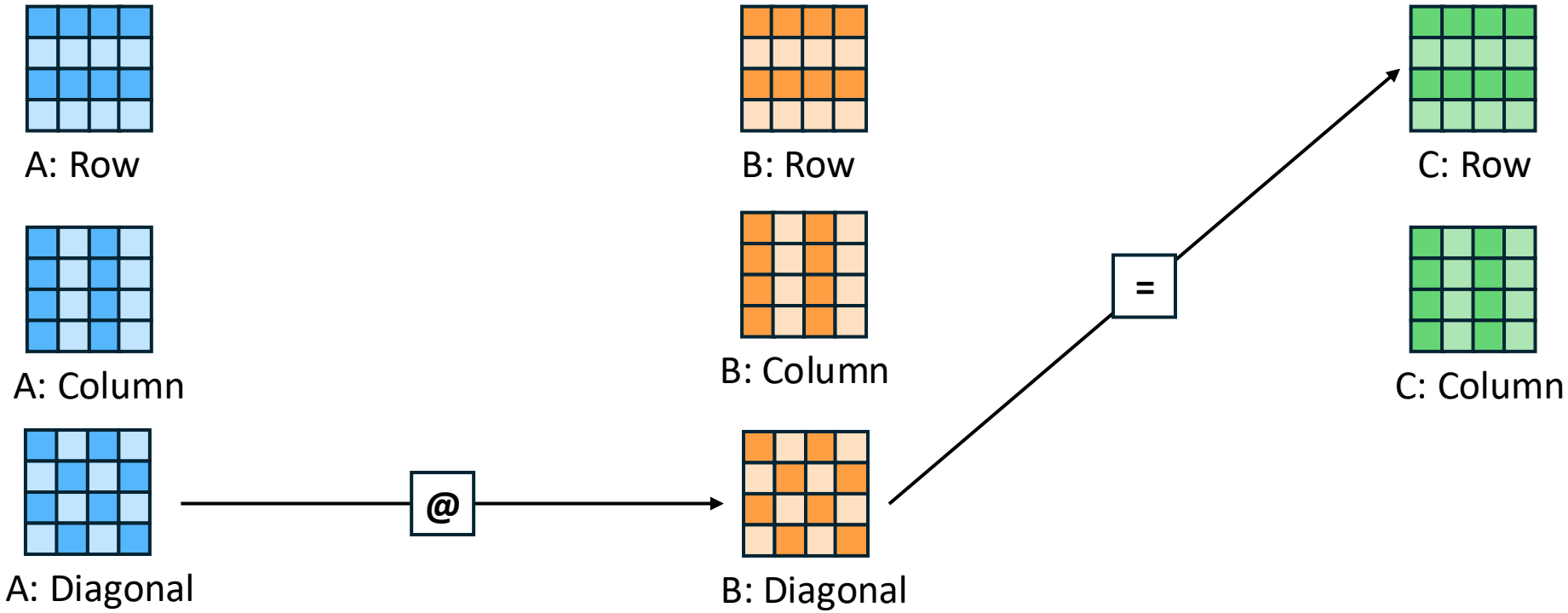
2. Find Cheapest Layouts for Tensor Operator



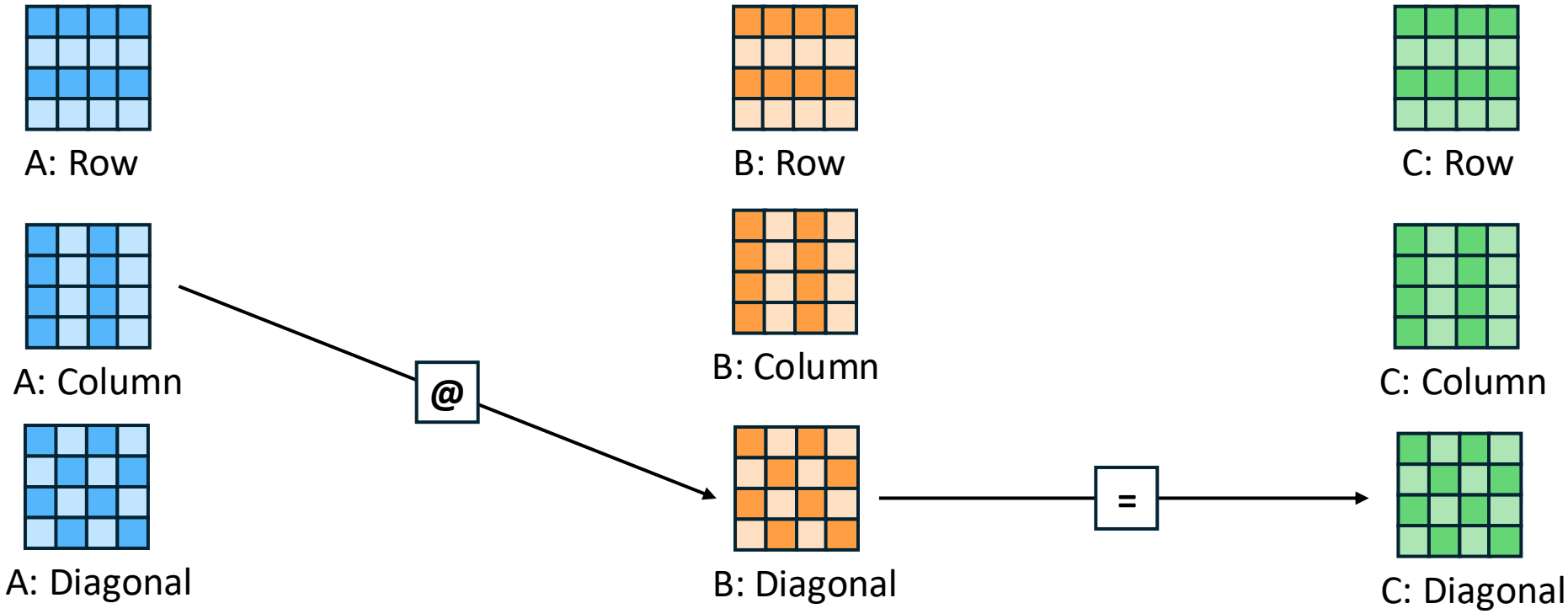
2. Find Cheapest Layouts for Tensor Operator



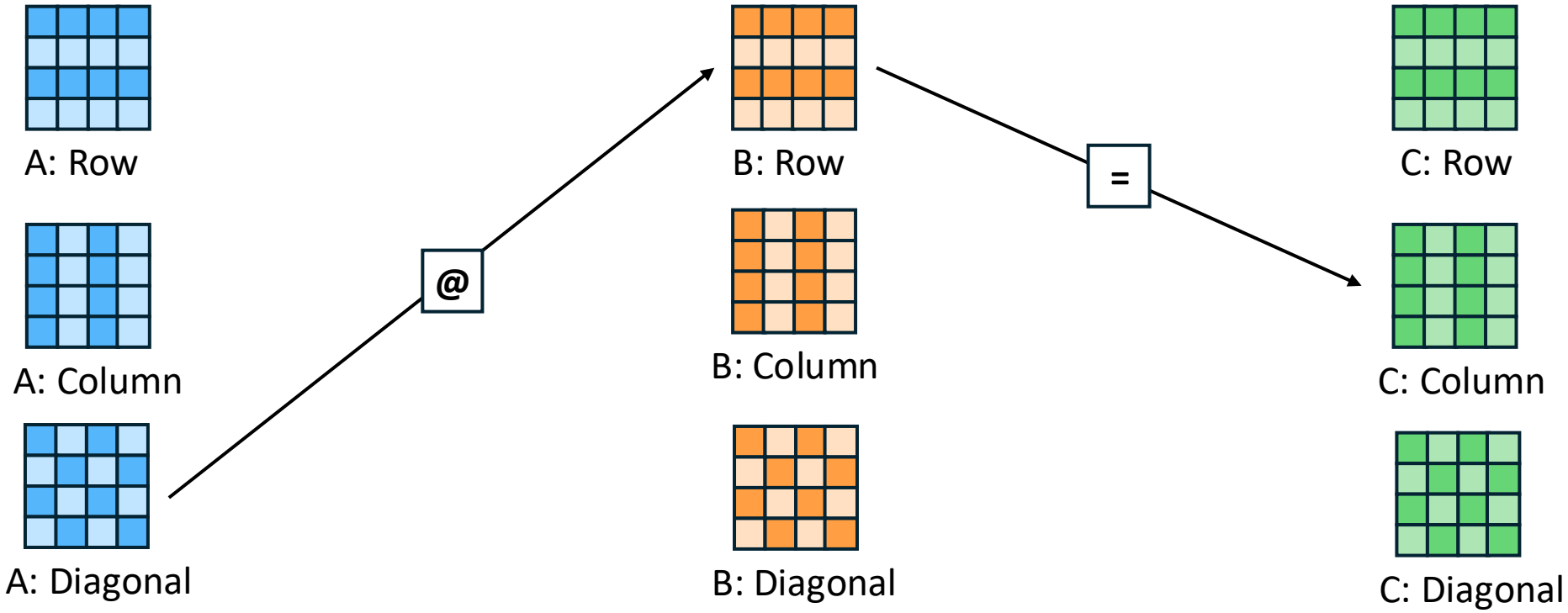
2. Find Cheapest Layouts for Tensor Operator



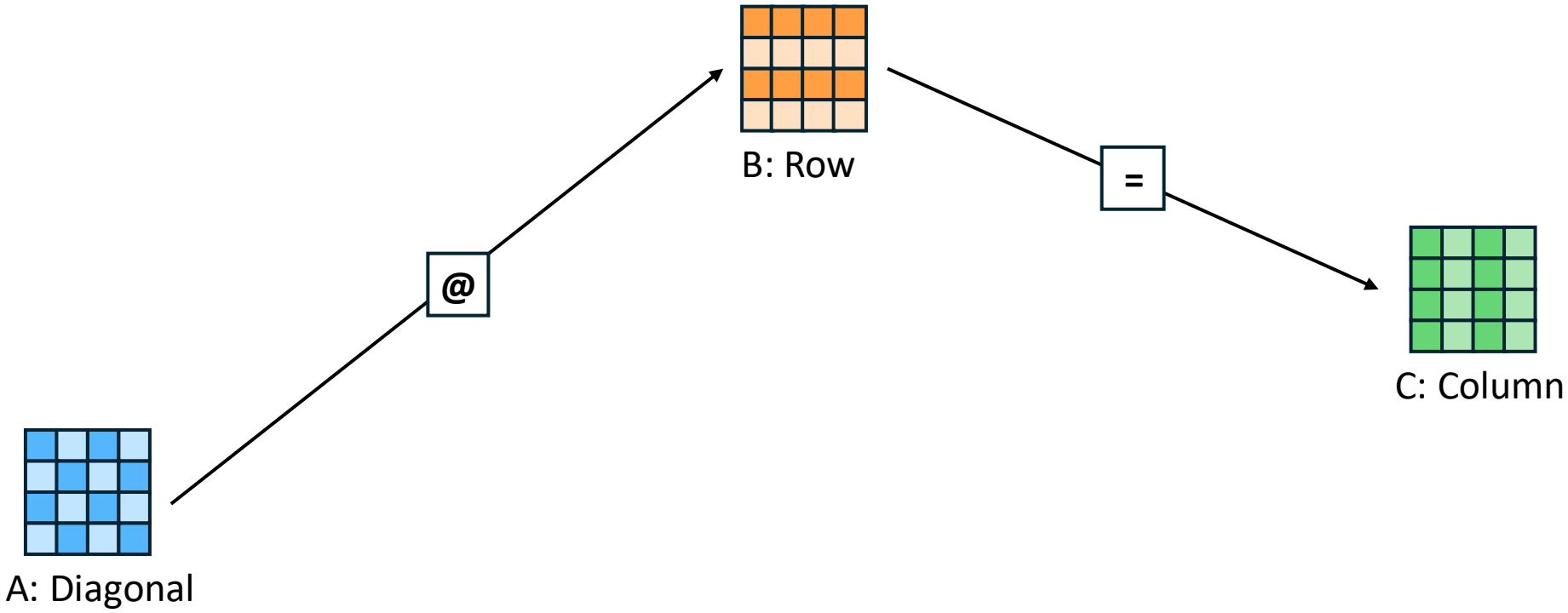
2. Find Cheapest Layouts for Tensor Operator



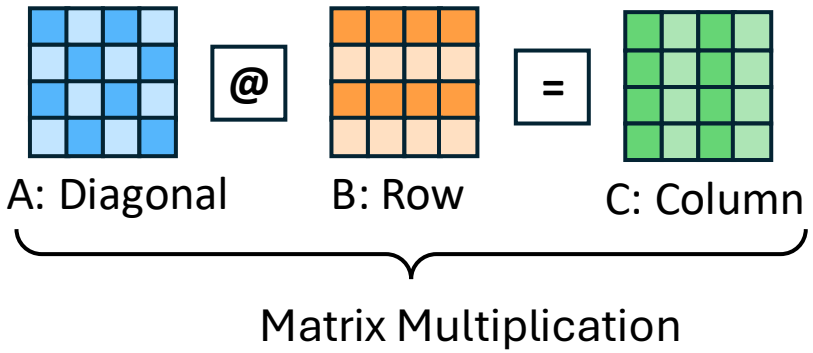
2. Find Cheapest Layouts for Tensor Operator



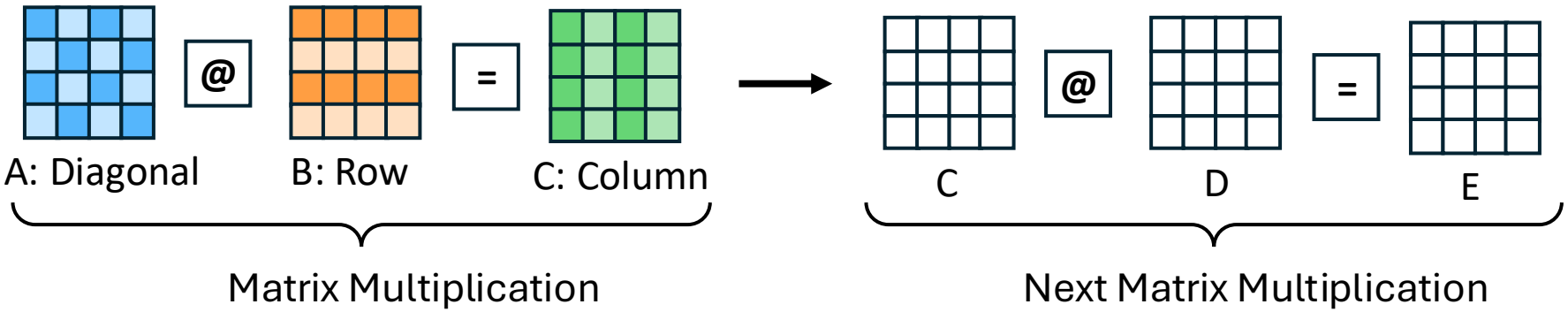
2. Find Cheapest Layouts for Tensor Operator



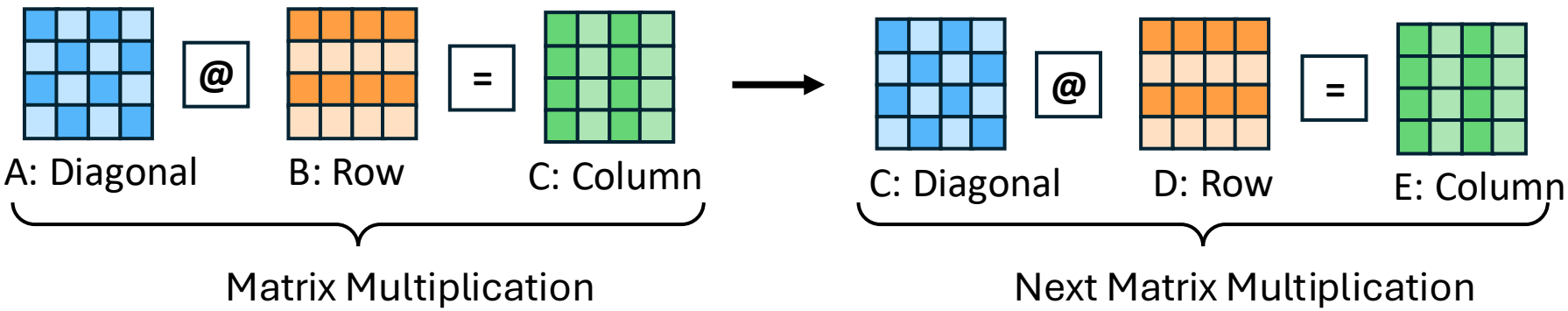
3. Find Layouts For Entire Program



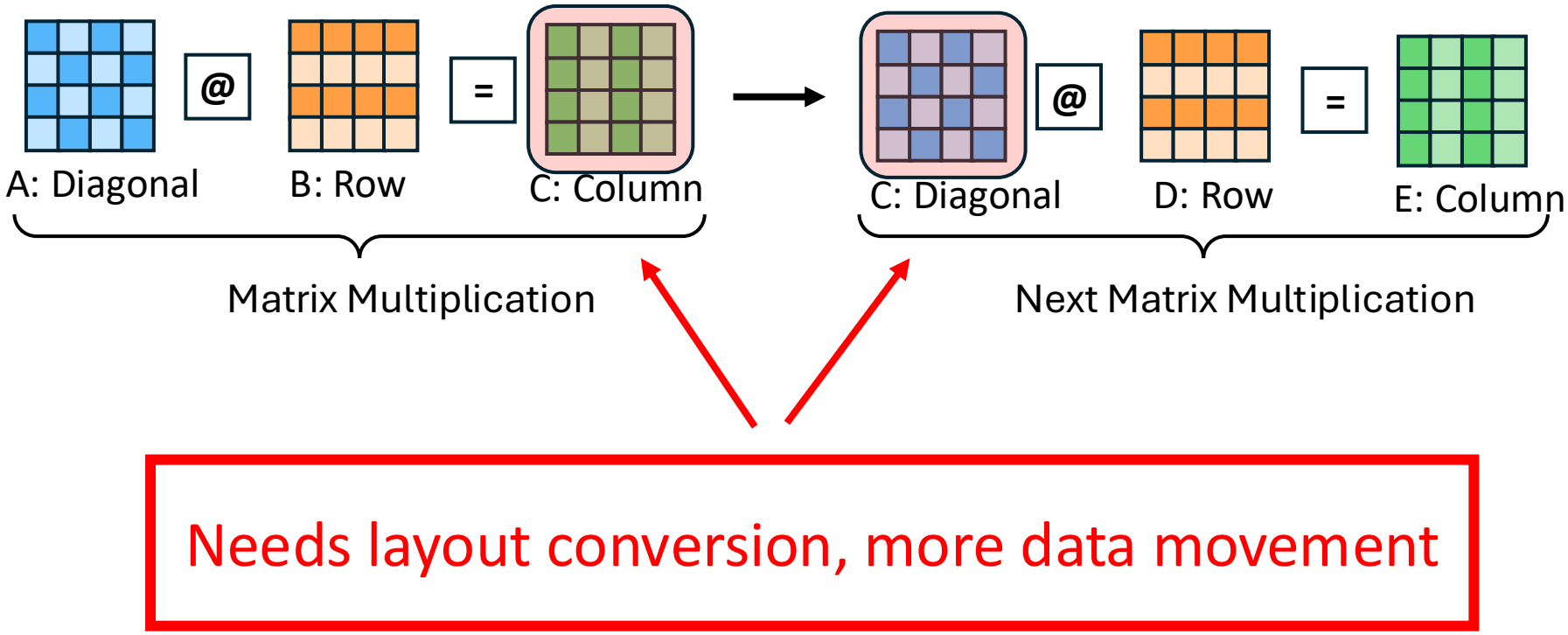
3. Find Layouts For Entire Program



3. Find Layouts For Entire Program



3. Find Layouts For Entire Program



4. Find Layouts *Again…*

4. Find Layouts *Again…*

- Change tensor operations or sizes

4. Find Layouts *Again…*

- Change tensor operations or sizes
- Change cryptographic parameters

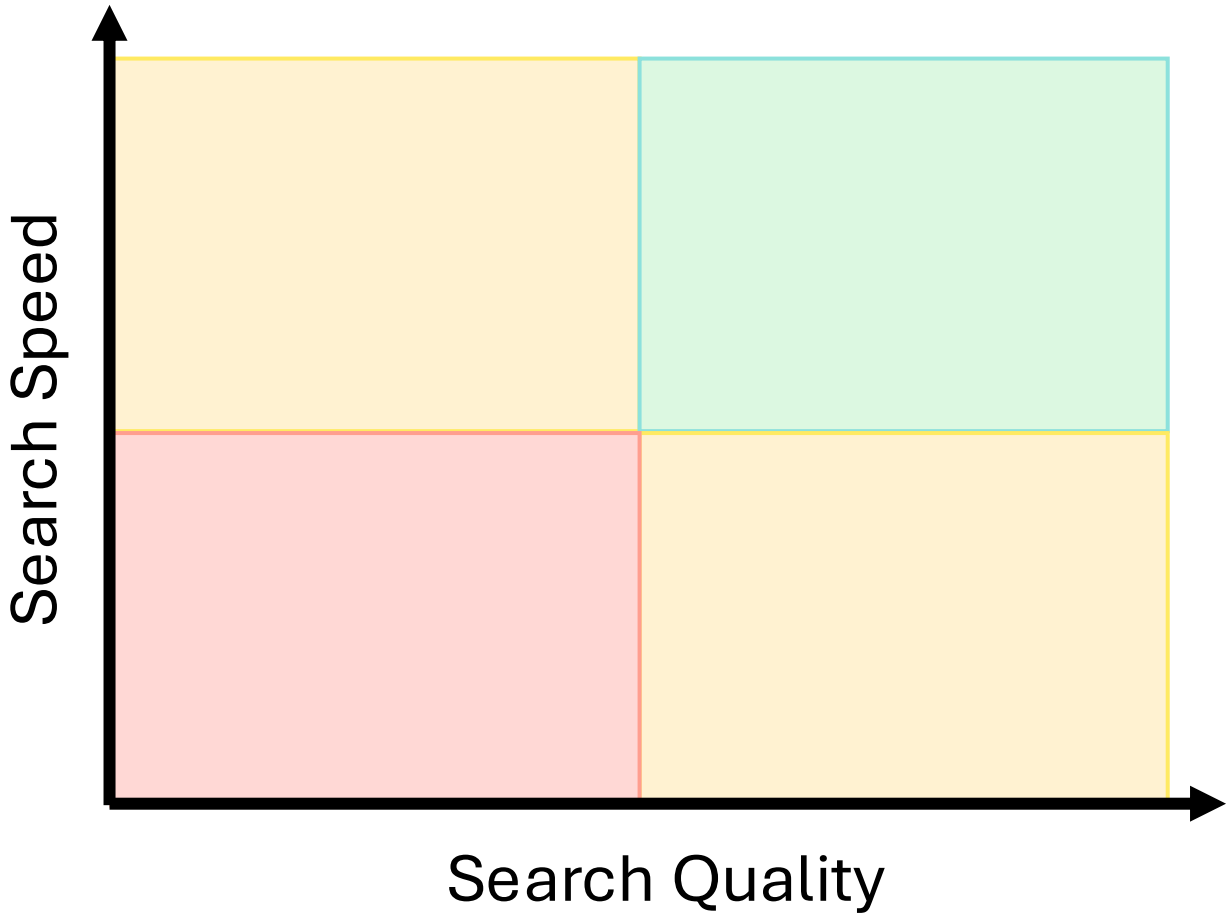
4. Find Layouts *Again…*

- Change tensor operations or sizes
- Change cryptographic parameters
- Change the application

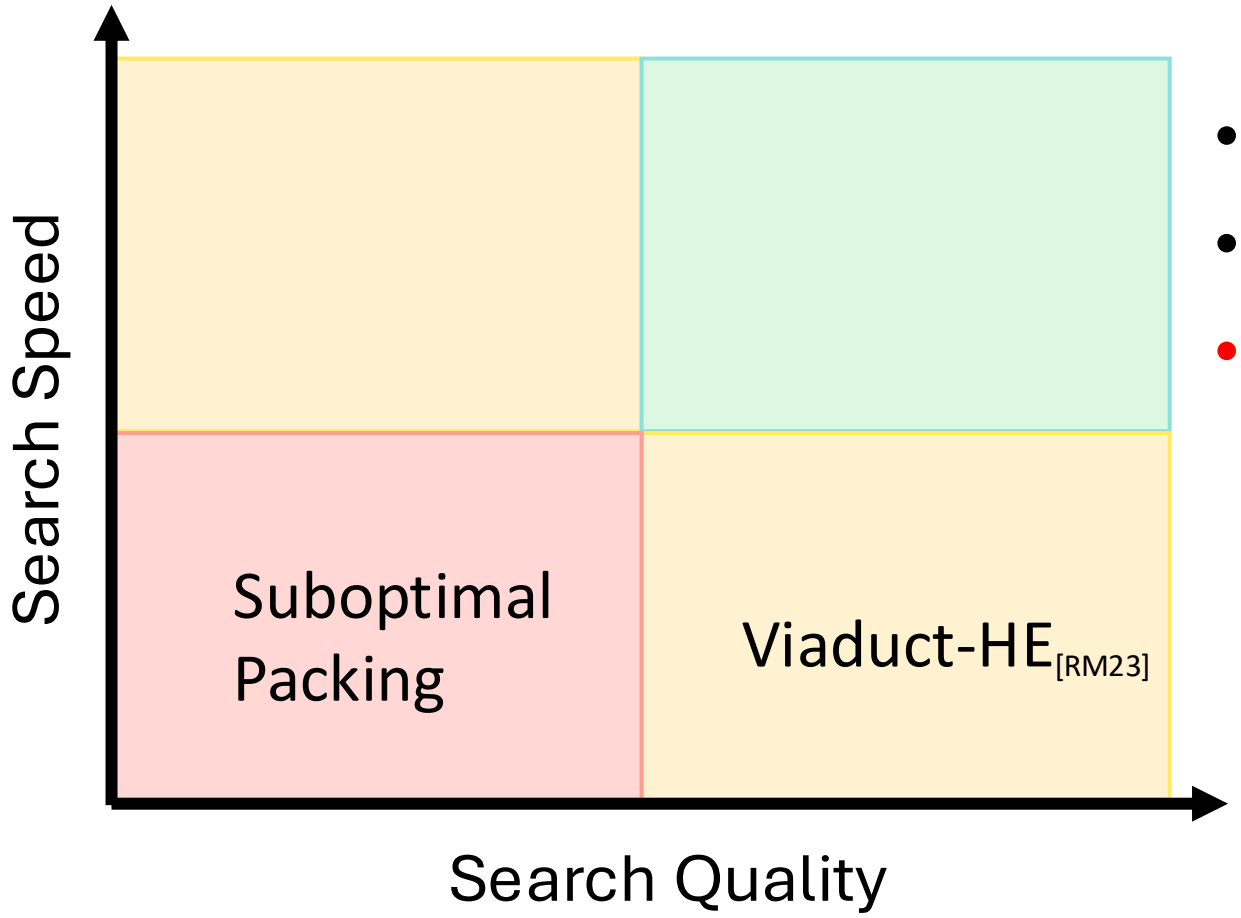
Layout assignment problem:

a search problem where an automated system explores numerous possible layouts to find an assignment that leads to the least costly FHE program.

Tension Between Search Quality and Speed

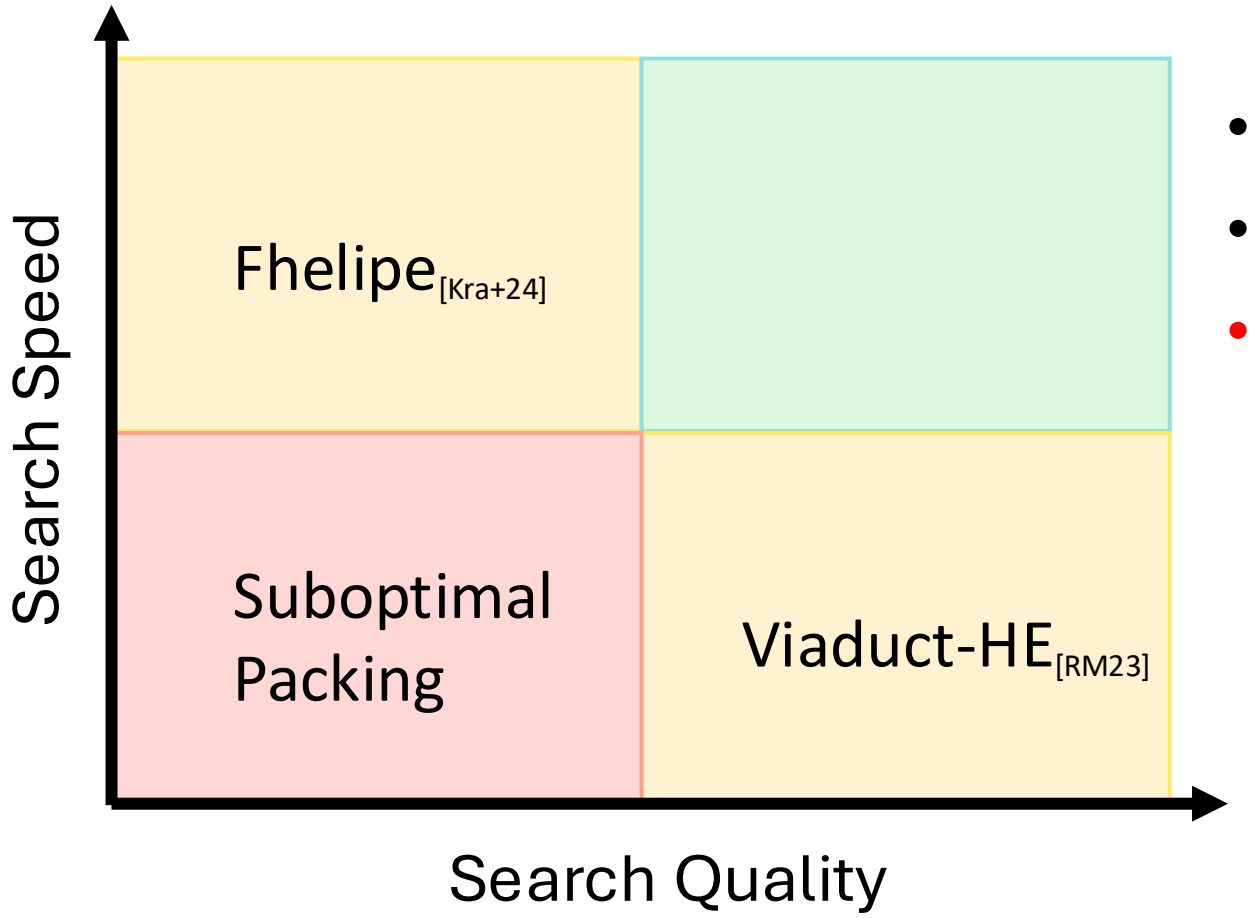


Viaduct-HE$_{[RM23]}$: ↑ Quality, ↓ Speed



- Supports complex layouts
- Brute-force search
- Array materialization

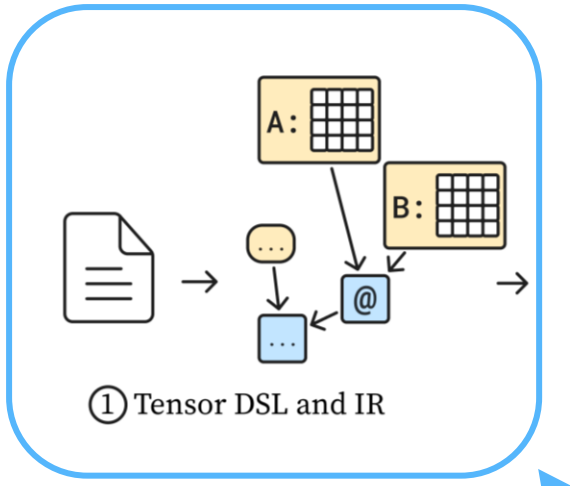
Fhelipe[Kra+24]: ↓ Quality, ↑ Speed



- Simple layout representation
- Fast compilation
- Doesn't support complex layouts, diagonalization

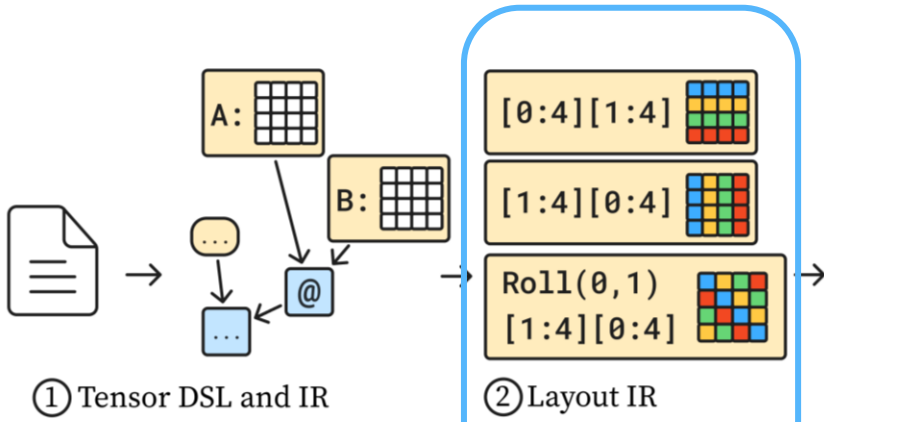
Our Approach: Rotom Compiler Pipeline

Rotom Has High-level Language Support



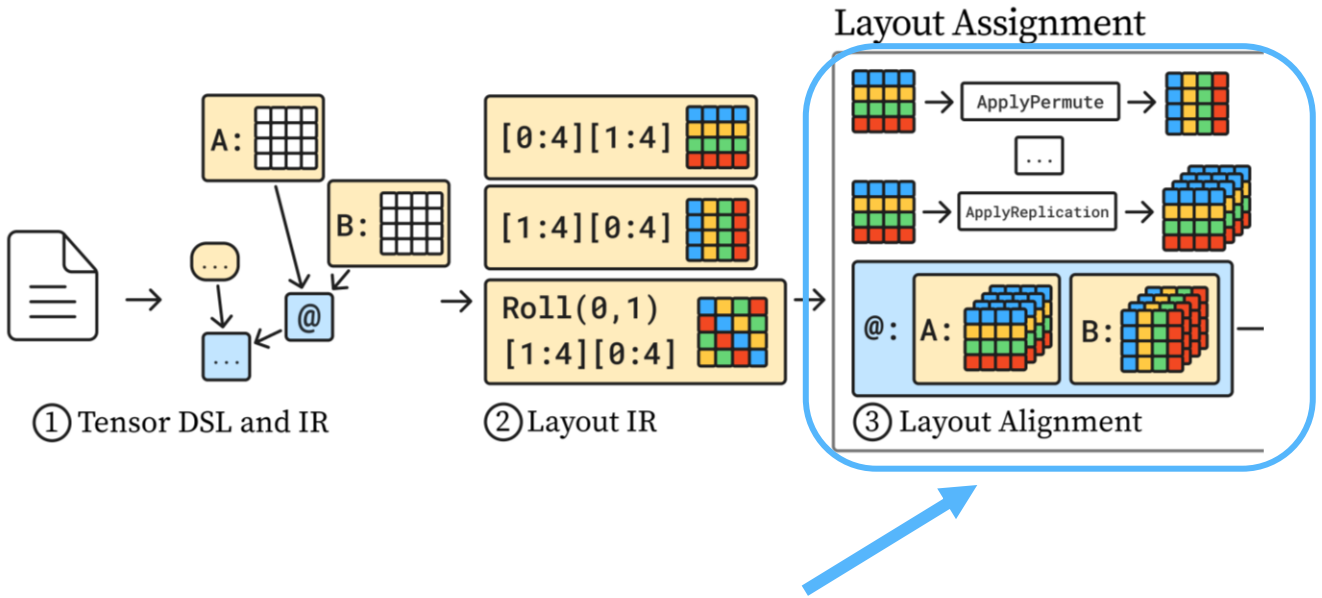
- High-level Python tensor DSL
- Compatible with PyTorch

Rotom Supports a Wide Search Space



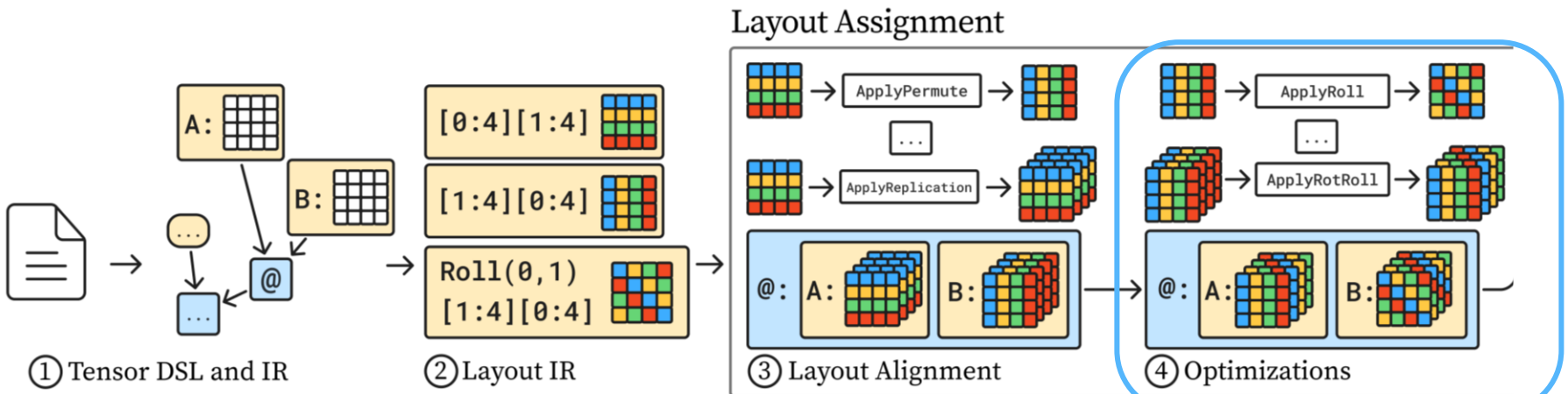
- Compact, expressive layout representation
- Diagonalization and tiling

Rotom Keeps Search Symbolic



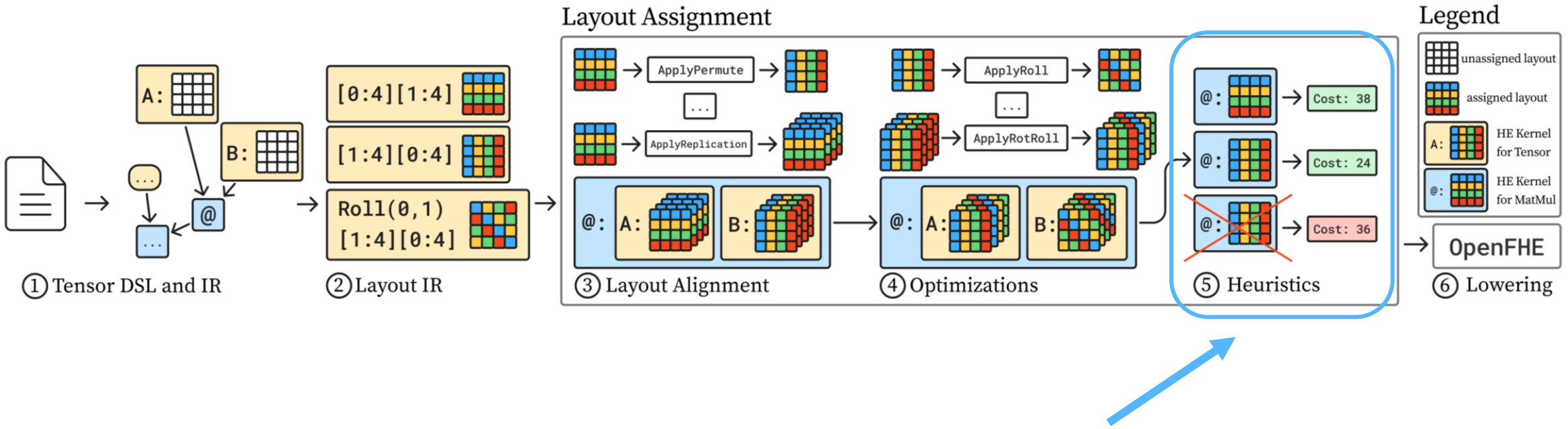
- Layout alignment rules to find compatible operand layouts
- Easily determines HE kernel and cost

Rotom Optimizes Search Quality



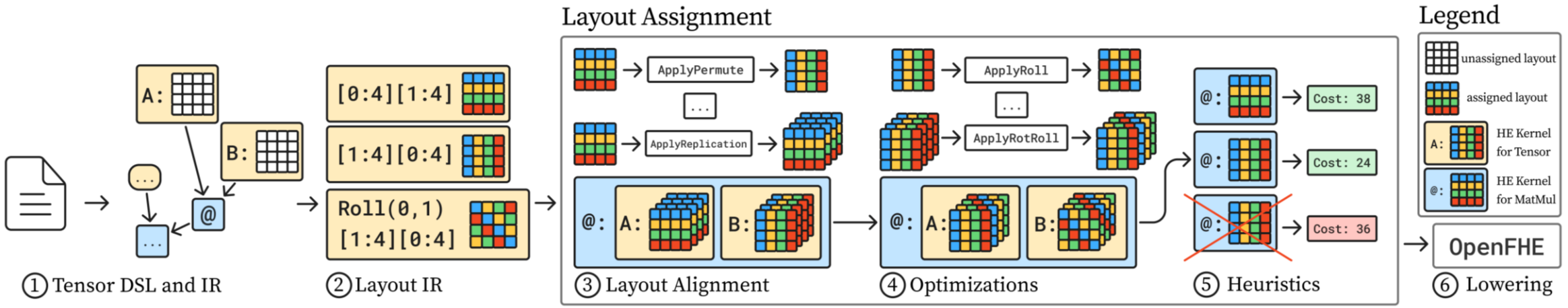
- Novel layout conversion operator → ApplyRoll
- Optimizations, e.g., baby-step giant-step

Rotom Maintains A Tractable Search Space



- Cost-based pruning heuristics
- Keep Pareto frontier of layouts

Focus for Today: ApplyRoll



1. Support complex layouts

2. Reduce data movement costs

How Does Rotom Represent Layouts?

Rotom's Layout Representation is *Compact*

- Defines layouts as nested loop traversals of a tensor

Rotom's Layout Representation is *Compact*

- Defines layouts as nested loop traversals of a tensor

```
for i in range(4):
  for j in range(4):
    ct[i] = A[i][j]
```

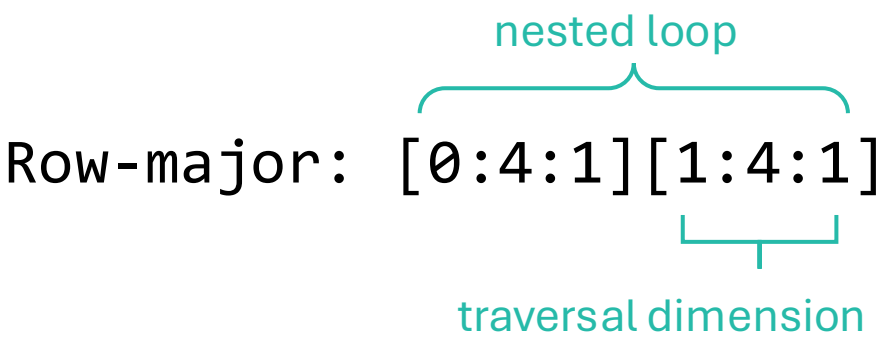
⟶ Row-major layout

Rotom's Layout Representation is *Compact*

- Defines layouts as nested loop traversals of a tensor

```
for i in range(4):
  for j in range(4):
    ct[i] = A[i][j]
```

→ Row-major layout

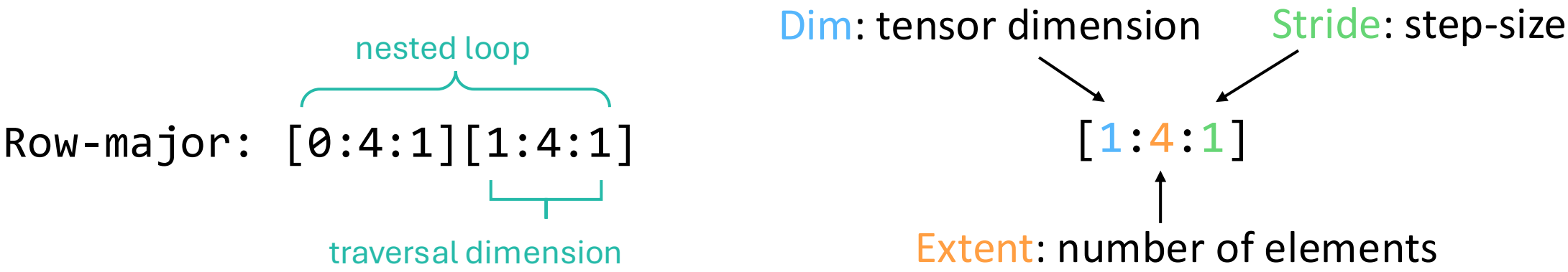


Rotom's Layout Representation is *Compact*

- Defines layouts as nested loop traversals of a tensor

```
for i in range(4):
  for j in range(4):
    ct[i] = A[i][j]
```

→ Row-major layout



Rotom's Layout Representation is *Flexible*

- Changing the loop traversal can represent different layouts

Rotom's Layout Representation is *Flexible*

- Changing the loop traversal can represent different layouts

```
for i in range(4):
  for j in range(4):
    ct[i] = A[j][i]
```

swap

→ Column-major layout

Rotom's Layout Representation is *Flexible*

- Changing the loop traversal can represent different layouts

```
for i in range(4):
  for j in range(4):
    ct[i] = A[j][i]
```



→ Column-major layout

```
   Row-major: [0:4:1][1:4:1]
Column-major: [1:4:1][0:4:1]
```



Rotom's Layout Representation is *Expressive*

- Represent complex layouts such as diagonalization

Rotom's Layout Representation is *Expressive*

- Represent complex layouts such as diagonalization

```
for i in range(4):
  for j in range(4):
    ct[i] = A[j][i+j]
```

modular addition

→ Diagonal-major layout

Rotom's Layout Representation is *Expressive*

- Represent complex layouts such as diagonalization

```
for i in range(4):
  for j in range(4):
    ct[i] = A[j][i+j]
```



Diagonal-major layout

```
Diagonal: roll(1,0)[1:4:1][0:4:1]
```

Rotom's Layout Representation is *Expressive*

- Represent complex layouts such as diagonalization

```
for i in range(4):
  for j in range(4):
    ct[i] = A[j][i+j]
```

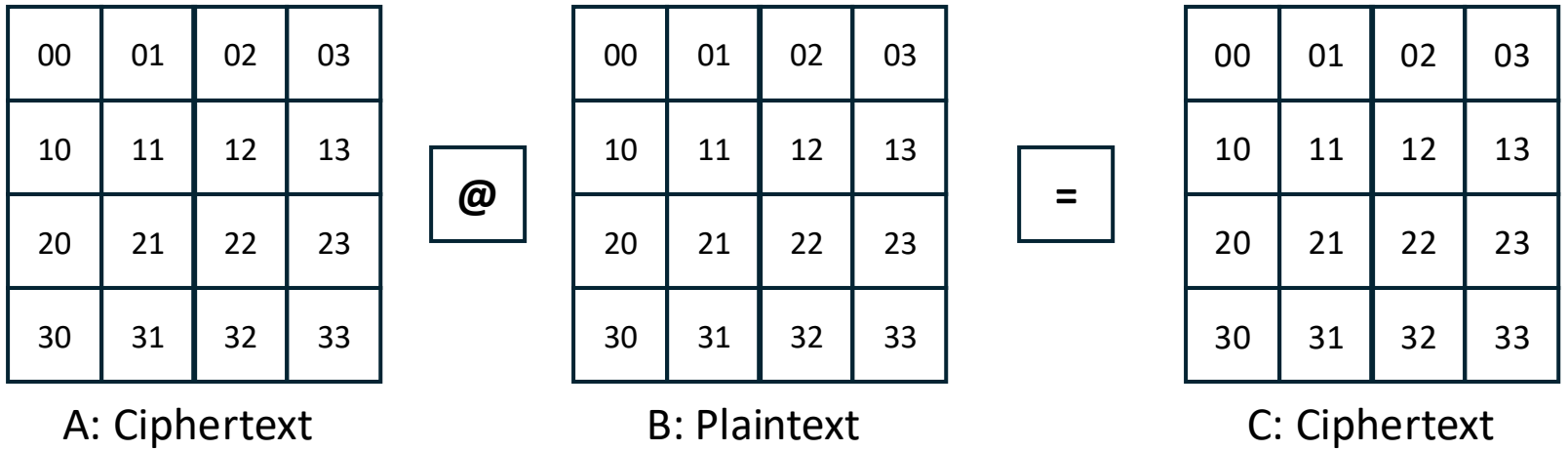
modular addition

→ Diagonal-major layout

```
Diagonal: roll(1,0)[1:4:1][0:4:1]
```

[0:4:1] = [0:4:1] + [1:4:1] % 4, where 0,1 are the index of the traversal dimension

Matrix Multiplication (Without Diagonals)



Prior Work: Assign A – Row-major Layout

00	01	02	03
10	11	12	13
20	21	22	23
30	31	32	33

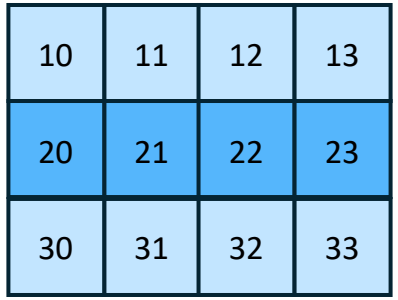
A: Ciphertext

Prior Work: Assign A – Row-major Layout

00	01	02	03
10	11	12	13
20	21	22	23
30	31	32	33

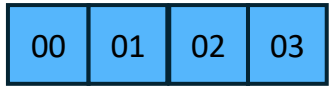
A: Ciphertext

Prior Work: Assign A – Row-major Layout



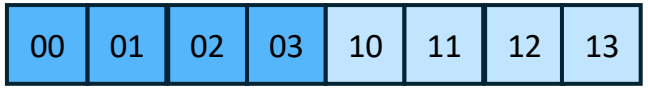
A: Ciphertext

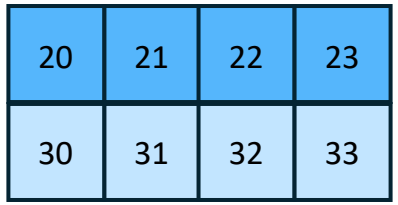
A: Row major Layout



Prior Work: Assign A – Row-major Layout

A: Row major Layout

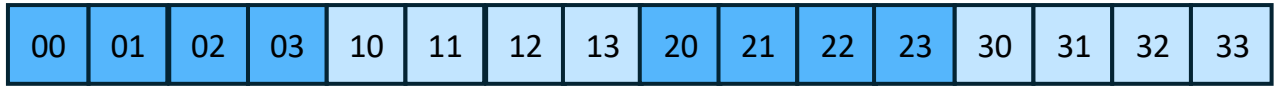




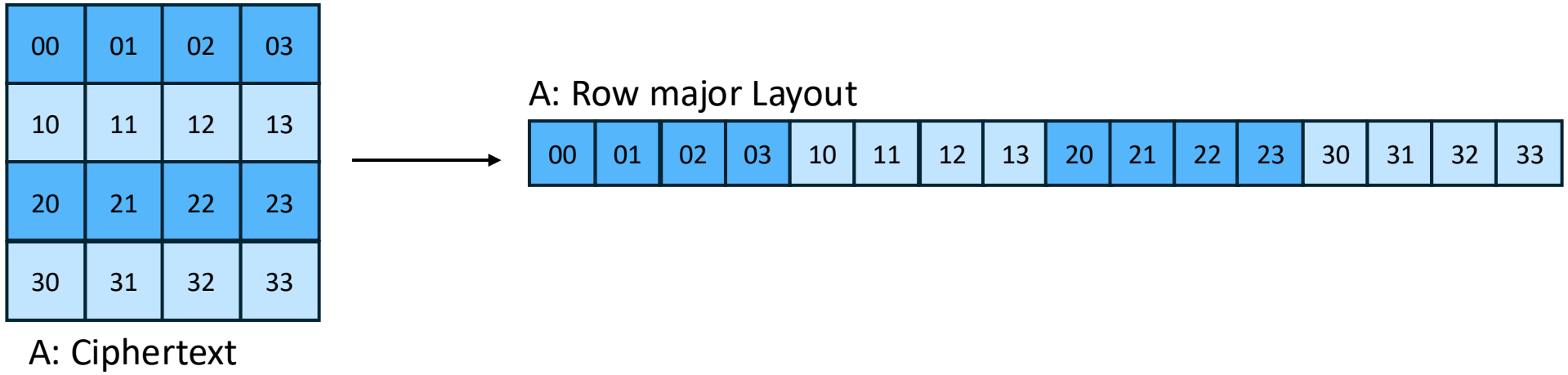
A: Ciphertext

Prior Work: Assign A – Row-major Layout

A: Row major Layout



Prior Work: Assign A – Row-major Layout



Prior Work: Assign B – Repeated Column Layout

00	01	02	03
10	11	12	13
20	21	22	23
30	31	32	33

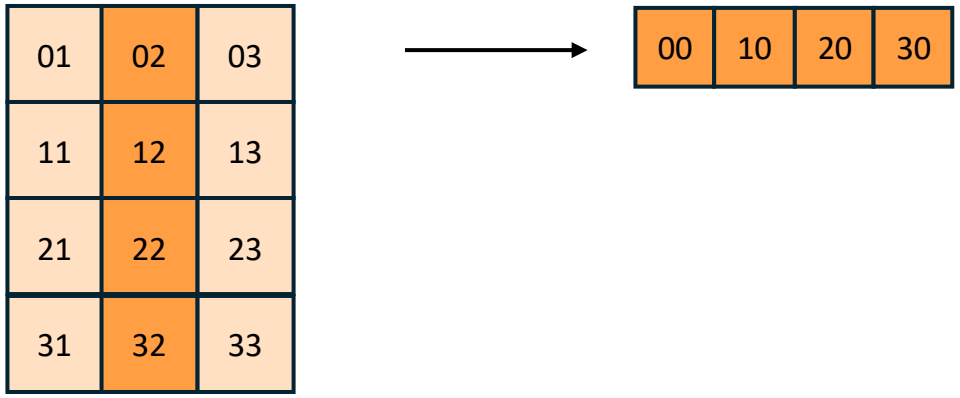
B: Plaintext

Prior Work: Assign B – Repeated Column Layout

00	01	02	03
10	11	12	13
20	21	22	23
30	31	32	33

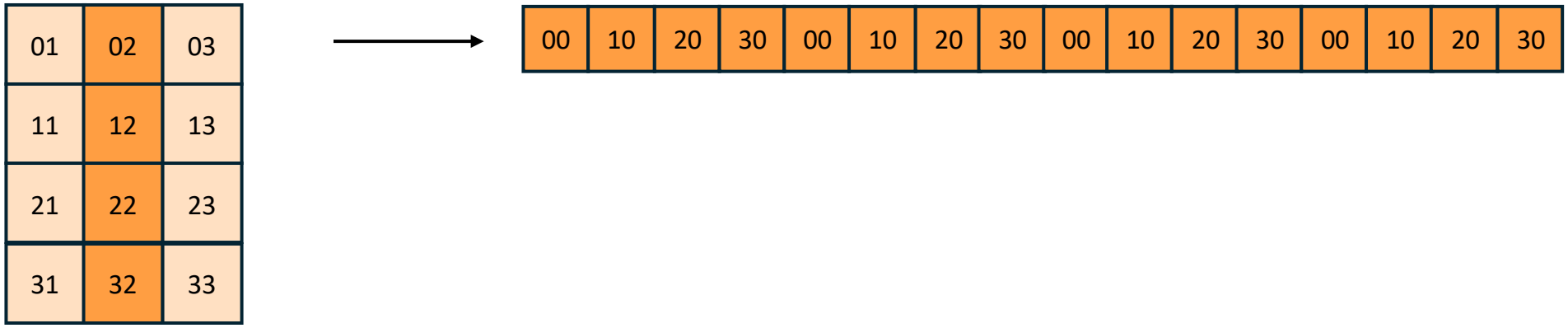
B: Plaintext

Prior Work: Assign B – Repeated Column Layout



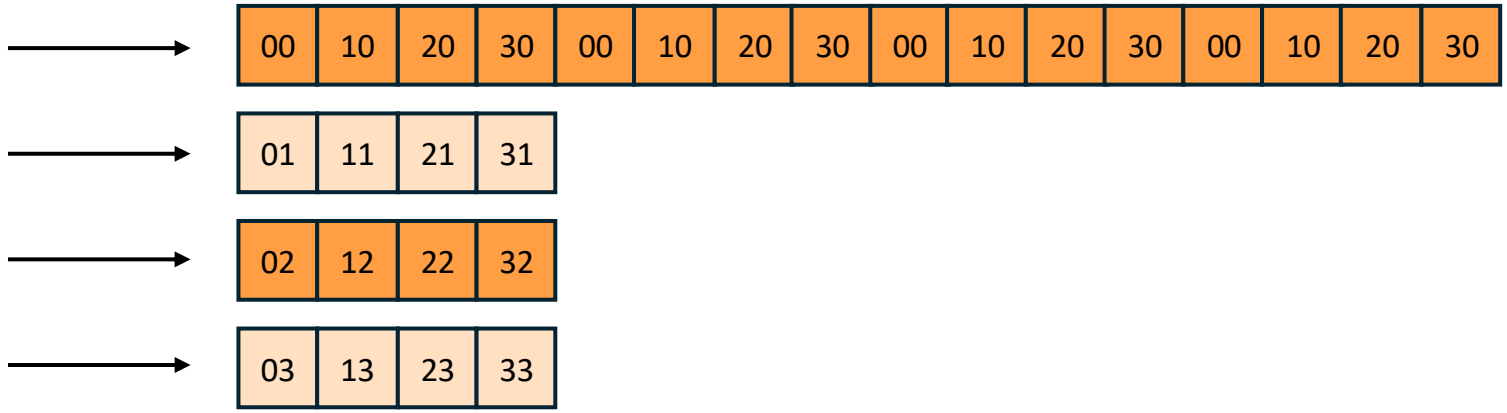
B: Plaintext

Prior Work: Assign B – Repeated Column Layout



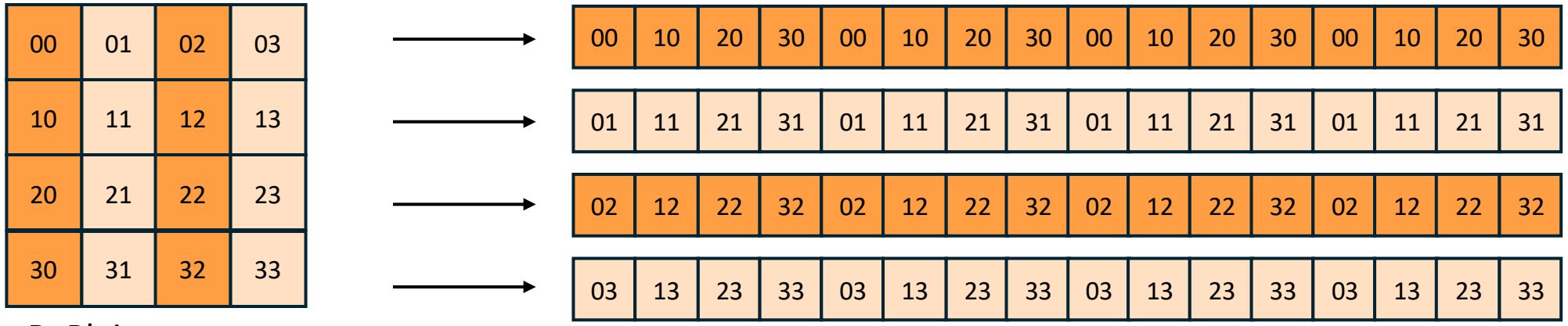
B: Plaintext

Prior Work: Assign B – Repeated Column Layout



B: Plaintext

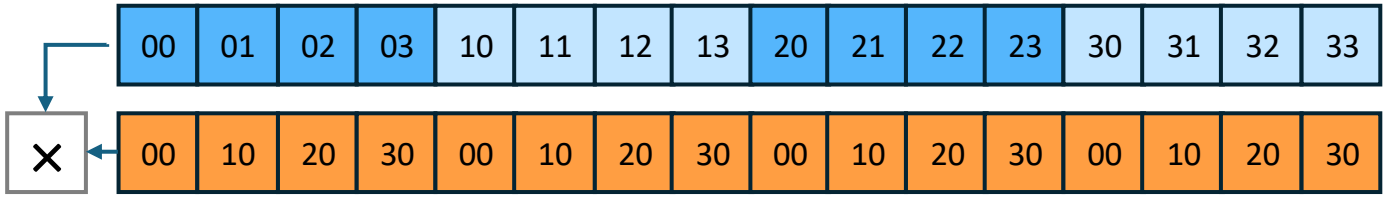
Prior Work: Assign B – Repeated Column Layout



B: Plaintext

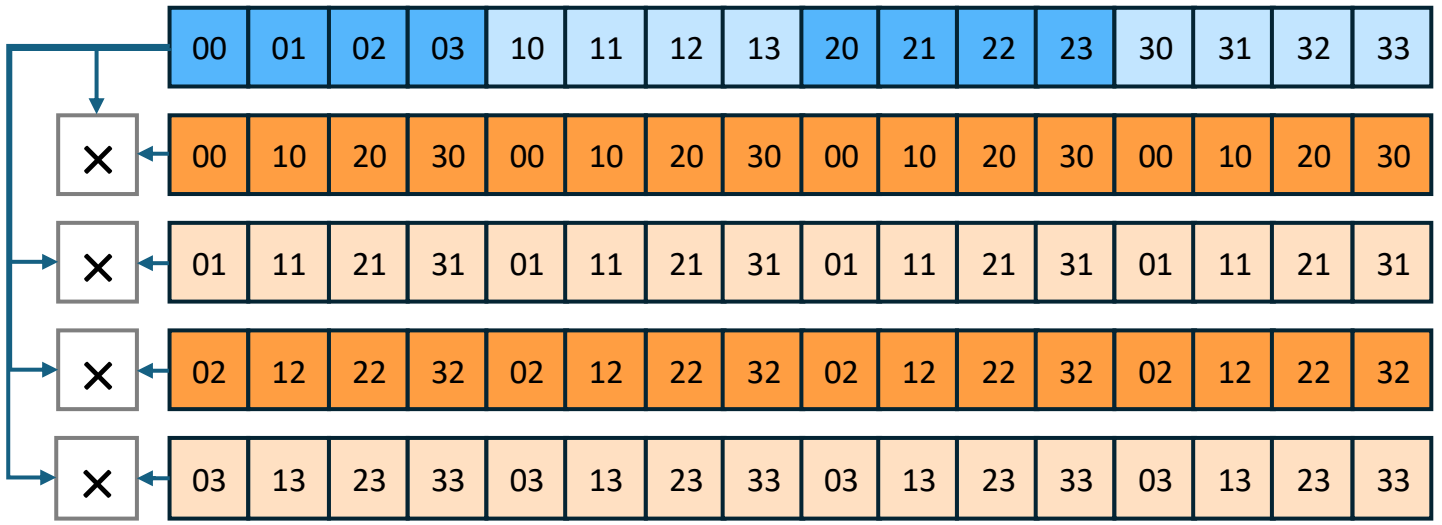
1. Multiply Each Row and Column

1. Multiply rows and columns (4 muls)



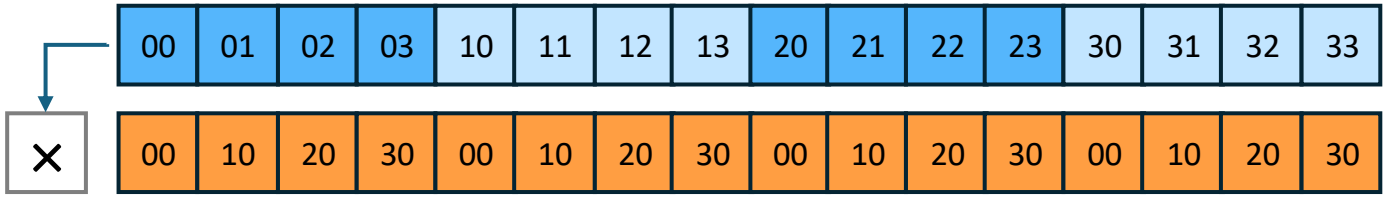
1. Multiply Each Row and Column

1. Multiply rows and columns (4 muls)



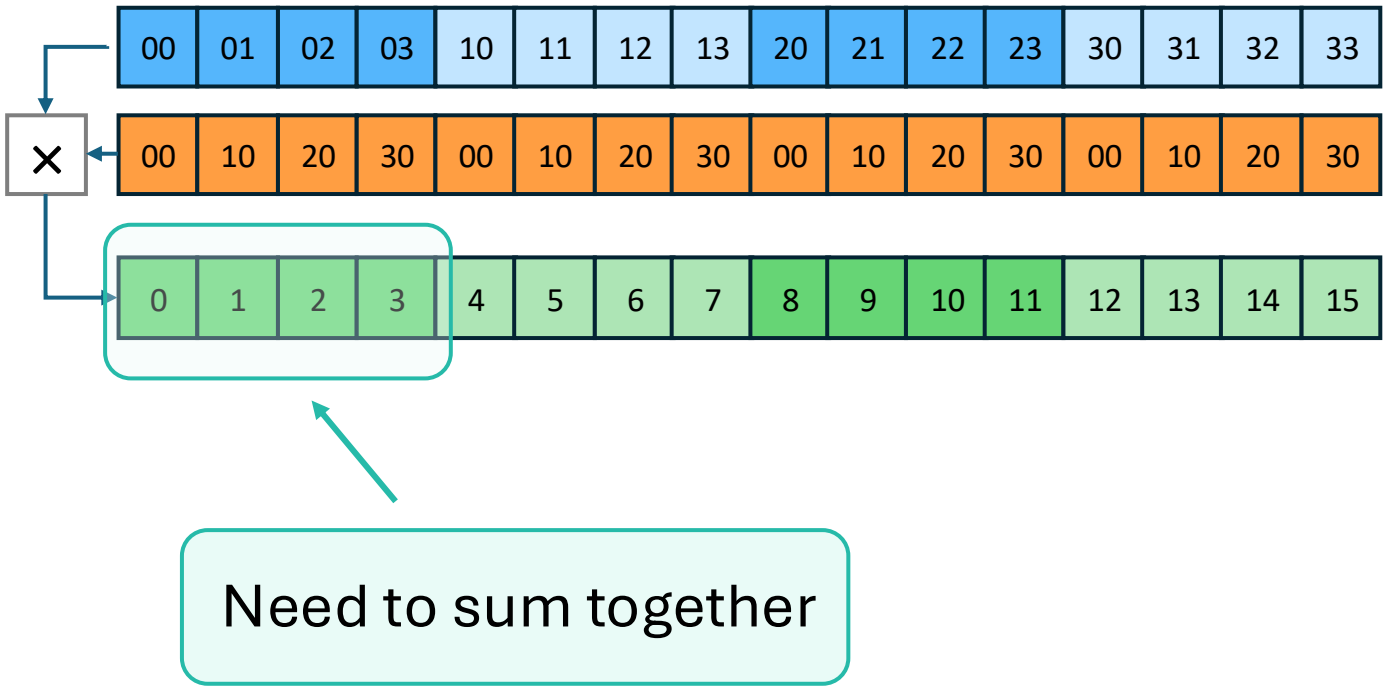
2. Inner Summation Within Each Ciphertext

1. Multiply rows and columns (4 muls)



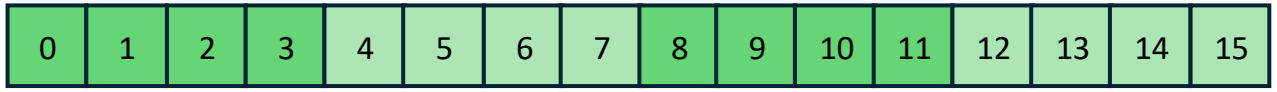
2. Inner Summation Within Each Ciphertext

1. Multiply rows and columns (4 muls)



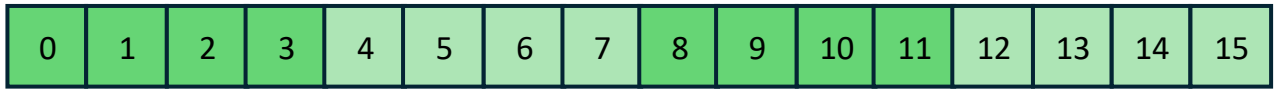
2. Inner Summation Within Each Ciphertext

1. Multiply rows and columns
   (4 muls)



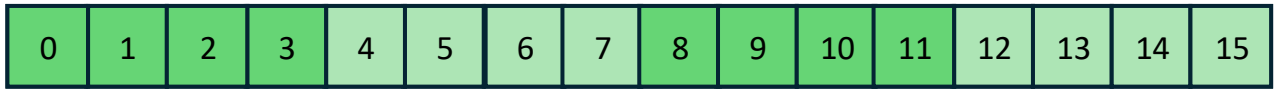
2. Inner Summation Within Each Ciphertext

1. Multiply rows and columns
   (4 muls)
2. Inner summation
   (8 rot, 8 add)



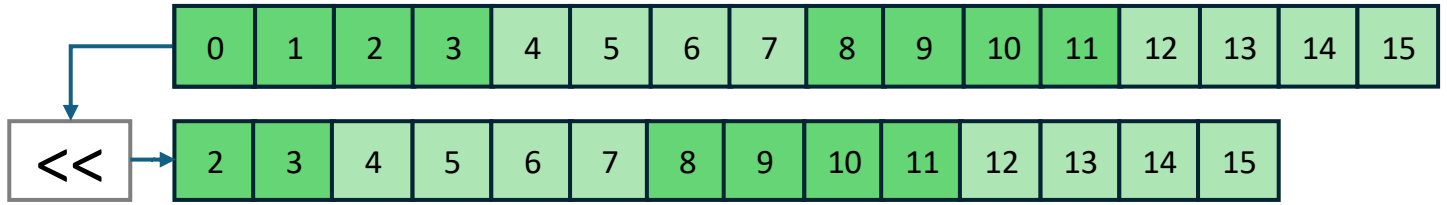
2. Inner Summation Within Each Ciphertext

1. Multiply rows and columns
   (4 muls)
2. Inner summation
   (8 rot, 8 add)



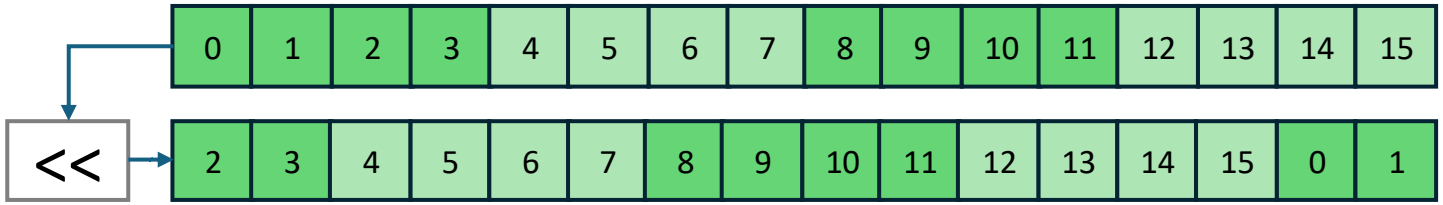
2. Inner Summation Within Each Ciphertext

1. Multiply rows and columns (4 muls)
2. Inner summation (8 rot, 8 add)



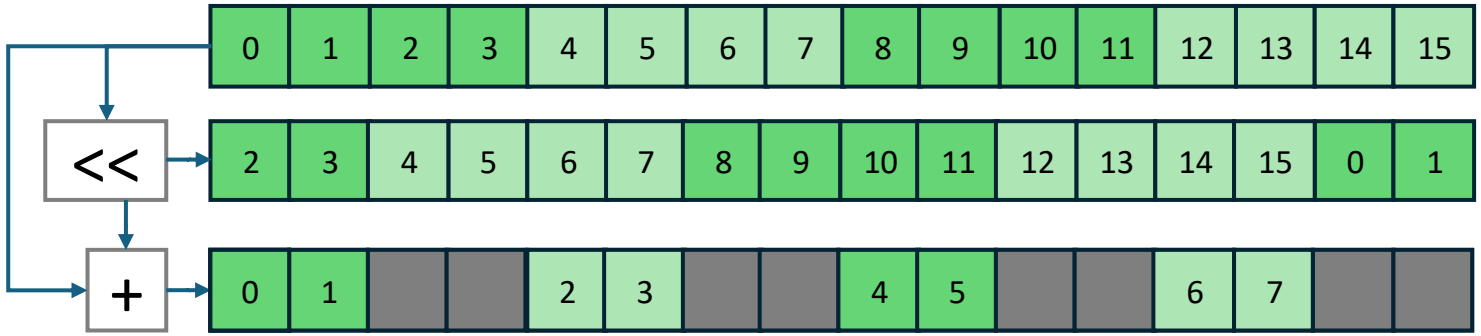
2. Inner Summation Within Each Ciphertext

1. Multiply rows and columns (4 muls)
2. Inner summation (8 rot, 8 add)



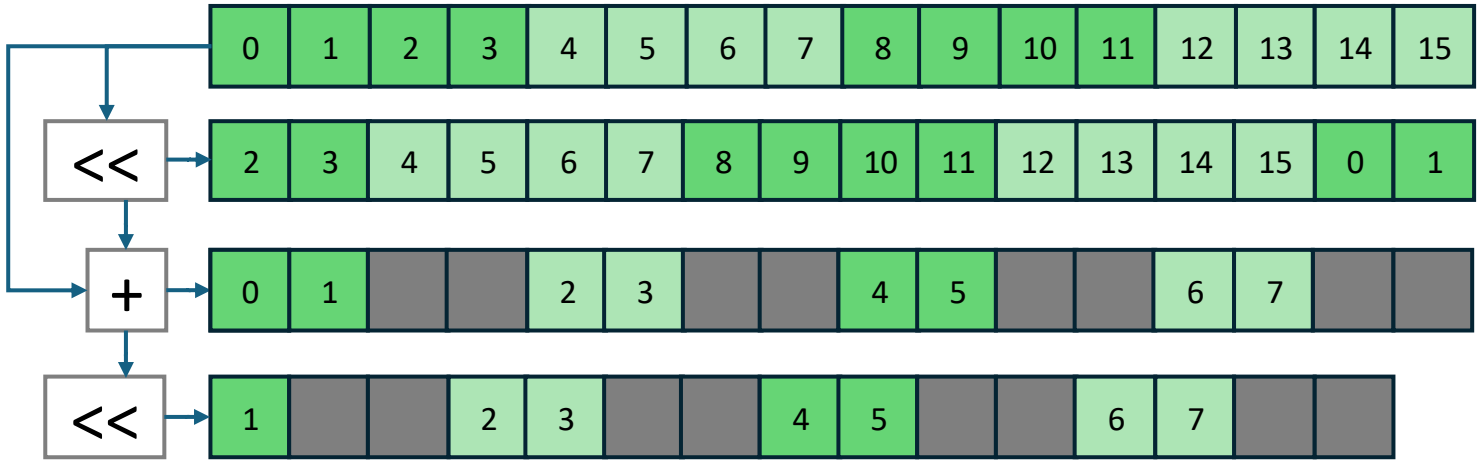
2. Inner Summation Within Each Ciphertext

1. Multiply rows and columns (4 muls)
2. Inner summation (8 rot, 8 add)



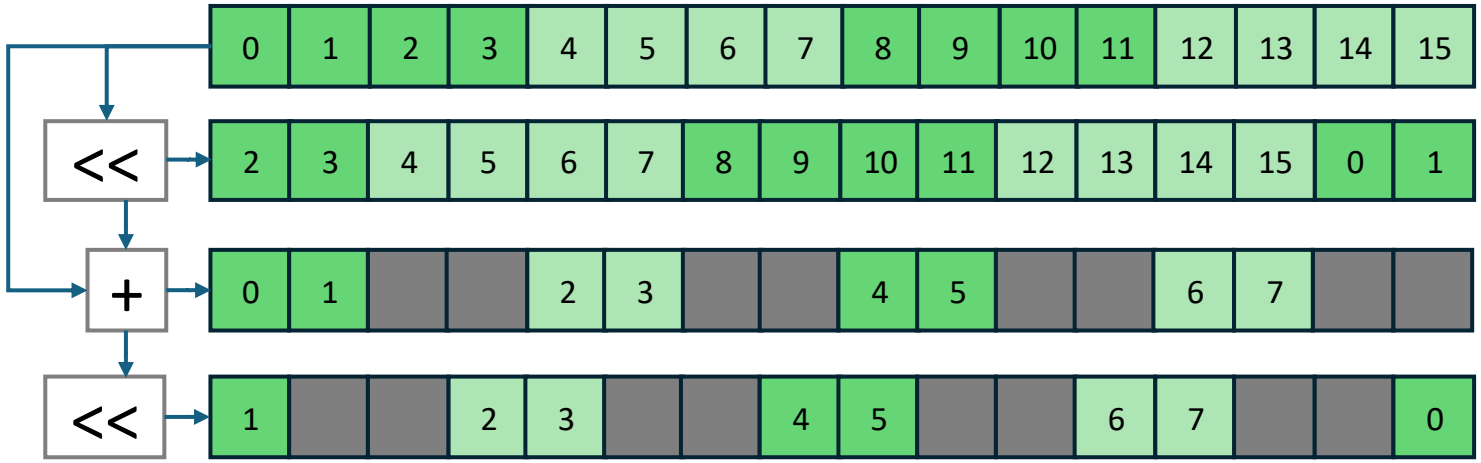
2. Inner Summation Within Each Ciphertext

1. Multiply rows and columns (4 muls)
2. Inner summation (8 rot, 8 add)



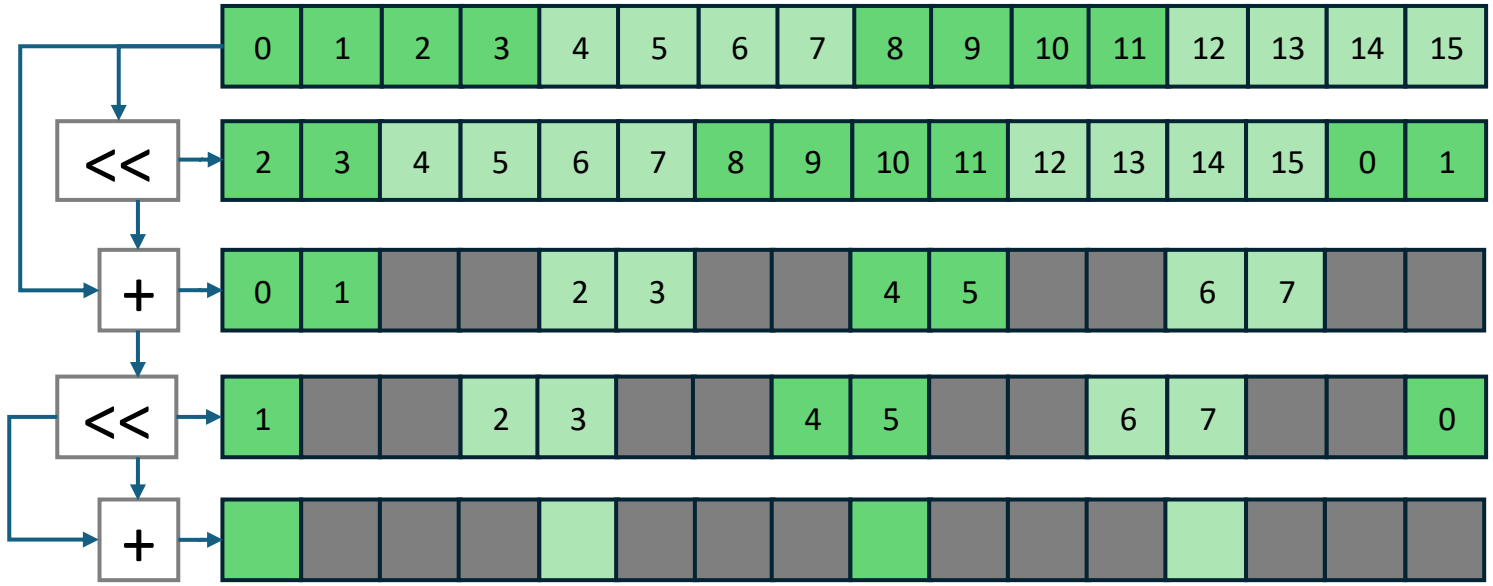
2. Inner Summation Within Each Ciphertext

1. Multiply rows and columns (4 muls)
2. Inner summation (8 rot, 8 add)



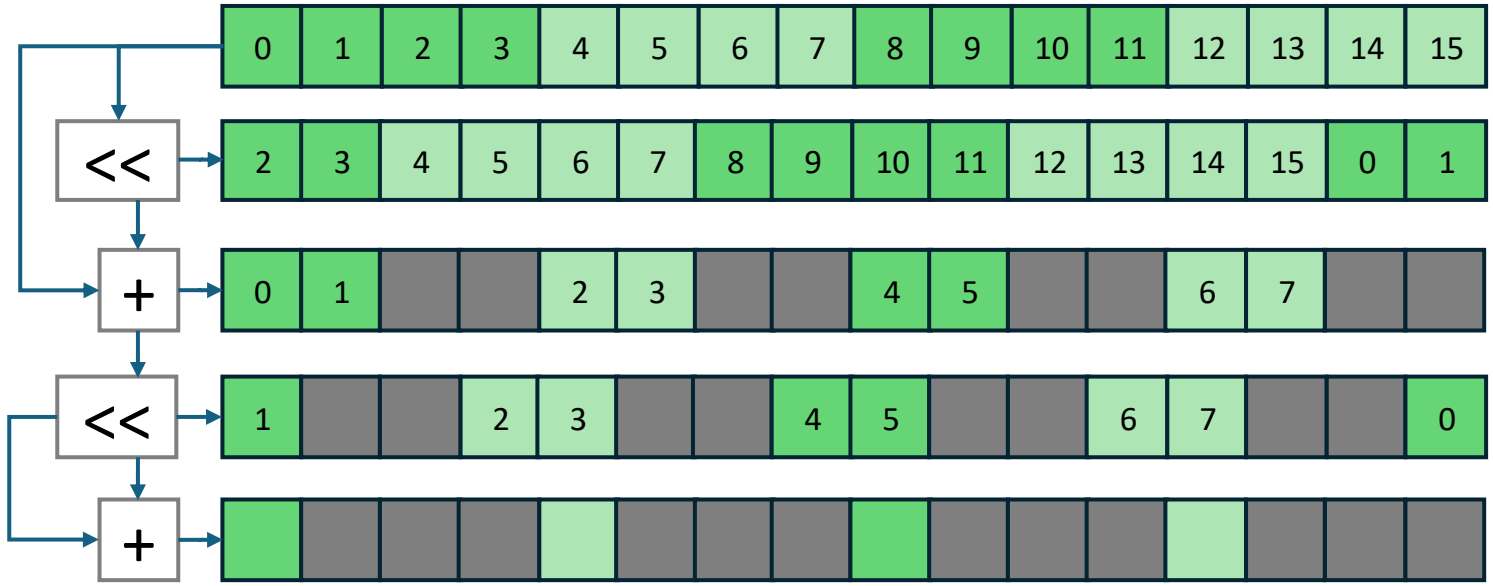
2. Inner Summation Within Each Ciphertext

1. Multiply rows and columns (4 muls)
2. Inner summation (8 rot, 8 add)



2. Inner Summation Within Each Ciphertext

1. Multiply rows and columns (4 muls)
2. Inner summation (8 rot, 8 add)



Repeat 4 times, once for each ciphertext

2. Inner Summation Within Each Ciphertext

1. Multiply rows and columns
   (4 muls)
2. Inner summation
   (8 rot, 8 add)



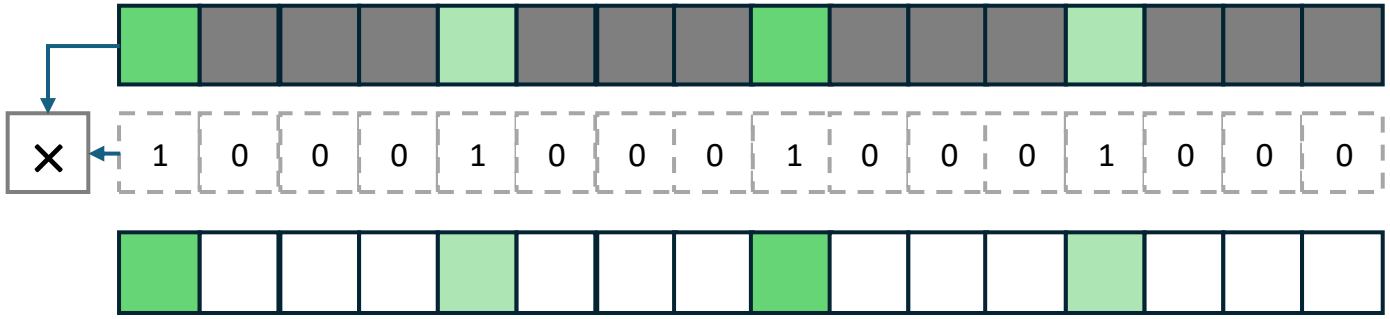
3. Mask Out Garbage Values

1. Multiply rows and columns
   (4 muls)
2. Inner summation
   (8 rot, 8 add)
3. Masking
   (4 mul)



3. Mask Out Garbage Values

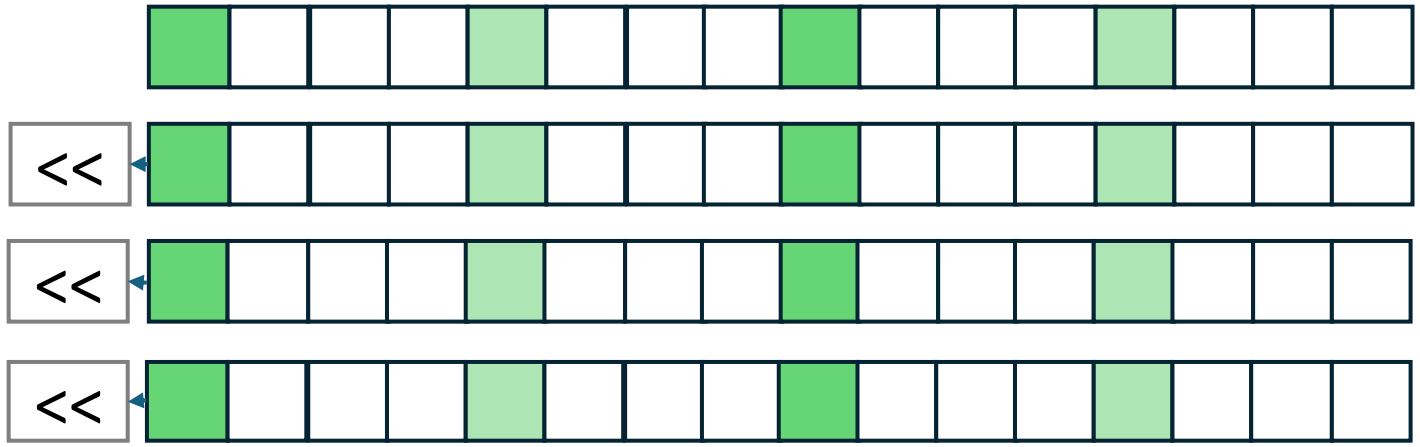
1. Multiply rows and columns
   (4 muls)
2. Inner summation
   (8 rot, 8 add)
3. Masking
   (4 mul)



Repeat 4 times, once for each ciphertext

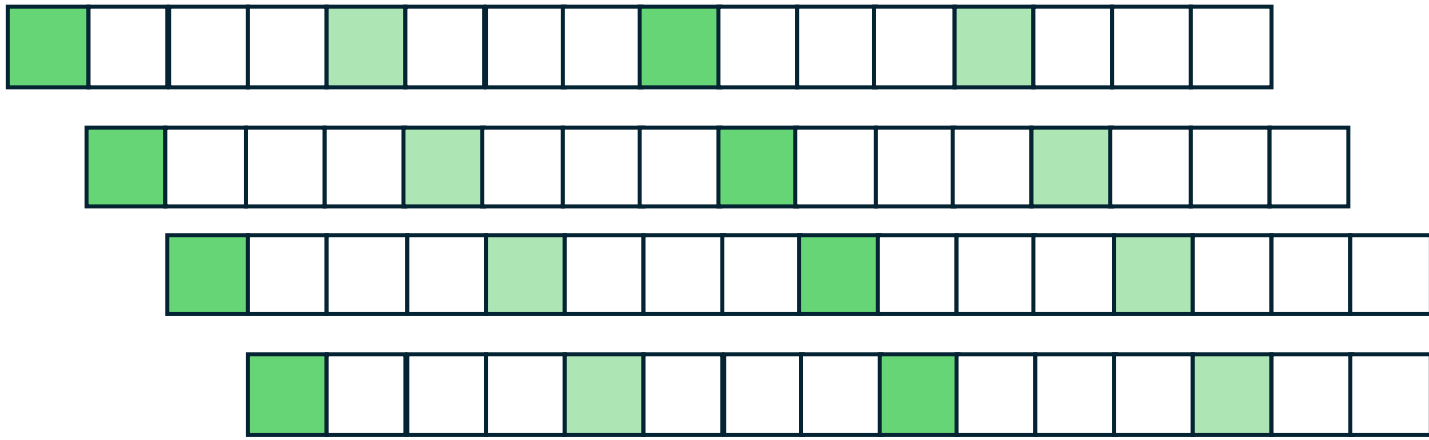
4. Compact Ciphertexts Together

1. Multiply rows and columns
   (4 muls)
2. Inner summation
   (8 rot, 8 add)
3. Masking
   (4 mul)
4. Compaction:
   (3 rot, 3 add)



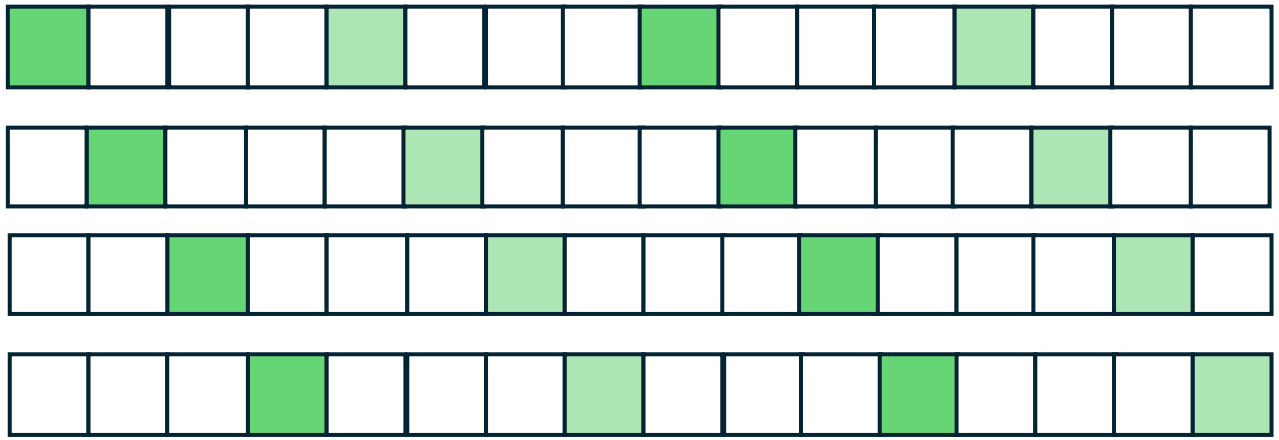
4. Compact Ciphertexts Together

1. Multiply rows and columns
   (4 muls)
2. Inner summation
   (8 rot, 8 add)
3. Masking
   (4 mul)
4. Compaction:
   (3 rot, 3 add)



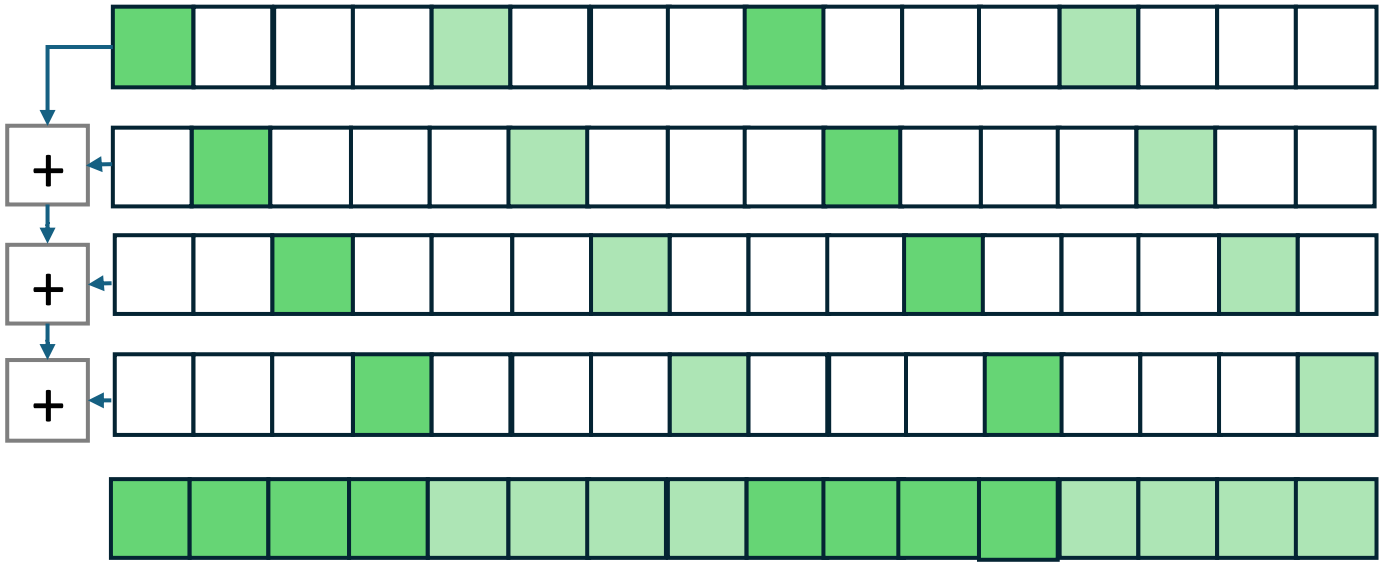
4. Compact Ciphertexts Together

1. Multiply rows and columns
   (4 muls)
2. Inner summation
   (8 rot, 8 add)
3. Masking
   (4 mul)
4. Compaction:
   (3 rot, 3 add)

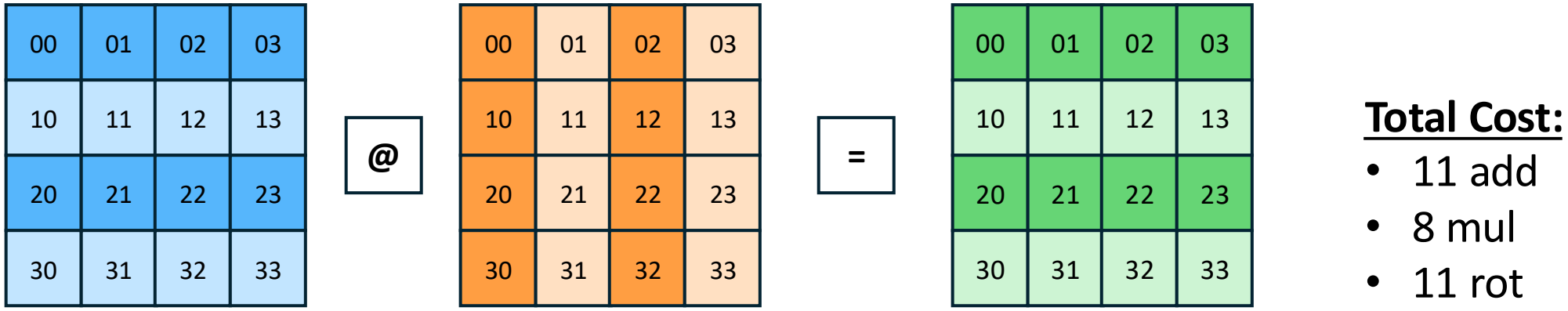


4. Compact Ciphertexts Together

1. Multiply rows and columns
   (4 muls)
2. Inner summation
   (8 rot, 8 add)
3. Masking
   (4 mul)
4. Compaction:
   (3 rot, 3 add)



Prior Work: Cost of Matrix Multiplication



Total Cost:

- 11 add
- 8 mul
- 11 rot

Problem: Inner Summation is Expensive

Requires data movement

- (<<) Rotations to align values to sum
- (×) Leaves garbage values that need to be masked out

Rotom Finds A Better Layout Assignment

- Summation *across the ciphertexts*
- Avoid rotation-and-summation within ciphertexts

Rotom Finds A Better Layout Assignment

- Summation *<u>across the ciphertexts</u>*
- Avoid rotation-and-summation within ciphertexts

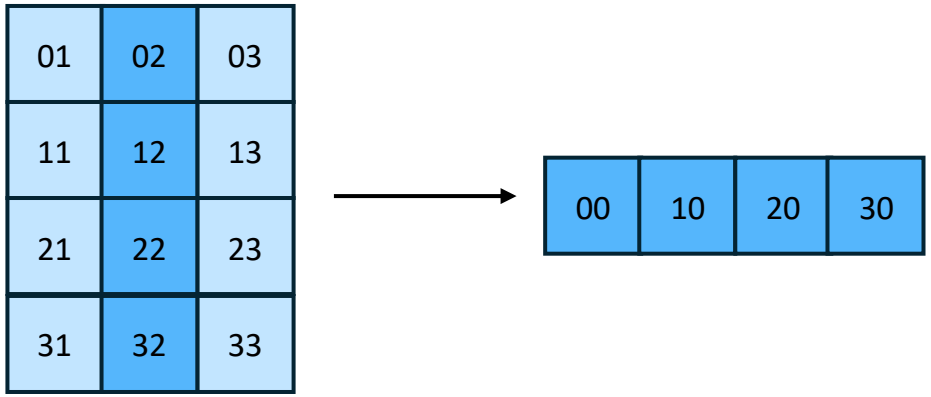
1. Start with simple layouts: row and column
2. Use ApplyRoll to diagonalize layouts
   - Rotom introduces this novel layout conversion operator

Rotom: Assign A – Column-major Layout

00	01	02	03
10	11	12	13
20	21	22	23
30	31	32	33

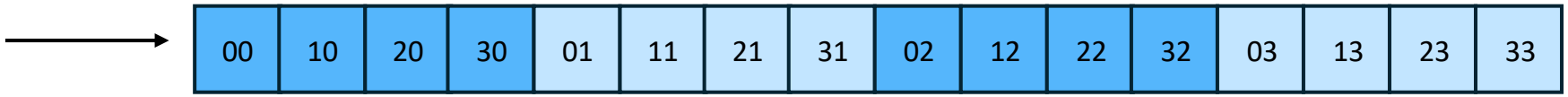
A: Ciphertext

Rotom: Assign A – Column-major Layout

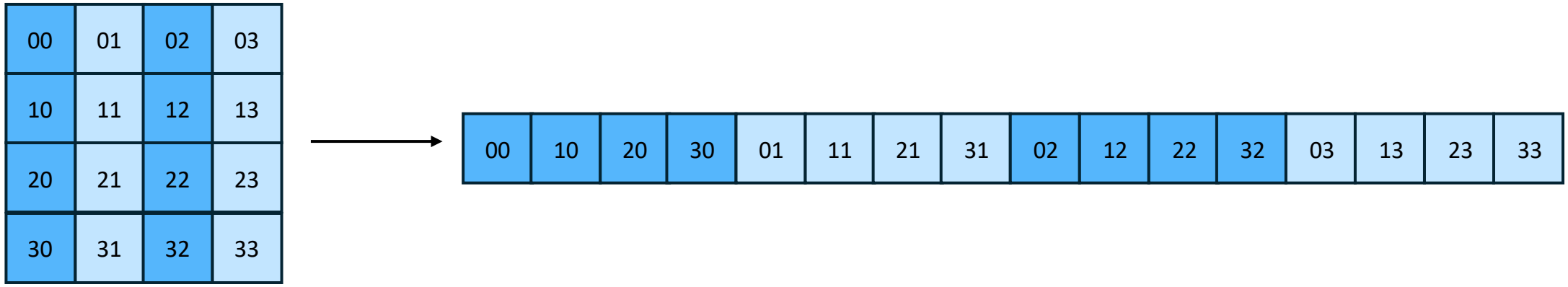


A: Ciphertext

Rotom: Assign A – Column-major Layout



Rotom: Assign A – Column-major Layout



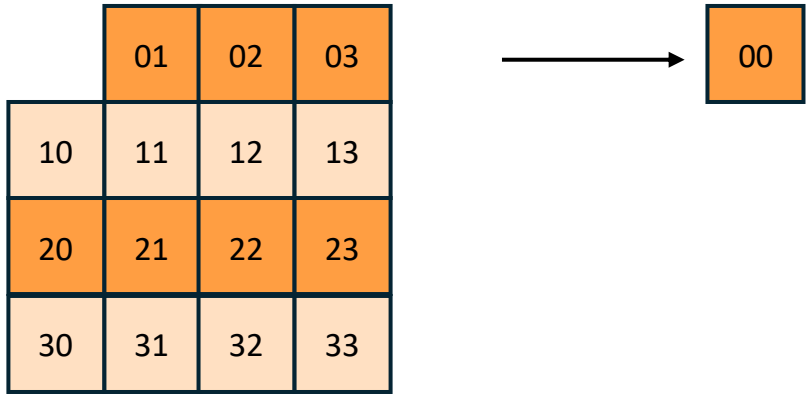
A: Ciphertext

Rotom: Assign B – Repeated Row Layout

00	01	02	03
10	11	12	13
20	21	22	23
30	31	32	33

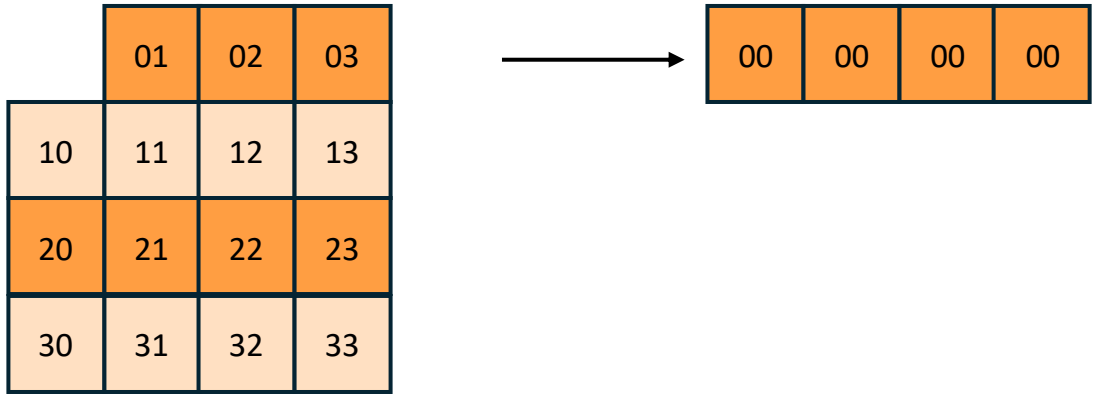
B: Plaintext

Rotom: Assign B – Repeated Row Layout



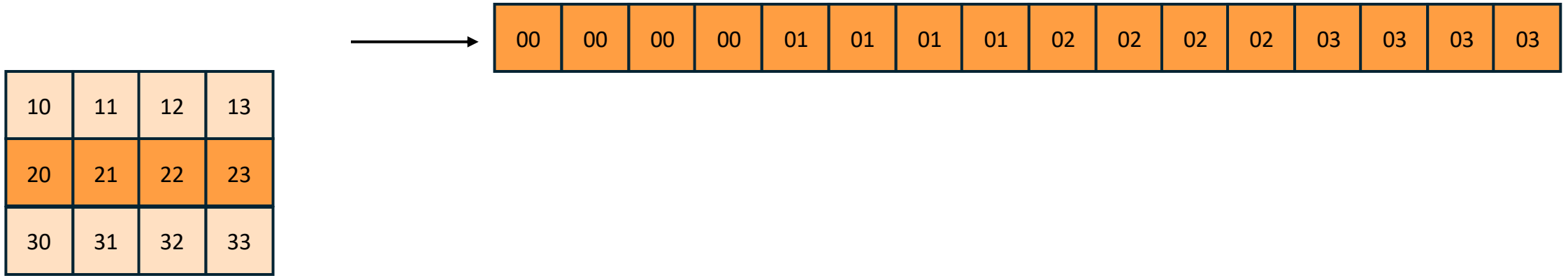
B: Plaintext

Rotom: Assign B – Repeated Row Layout

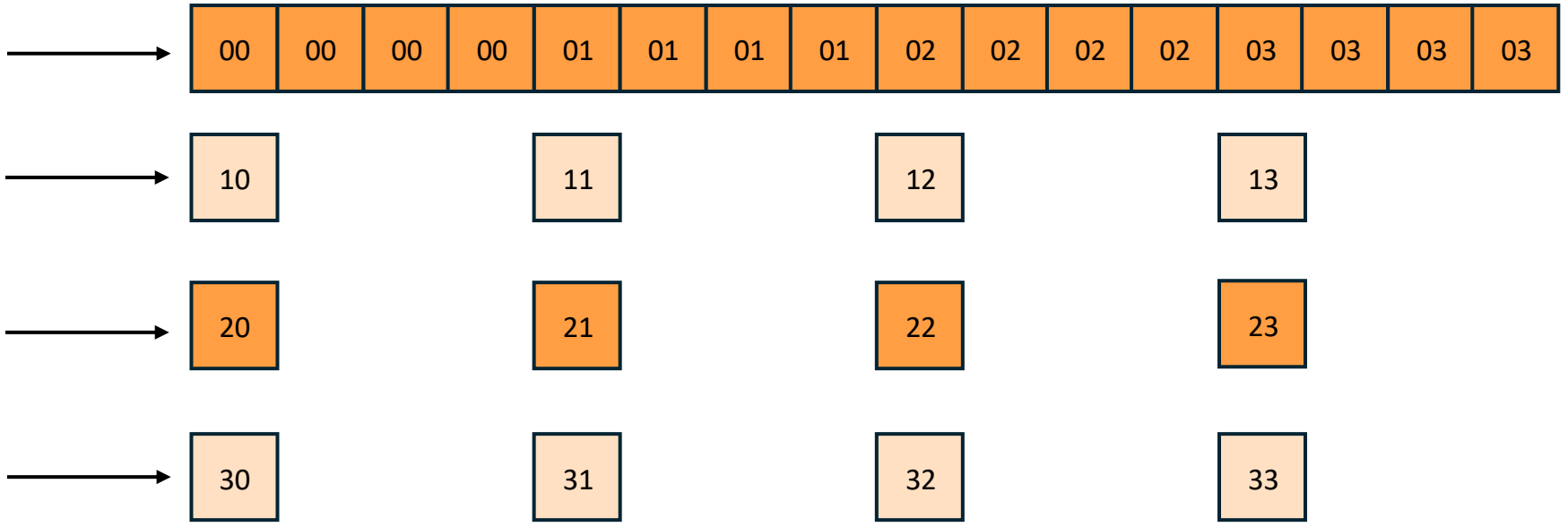


B: Plaintext

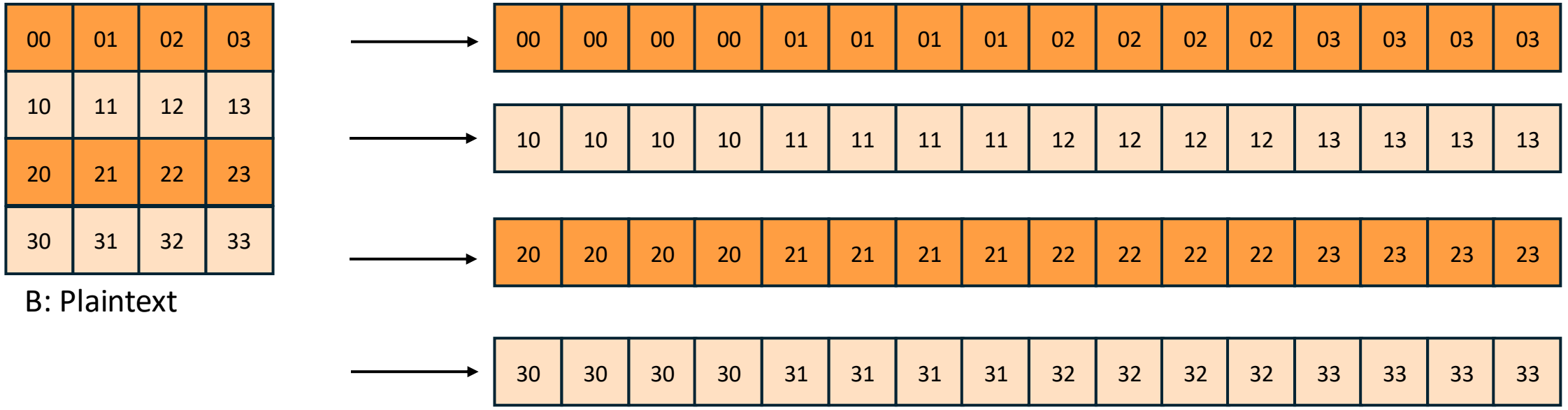
Rotom: Assign B – Repeated Row Layout



Rotom: Assign B – Repeated Row Layout

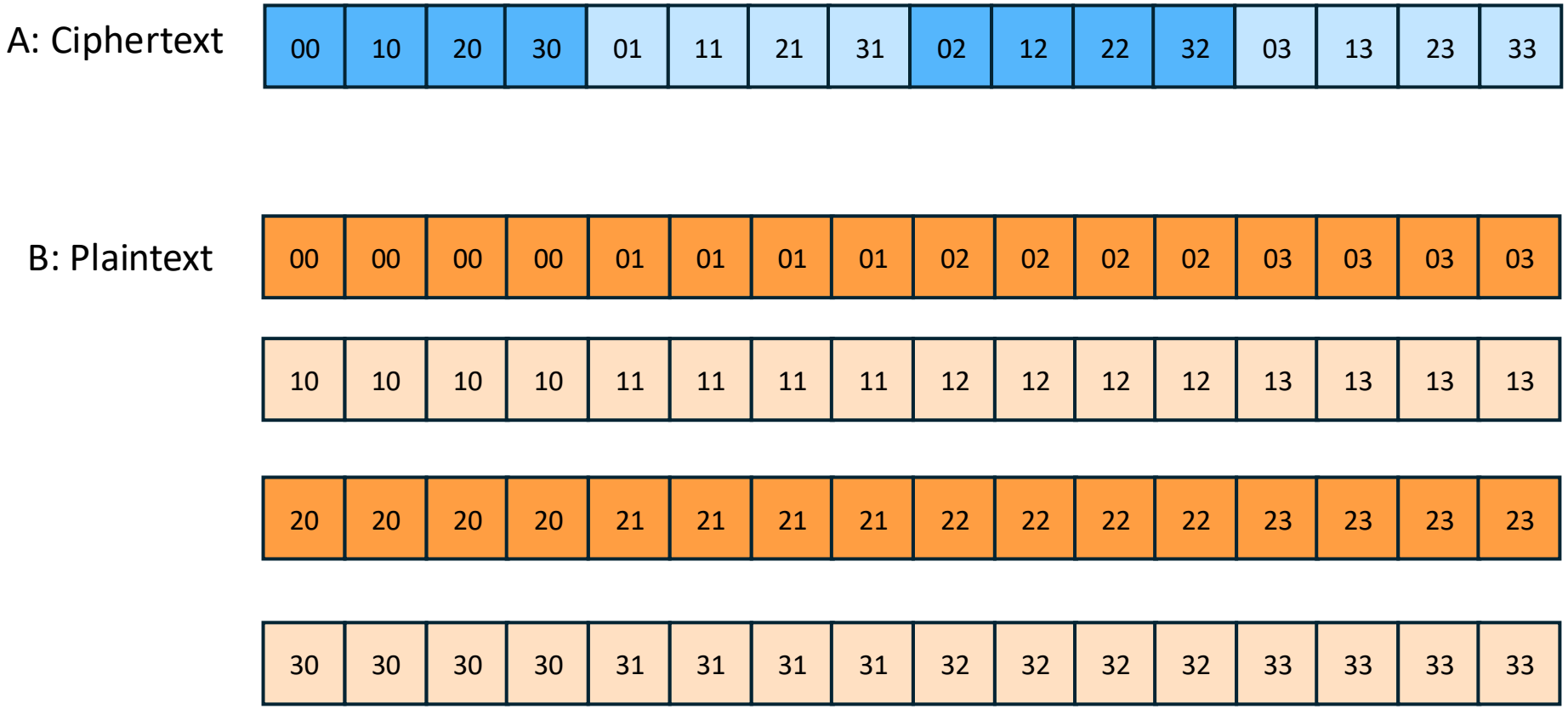


Rotom: Assign B – Repeated Row Layout

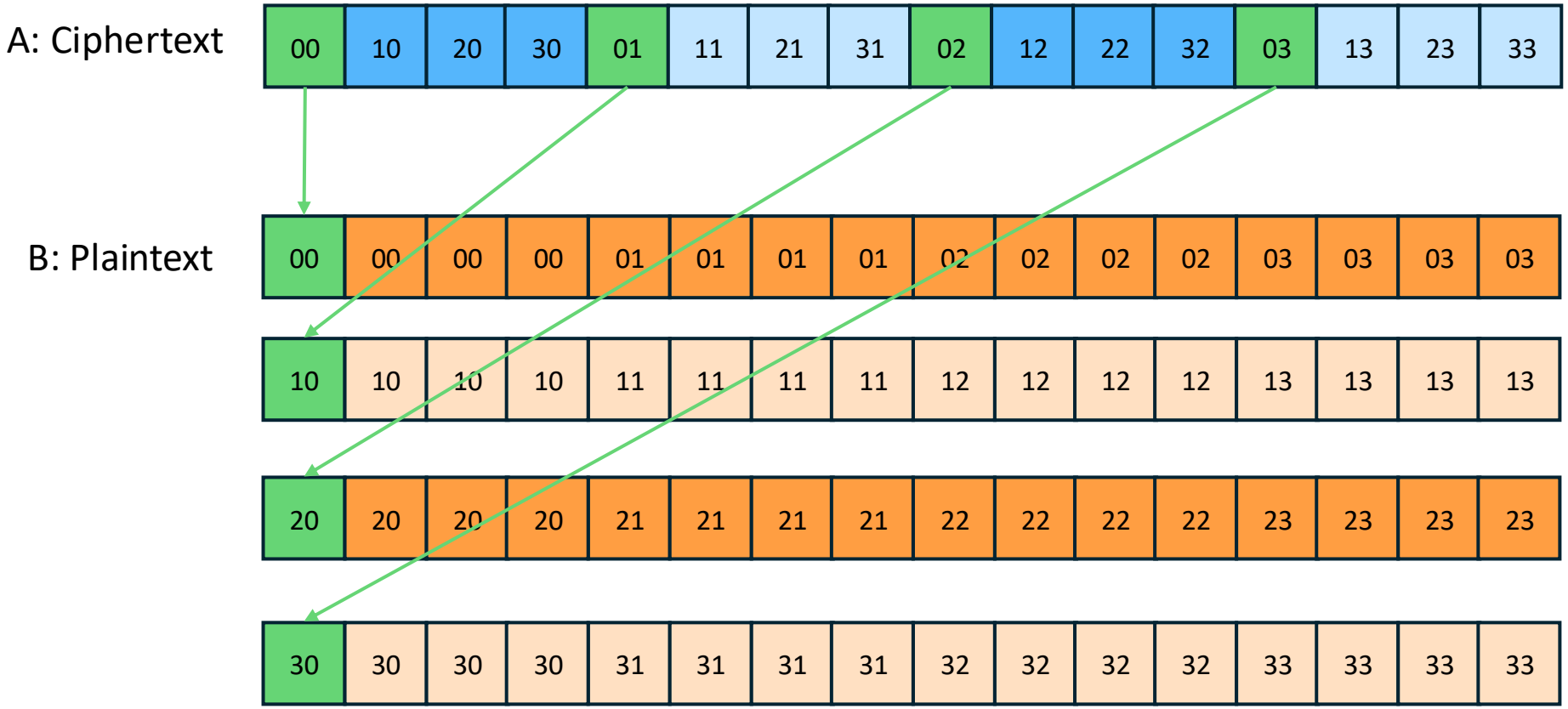


B: Plaintext

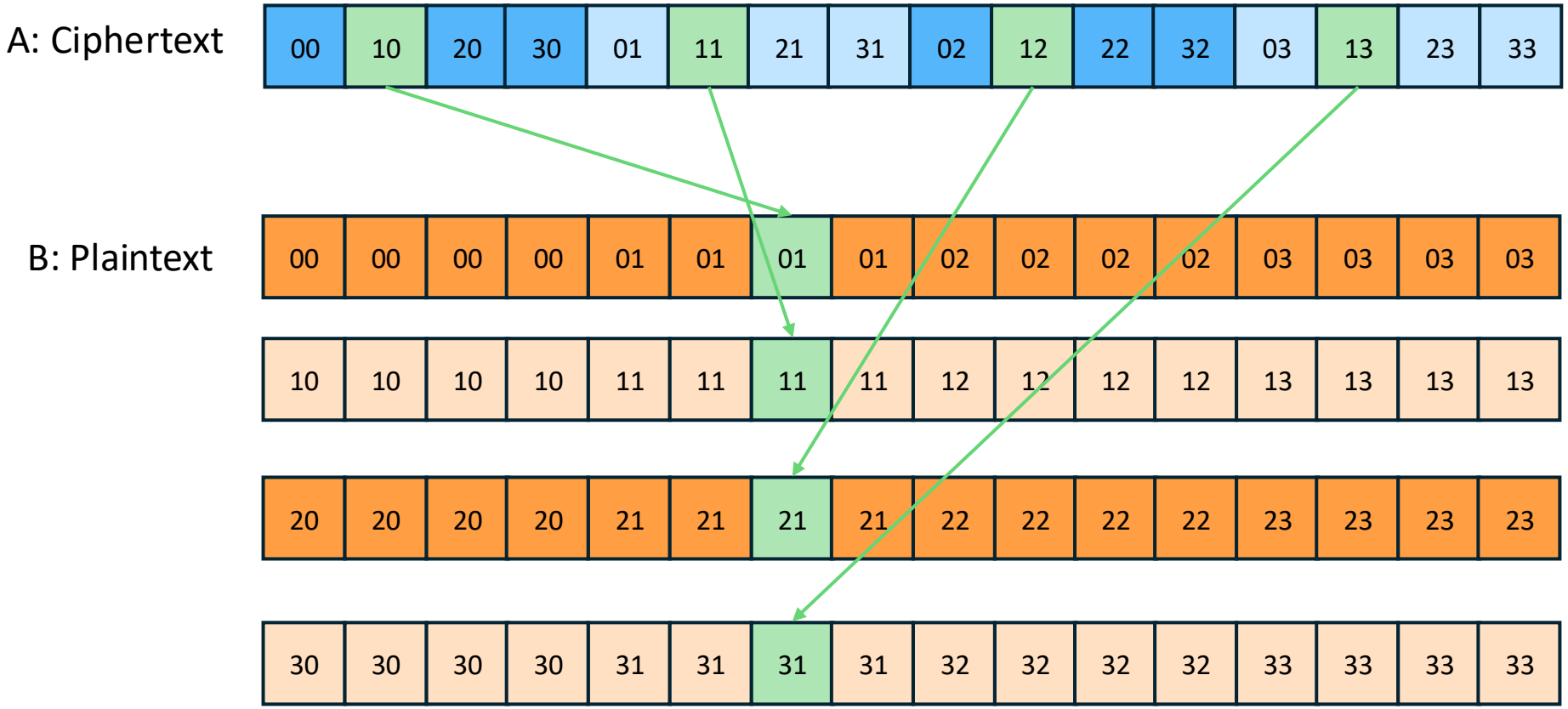
Column × Row *Doesn't* Align Summation



Column × Row *Doesn't* Align Summation

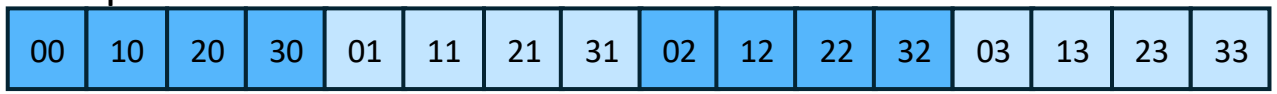


Column × Row *Doesn't* Align Summation



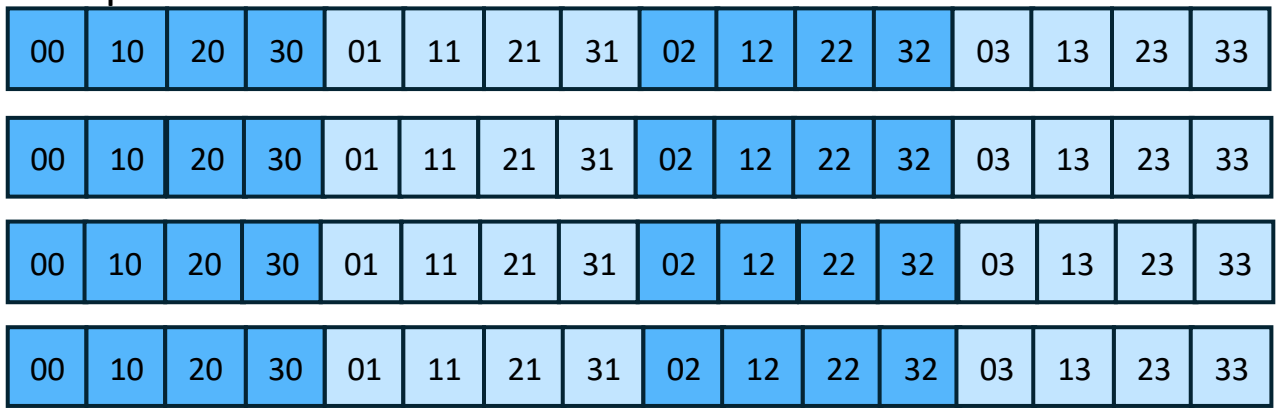
1. ApplyRoll – Diagonalize A

A: Ciphertext



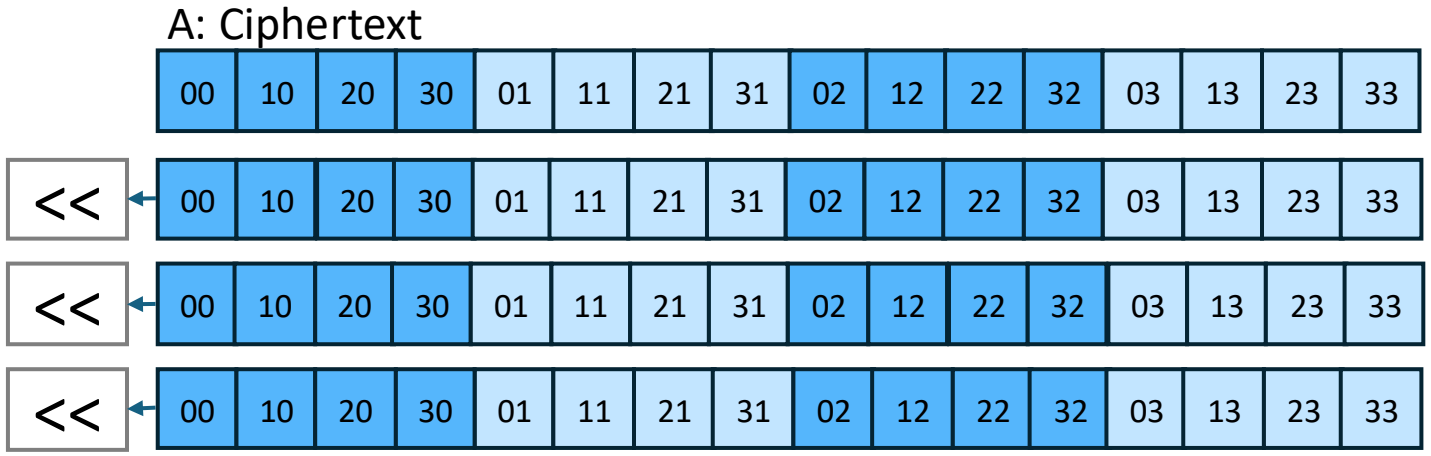
1. ApplyRoll – Diagonalize A

A: Ciphertext

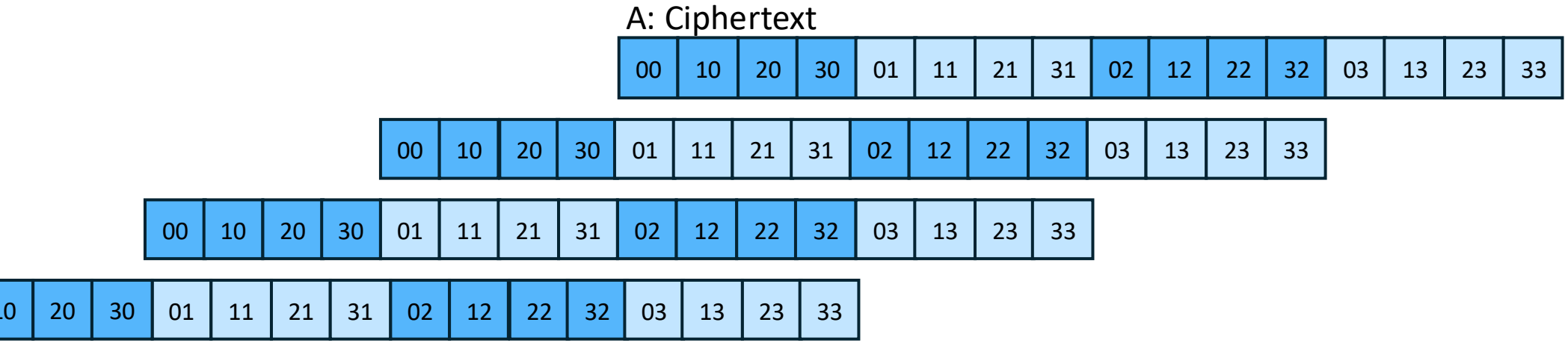


1. ApplyRoll – Diagonalize A

1. ApplyRoll to A (3 rot)



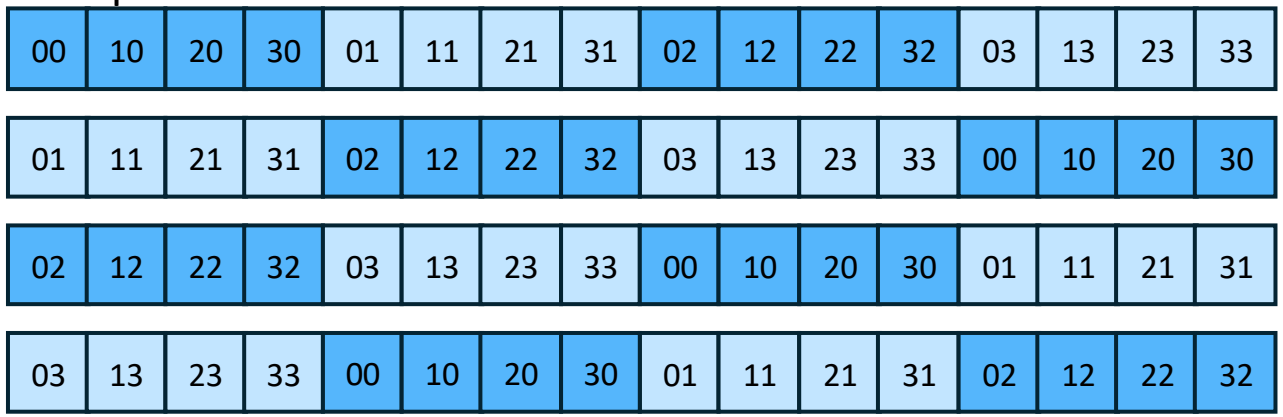
1. ApplyRoll – Diagonalize A



1. ApplyRoll – Diagonalize A

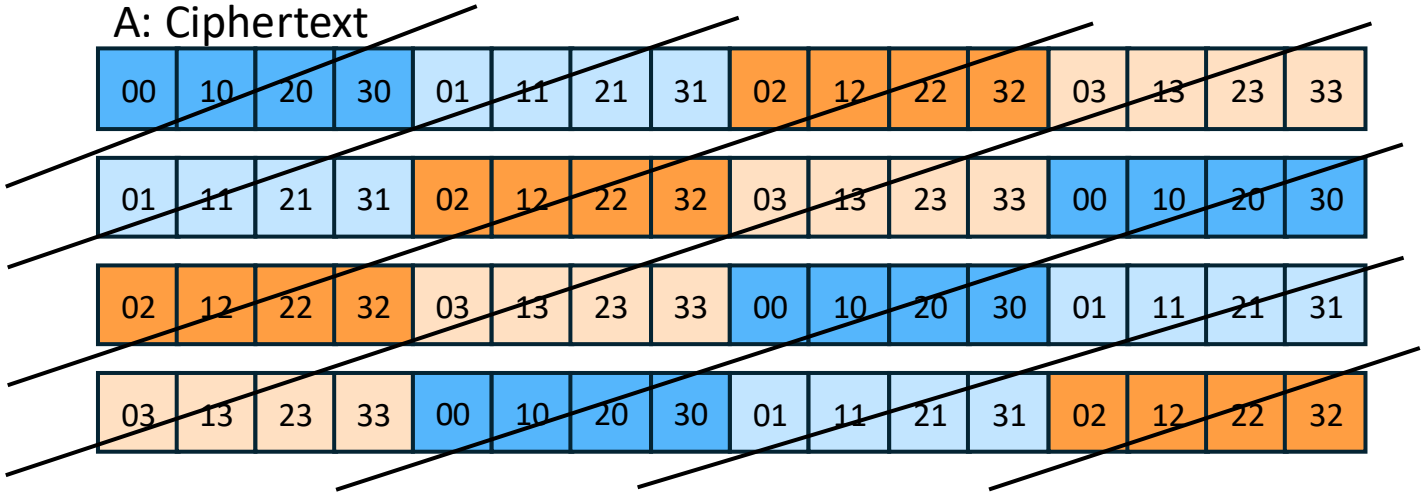
1. ApplyRoll to A (3 rot)

A: Ciphertext



1. ApplyRoll – Diagonalize A

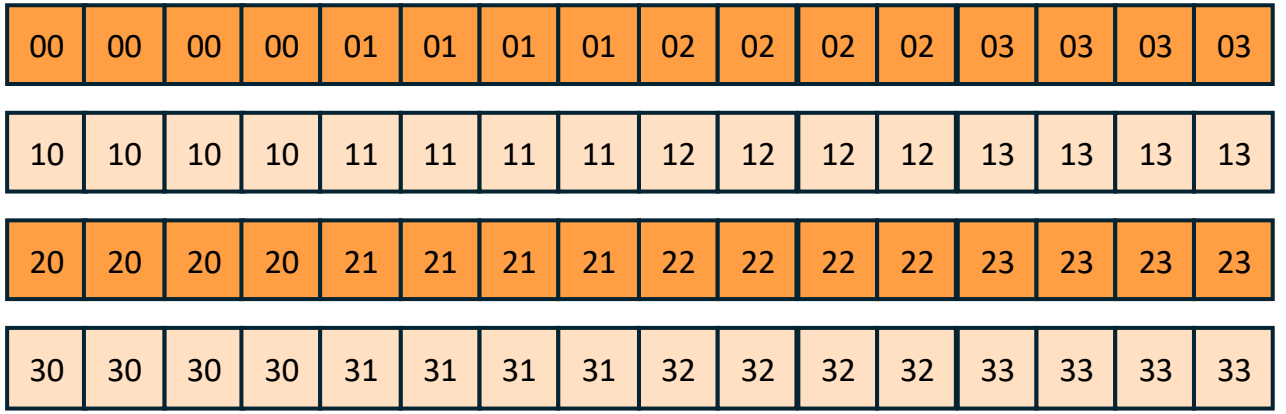
1. ApplyRoll to A (3 rot)



2. ApplyRoll – Diagonalize B

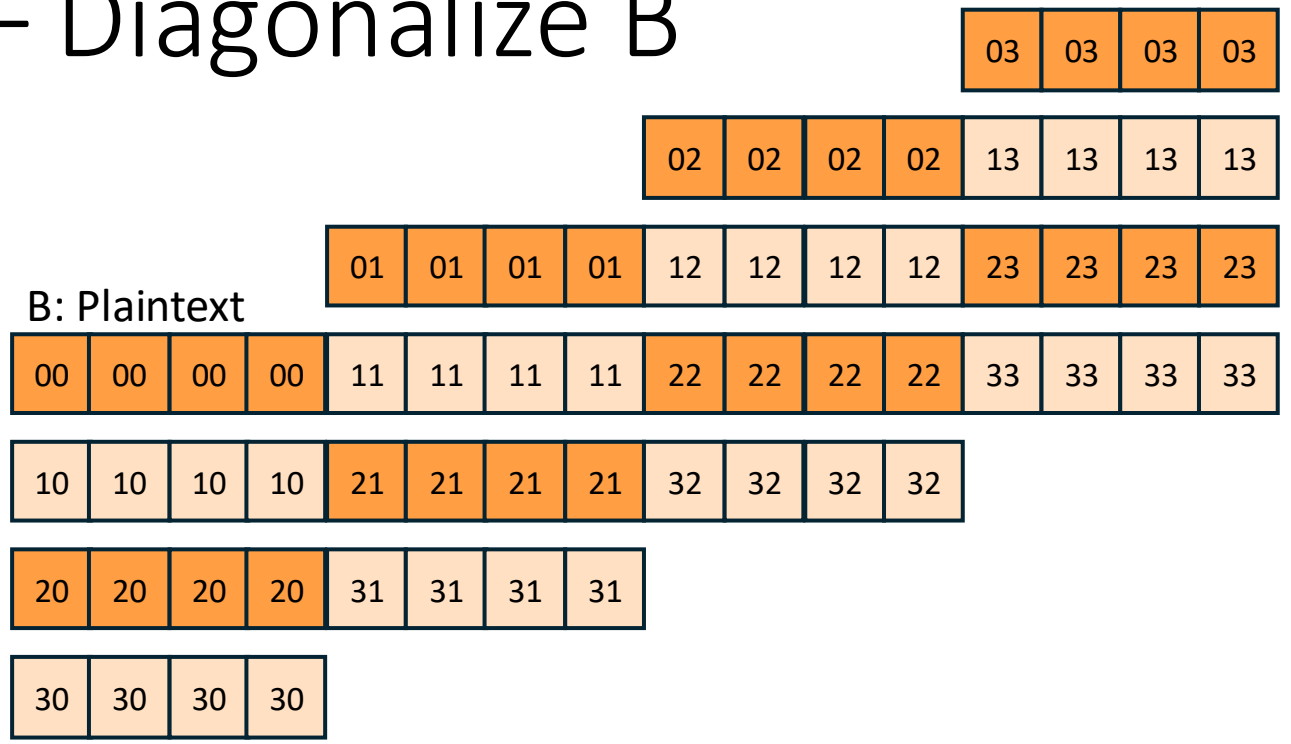
1. ApplyRoll to A
   (3 rot)

B: Plaintext



2. ApplyRoll – Diagonalize B

1. ApplyRoll to A (3 rot)
2. ApplyRoll to B (Free)



2. ApplyRoll – Diagonalize B

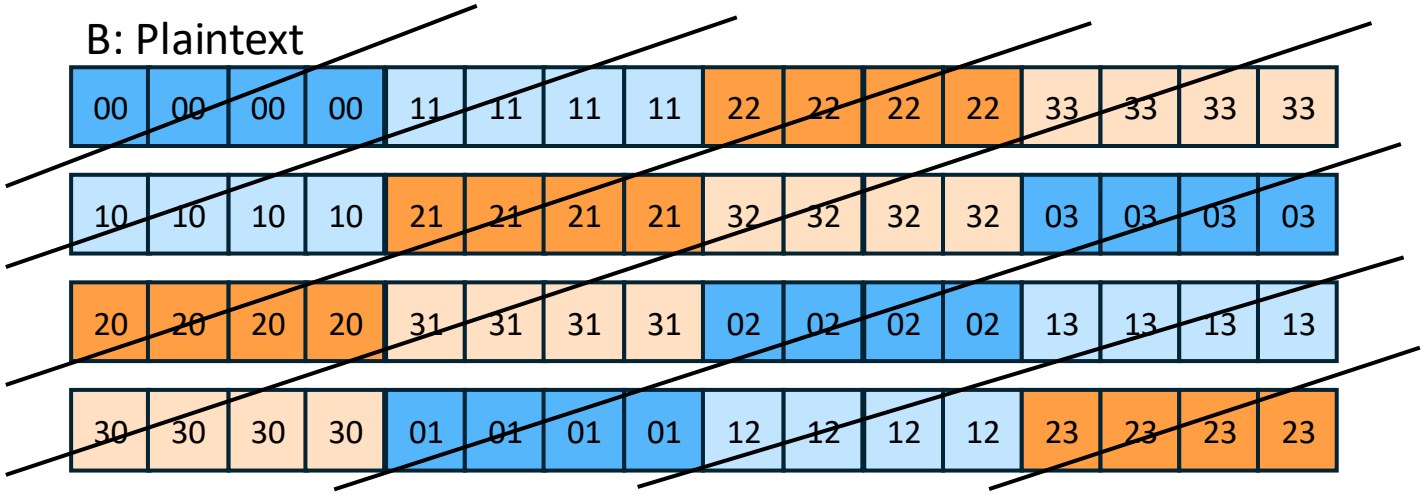
1. ApplyRoll to A (3 rot)
2. ApplyRoll to B (Free)

B: Plaintext

00	00	00	00	11	11	11	11	22	22	22	22	33	33	33	33
10	10	10	10	21	21	21	21	32	32	32	32	03	03	03	03
20	20	20	20	31	31	31	31	02	02	02	02	13	13	13	13
30	30	30	30	01	01	01	01	12	12	12	12	23	23	23	23

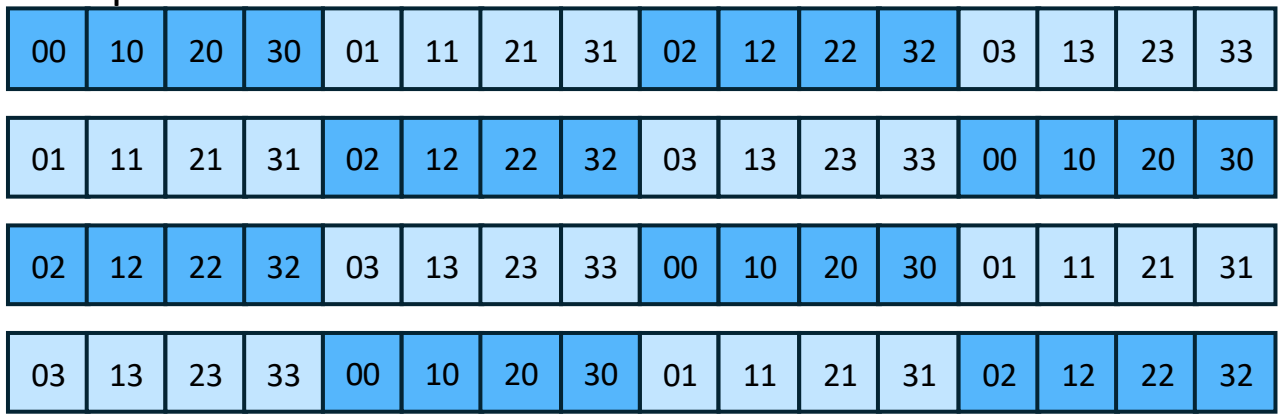
2. ApplyRoll – Diagonalize B

1. ApplyRoll to A (3 rot)
2. ApplyRoll to B (Free)

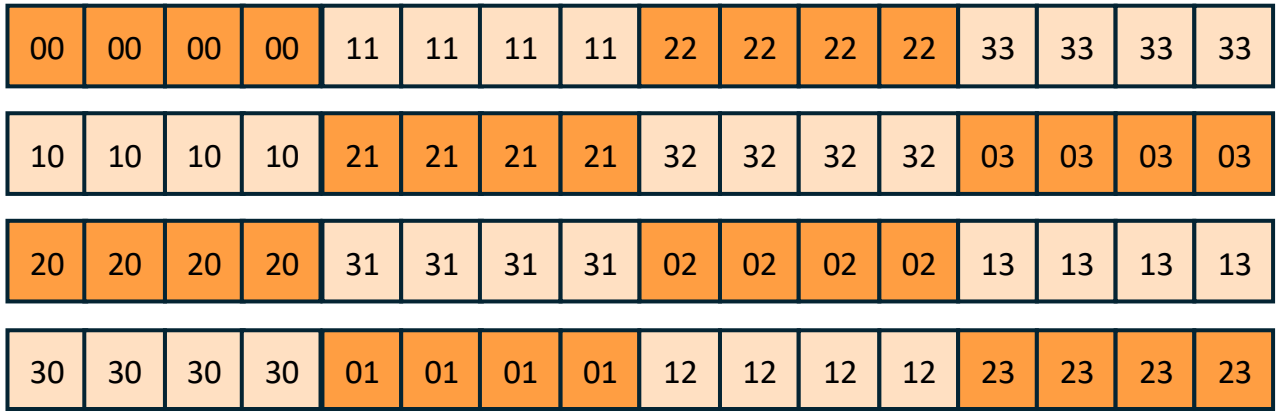


Diagonalization Aligns Summation

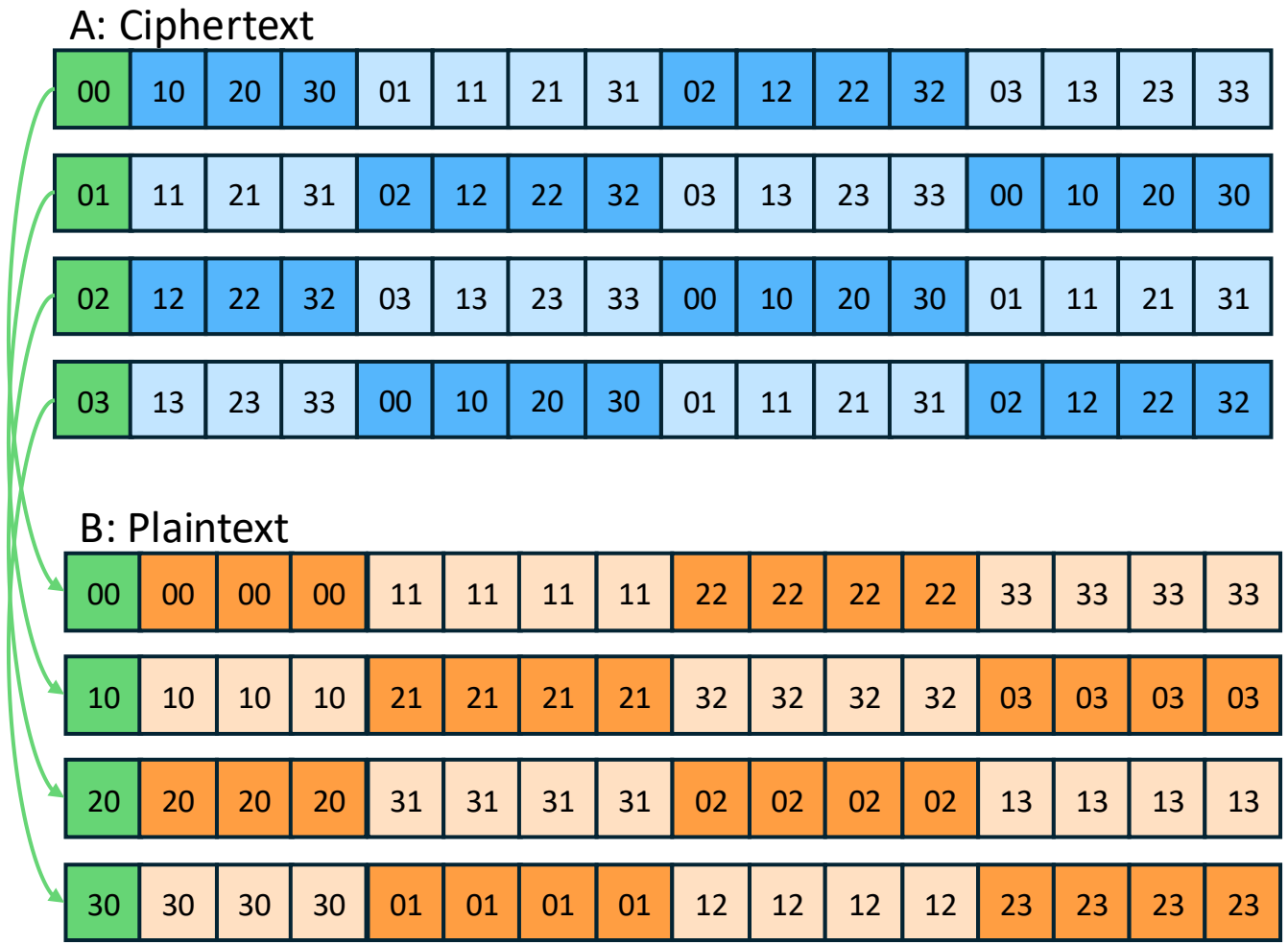
A: Ciphertext



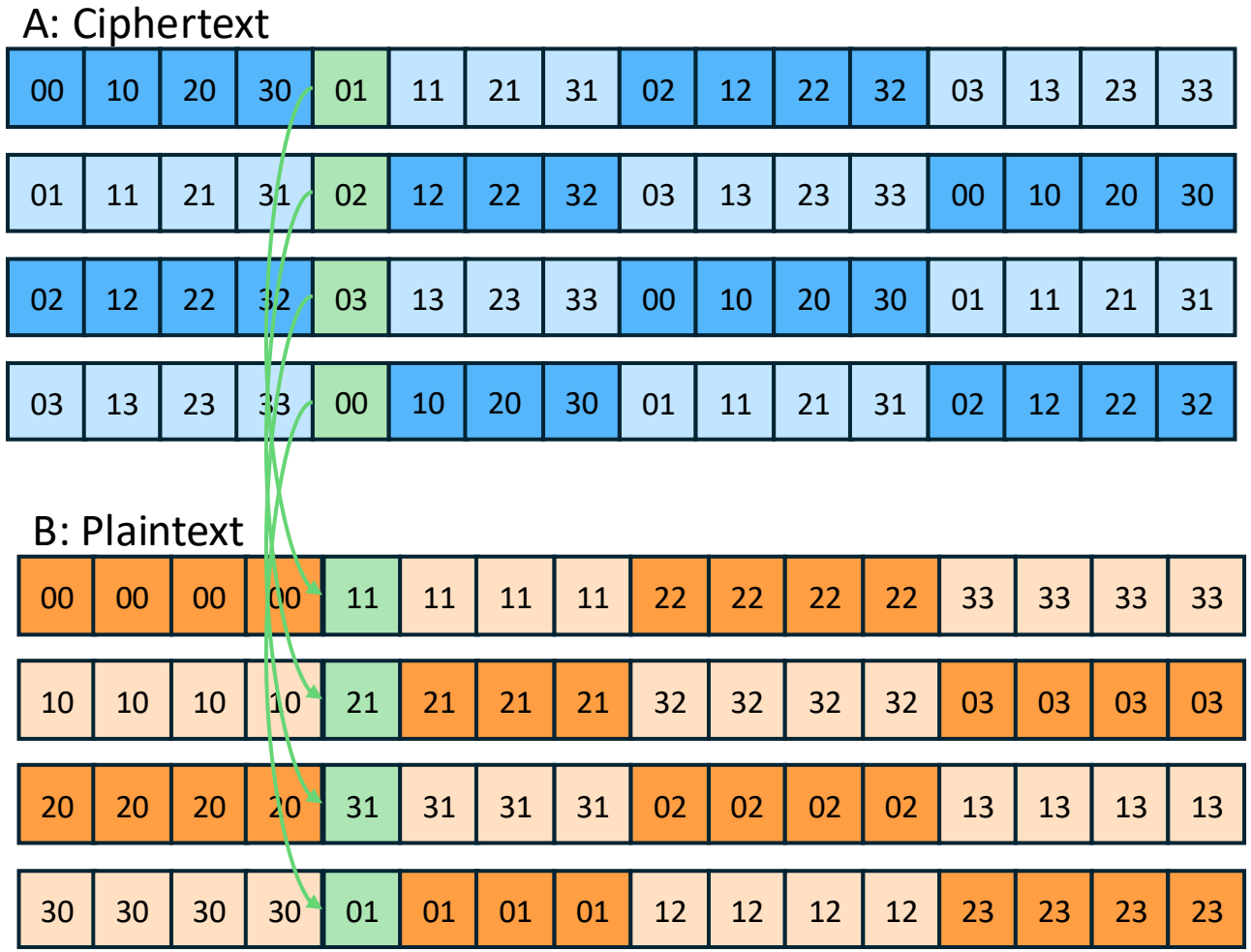
B: Plaintext



Diagonalization Aligns Summation



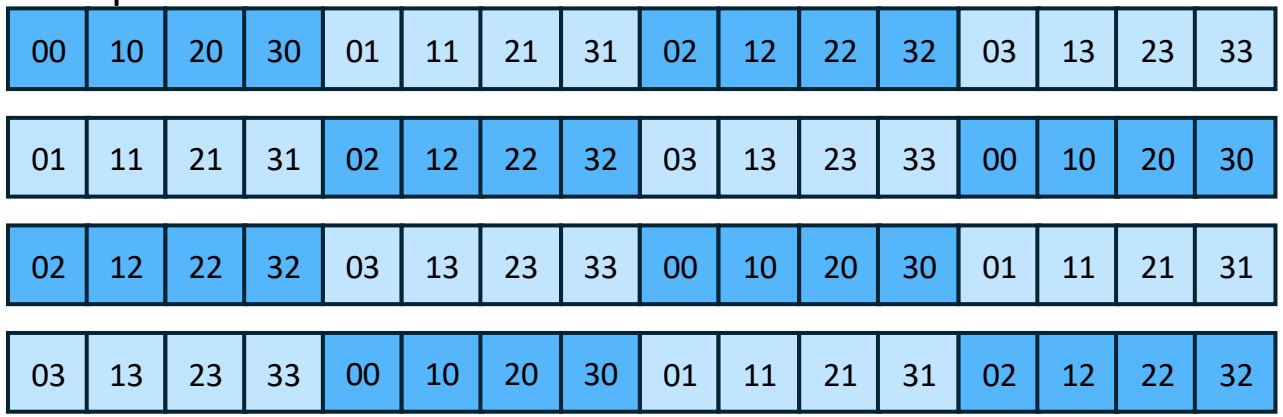
Diagonalization Aligns Summation



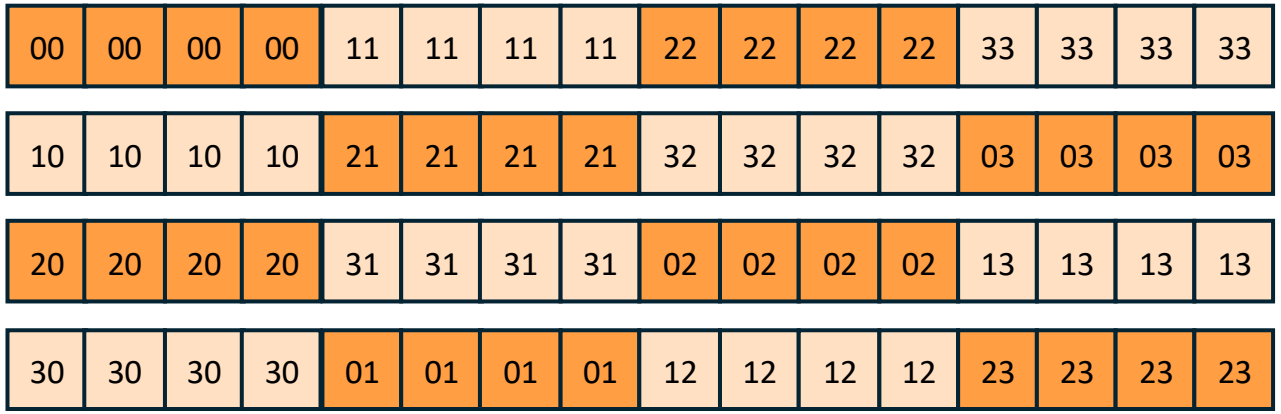
3. Multiply A and B

1. ApplyRoll to A (3 rot)
2. ApplyRoll to B (Free)
3. Multiply (4 mul)

A: Ciphertext

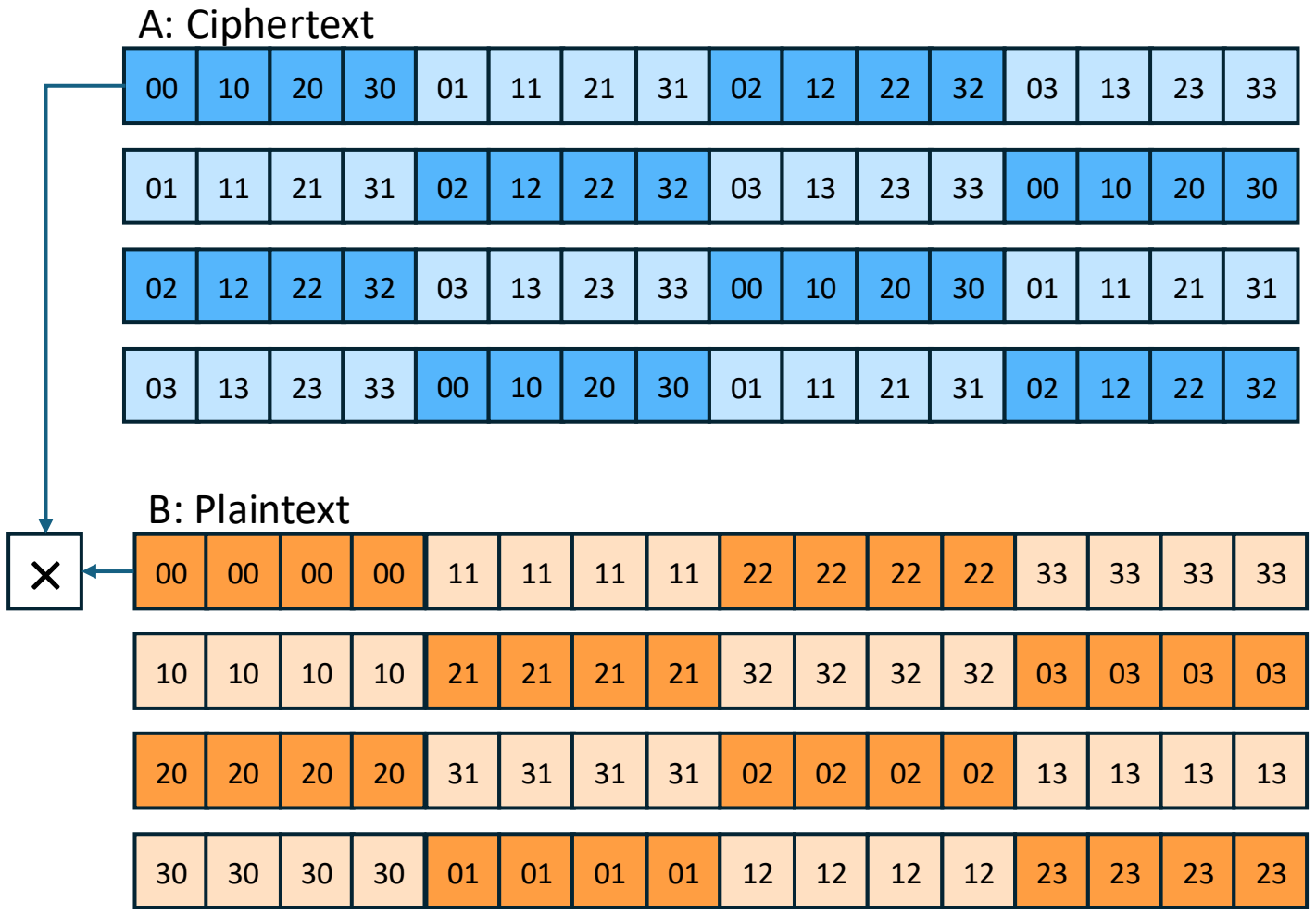


B: Plaintext



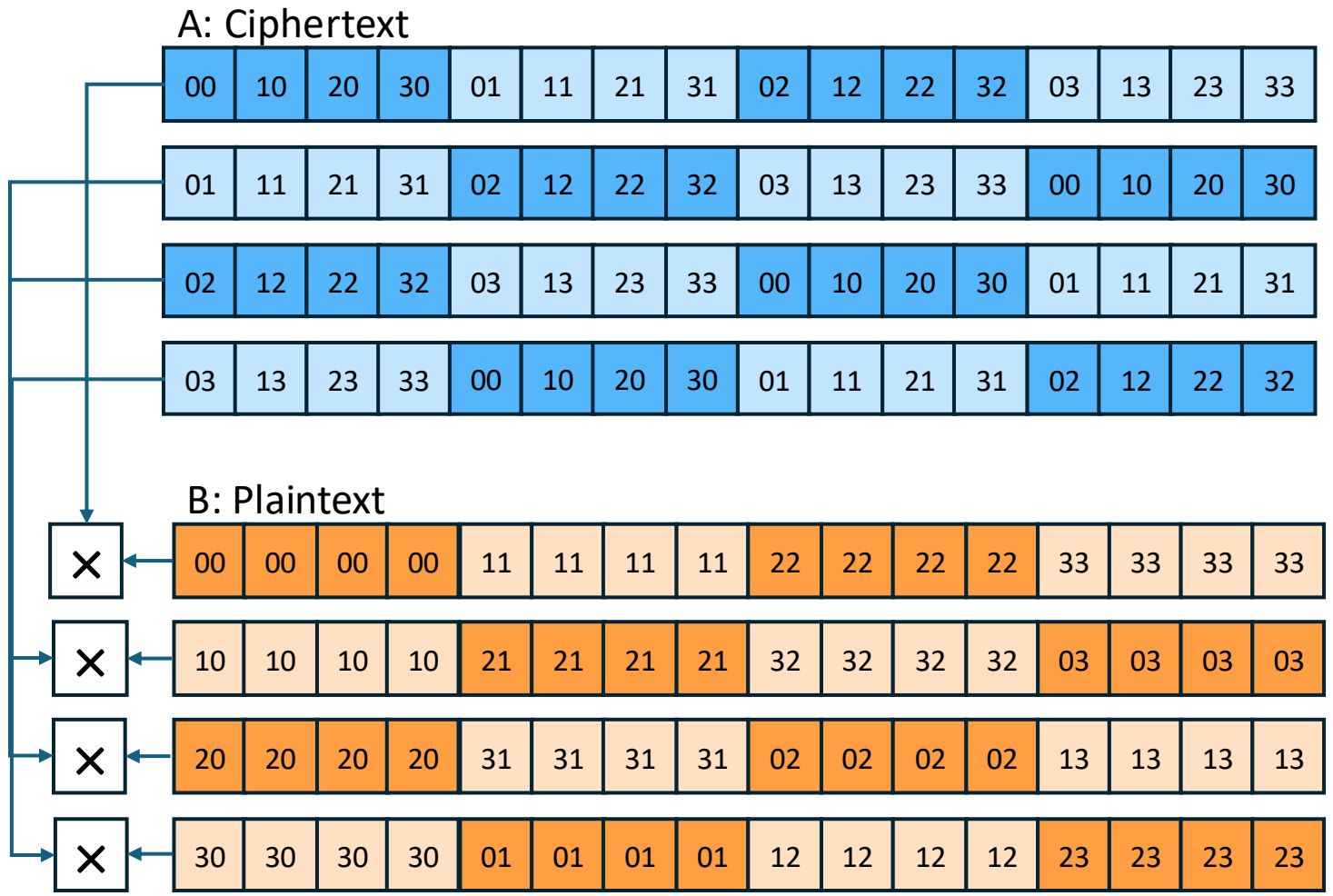
3. Multiply A and B

1. ApplyRoll to A (3 rot)
2. ApplyRoll to B (Free)
3. Multiply (4 mul)



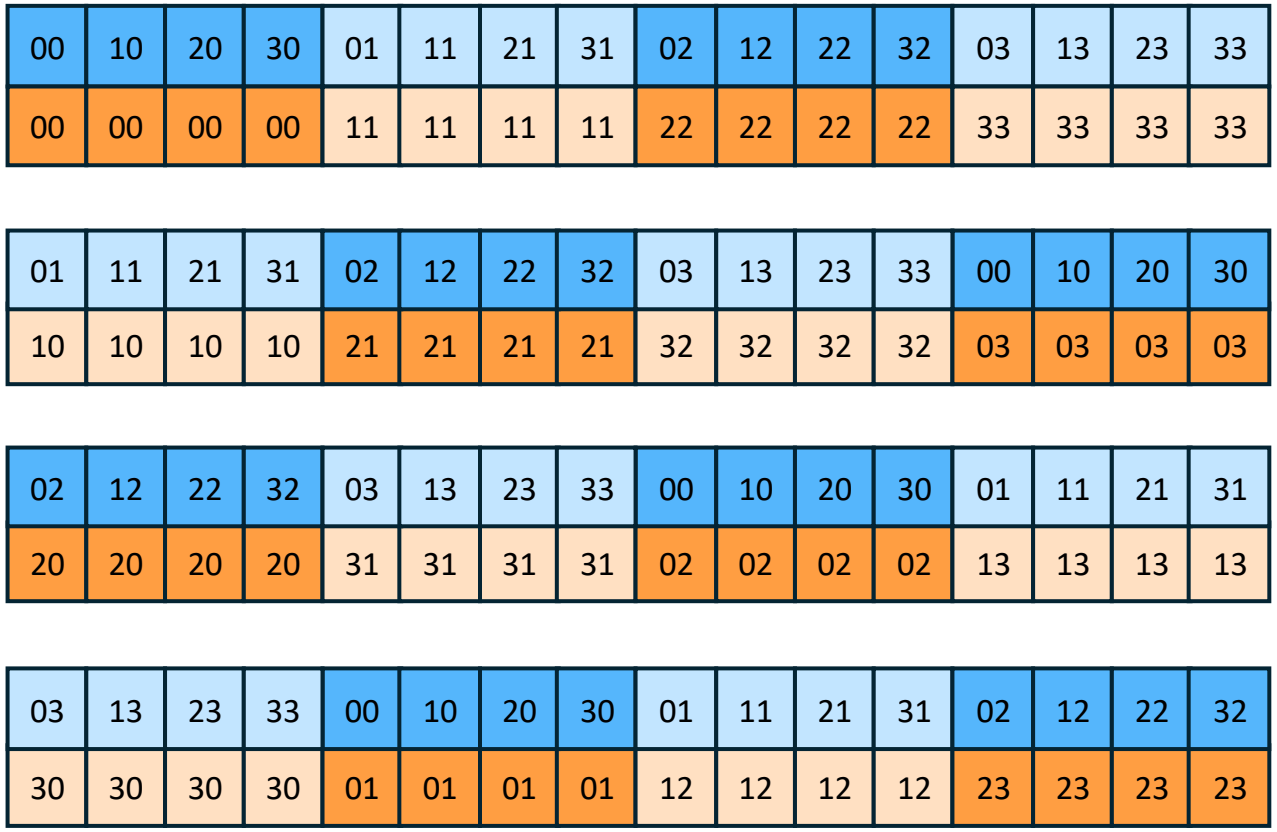
3. Multiply A and B

1. ApplyRoll to A (3 rot)
2. ApplyRoll to B (Free)
3. Multiply (4 mul)



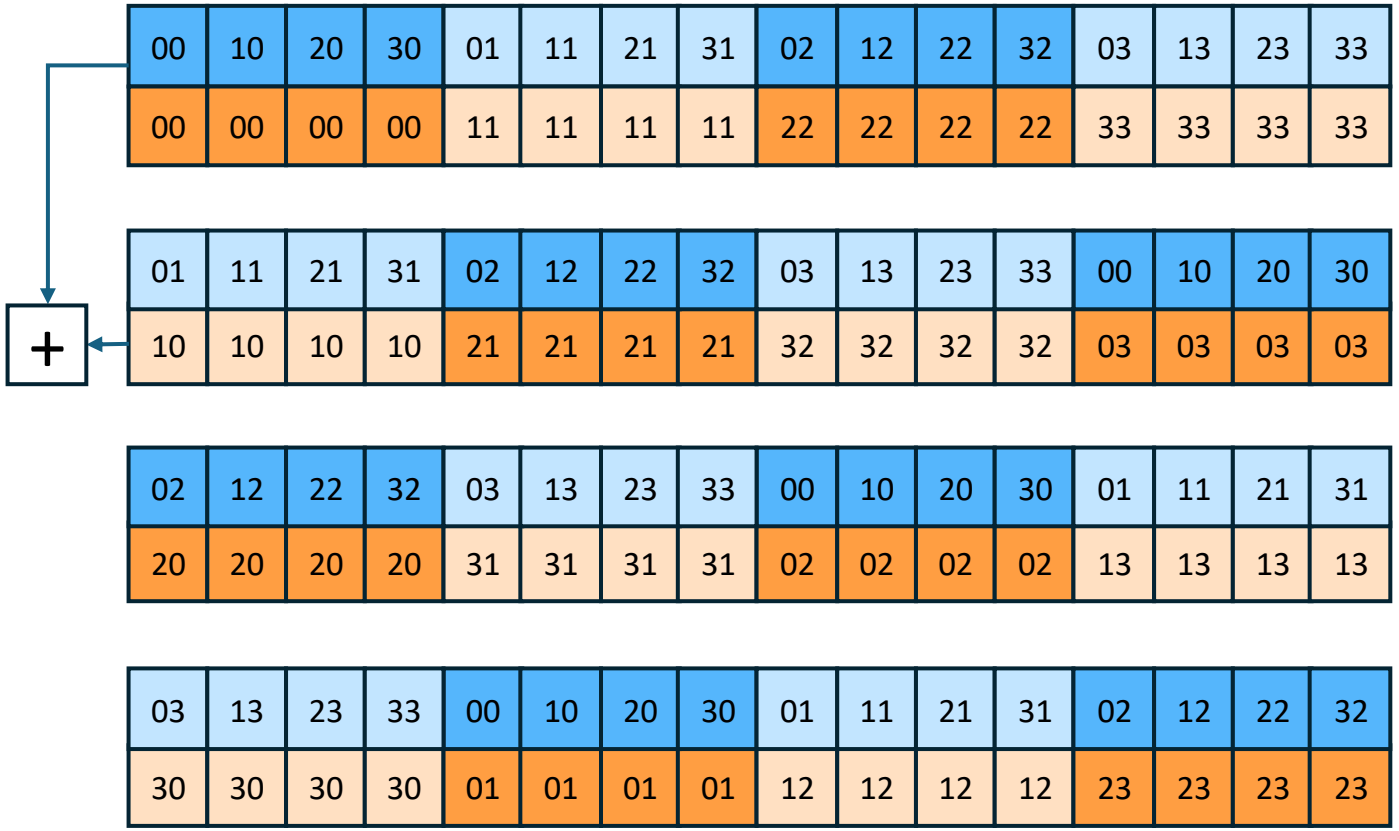
4. Sum Across Ciphertexts

1. ApplyRoll to A (3 rot)
2. ApplyRoll to B (Free)
3. Multiply (4 mul)



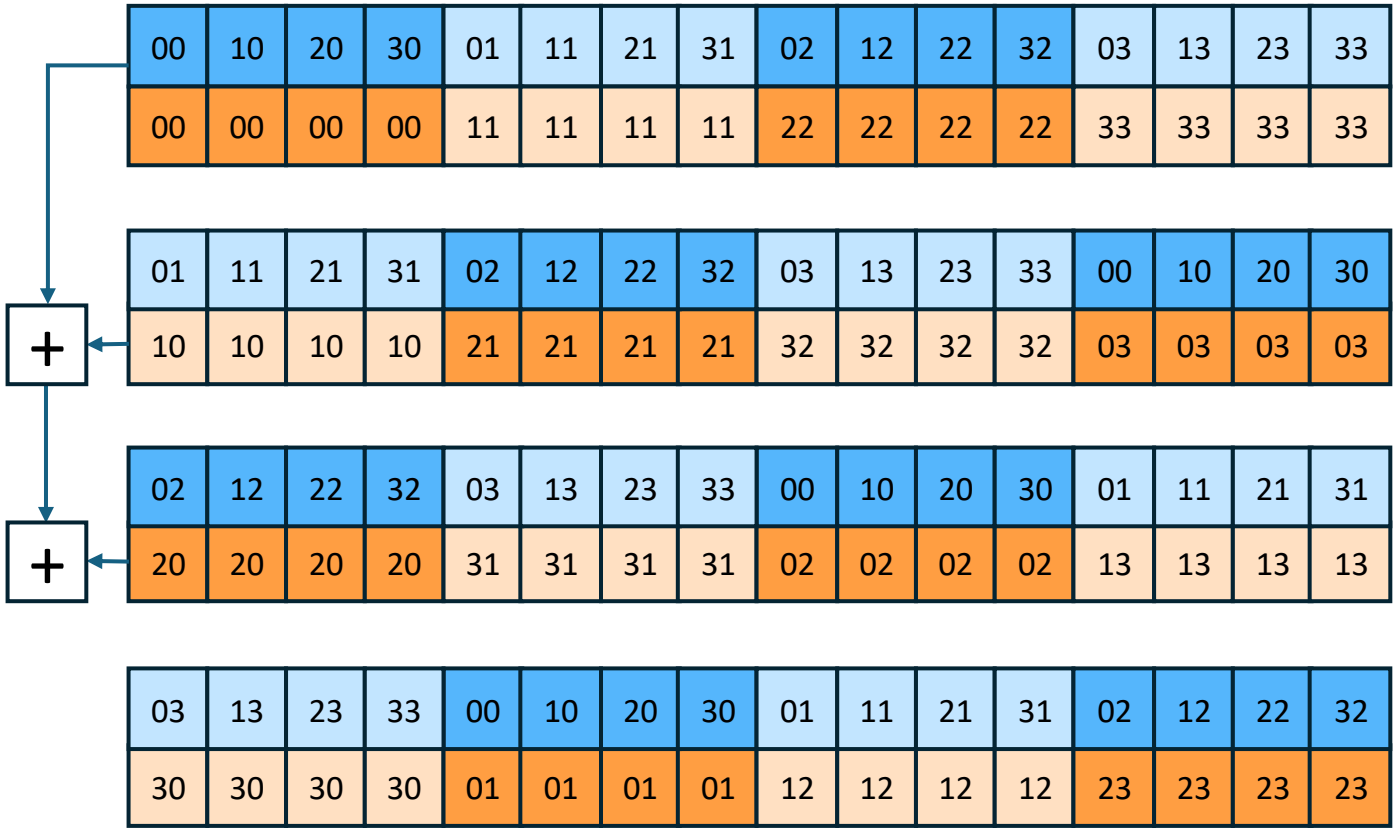
4. Sum Across Ciphertexts

1. ApplyRoll to A (3 rot)
2. ApplyRoll to B (Free)
3. Multiply (4 mul)



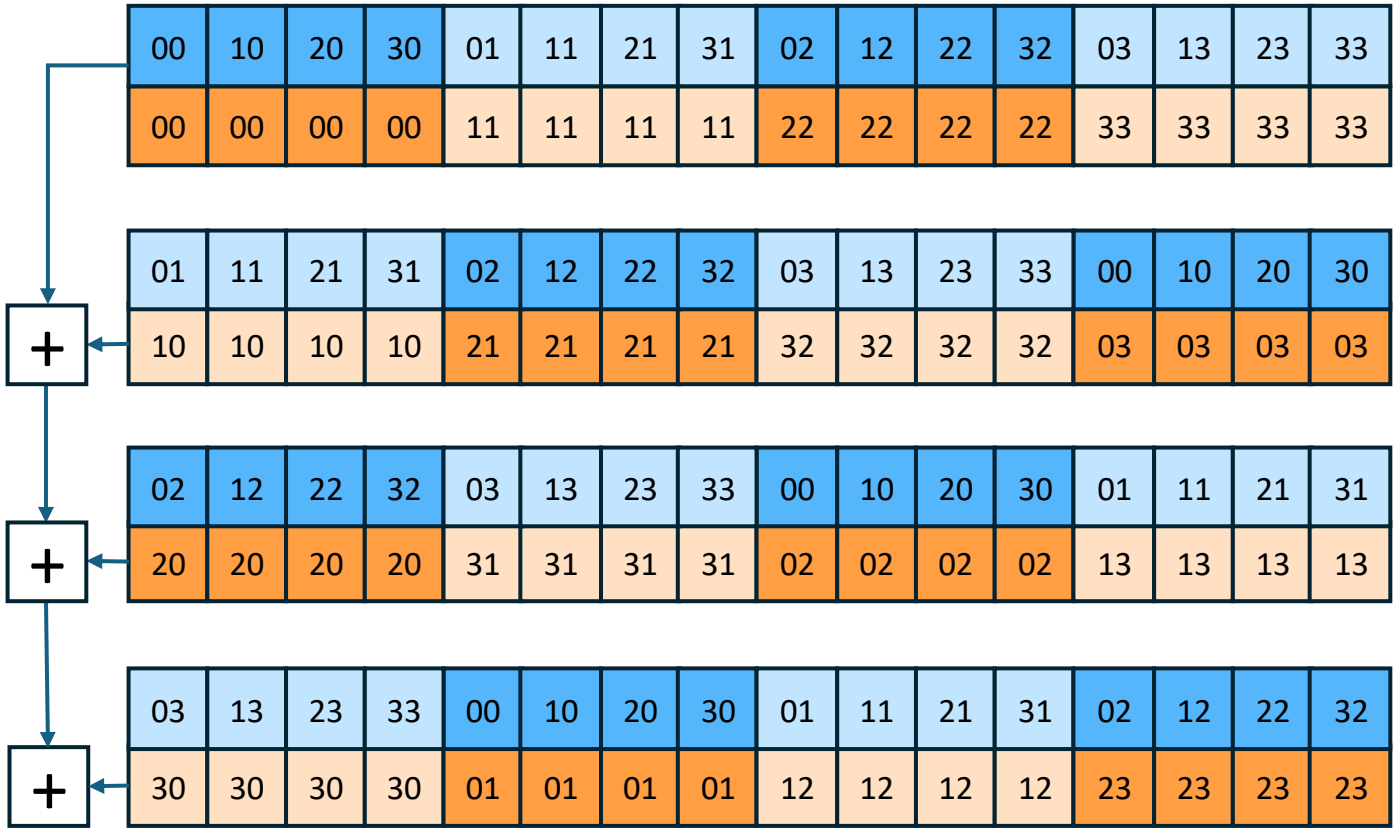
4. Sum Across Ciphertexts

1. ApplyRoll to A (3 rot)
2. ApplyRoll to B (Free)
3. Multiply (4 mul)



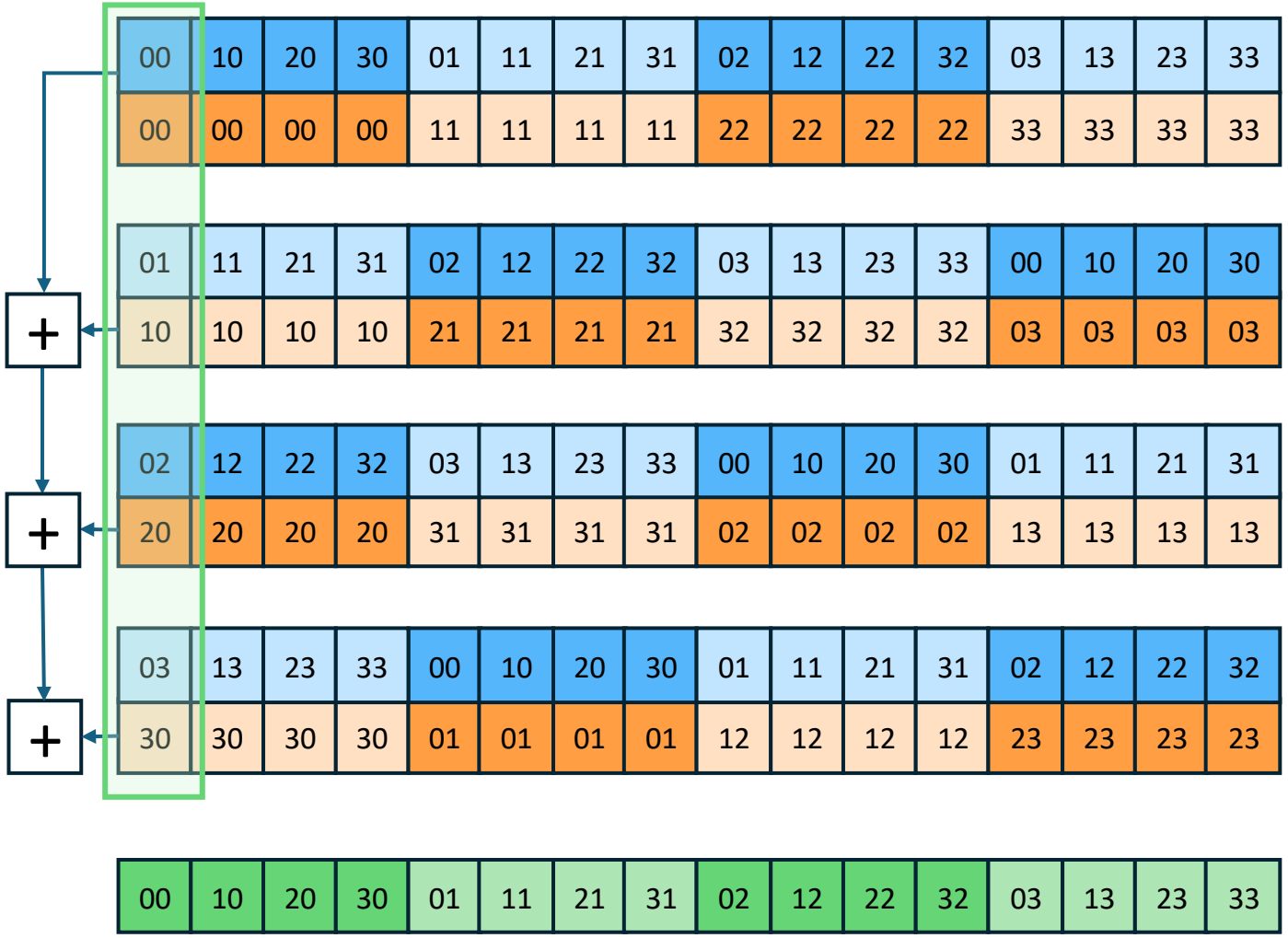
4. Sum Across Ciphertexts

1. ApplyRoll to A (3 rot)
2. ApplyRoll to B (Free)
3. Multiply (4 mul)

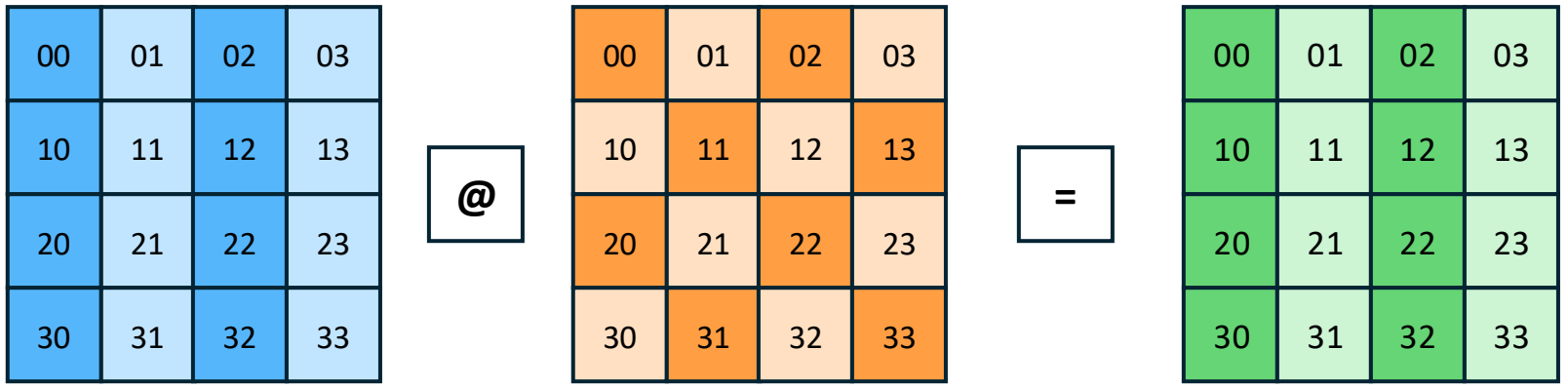


4. Sum Across Ciphertexts

1. ApplyRoll to A (3 rot)
2. ApplyRoll to B (Free)
3. Multiply (4 mul)



Rotom's Layout Assignment Beats Prior Work

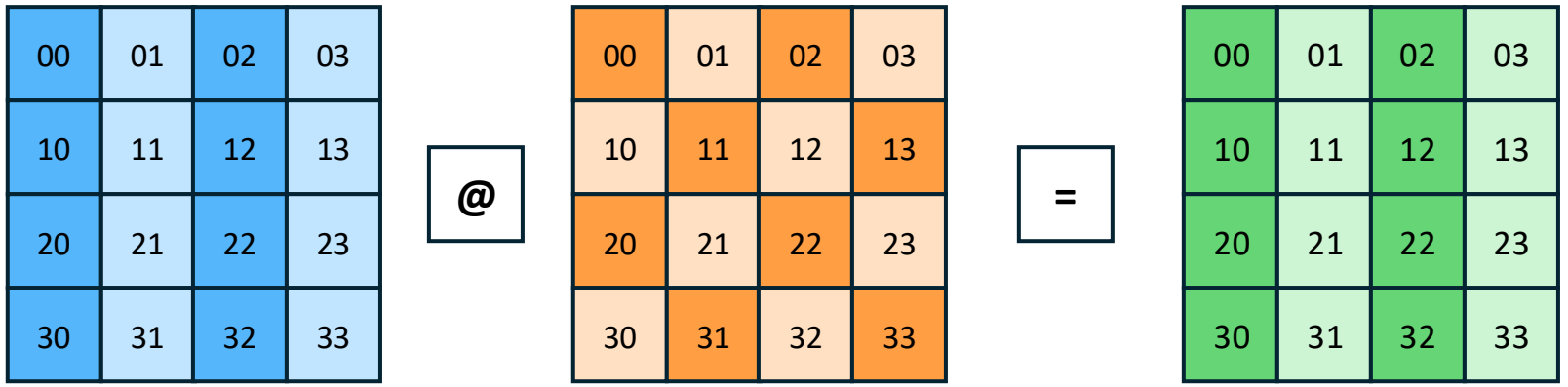


Rolled Column | Rolled Row | Column-major

Prior → Rotom:

- 11 → 3 add
- 8 → 4 mul
- 11 → 3 rot

Rotom's Layout Assignment Beats Prior Work



Rolled Column | Rolled Row | Column-major

Prior → Rotom:

- 11 → 3 add
- 8 → 4 mul
- 11 → 3 rot

In this example:

- ≈2× less multiplications (no masking)
- ≈3.6× less rotations (ApplyRoll)

Implementation







- Python Tensor DSL & PyTorch
- Targets OpenFHE and Google HEIR
- Interactive website:
  - https://cmu-cryptosystems.github.io/Rotom/

Evaluation



- Compared against Viaduct-HE[RM23] and Fhelipe[Kra+24]
- Six realistic tensor workloads

Evaluation





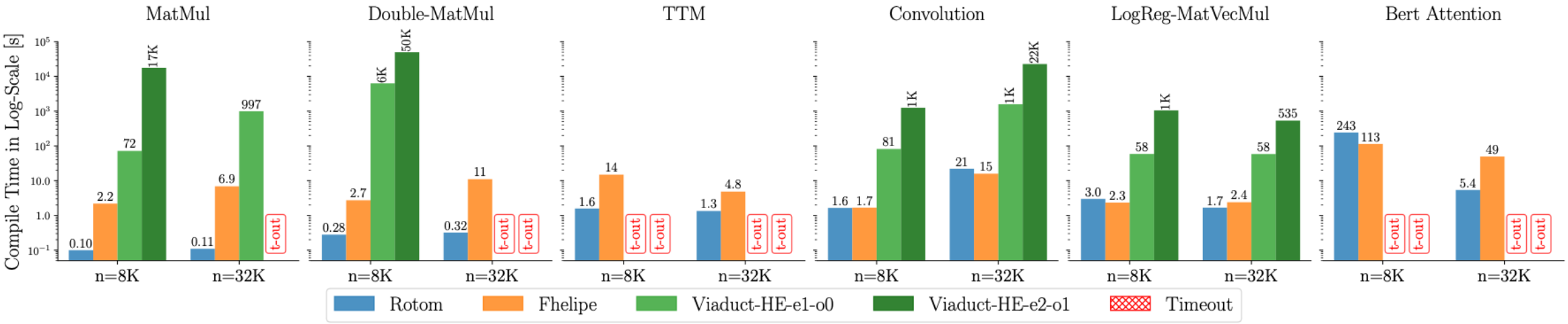


- Compared against Viaduct-HE[RM23] and Fhelipe[Kra+24]
- Six realistic tensor workloads

- Search Speed – Compile Time
- Search Quality – Execution Time

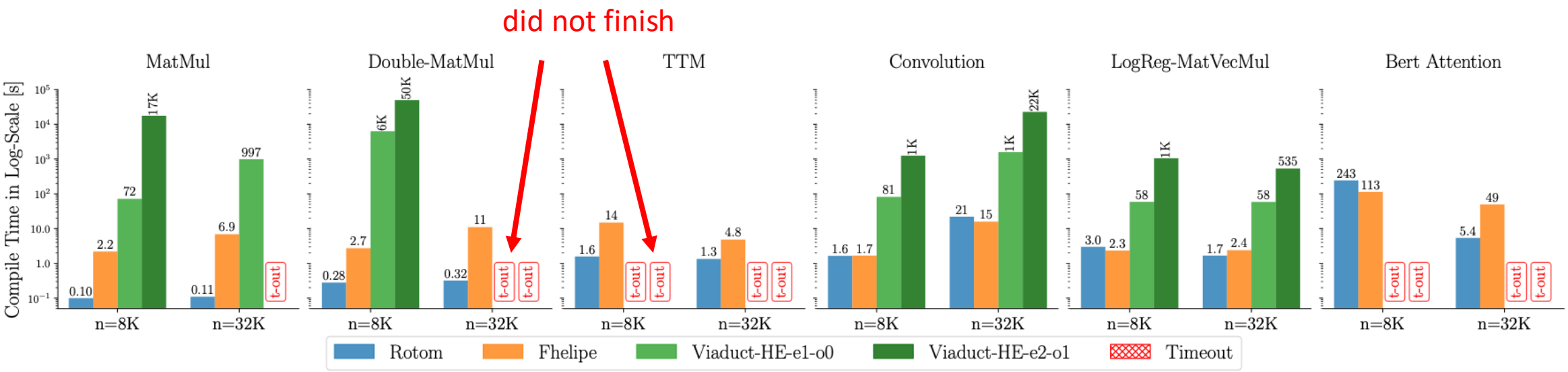
Compile Time

Rotom compiles all benchmarks in under 5 minutes (most in seconds)



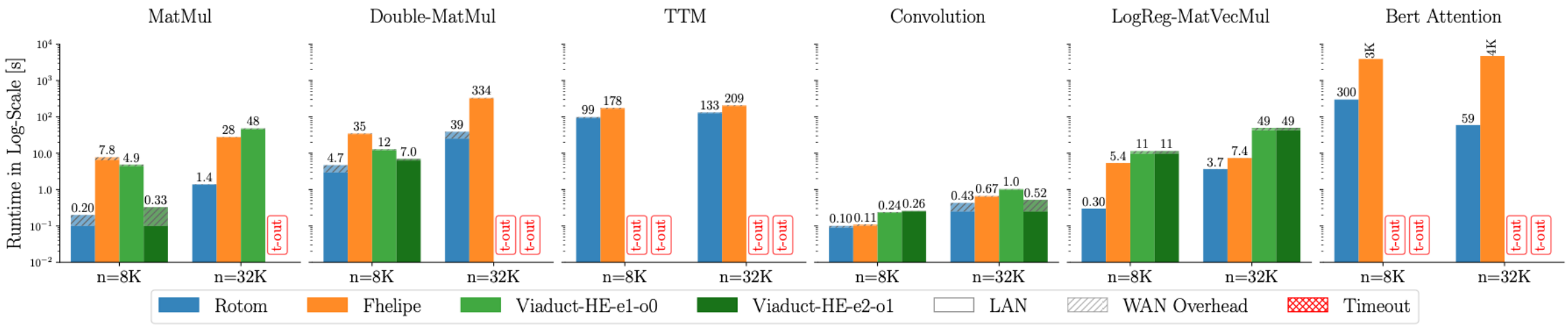
Compile Time

Rotom compiles all benchmarks in under 5 minutes (most in seconds)



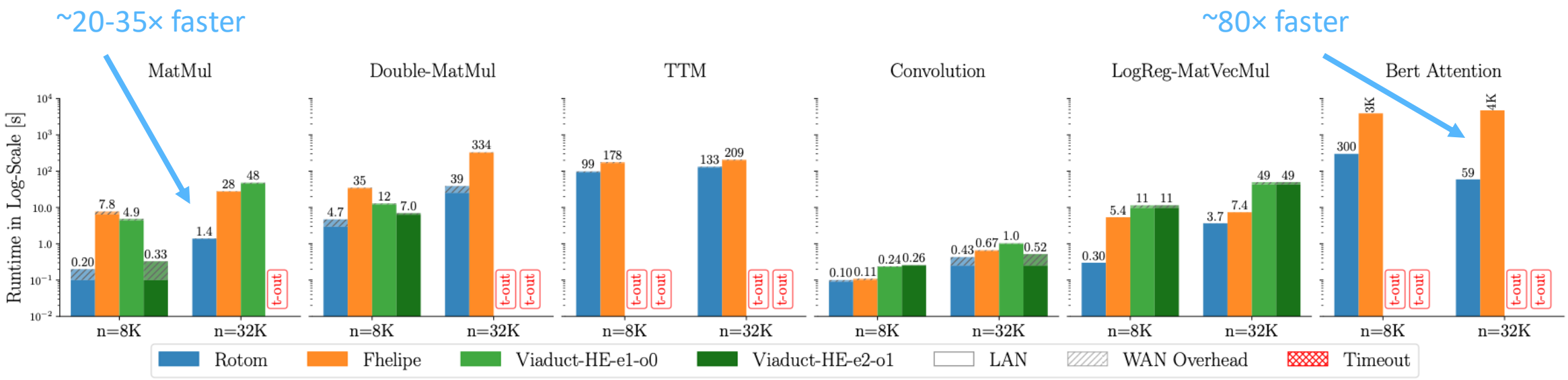
Execution Time

Rotom executes all benchmarks faster than prior systems.

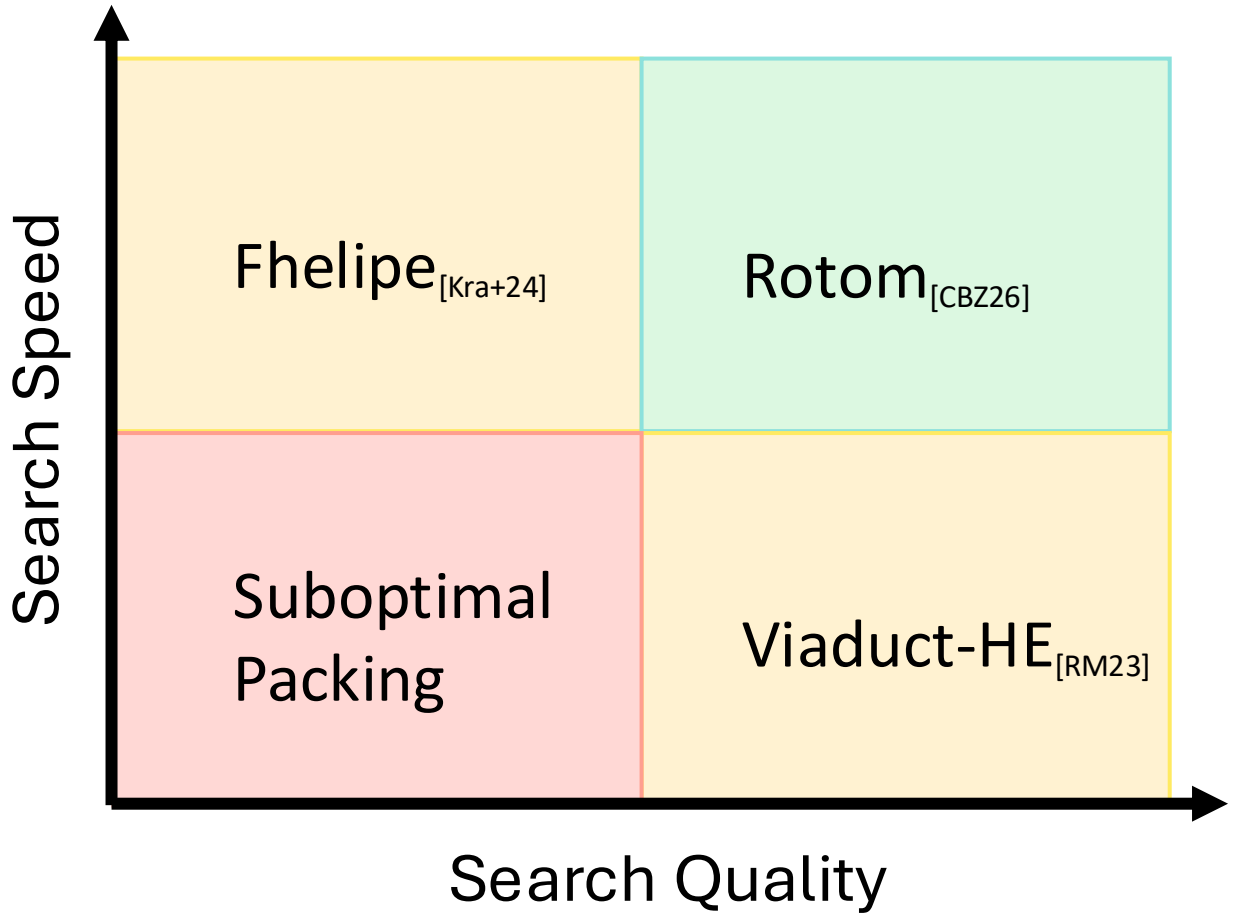


Execution Time

Rotom executes all benchmarks faster than prior systems.



Rotom Achieves the Best of Both Worlds!



Rotom is a HE Tensor Compiler that Efficiently Solves Layout Assignment

Edward Chen
ejchen@cmu.edu
https://github.com/cmu-cryptosystems/Rotom



Link to our repo!



Link to our website!