Artifacts Available V1.1

Artifacts Evaluated Functional V1.1

Results Reproduced V1.1

pLUTo

Enabling Massively Parallel Computation in DRAM via Lookup Tables

João Dinis Ferreira Gabriel Falcao Juan Gómez-Luna Mohammed Alser
Lois Orosa Mohammad Sadrosadati Jeremie S. Kim Geraldo F. Oliveira
Taha Shahroodi Anant Nori Onur Mutlu

SAFARI
ETH Zürich
TU Delft
instituto de telecomunicações
CESGA GALICIA SUPERCOMPUTING CENTER
intel

Summary

Background: Processing-in-Memory (PiM) alleviates the performance and energy bottlenecks caused by data movement in modern applications

- *Processing-near-Memory (PnM): adds* logic elements near memory arrays
- *Processing-using-Memory (PuM): uses* the analog properties of memory for computation

Problem: Existing Processing-using-DRAM architectures *only support* a limited range of operations (data movement, bitwise logic, bit shifting)

- This *limits* Processing-using-DRAM's applicability to a narrow set of applications

Goal: Extend the applicability of Processing-using-DRAM by designing a PuM substrate with support for complex operations

pLUTo: A Processing-using-DRAM substrate that replaces complex operations with equivalent memory lookups

- *pLUTo LUT Query* operation enables bulk in-memory table lookups
- *Three pLUTo designs* target different performance/energy/area tradeoffs
- *pLUTo API* and *pLUTo Compiler* facilitate programmer adoption

Key Results: Our extensive evaluation shows that pLUTo

- *Greatly* outperforms CPU/GPU/PnM baselines, both in performance and energy
- Incurs *small* DRAM area overheads (between 10.2% and 23.1%)

https://github.com/CMU-SAFARI/pLUTo

Contents

Introduction

pLUTo

Overview

pLUTo Designs

System Integration

Evaluation

Conclusion

Contents

Introduction

pLUTo

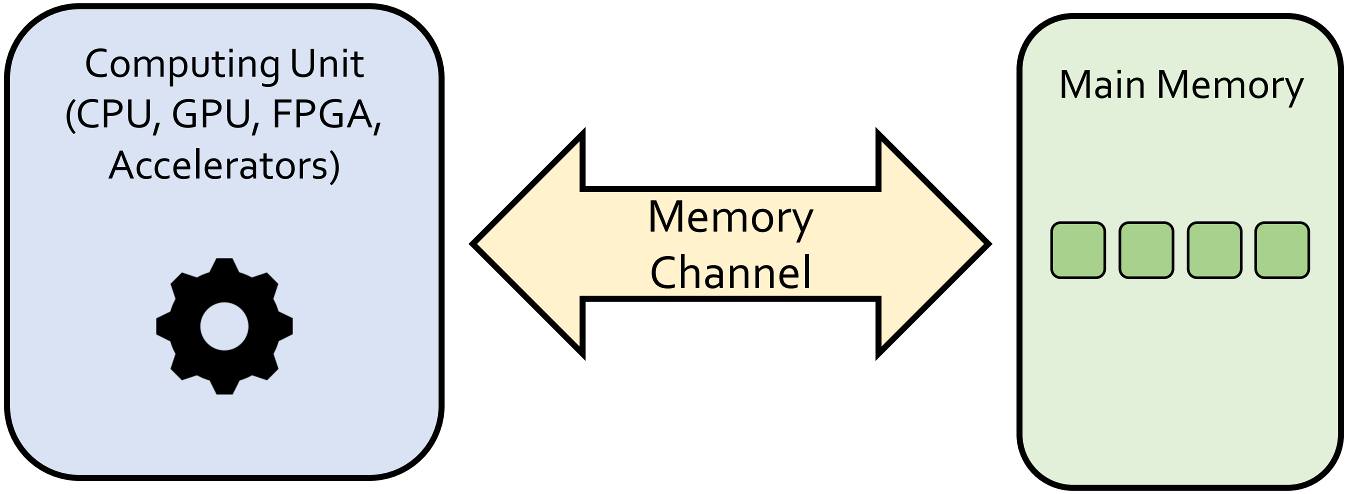
- Overview
- pLUTo Designs
- System Integration

Evaluation

Conclusion

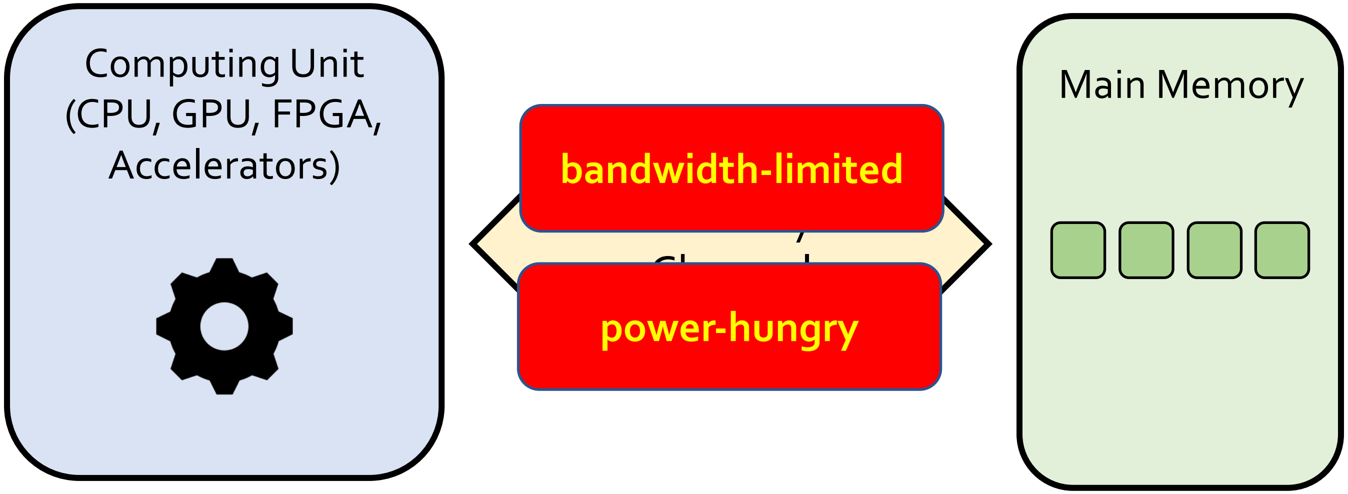
Data Movement Bottleneck

Data movement is a major bottleneck
in modern computer architectures



Data Movement Bottleneck

Data movement is a major bottleneck
in modern computer architectures



Computing Unit (CPU, GPU, FPGA, Accelerators)

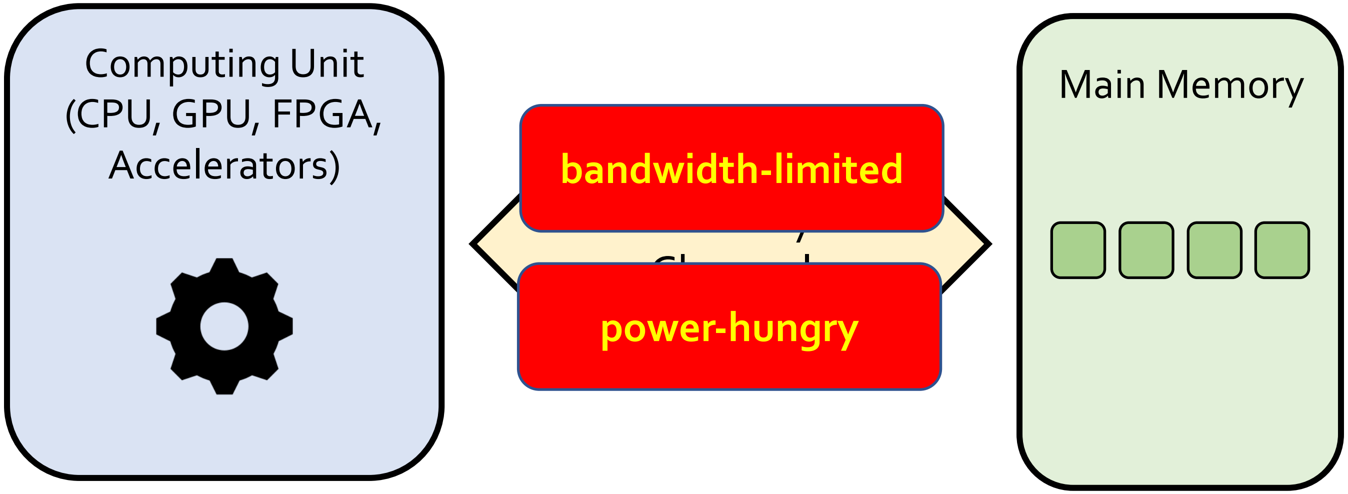
bandwidth-limited

power-hungry

Main Memory

Data Movement Bottleneck

Data movement is a major bottleneck in modern computer architectures



Computing Unit (CPU, GPU, FPGA, Accelerators)

bandwidth-limited

power-hungry

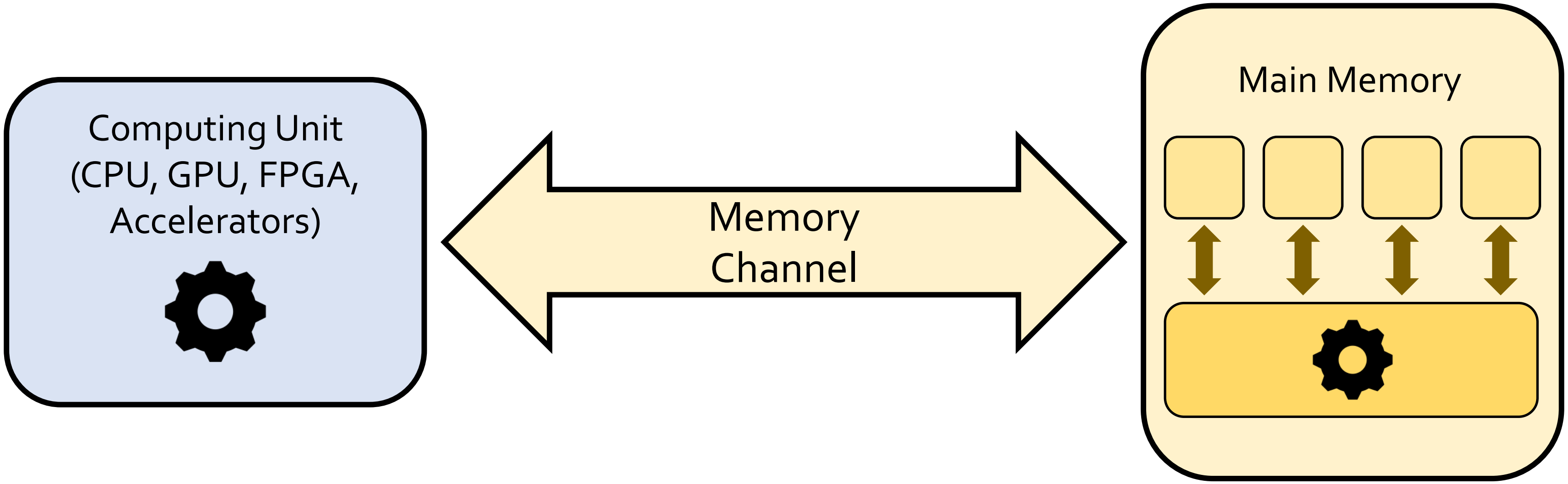
Main Memory

Over 60% of the total system energy is spent on data movement[1]

[1] A. Boroumand et al., "Google Workloads for Consumer Devices: Mitigating Data Movement Bottlenecks," ASPLOS, 2018

Processing-in-Memory

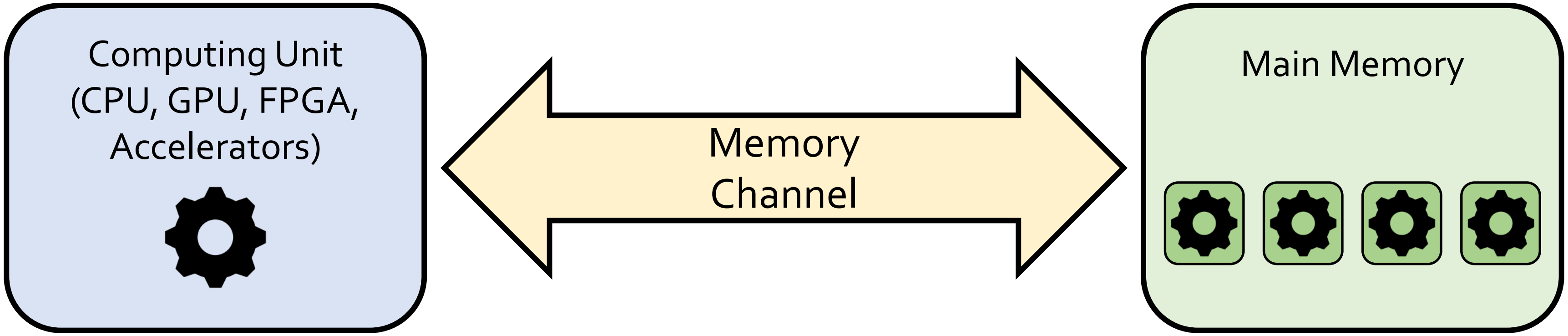
1. Processing-near-Memory



Processing-near-Memory leverages additional logic placed on the same die as memory or on the logic layer of 3D-stacked memory

Processing-in-Memory

2. Processing-using-Memory



Processing-using-Memory leverages the operational principles of memory to perform computation

Limitations of Processing-using-DRAM

Data Movement	*RowClone, Seshadri+ 2013* *LISA, Chang+ 2013*
Bitwise Operations	*Ambit, Seshadri+ 2017*
Bit Shifting	*DRISA, Li+ 2017*
Arithmetic Operations	*SIMDRAM, Hajinazar & Oliveira+ 2021*

Existing Processing-using-DRAM architectures only support a limited range of operations

The Goal of pLUTo

Extend Processing-using-DRAM to support the execution of *arbitrarily complex operations*

Contents

Introduction

pLUTo

Overview

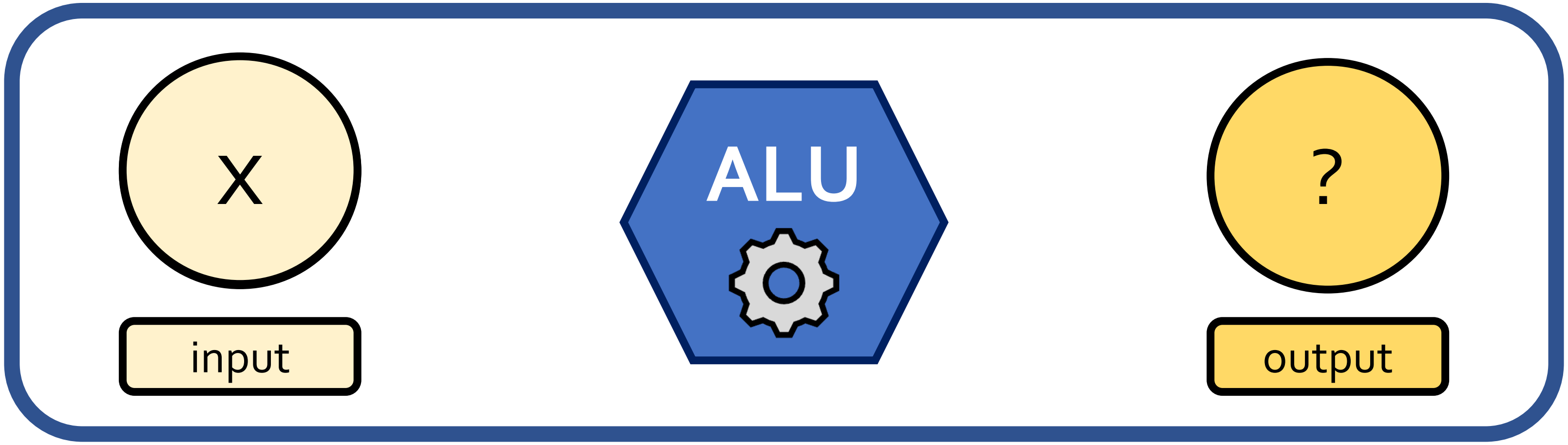
pLUTo Designs

System Integration

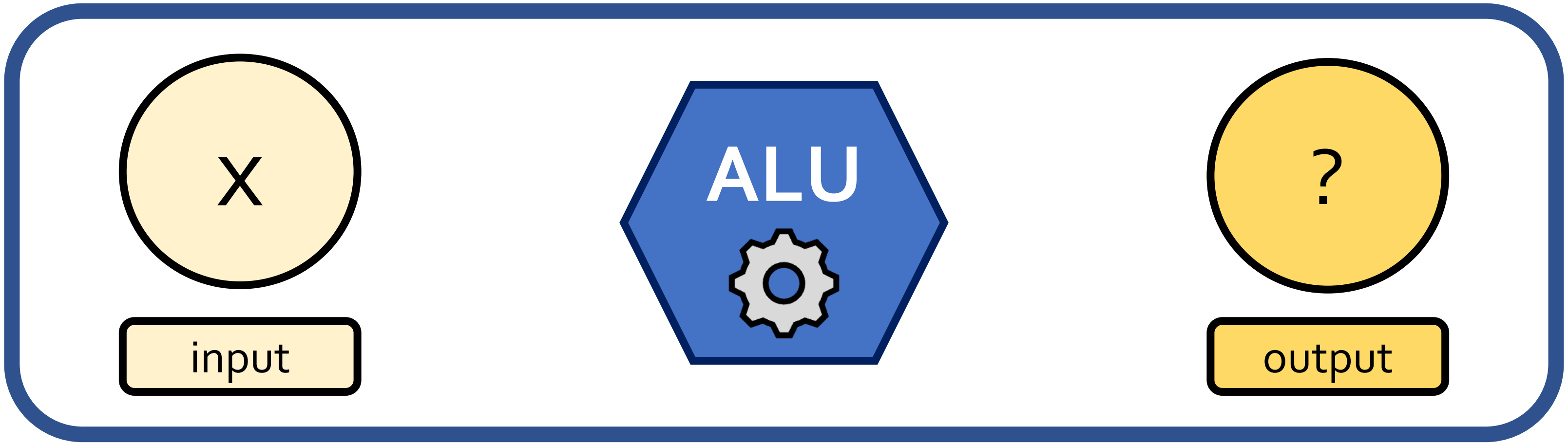
Evaluation

Conclusion

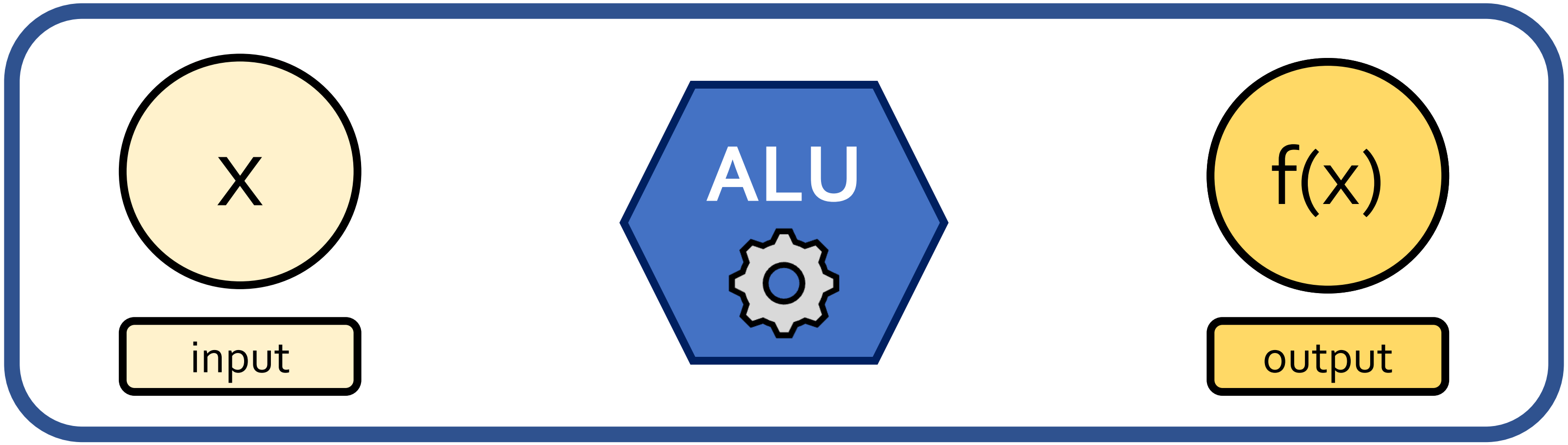
pLUTo: Key Idea



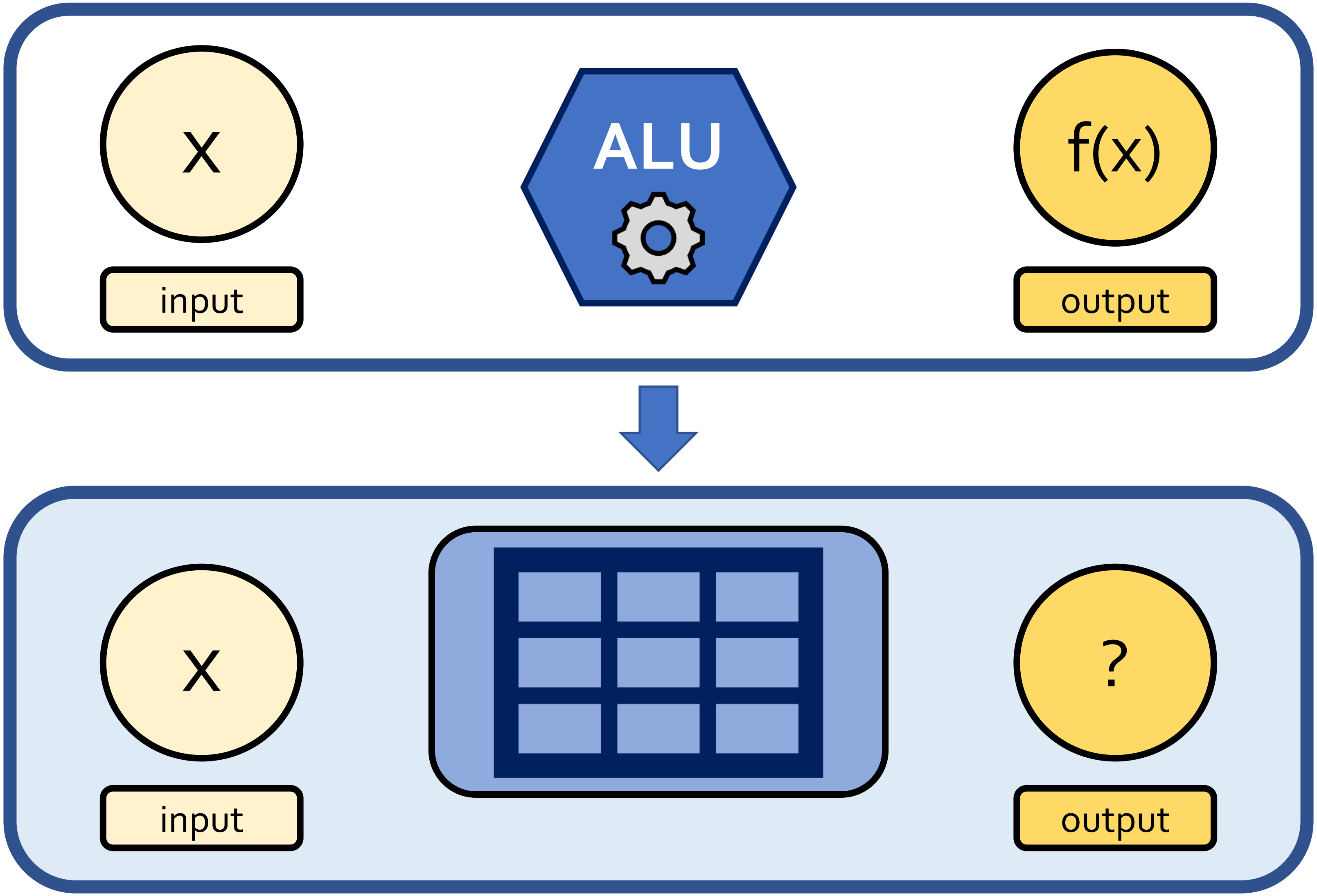
pLUTo: Key Idea



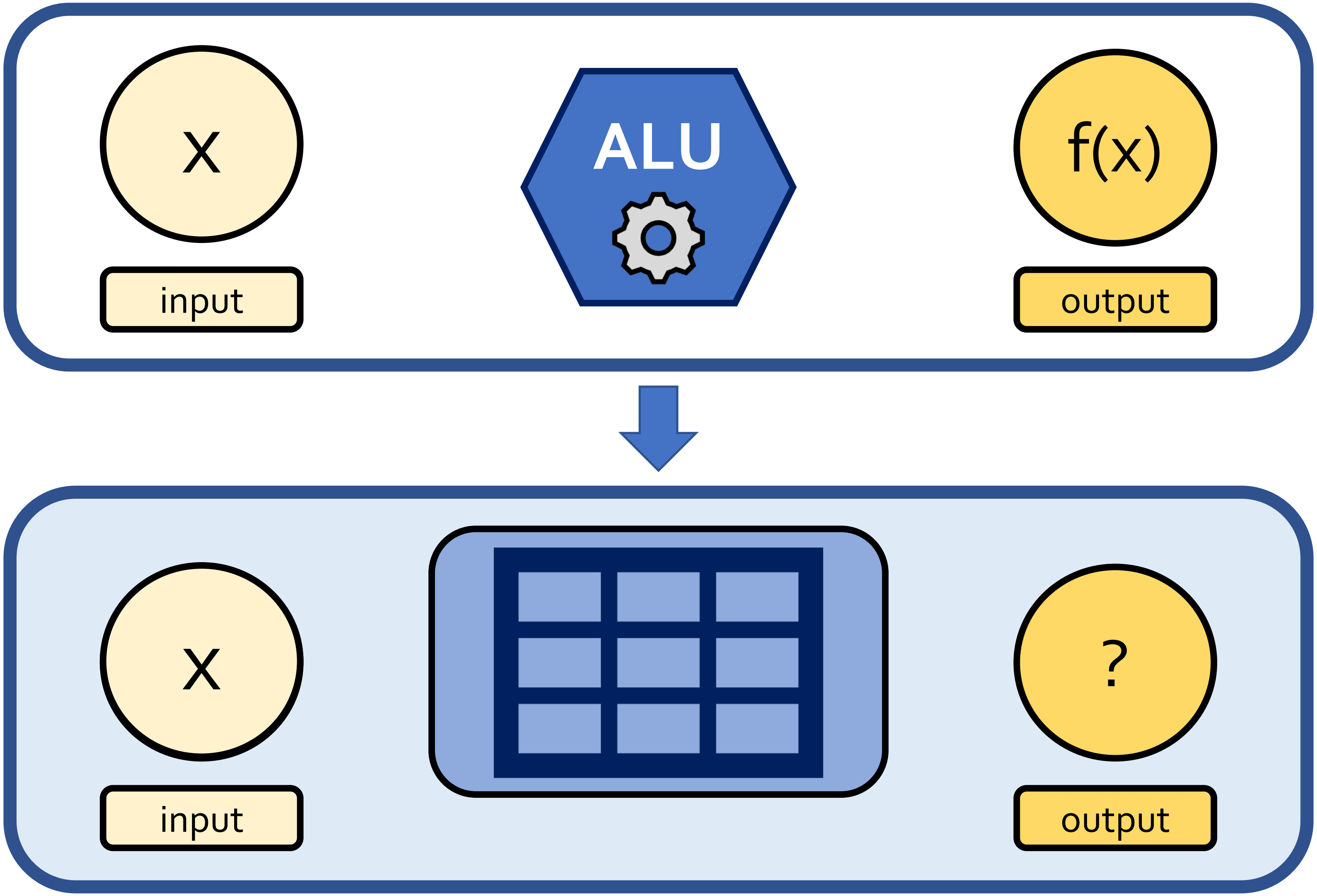
pLUTo: Key Idea



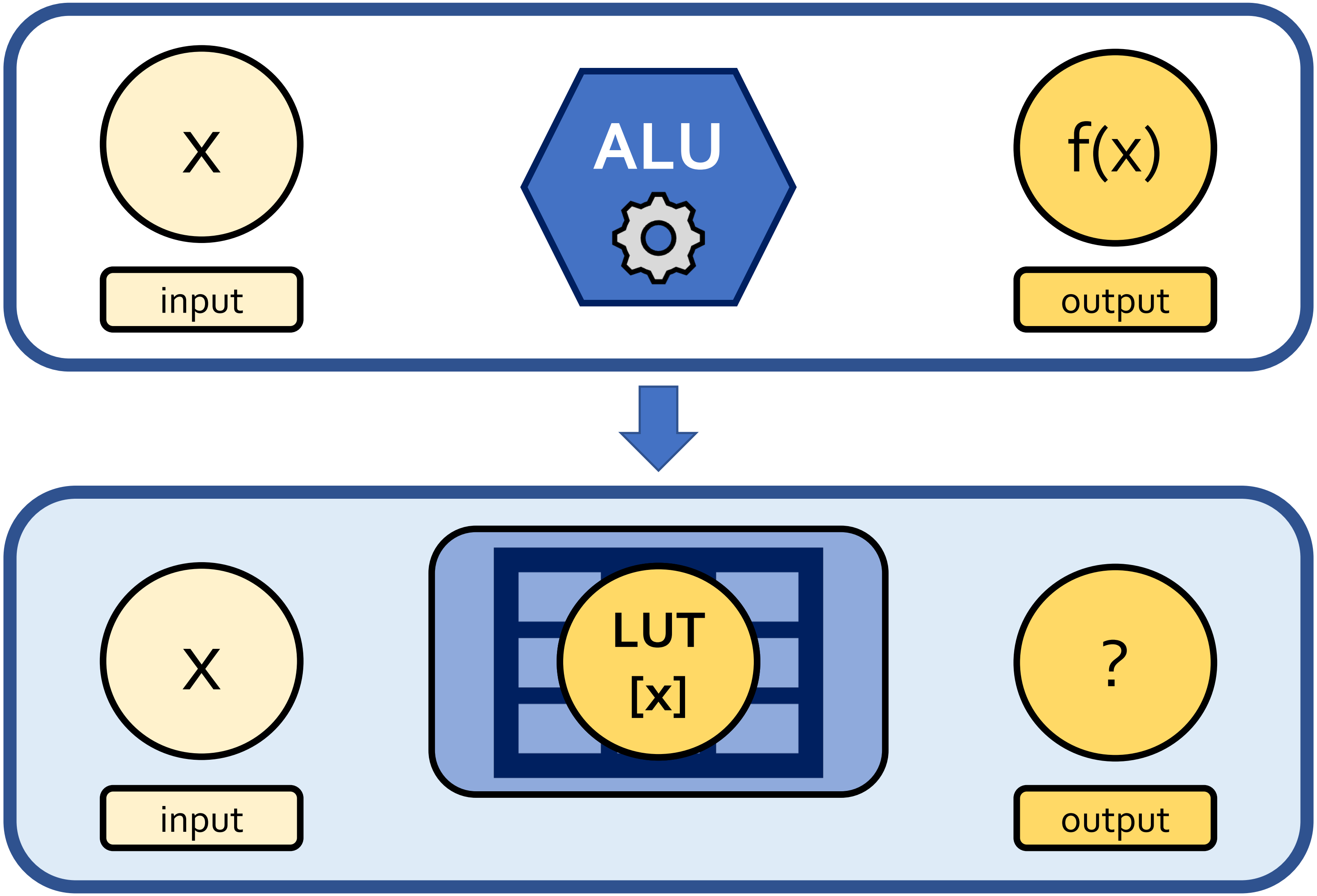
pLUTo: Key Idea



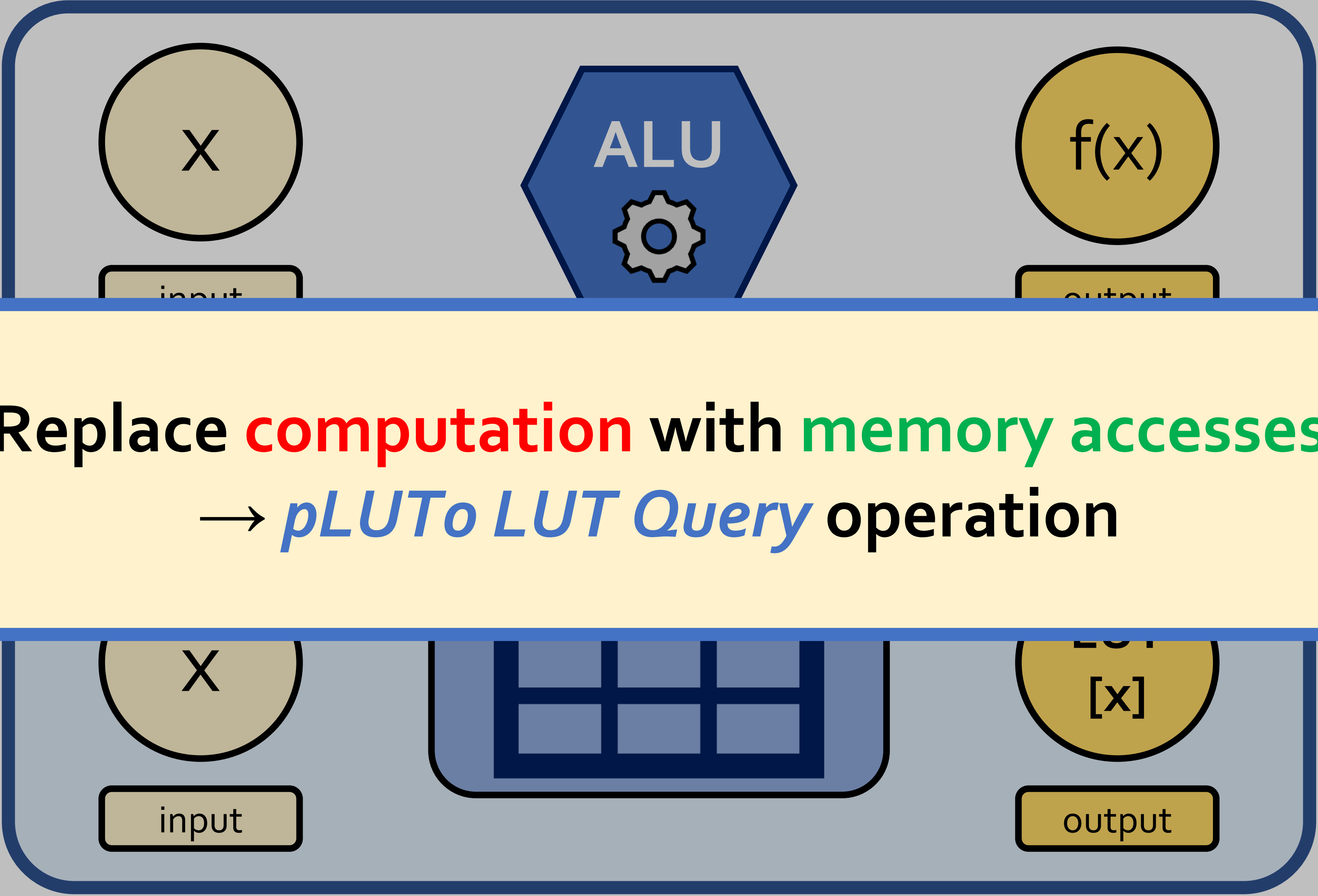
pLUTo: Key Idea



pLUTo: Key Idea



pLUTo: Key Idea



Replace computation with memory accesses
→ *pLUTo LUT Query* operation

The pLUTo LUT Query: Setup

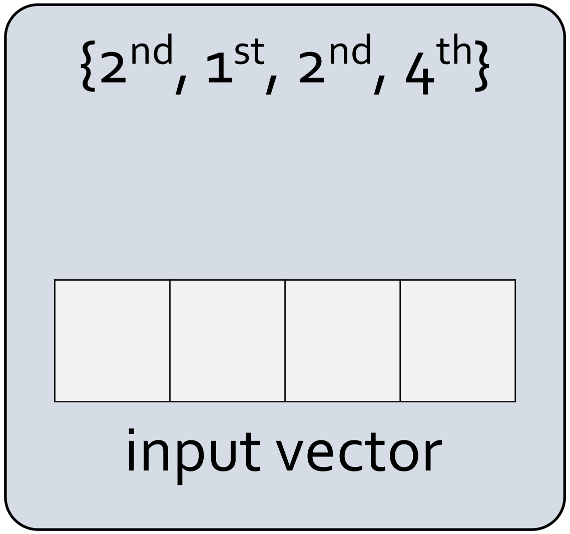
Desired LUT Query

Return {2nd, 1st, 2nd, 4th} *prime numbers*

The pLUTo LUT Query: Setup

Desired LUT Query

Return {2nd, 1st, 2nd, 4th} *prime numbers*



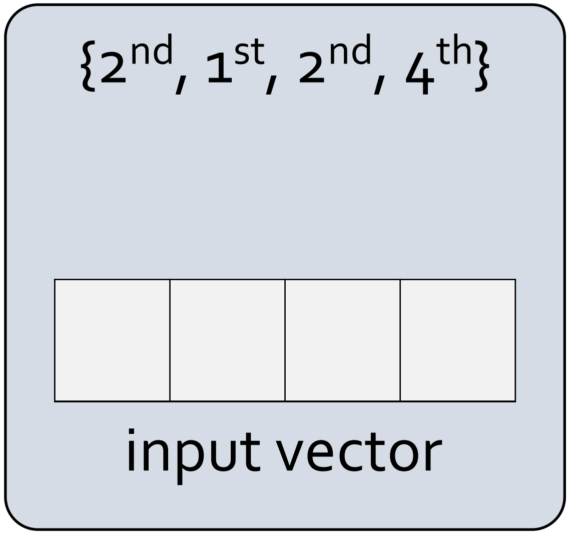
{2nd, 1st, 2nd, 4th}

input vector

The pLUTo LUT Query: Setup

Desired LUT Query

Return {2nd, 1st, 2nd, 4th} *prime numbers*



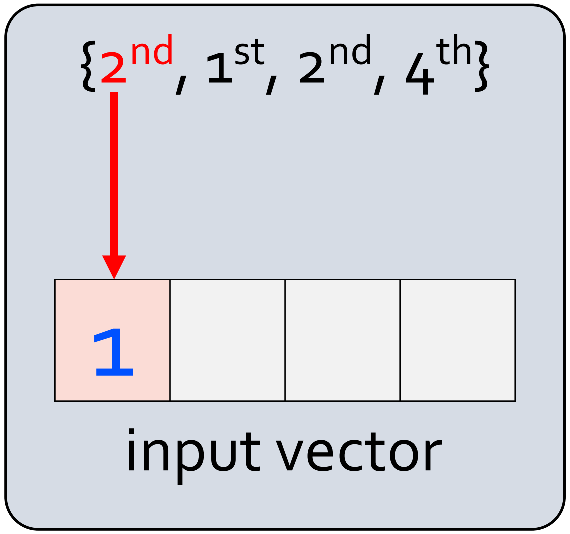
	Prime numbers	
LUT index	i	f(i)
0	1st	2
1	2nd	3
2	3rd	5
3	4th	7

lookup table

The pLUTo LUT Query: Setup

Desired LUT Query

Return {2nd, 1st, 2nd, 4th} *prime numbers*



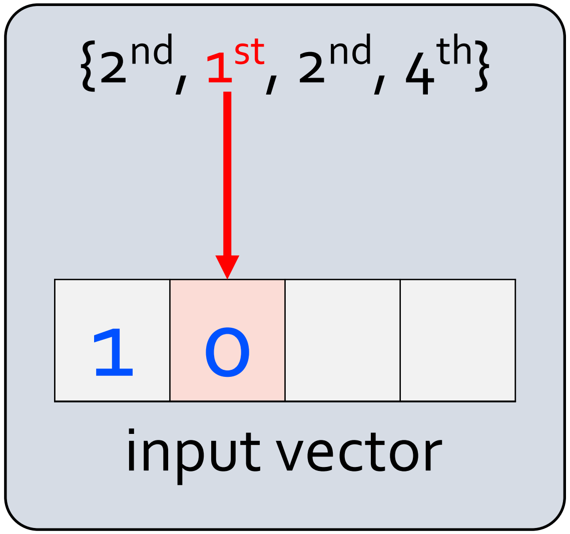
LUT index	Prime numbers	
	i	f(i)
0	1st	2
1	2nd	3
2	3rd	5
3	4th	7

lookup table

The pLUTo LUT Query: Setup

Desired LUT Query

Return {2nd, 1st, 2nd, 4th} *prime numbers*



{2nd, 1st, 2nd, 4th}

1	0		

input vector

	Prime numbers	
LUT index	i	f(i)
0	1st	2
1	2nd	3
2	3rd	5
3	4th	7

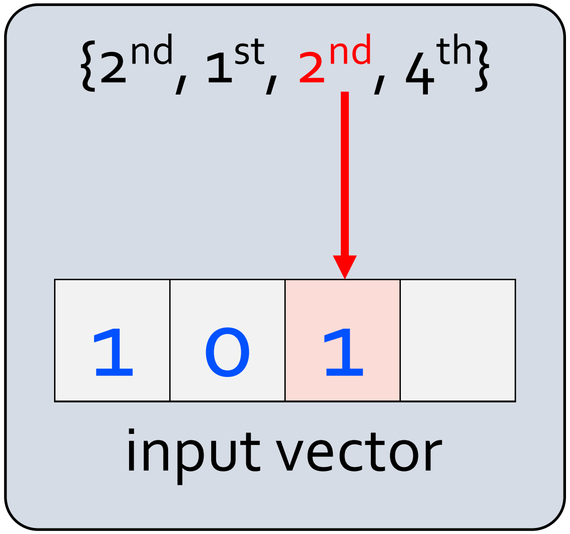
lookup table

SAFARI

The pLUTo LUT Query: Setup

Desired LUT Query

Return {2nd, 1st, 2nd, 4th} *prime numbers*


{2nd, 1st, 2nd, 4th}

1	0	1	

input vector

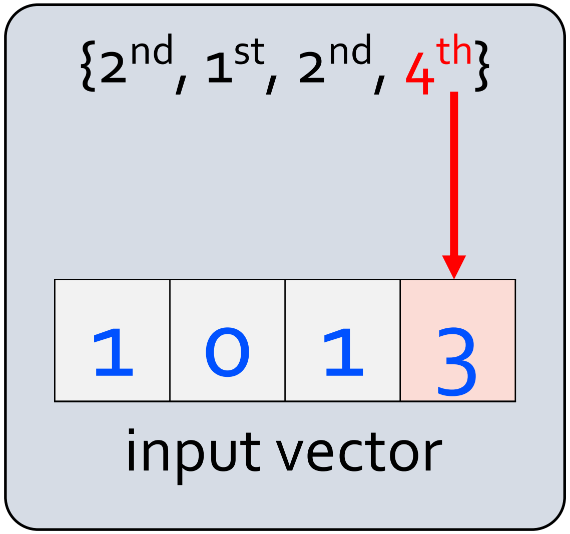
	Prime numbers	
LUT index	i	f(i)
0	1st	2
1	2nd	3
2	3rd	5
3	4th	7

lookup table

The pLUTo LUT Query: Setup

Desired LUT Query

Return {2nd, 1st, 2nd, 4th} *prime numbers*



	Prime numbers	
LUT index	i	f(i)
0	1st	2
1	2nd	3
2	3rd	5
3	4th	7

lookup table

The pLUTo LUT Query: Setup

Desired LUT Query

Return {2nd, 1st, 2nd, 4th} *prime numbers*

{2nd, 1st, 2nd, 4th}

1	0	1	3

input vector

	Prime numbers	
LUT index	i	f(i)
0	1st	2
1	2nd	3
2	3rd	5
3	4th	7

lookup table

output vector

The pLUTo LUT Query: Setup

Desired LUT Query

Return {2^{nd}, 1^{st}, 2^{nd}, 4^{th}} *prime numbers*

{2^{nd}, 1^{st}, 2^{nd}, 4^{th}}

1	0	1	3

input vector

	Prime numbers	
LUT index	i	f(i)
0	1^{st}	2
1	2^{nd}	3
2	3^{rd}	5
3	4^{th}	7

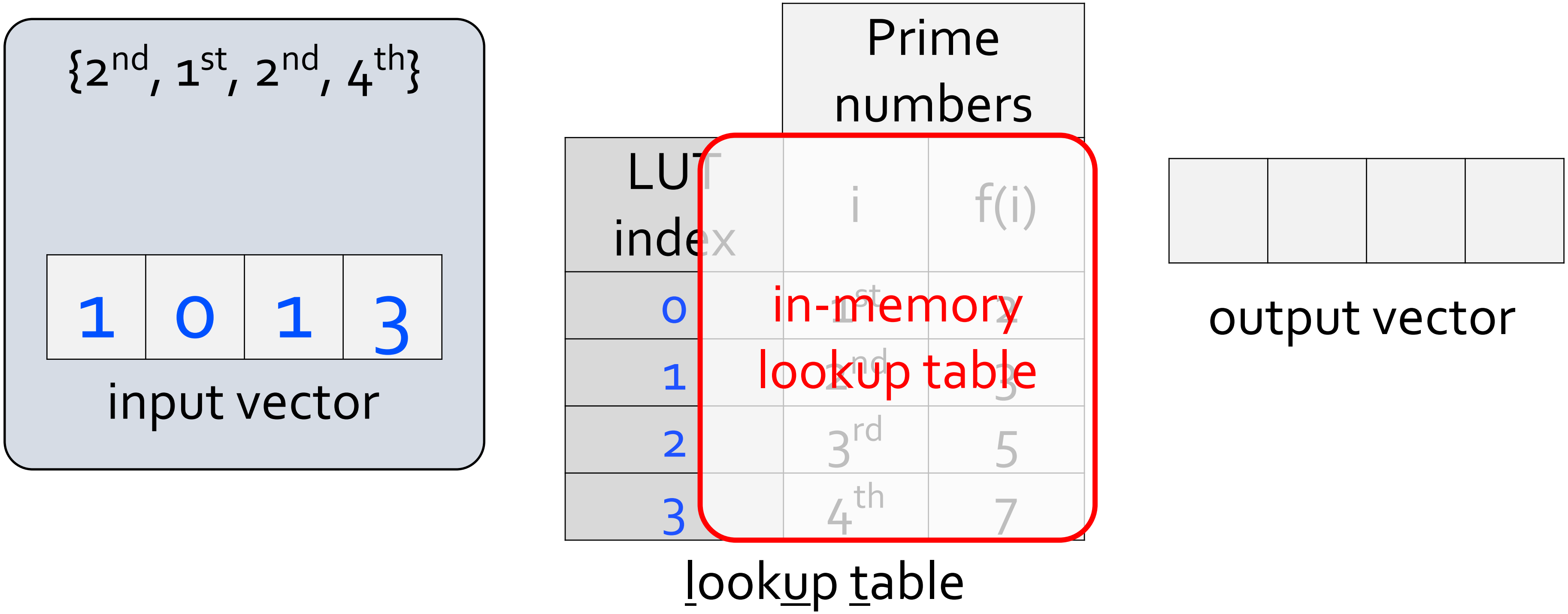
lookup table

output vector

The pLUTo LUT Query: Setup

Desired LUT Query

Return {2nd, 1st, 2nd, 4th} *prime numbers*



{2nd, 1st, 2nd, 4th}

1	0	1	3

input vector

LUT index	Prime numbers	
	i	f(i)
0	1st	2
1	2nd	3
2	3rd	5
3	4th	7

in-memory lookup table

lookup table

output vector

The pLUTo LUT Query: Setup

Desired LUT Query

Return {2nd, 1st, 2nd, 4th} *prime numbers*

{2nd, 1st, 2nd, 4th}

1	0	1	3

input vector

		Prime numbers
	LUT index	f(i)
1st	0	2
2nd	1	3
3rd	2	5
4th	3	7

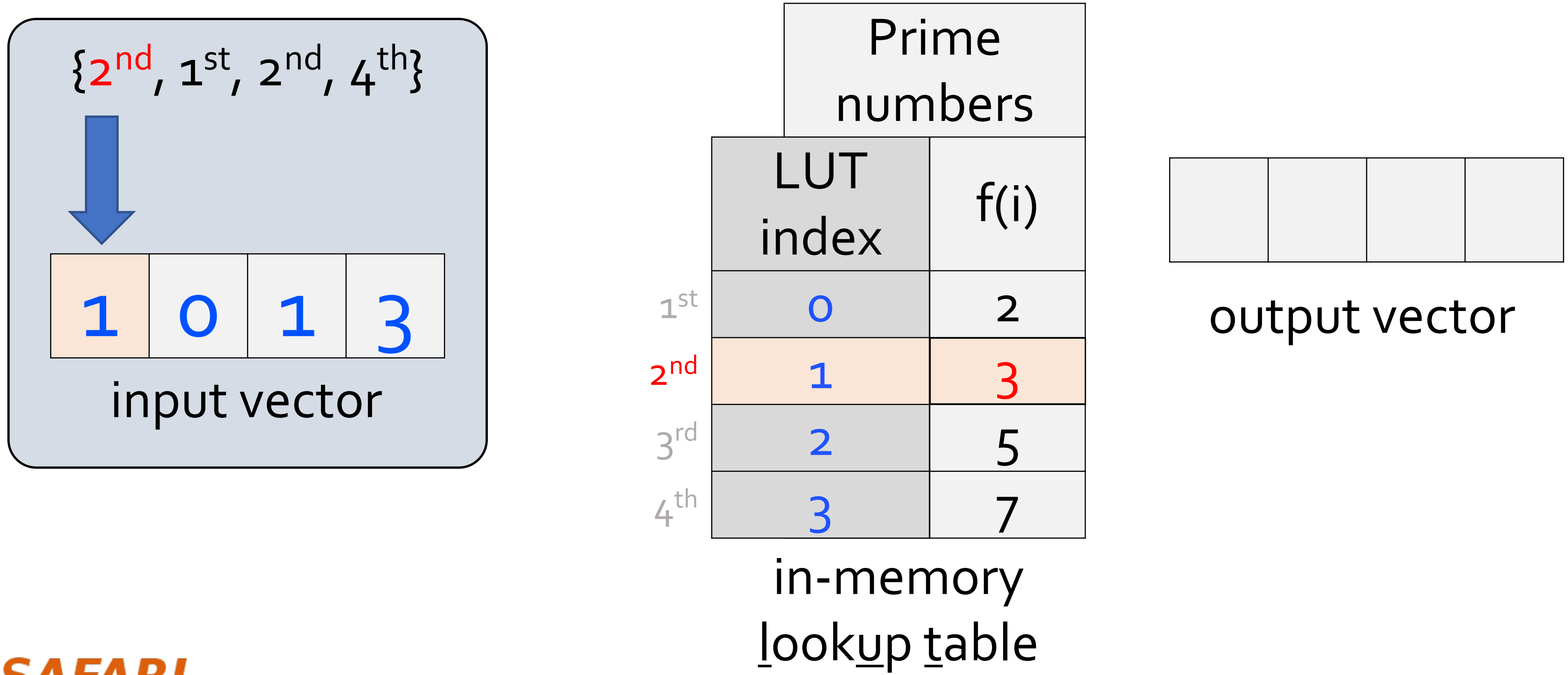
in-memory lookup table

output vector

The pLUTo LUT Query: Operation

Desired LUT Query

Return {2nd, 1st, 2nd, 4th} *prime numbers*



{2nd, 1st, 2nd, 4th}

1	0	1	3

input vector

		Prime numbers
	LUT index	f(i)
1st	0	2
2nd	1	3
3rd	2	5
4th	3	7

in-memory lookup table

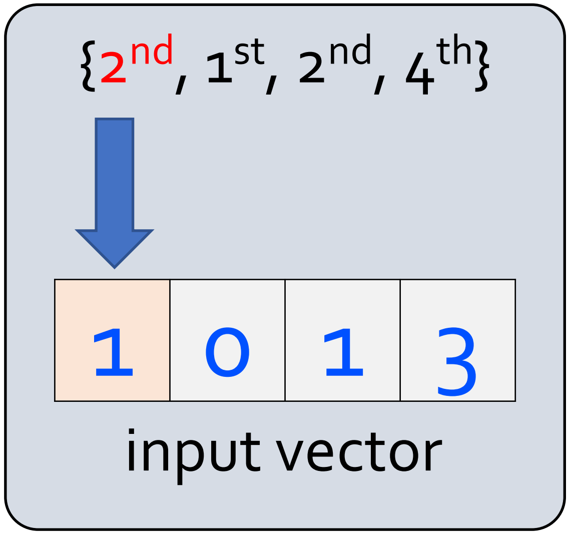
output vector

SAFARI

The pLUTo LUT Query: Operation

Desired LUT Query

Return {2nd, 1st, 2nd, 4th} *prime numbers*



{2nd, 1st, 2nd, 4th}

1	0	1	3

input vector

		Prime numbers
	LUT index	f(i)
1st	0	2
2nd	1	3
3rd	2	5
4th	3	7

in-memory lookup table

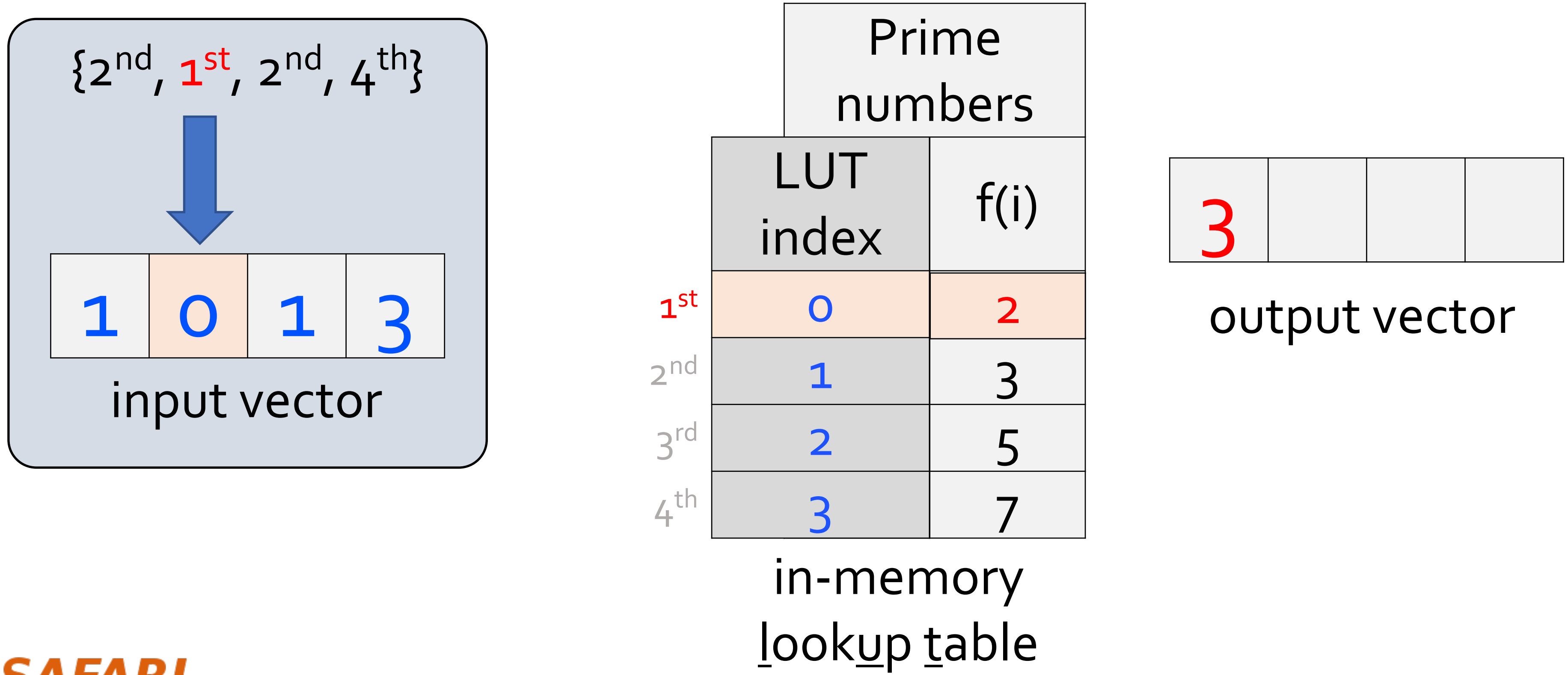
3			

output vector

The pLUTo LUT Query: Operation

Desired LUT Query

Return {2nd, 1st, 2nd, 4th} *prime numbers*



{2nd, 1st, 2nd, 4th}

1	0	1	3

input vector

	LUT index	Prime numbers f(i)
1st	0	2
2nd	1	3
3rd	2	5
4th	3	7

in-memory lookup table

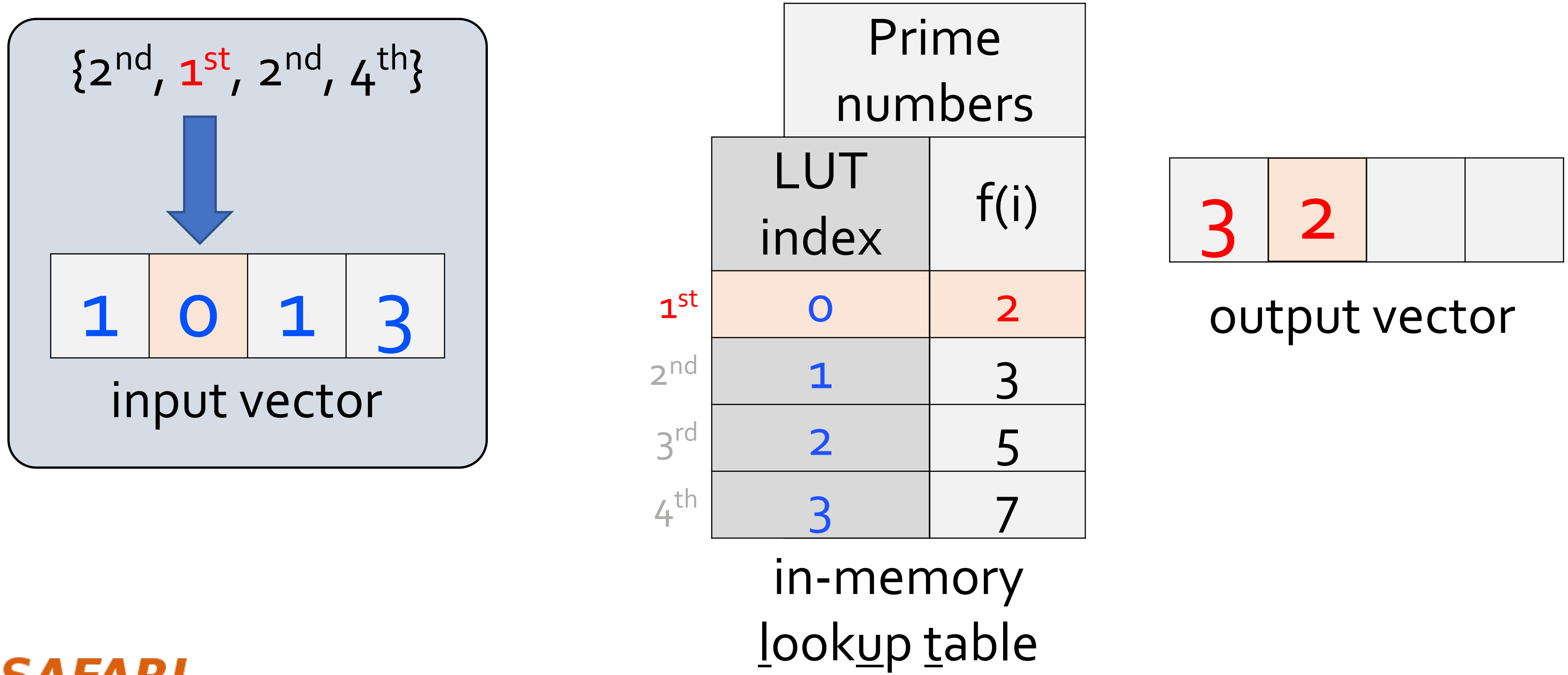
3			

output vector

The pLUTo LUT Query: Operation

Desired LUT Query

Return {2nd, 1st, 2nd, 4th} *prime numbers*



{2nd, 1st, 2nd, 4th}

1	0	1	3

input vector

		Prime numbers
	LUT index	f(i)
1st	0	2
2nd	1	3
3rd	2	5
4th	3	7

in-memory lookup table

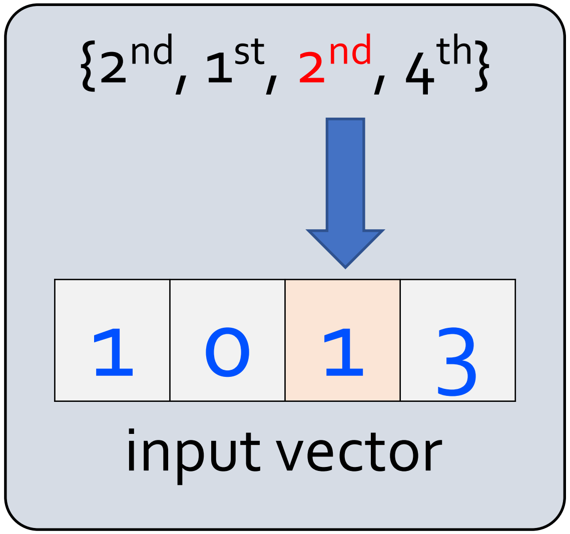
3	2		

output vector

The pLUTo LUT Query: Operation

Desired LUT Query

Return {2nd, 1st, 2nd, 4th} *prime numbers*


{2nd, 1st, 2nd, 4th}

1	0	1	3

input vector

		Prime numbers
	LUT index	f(i)
1st	0	2
2nd	1	3
3rd	2	5
4th	3	7

in-memory lookup table

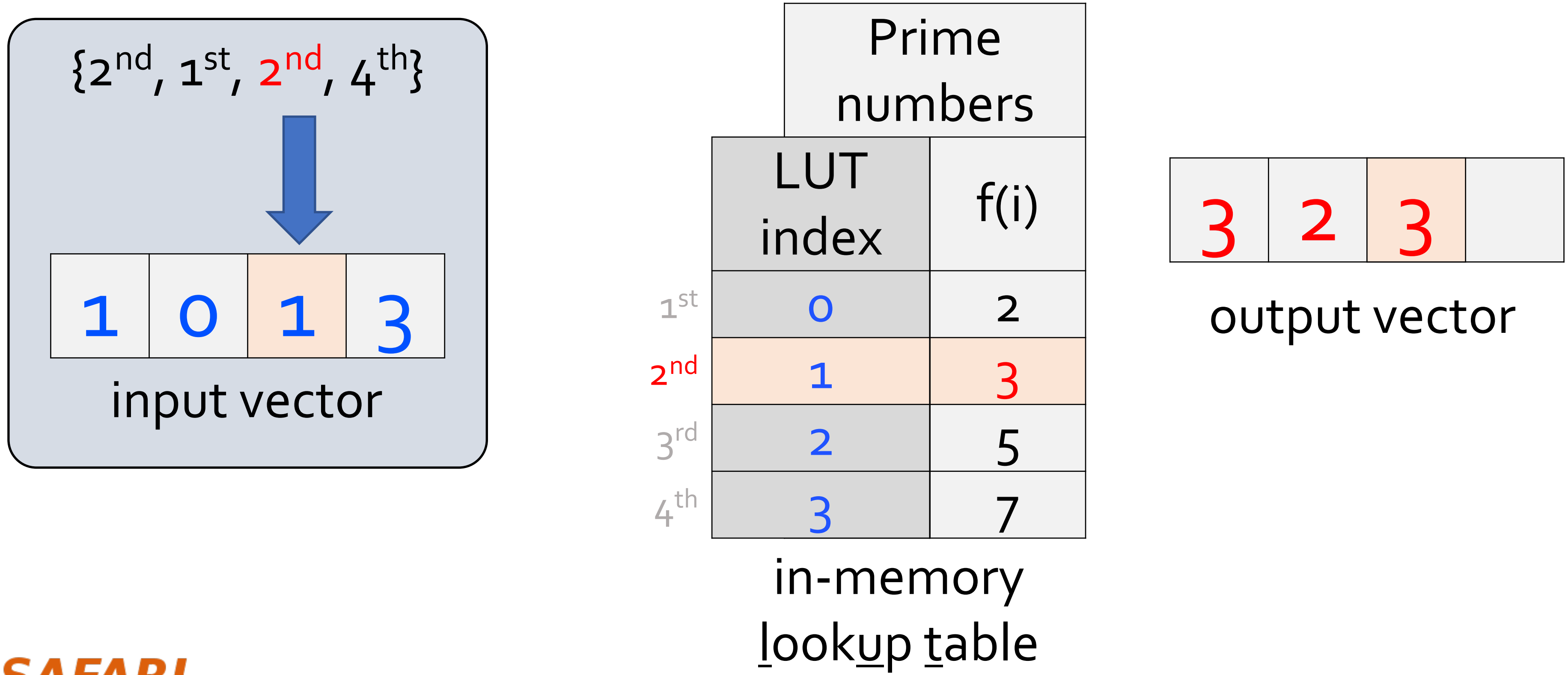
3	2		

output vector

The pLUTo LUT Query: Operation

Desired LUT Query

Return {2nd, 1st, 2nd, 4th} *prime numbers*



{2nd, 1st, 2nd, 4th}

1	0	1	3

input vector

		Prime numbers
	LUT index	f(i)
1st	0	2
2nd	1	3
3rd	2	5
4th	3	7

in-memory lookup table

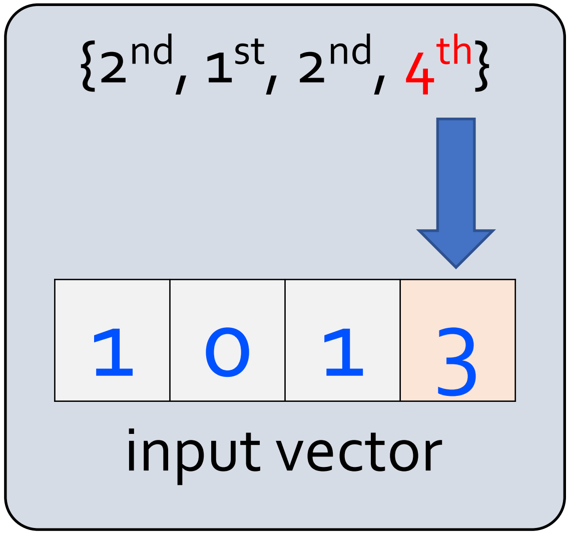
3	2	3	

output vector

The pLUTo LUT Query: Operation

Desired LUT Query

Return {2nd, 1st, 2nd, 4th} *prime numbers*



{2nd, 1st, 2nd, 4th}

1	0	1	3

input vector

		Prime numbers
	LUT index	f(i)
1st	0	2
2nd	1	3
3rd	2	5
4th	3	7

in-memory lookup table

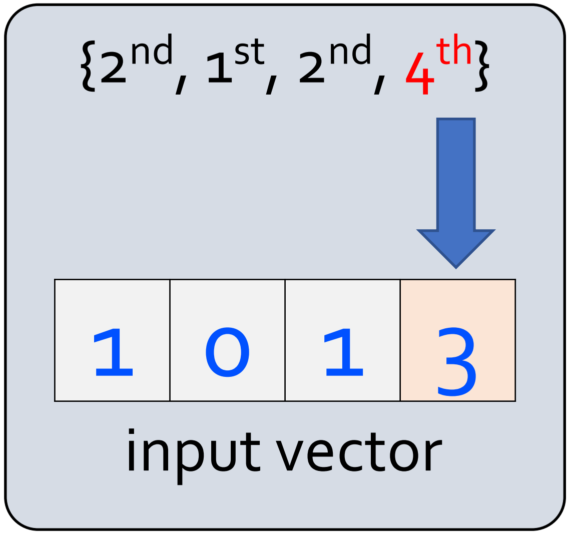
3	2	3	

output vector

The pLUTo LUT Query: Operation

Desired LUT Query

Return {2nd, 1st, 2nd, 4th} *prime numbers*



		Prime numbers
	LUT index	f(i)
1st	0	2
2nd	1	3
3rd	2	5
4th	3	7

in-memory lookup table

3	2	3	7

output vector

In-DRAM pLUTo LUT Query: Setup

LUT Query:
Return {2nd, 1st, 2nd, 4th}
prime numbers

	Prime numbers	
LUT index	i	f(i)
0	1st	2
1	2nd	3
2	3rd	5
3	4th	7

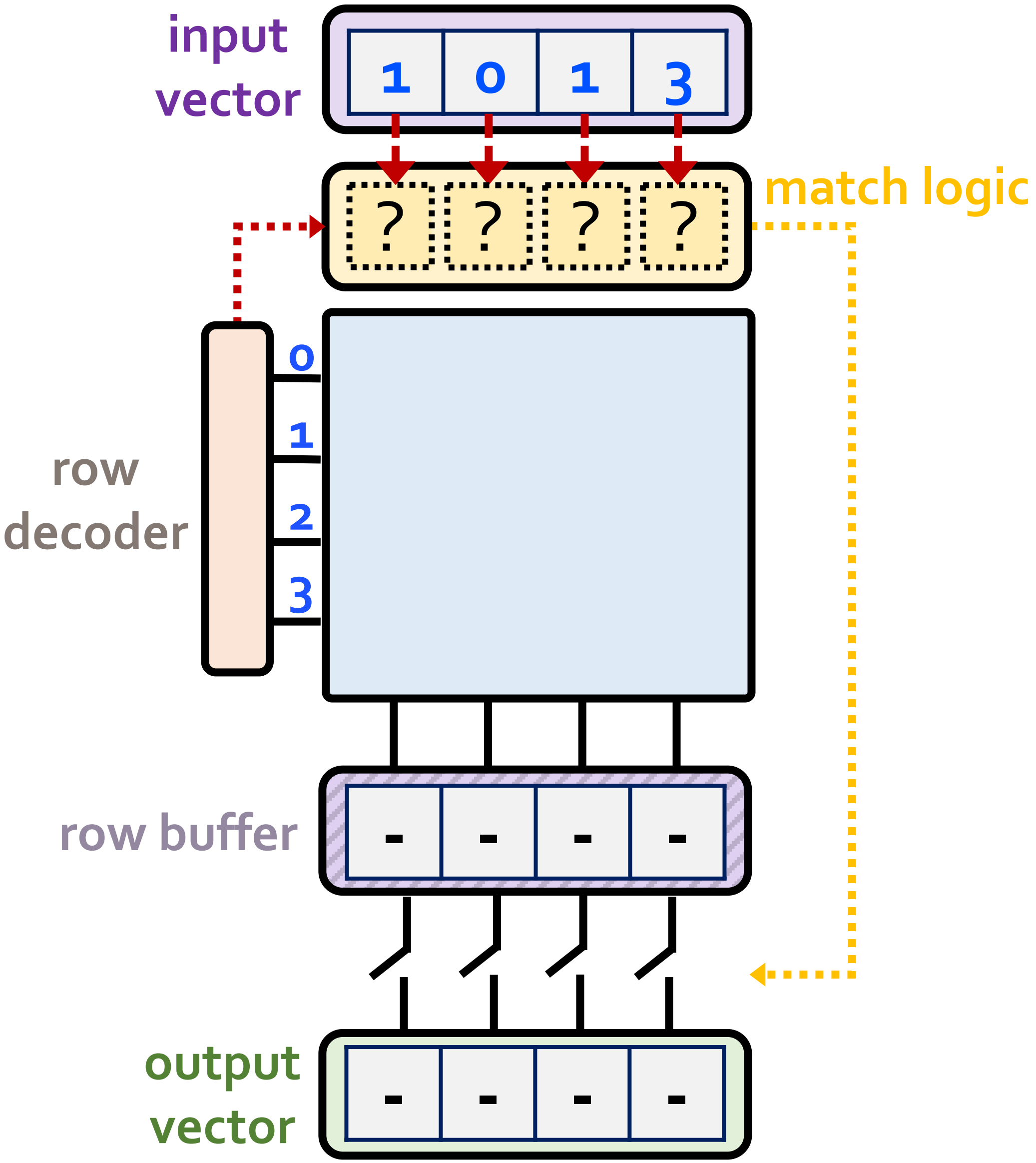
lookup table

1	0	1	3

input vector

3	2	3	7

output vector



In-DRAM pLUTo LUT Query: Setup

LUT Query:
Return {2nd, 1st, 2nd, 4th}
prime numbers

	Prime numbers	
LUT index	i	f(i)
0	1st	2
1	2nd	3
2	3rd	5
3	4th	7

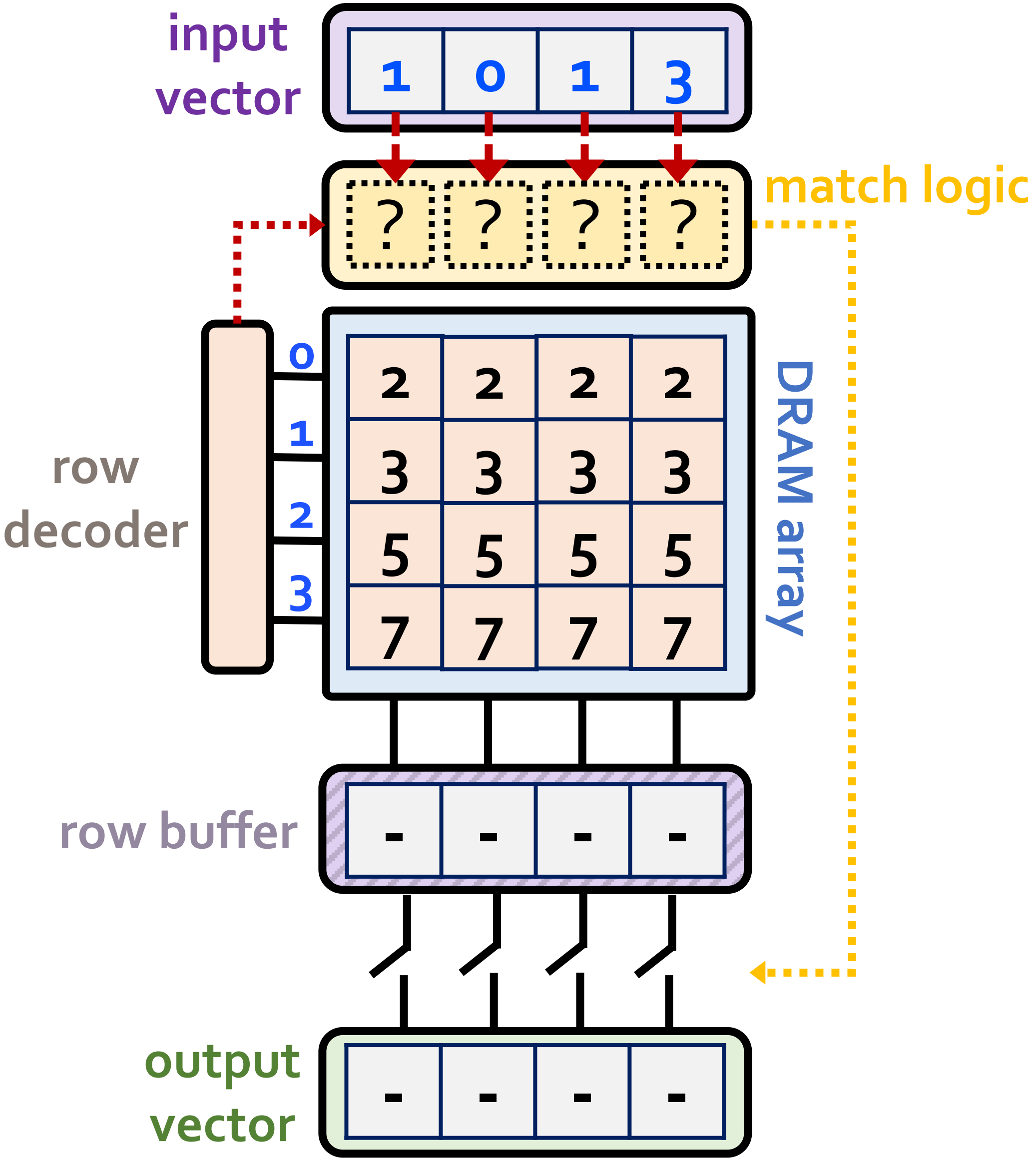
lookup table

1	0	1	3

input vector

3	2	3	7

output vector



SAFARI

In-DRAM pLUTo LUT Query: Setup

LUT Query:
Return {2nd, 1st, 2nd, 4th}
prime numbers

	Prime numbers	
LUT index	i	f(i)
0	1st	2
1	2nd	3
2	3rd	5
3	4th	7

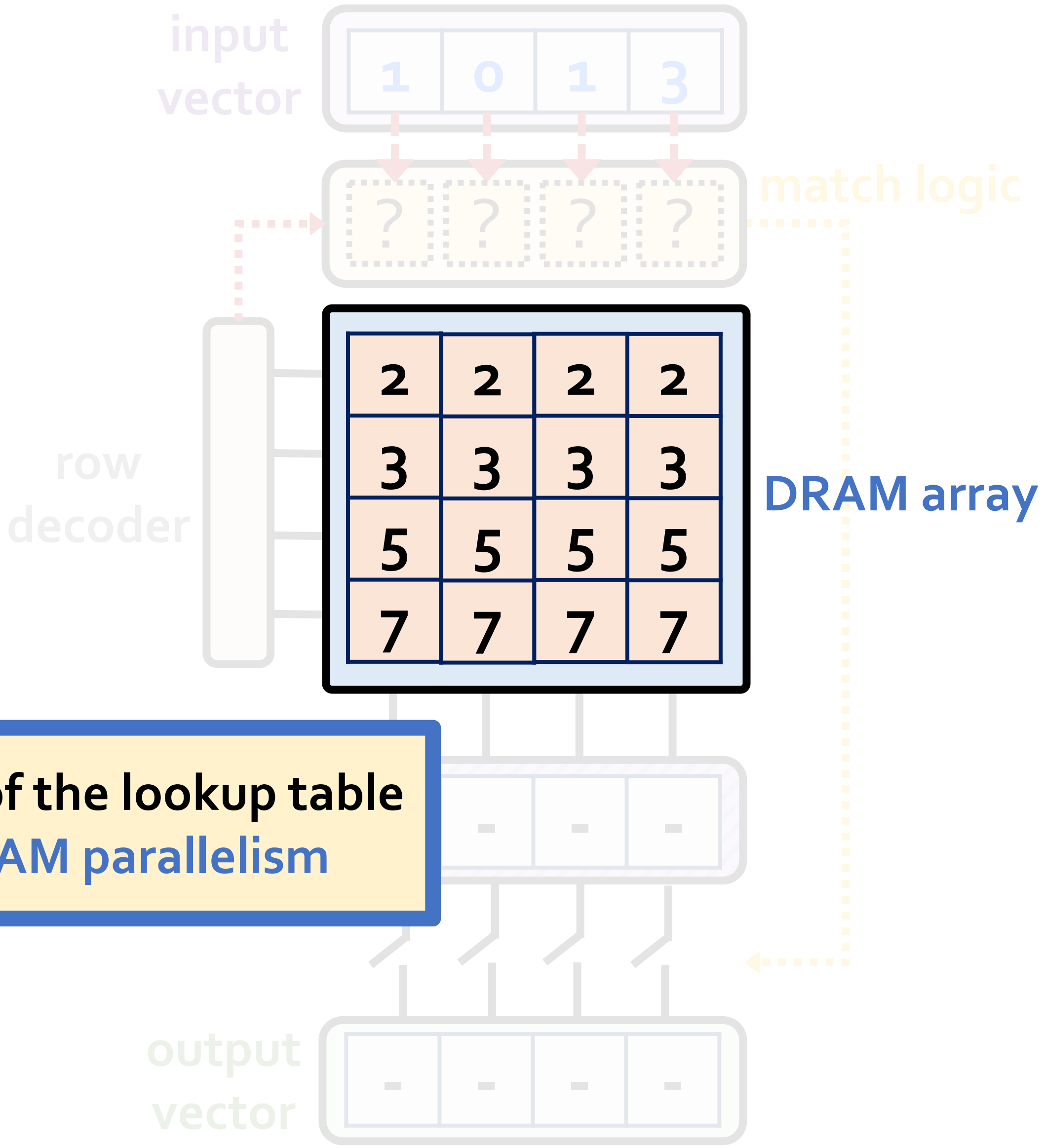
lookup ta

1	0	1

input vector

3	2	3	7

output vector



2	2	2	2
3	3	3	3
5	5	5	5
7	7	7	7

Multiple copies of the lookup table
→ *exploit* DRAM parallelism

In-DRAM pLUTo LUT Query: Setup

LUT Query:
Return {2^{nd}, 1^{st}, 2^{nd}, 4^{th}}
prime numbers

	Prime numbers	
LUT index	i	f(i)
0	1st	2
1	2nd	3
2	3rd	5
3	4th	7

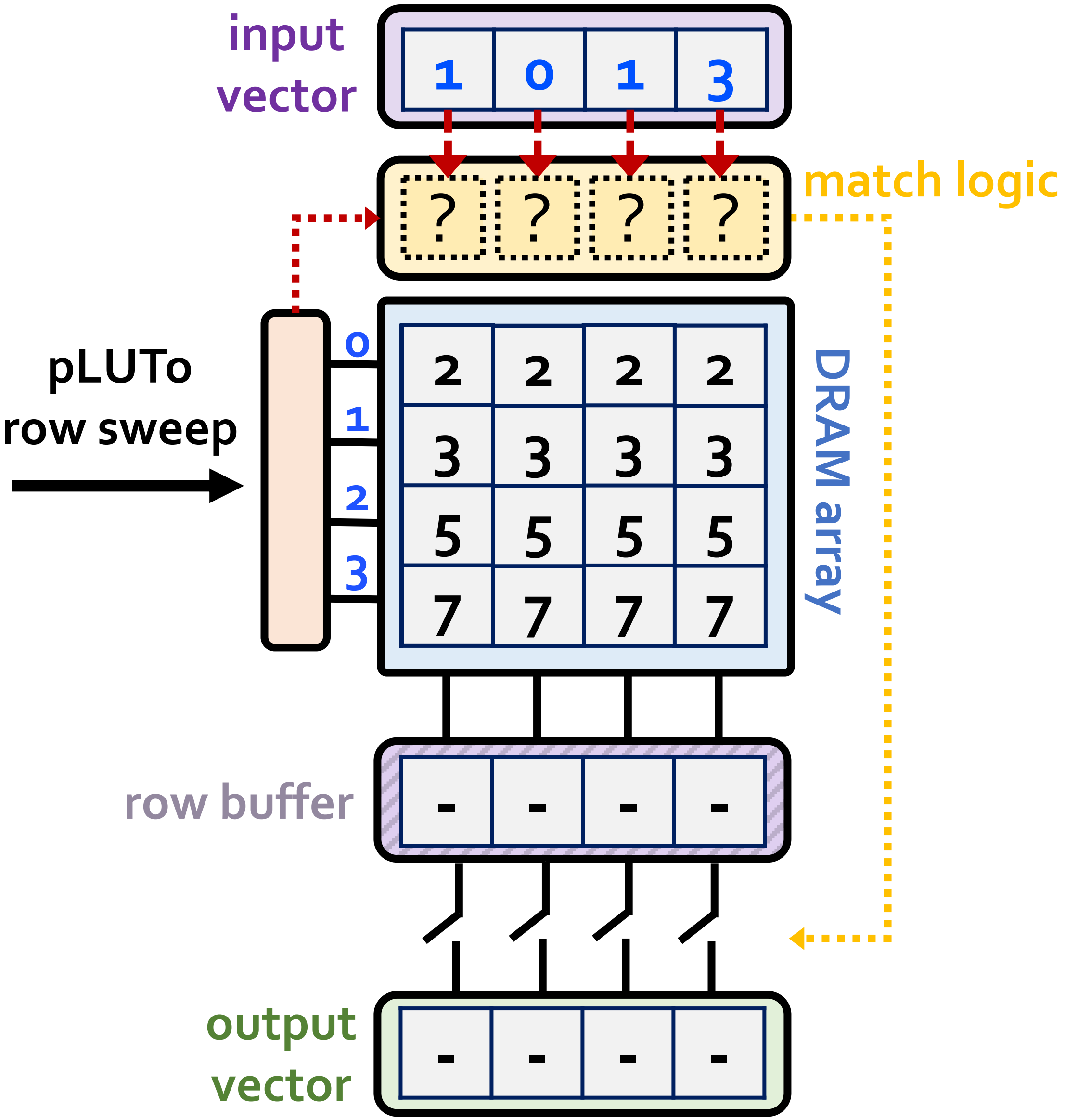
lookup table

1	0	1	3

input vector

3	2	3	7

output vector



In-DRAM pLUTo LUT Query: Step 1

LUT Query:
Return {2nd, 1st, 2nd, 4th}
prime numbers

	Prime numbers	
LUT index	i	f(i)
0	1st	2
1	2nd	3
2	3rd	5
3	4th	7

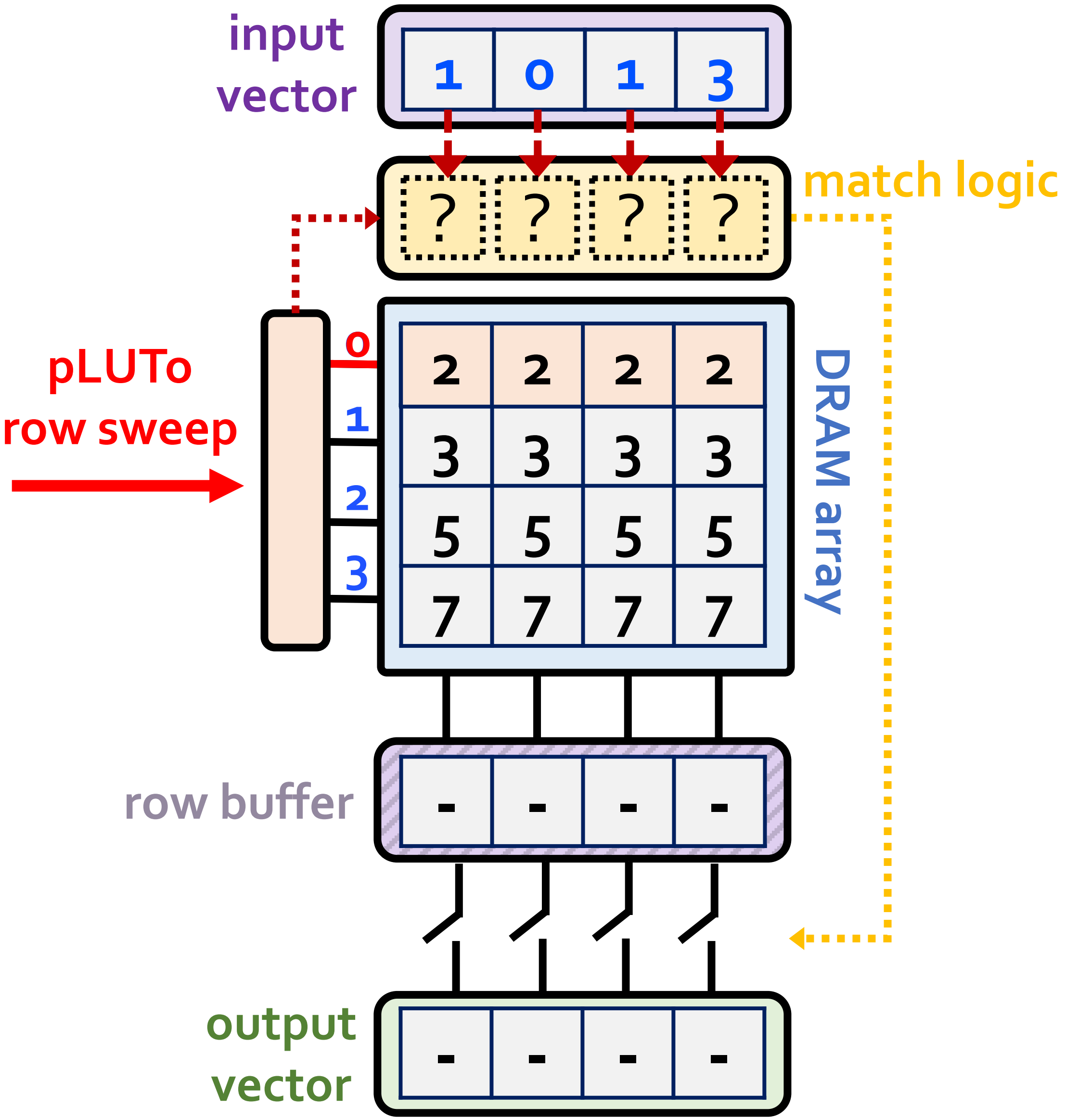
lookup table

1	0	1	3

input vector

3	2	3	7

output vector



In-DRAM pLUTo LUT Query: Step 1

LUT Query:
Return {2nd, 1st, 2nd, 4th}
prime numbers

LUT index	Prime numbers	
	i	f(i)
0	1st	2
1	2nd	3
2	3rd	5
3	4th	7

lookup table

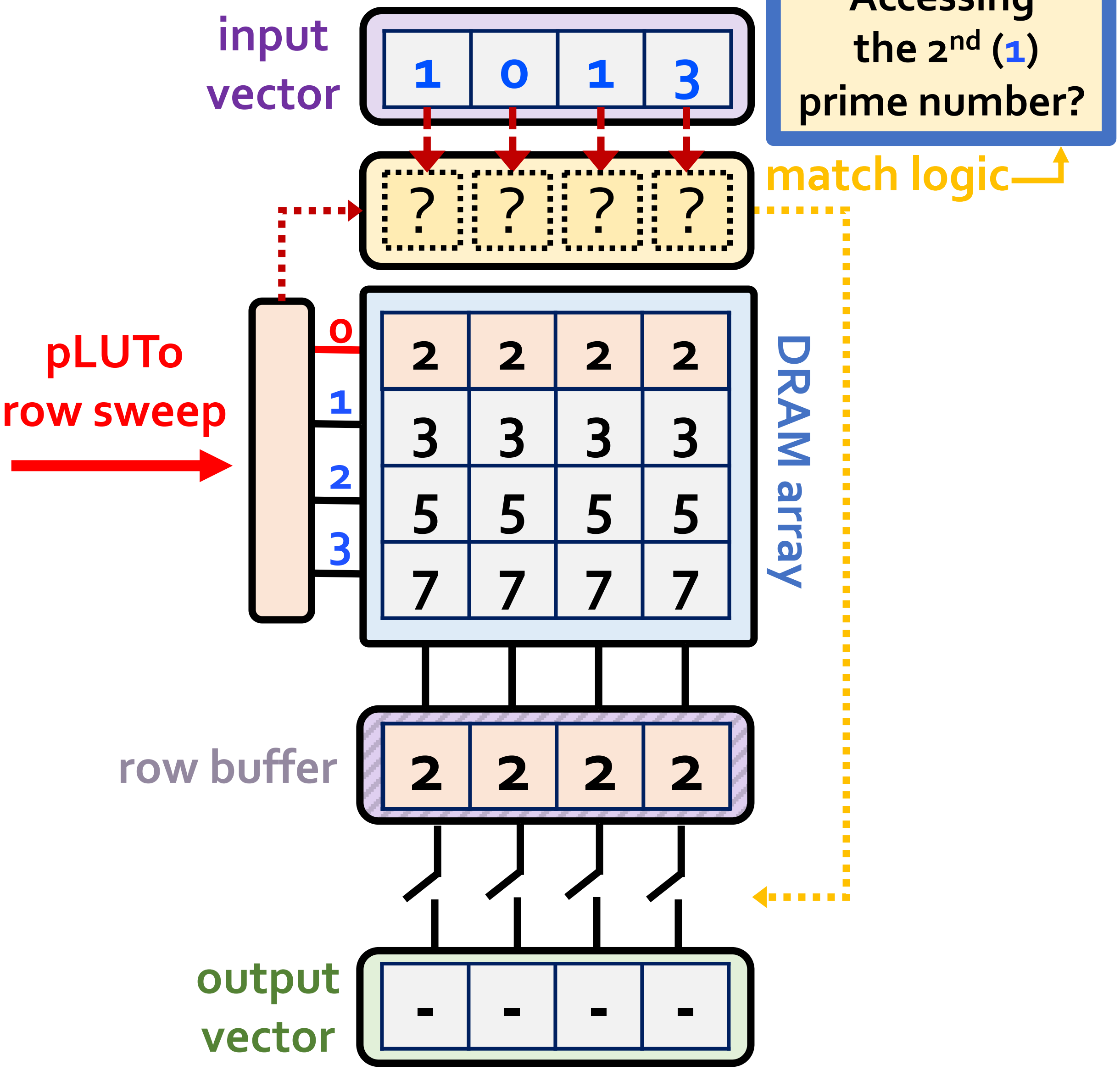
1	0	1	3

input vector

3	2	3	7

output vector

Accessing the 2nd (1) prime number?



In-DRAM pLUTo LUT Query: Step 1

LUT Query:
Return {2nd, 1st, 2nd, 4th}
prime numbers

	Prime numbers	
LUT index	i	f(i)
0	1st	2
1	2nd	3
2	3rd	5
3	4th	7

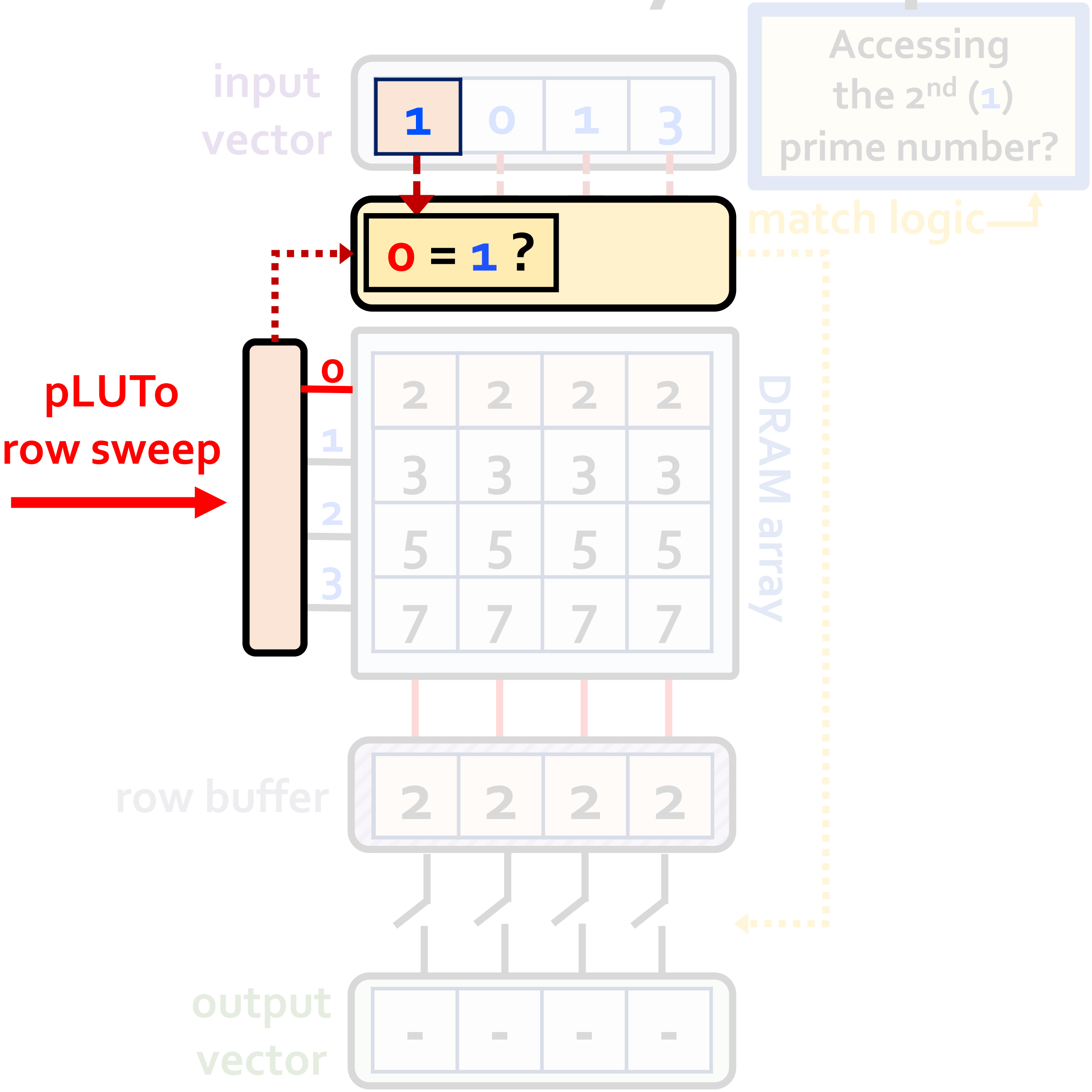
lookup table

1	0	1	3

input vector

3	2	3	7

output vector



In-DRAM pLUTo LUT Query: Step 1

LUT Query:
Return {2nd, 1st, 2nd, 4th}
prime numbers

	Prime numbers	
LUT index	i	f(i)
0	1st	2
1	2nd	3
2	3rd	5
3	4th	7

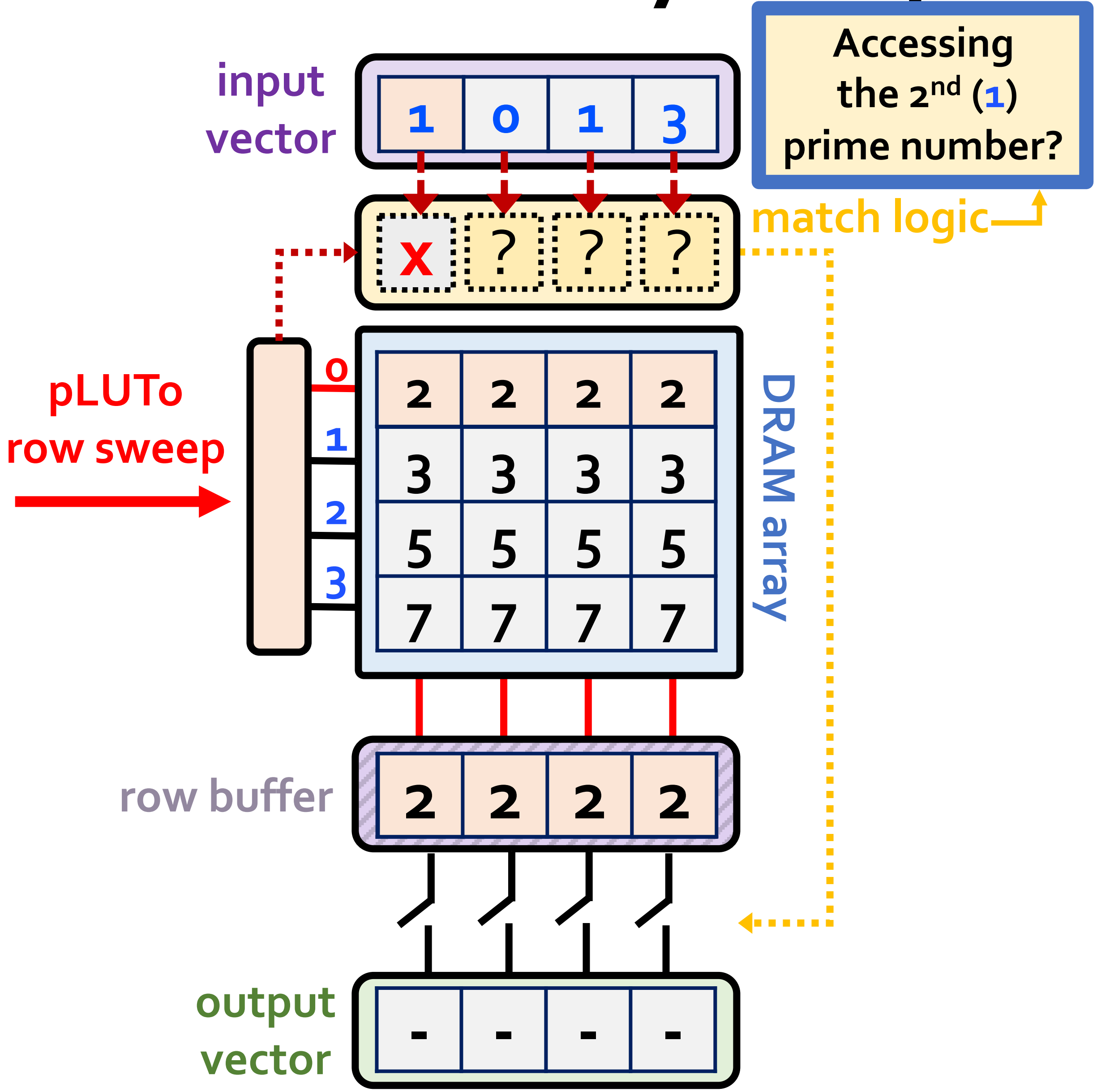
lookup table

1	0	1	3

input vector

3	2	3	7

output vector



Accessing the 2nd (1) prime number?

In-DRAM pLUTo LUT Query: Step 1

LUT Query:
Return {2nd, 1st, 2nd, 4th}
prime numbers

	Prime numbers	
LUT index	i	f(i)
0	1st	2
1	2nd	3
2	3rd	5
3	4th	7

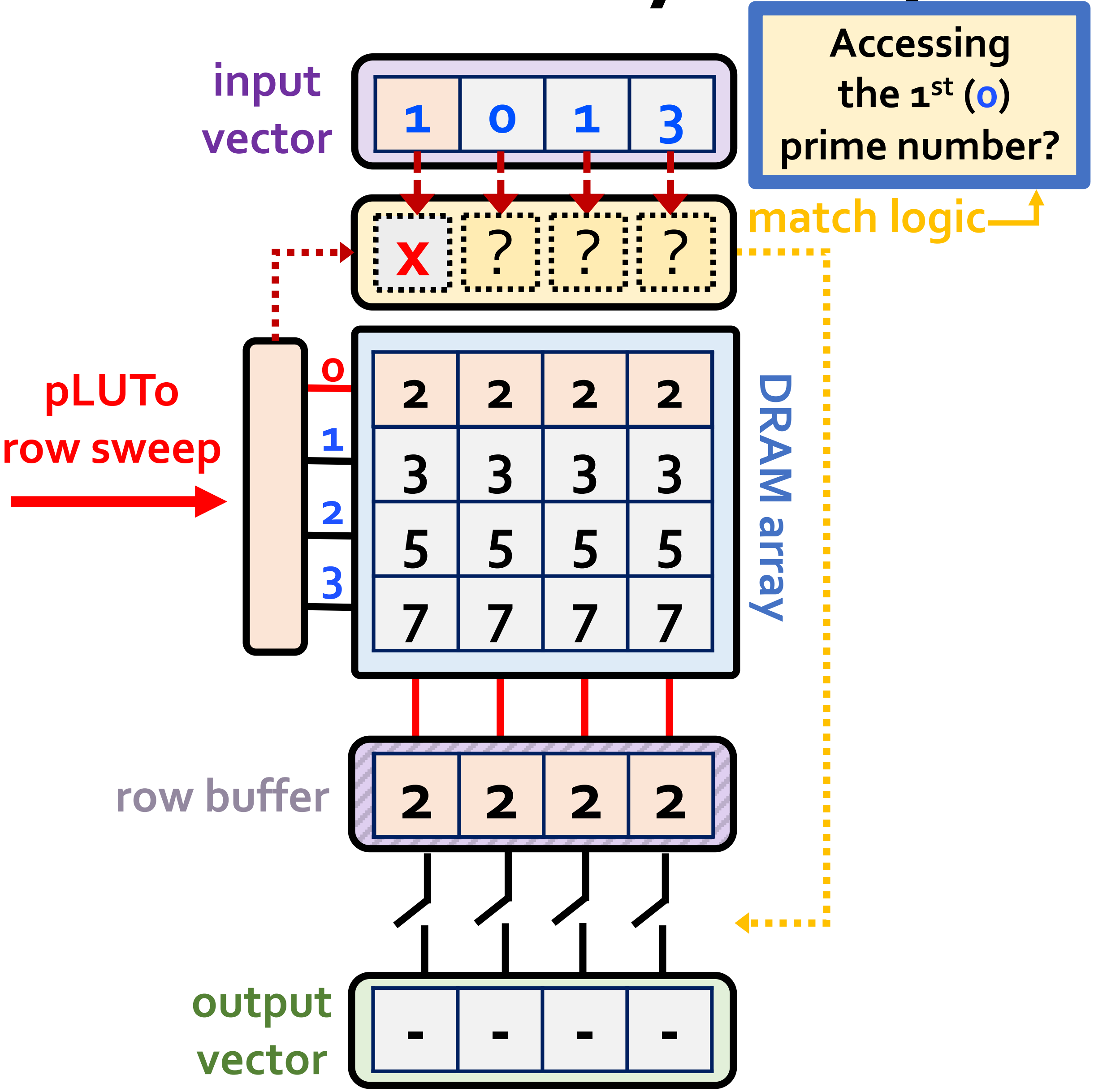
lookup table

1	0	1	3

input vector

3	2	3	7

output vector



In-DRAM pLUTo LUT Query: Step 1

LUT Query:
Return {2nd, 1st, 2nd, 4th}
prime numbers

	Prime numbers	
LUT index	i	f(i)
0	1st	2
1	2nd	3
2	3rd	5
3	4th	7

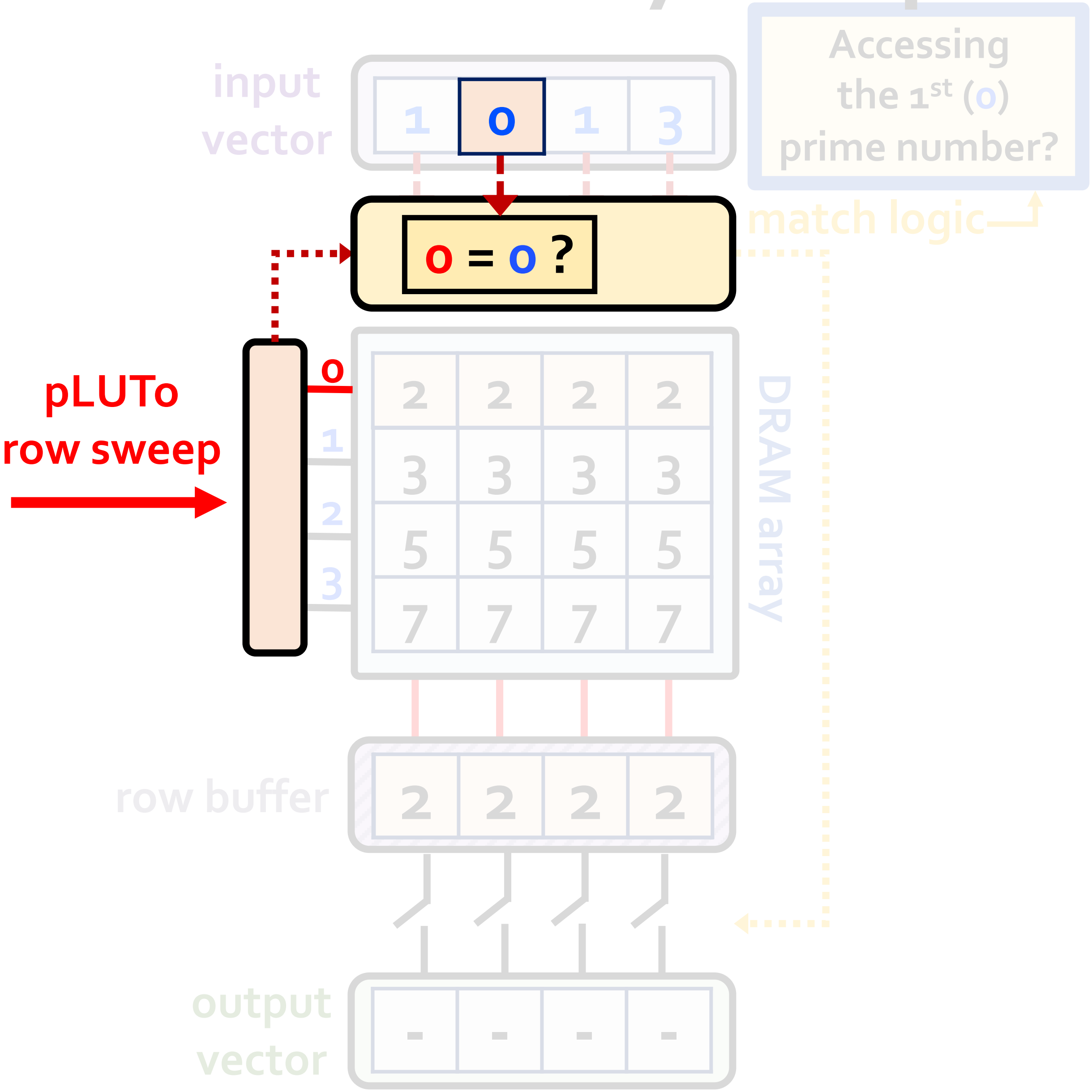
lookup table

1	0	1	3

input vector

3	2	3	7

output vector



Accessing the 1st (0) prime number?

In-DRAM pLUTo LUT Query: Step 1

LUT Query:
Return {2nd, 1st, 2nd, 4th} *prime numbers*

	Prime numbers	
LUT index	i	f(i)
0	1st	2
1	2nd	3
2	3rd	5
3	4th	7

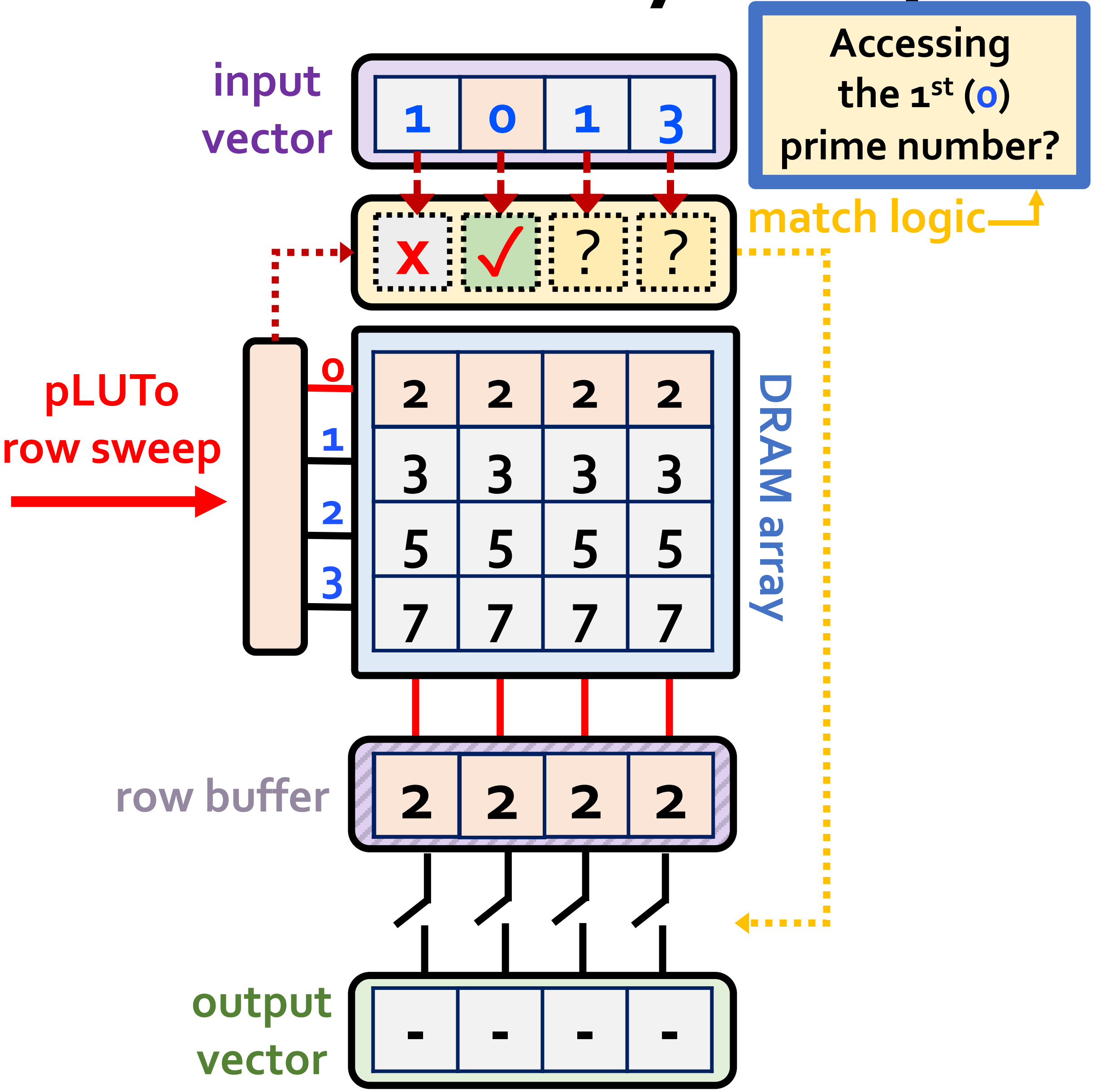
lookup table

1	0	1	3

input vector

3	2	3	7

output vector



Accessing the 1st (0) prime number?

In-DRAM pLUTo LUT Query: Step 1

LUT Query:
Return {2nd, 1st, 2nd, 4th}
prime numbers

	Prime numbers	
LUT index	i	f(i)
0	1st	2
1	2nd	3
2	3rd	5
3	4th	7

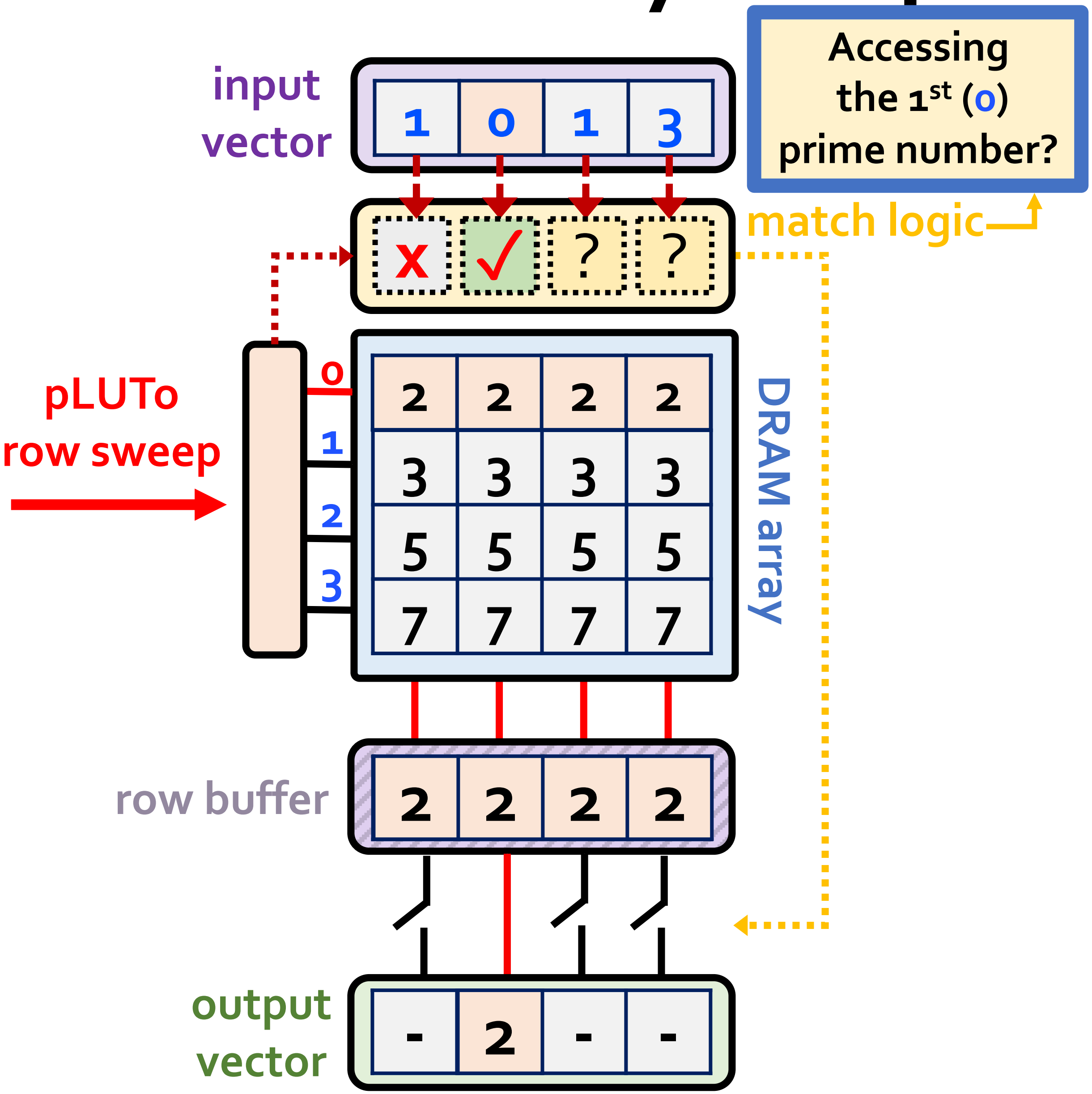
lookup table

1	0	1	3

input vector

3	2	3	7

output vector



Accessing the 1st (0) prime number?

In-DRAM pLUTo LUT Query: Step 1

LUT Query:
Return {2nd, 1st, 2nd, 4th}
prime numbers

	Prime numbers	
LUT index	i	f(i)
0	1st	2
1	2nd	3
2	3rd	5
3	4th	7

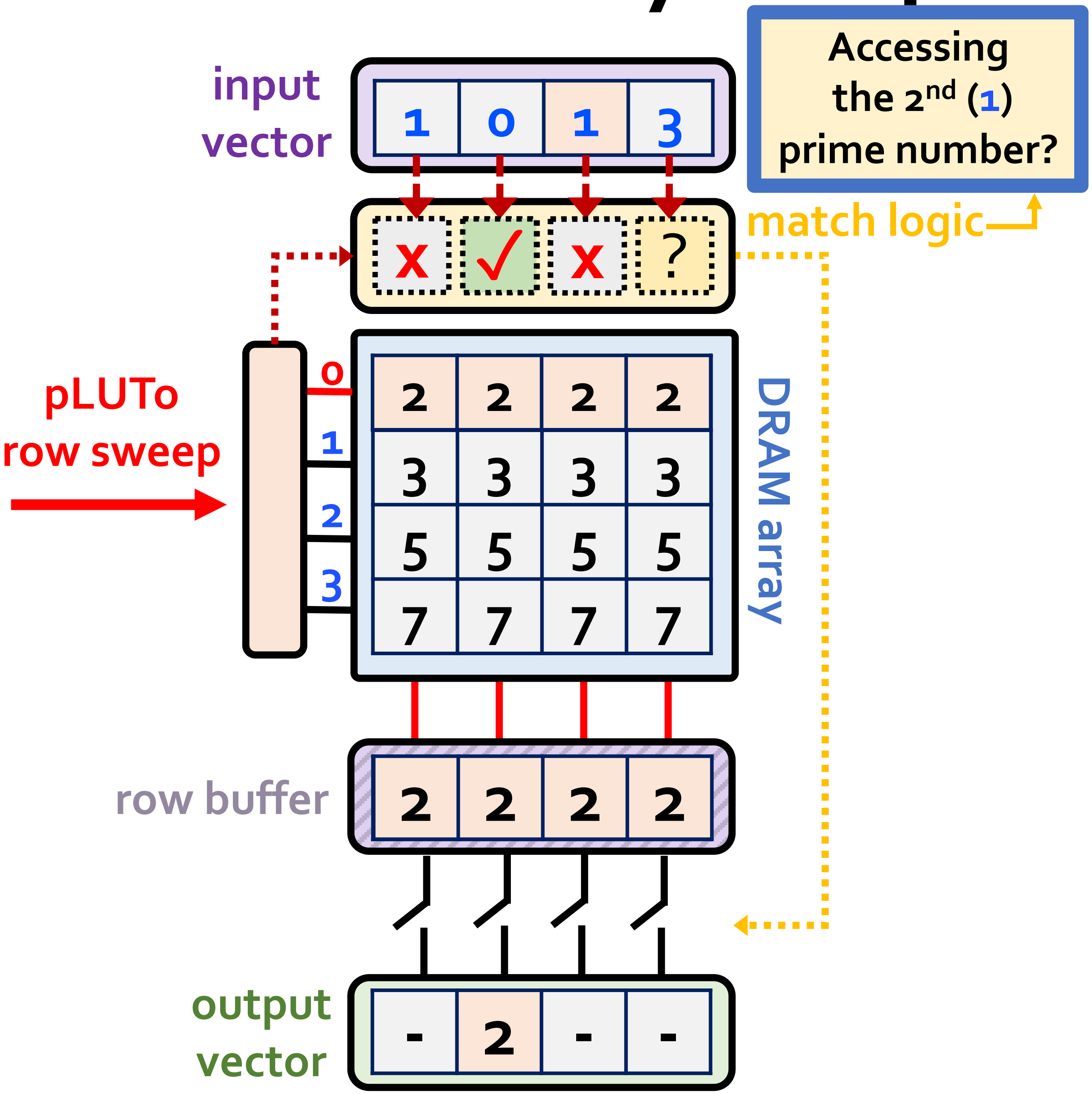
lookup table

1	0	1	3

input vector

3	2	3	7

output vector



In-DRAM pLUTo LUT Query: Step 1

LUT Query:
Return {2nd, 1st, 2nd, 4th}
prime numbers

		Prime numbers
LUT index	i	f(i)
0	1st	2
1	2nd	3
2	3rd	5
3	4th	7

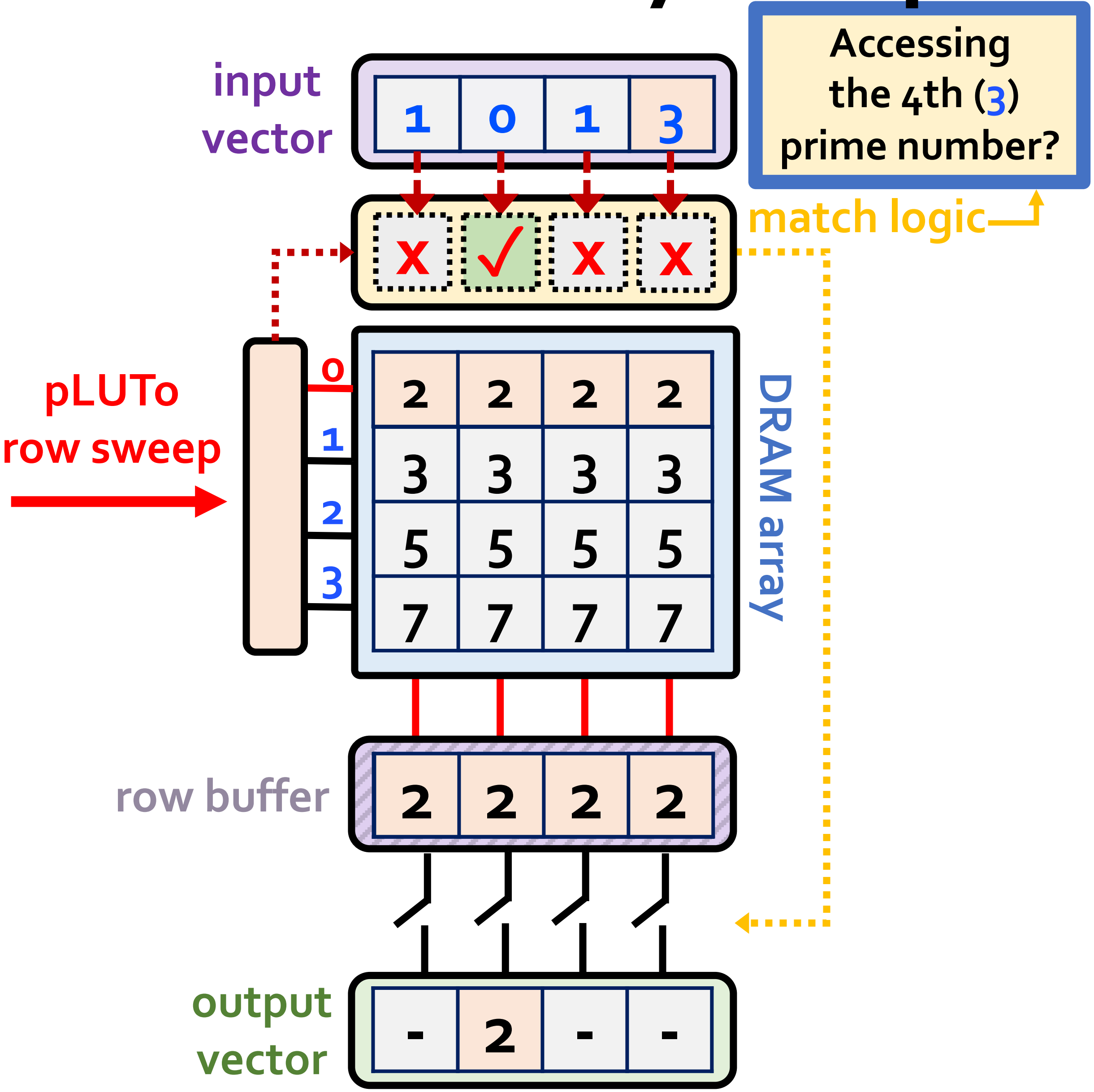
lookup table

1	0	1	3

input vector

3	2	3	7

output vector



Accessing the 4th (3) prime number?

In-DRAM pLUTo LUT Query: Step 2

LUT Query:
Return {2nd, 1st, 2nd, 4th}
prime numbers

	Prime numbers	
LUT index	i	f(i)
0	1st	2
1	2nd	3
2	3rd	5
3	4th	7

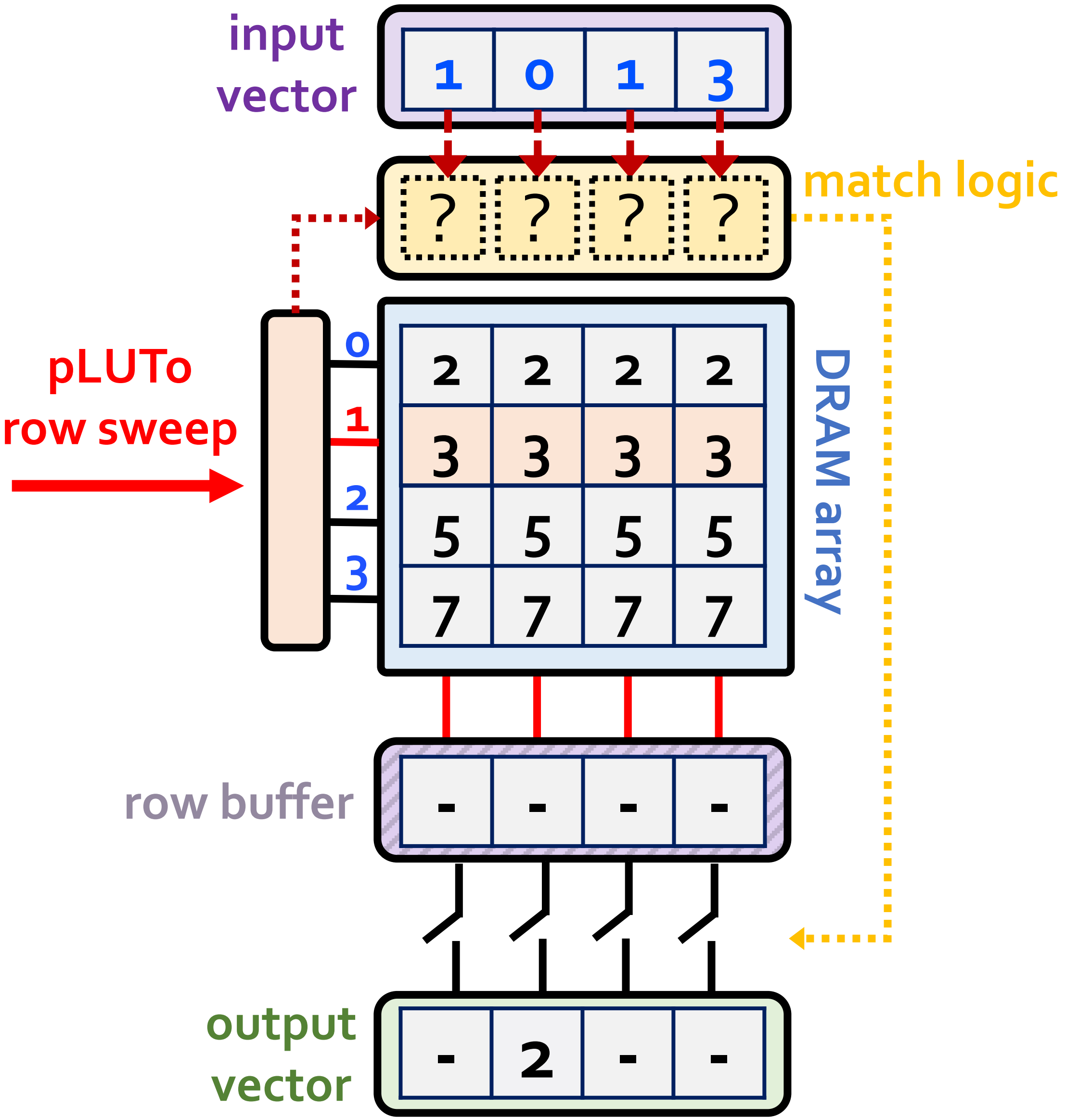
lookup table

1	0	1	3

input vector

3	2	3	7

output vector



In-DRAM pLUTo LUT Query: Step 2

LUT Query:
Return {2nd, 1st, 2nd, 4th}
prime numbers

	Prime numbers	
LUT index	i	f(i)
0	1st	2
1	2nd	3
2	3rd	5
3	4th	7

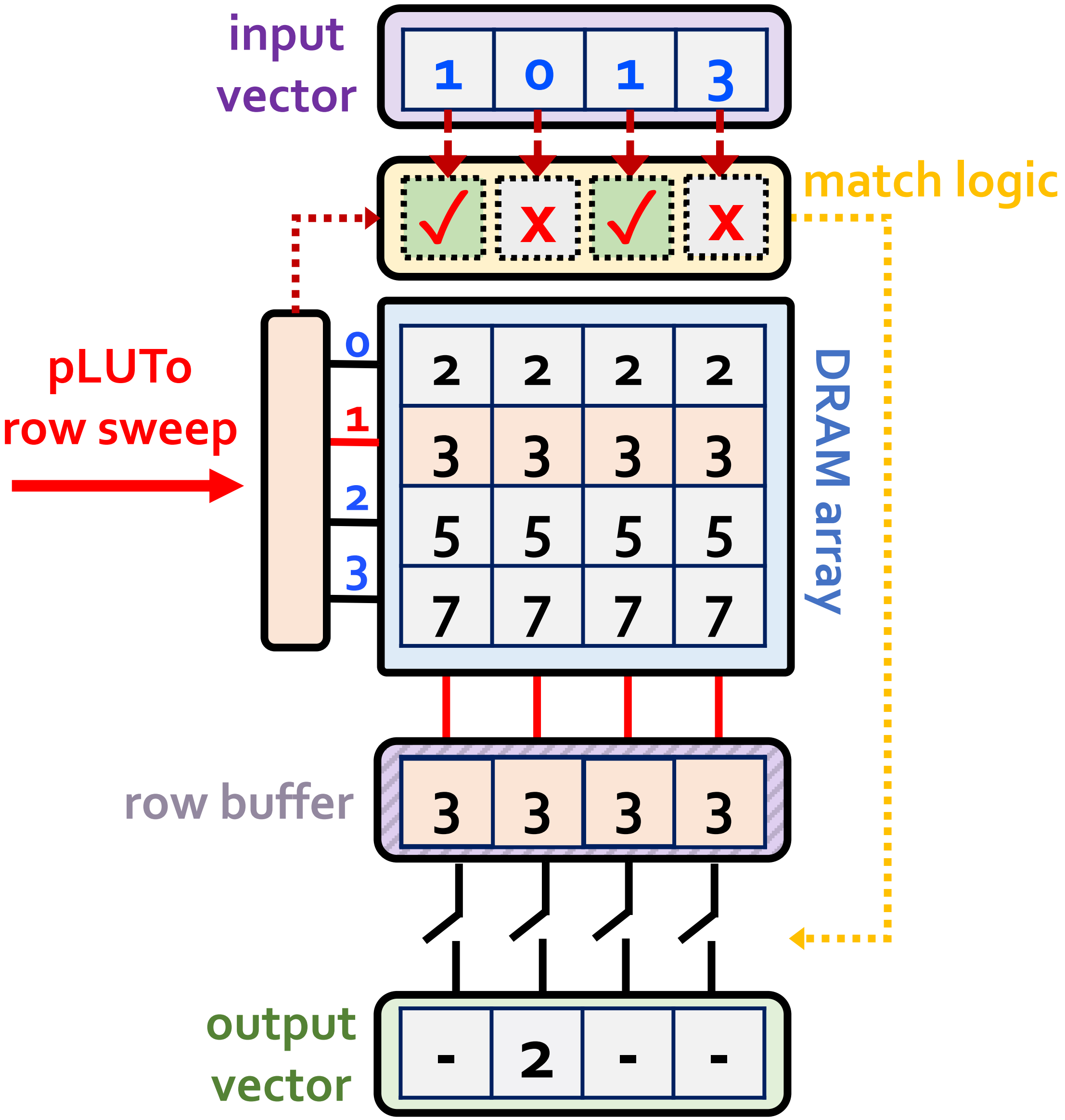
lookup table

1	0	1	3

input vector

3	2	3	7

output vector



In-DRAM pLUTo LUT Query: Step 2

LUT Query:
Return {2nd, 1st, 2nd, 4th}
prime numbers

	Prime numbers	
LUT index	i	f(i)
0	1st	2
1	2nd	3
2	3rd	5
3	4th	7

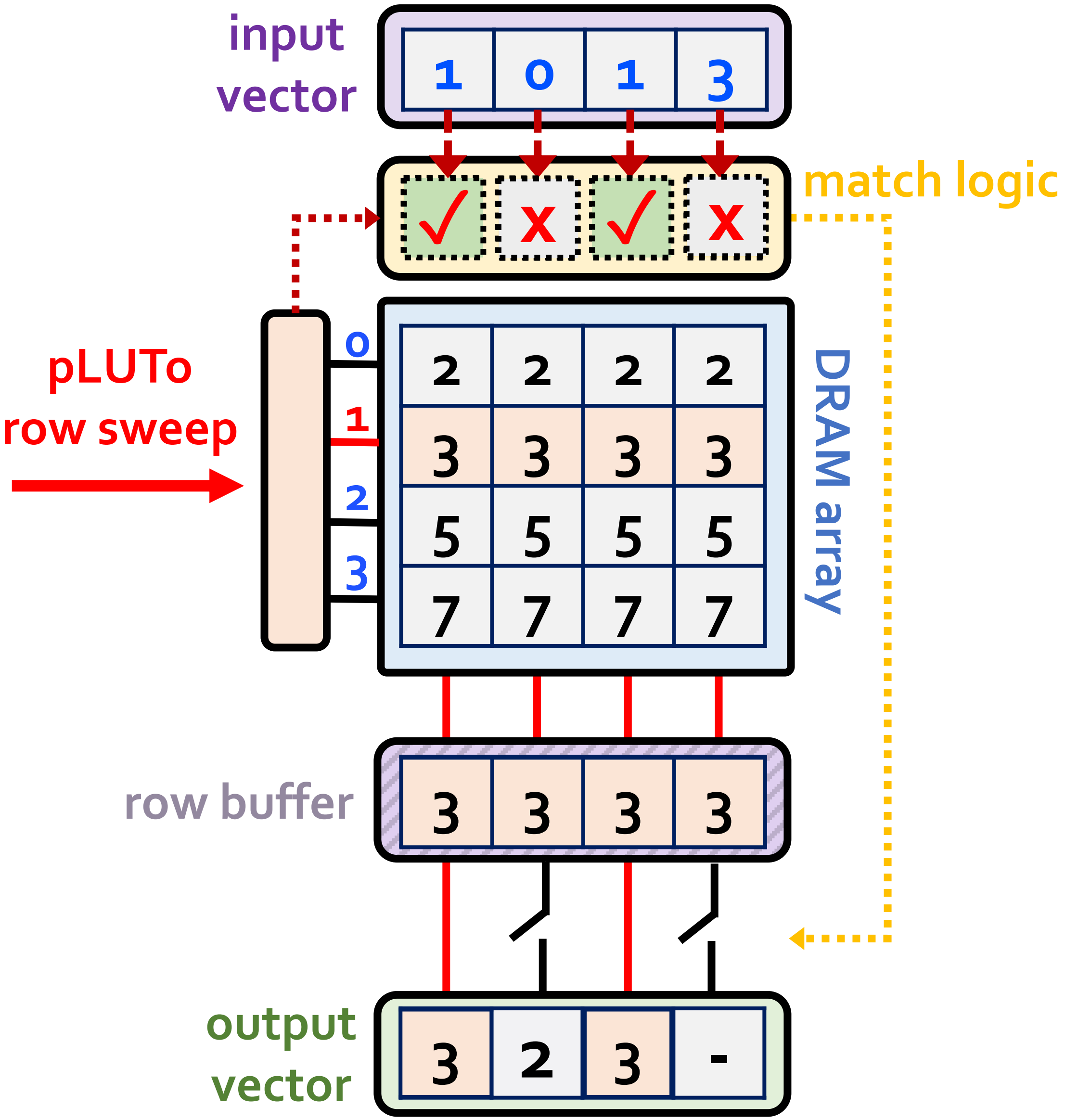
lookup table

1	0	1	3

input vector

3	2	3	7

output vector



In-DRAM pLUTo LUT Query: Step 3

LUT Query:
Return {2nd, 1st, 2nd, 4th}
prime numbers

	Prime numbers	
LUT index	i	f(i)
0	1st	2
1	2nd	3
2	3rd	5
3	4th	7

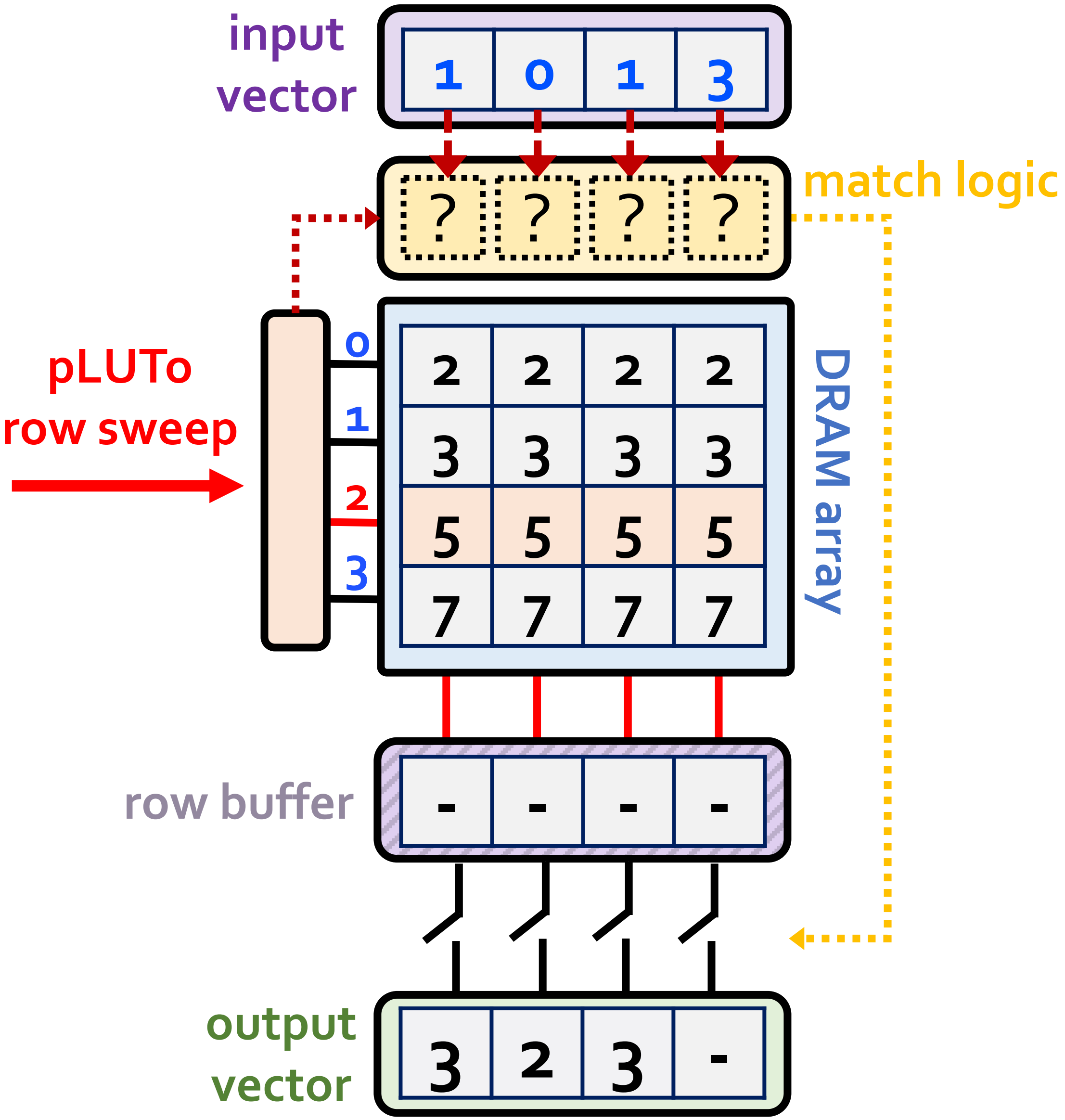
lookup table

1	0	1	3

input vector

3	2	3	7

output vector



In-DRAM pLUTo LUT Query: Step 3

LUT Query:
Return {2nd, 1st, 2nd, 4th}
prime numbers

	Prime numbers	
LUT index	i	f(i)
0	1st	2
1	2nd	3
2	3rd	5
3	4th	7

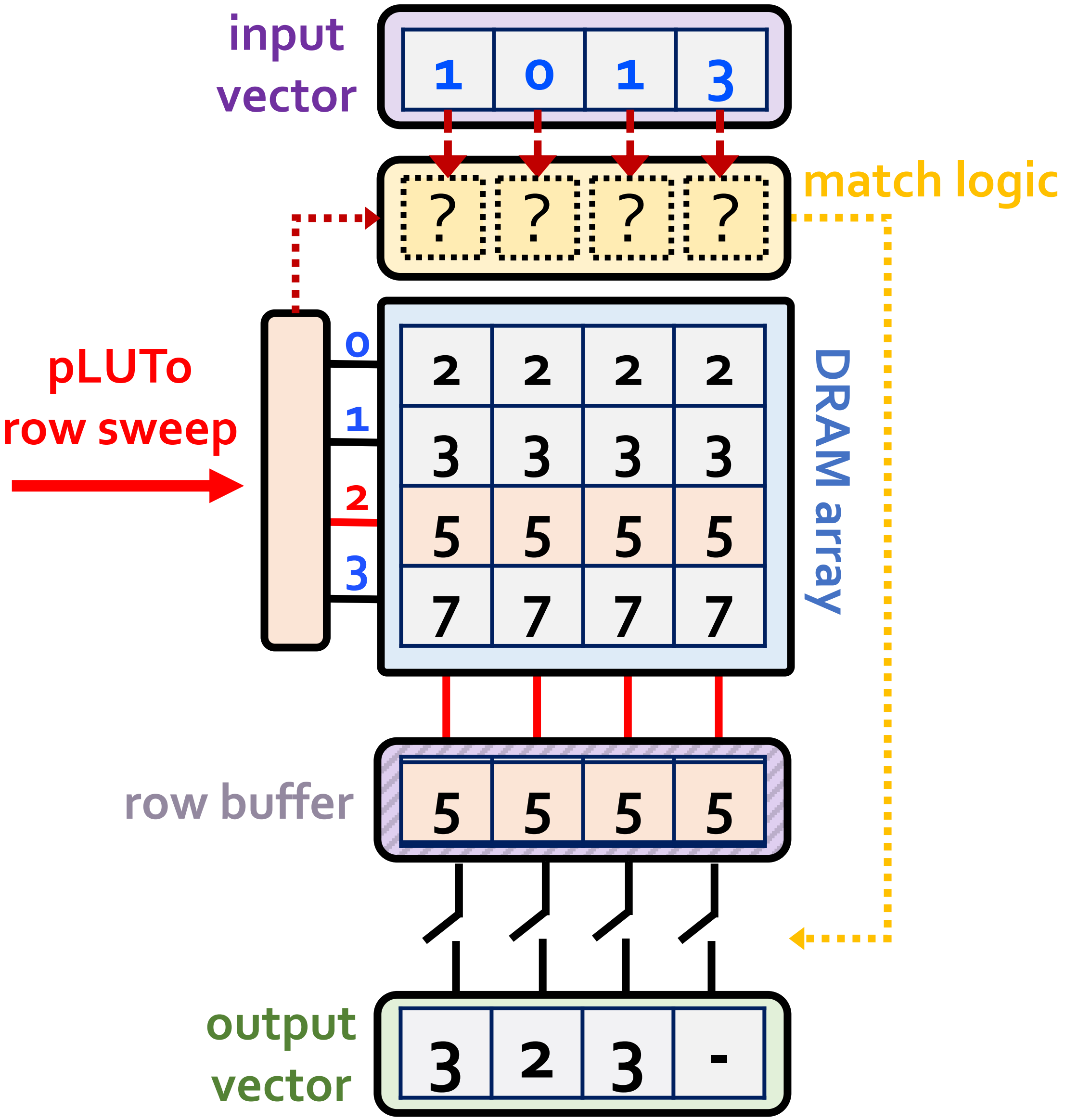
lookup table

1	0	1	3

input vector

3	2	3	7

output vector



In-DRAM pLUTo LUT Query: Step 3

LUT Query:
Return {2nd, 1st, 2nd, 4th}
prime numbers

		Prime numbers
LUT index	i	f(i)
0	1st	2
1	2nd	3
2	3rd	5
3	4th	7

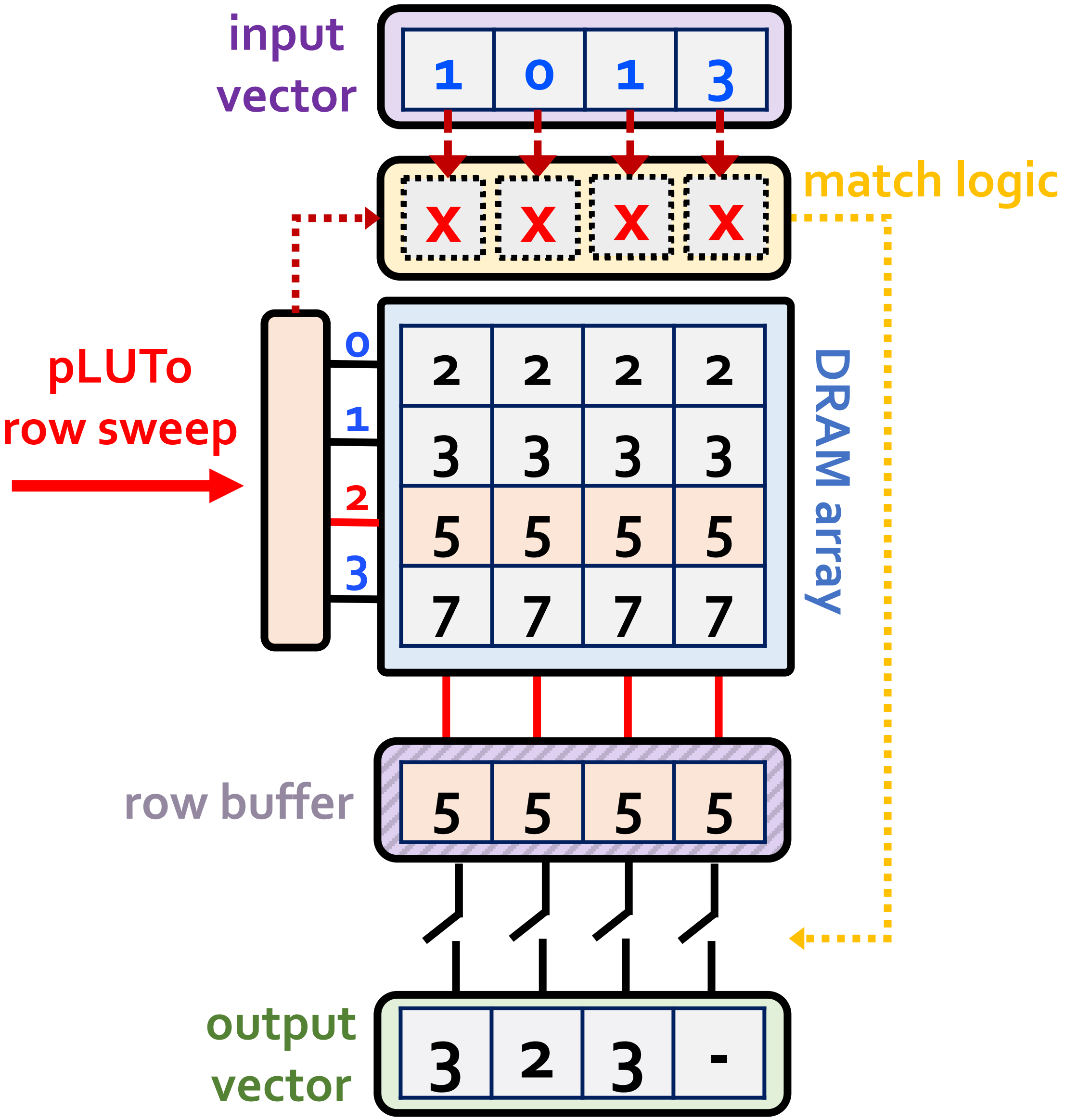
lookup table

1	0	1	3

input vector

3	2	3	7

output vector



In-DRAM pLUTo LUT Query: Step 4

LUT Query:
Return {2nd, 1st, 2nd, 4th}
prime numbers

	Prime numbers	
LUT index	i	f(i)
0	1st	2
1	2nd	3
2	3rd	5
3	4th	7

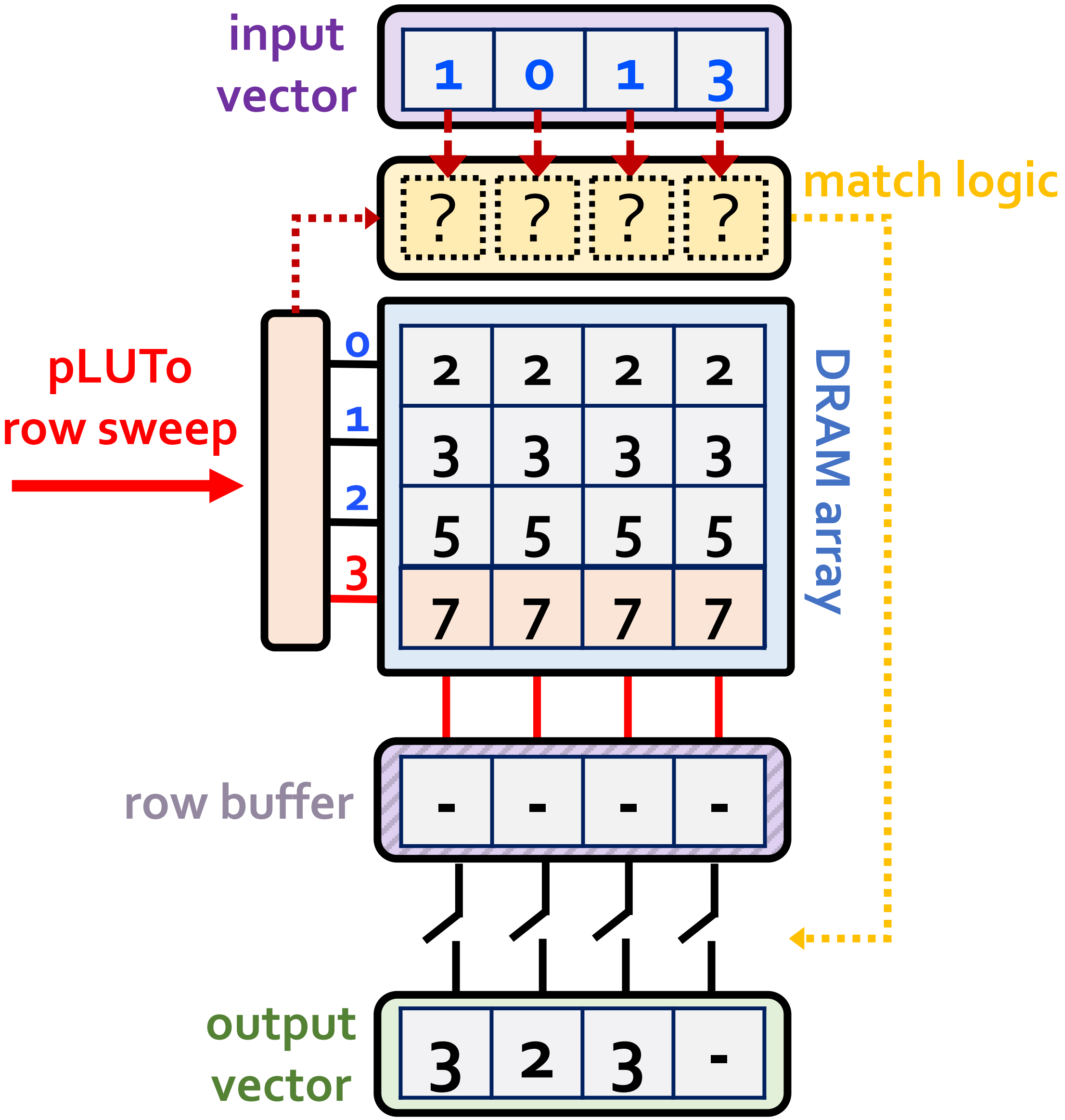
lookup table

1	0	1	3

input vector

3	2	3	7

output vector



In-DRAM pLUTo LUT Query: Step 4

LUT Query:
Return {2nd, 1st, 2nd, 4th}
prime numbers

	Prime numbers	
LUT index	i	f(i)
0	1st	2
1	2nd	3
2	3rd	5
3	4th	7

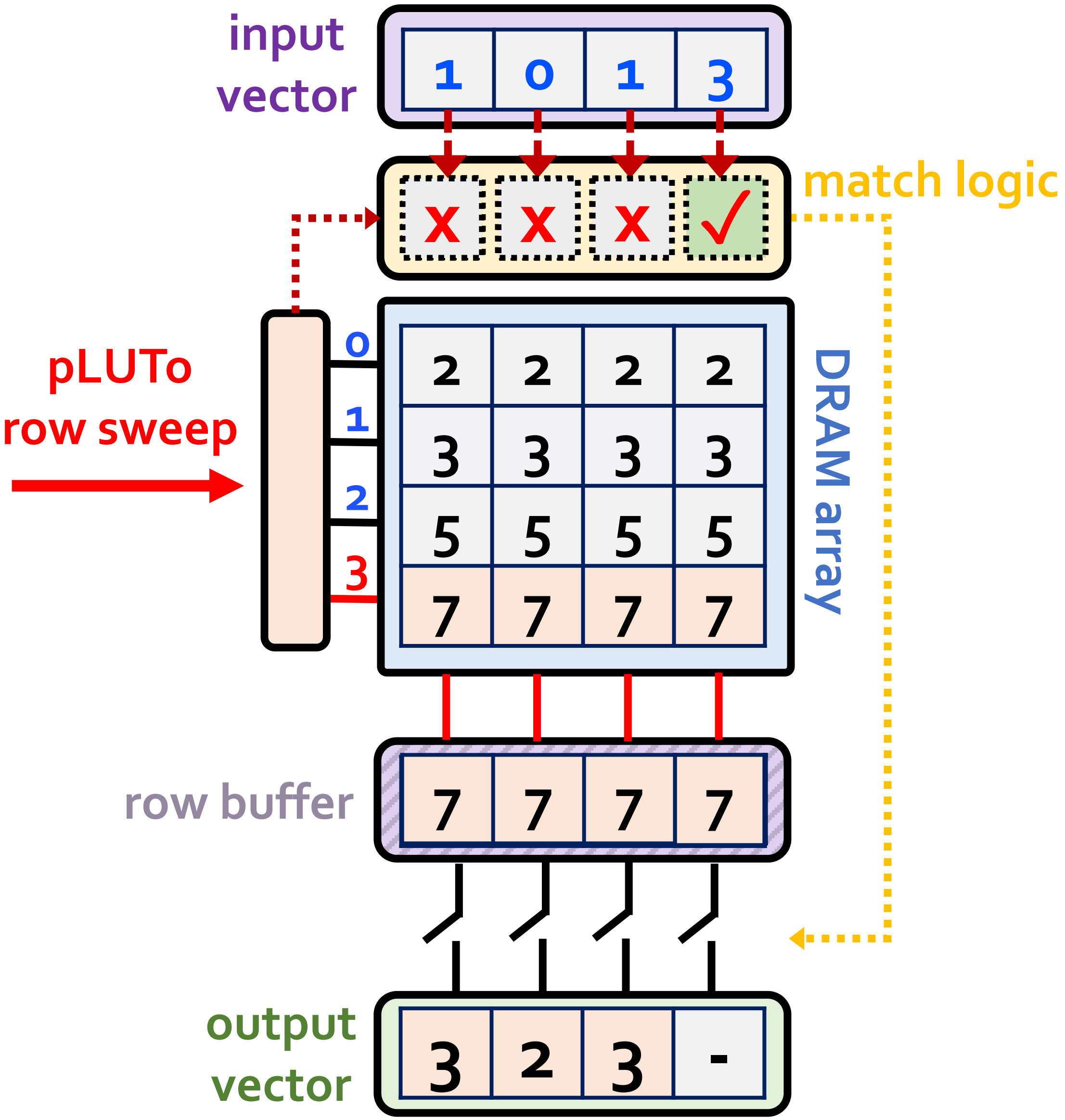
lookup table

1	0	1	3

input vector

3	2	3	7

output vector



In-DRAM pLUTo LUT Query: Step 4

LUT Query:
Return {2nd, 1st, 2nd, 4th}
prime numbers

	Prime numbers	
LUT index	i	f(i)
0	1st	2
1	2nd	3
2	3rd	5
3	4th	7

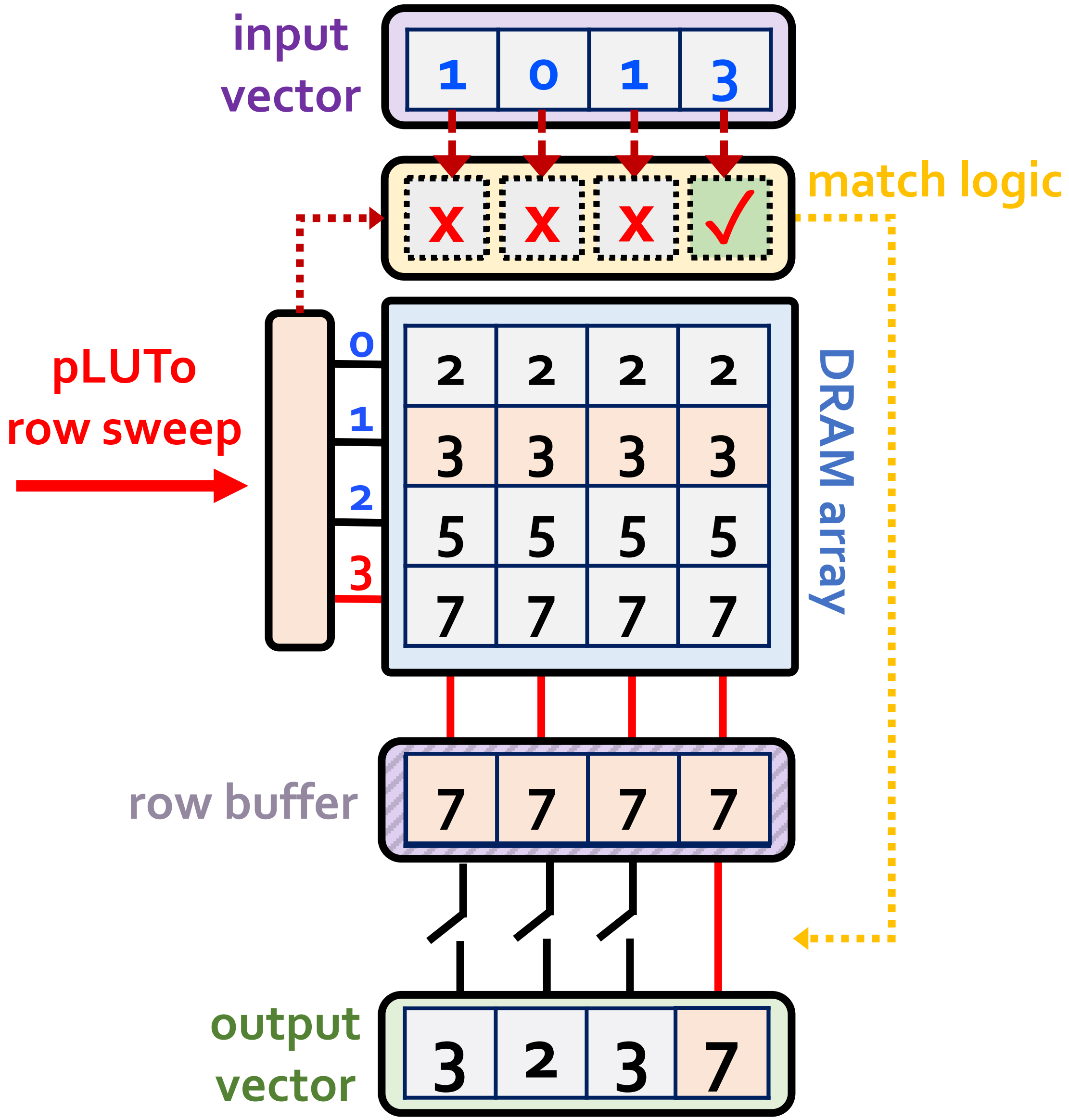
lookup table

1	0	1	3

input vector

3	2	3	7

output vector



Contents

Introduction

pLUTo

Overview

pLUTo Designs

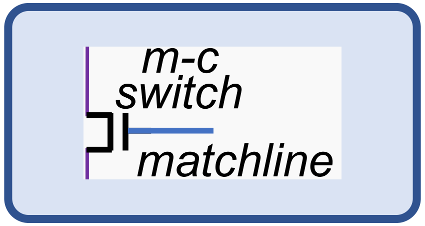
System Integration

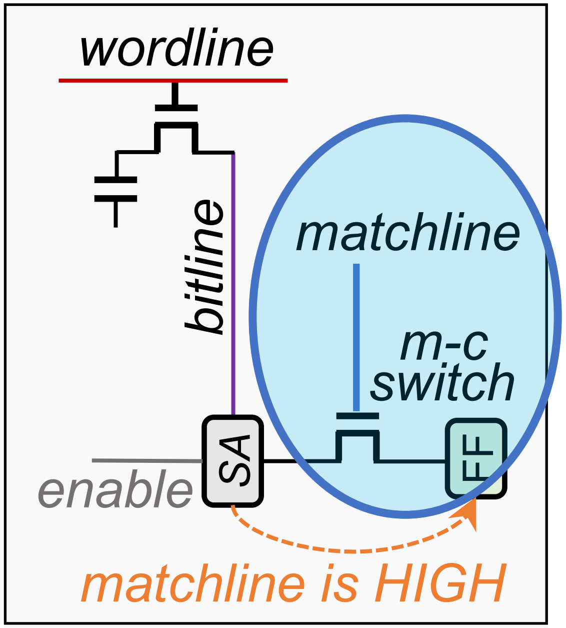
Evaluation

Conclusion

pLUTo Designs

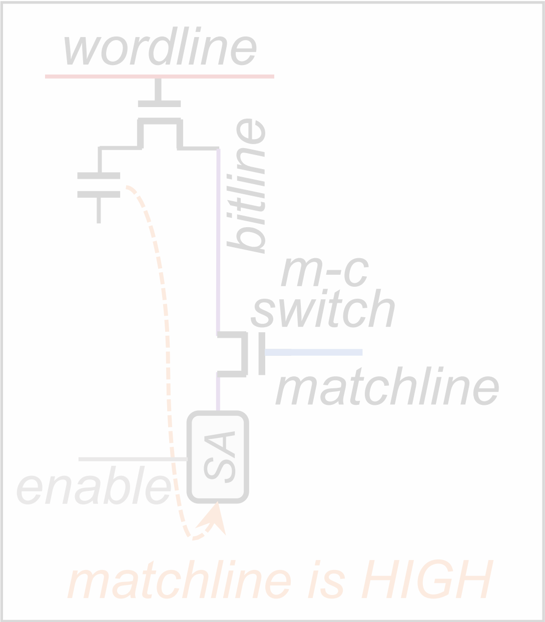
- **Match Logic:** shared by the three pLUTo designs





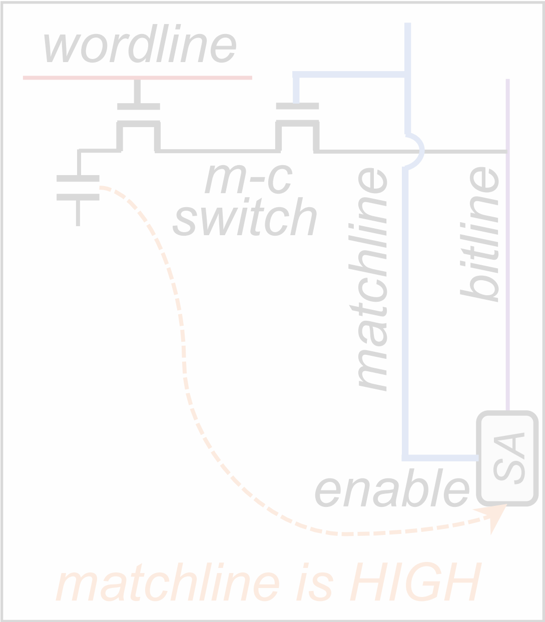
BSA

Buffered Sense Amplifier



GSA

Gated Sense Amplifier

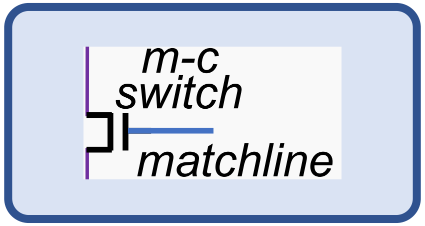


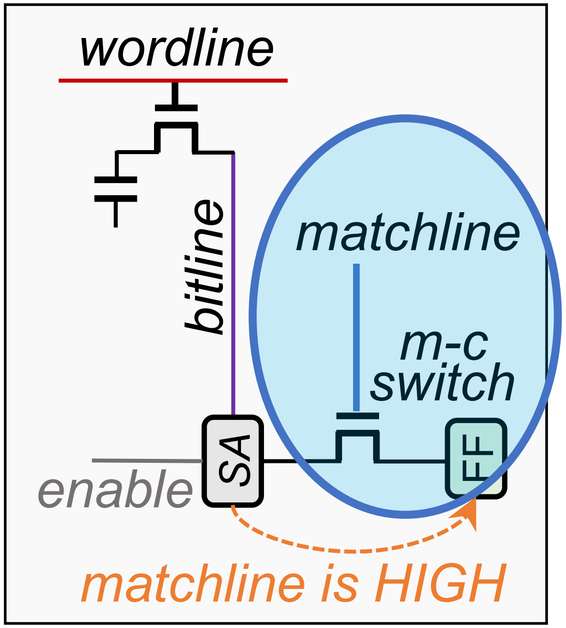
GMC

Gated Memory Cell

pLUTo Designs

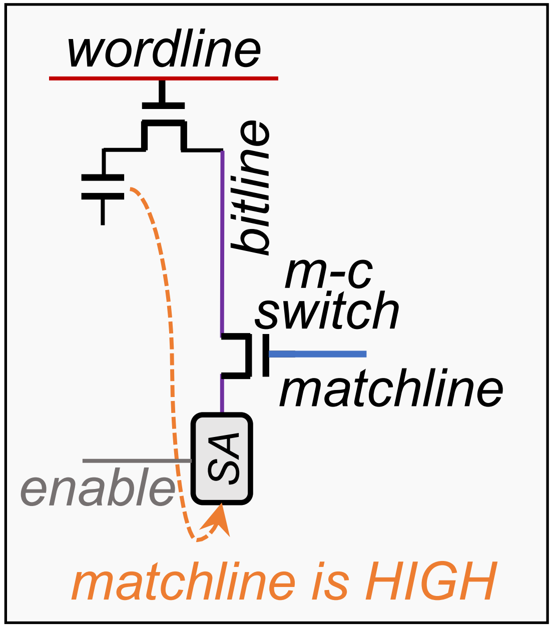
- **Match Logic:** shared by the three pLUTo designs





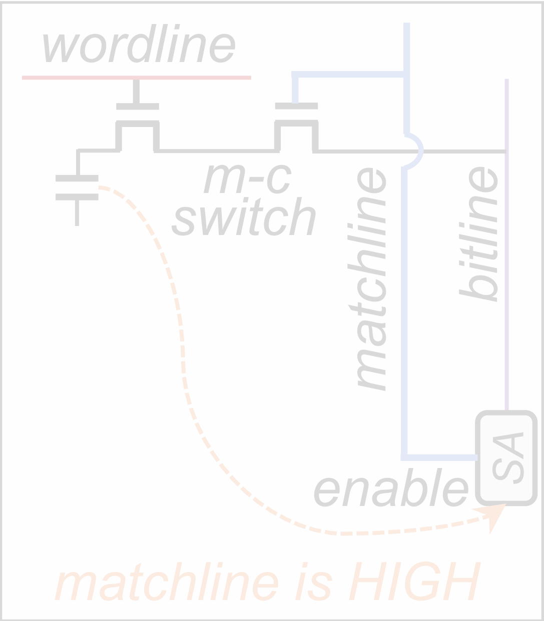
BSA

Buffered Sense Amplifier



GSA

Gated Sense Amplifier

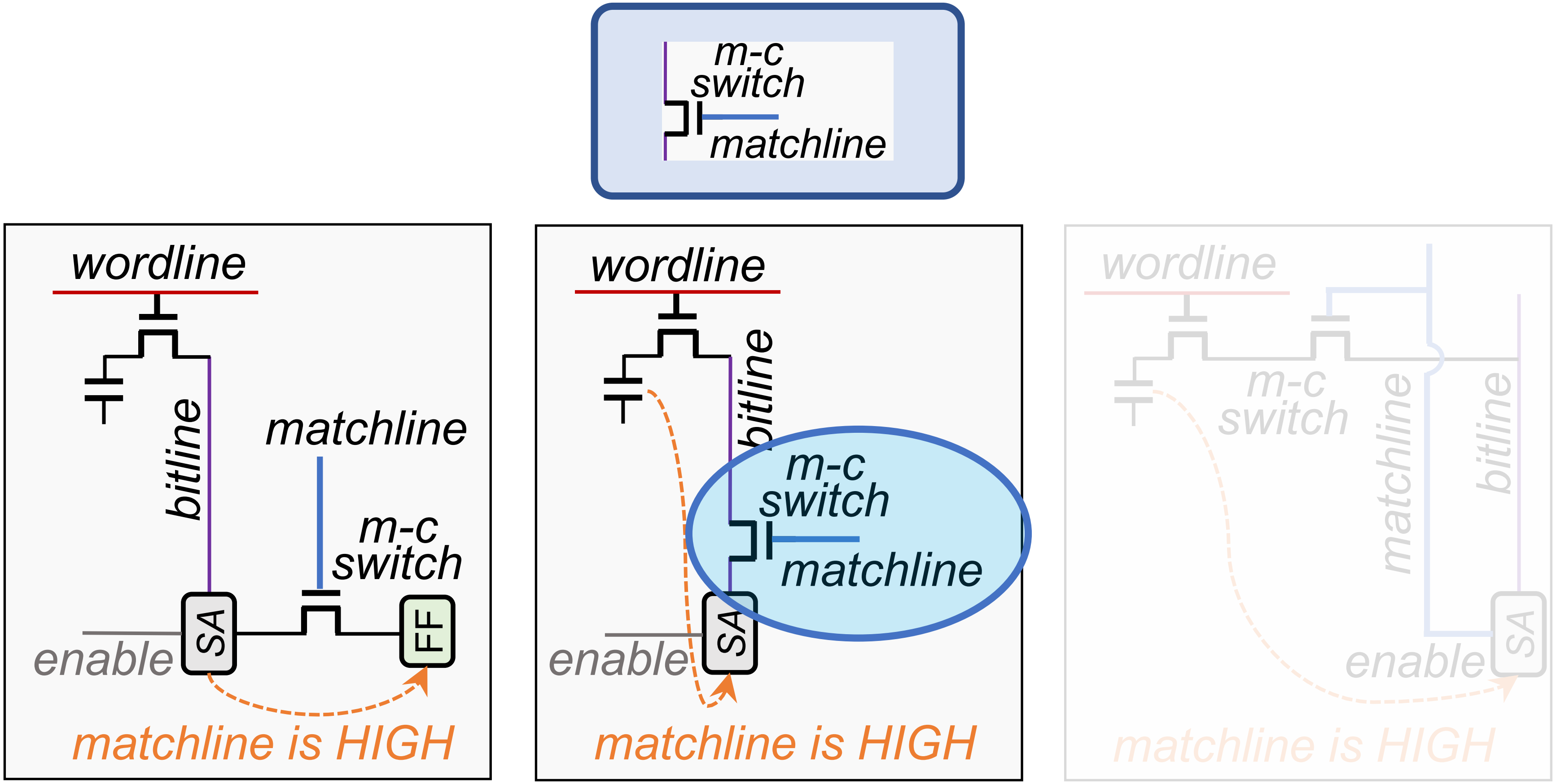


GMC

Gated Memory Cell

pLUTo Designs

- **Match Logic:** shared by the three pLUTo designs



BSA
Buffered Sense Amplifier

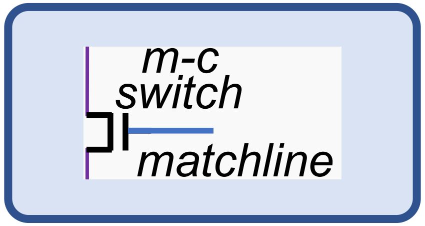
GSA
Gated Sense Amplifier

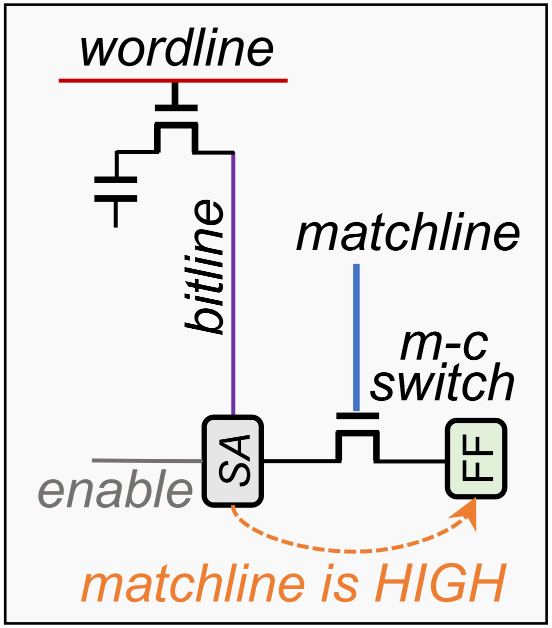
GMC
Gated Memory Cell

SAFARI

pLUTo Designs

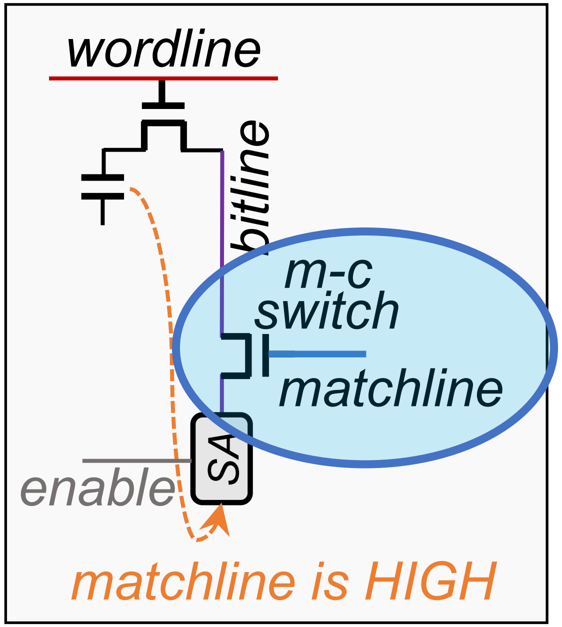
- **Match Logic:** shared by the three pLUTo designs





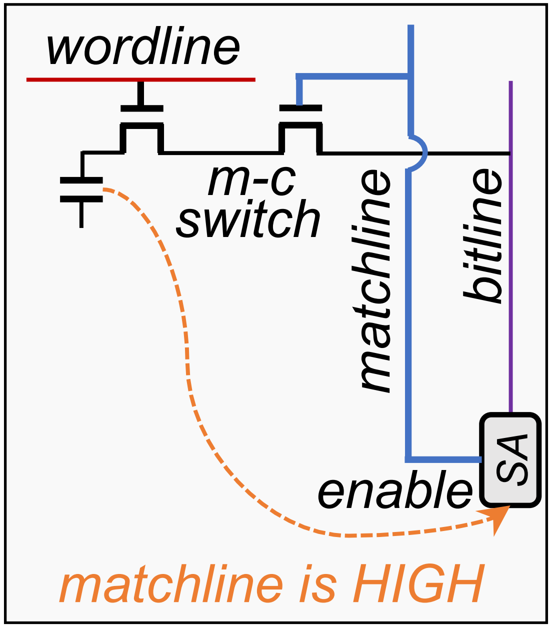
BSA

Buffered Sense Amplifier



GSA

Gated Sense Amplifier

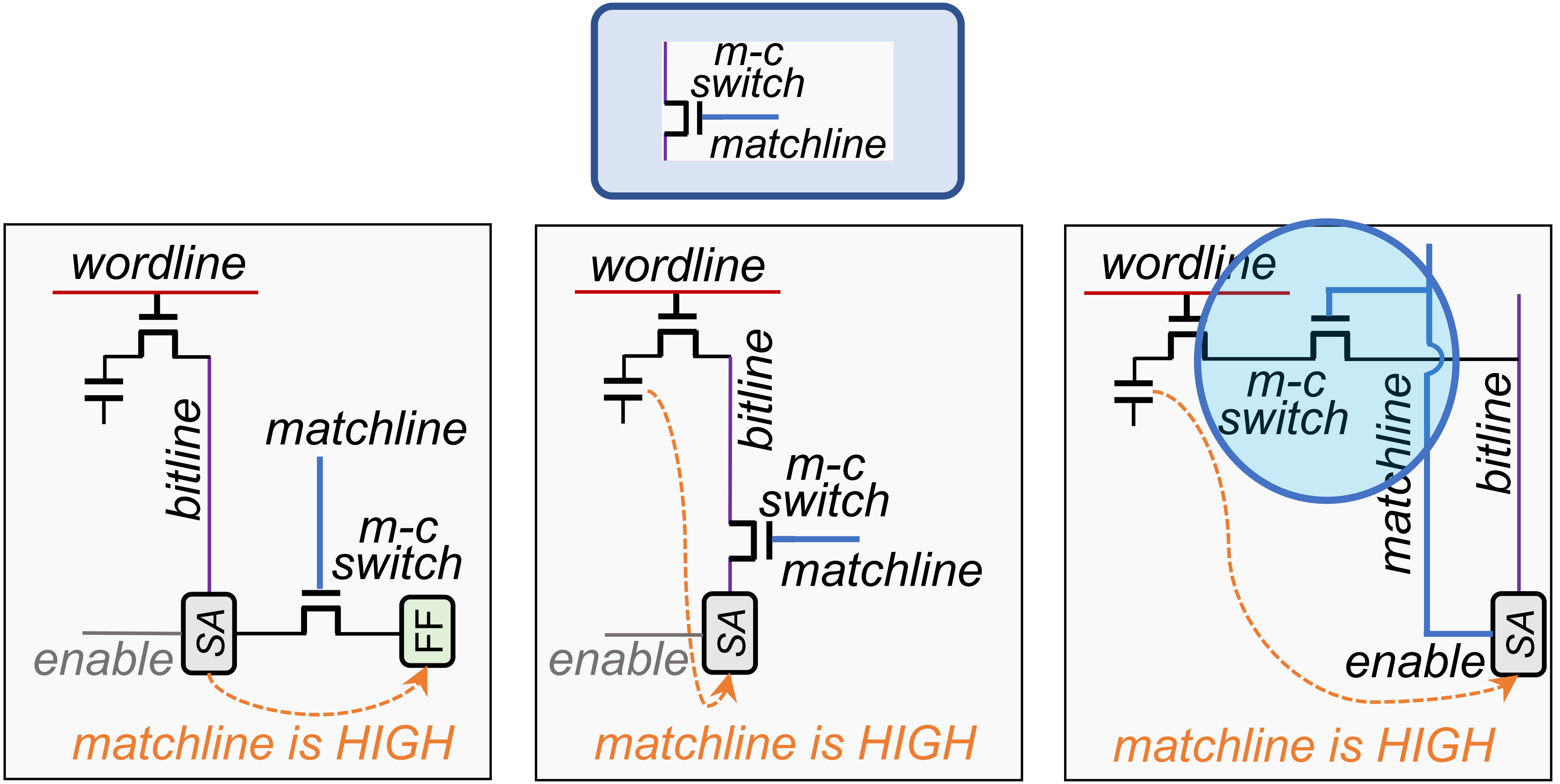


GMC

Gated Memory Cell

pLUTo Designs

- **Match Logic:** shared by the three pLUTo designs



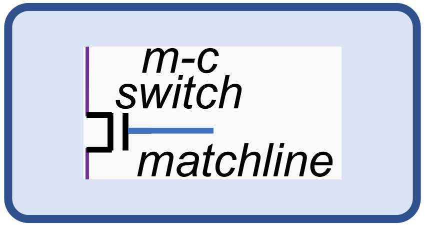
BSA
Buffered Sense Amplifier

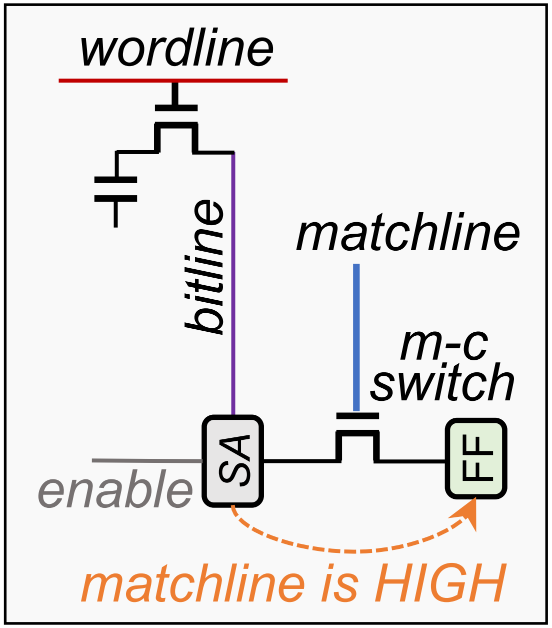
GSA
Gated Sense Amplifier

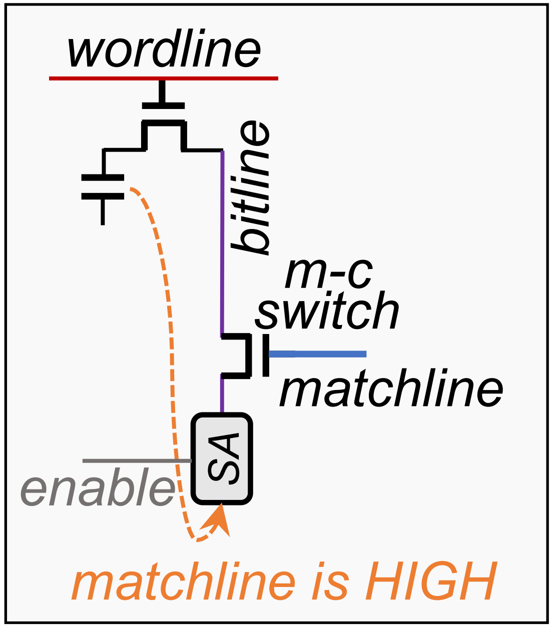
GMC
Gated Memory Cell

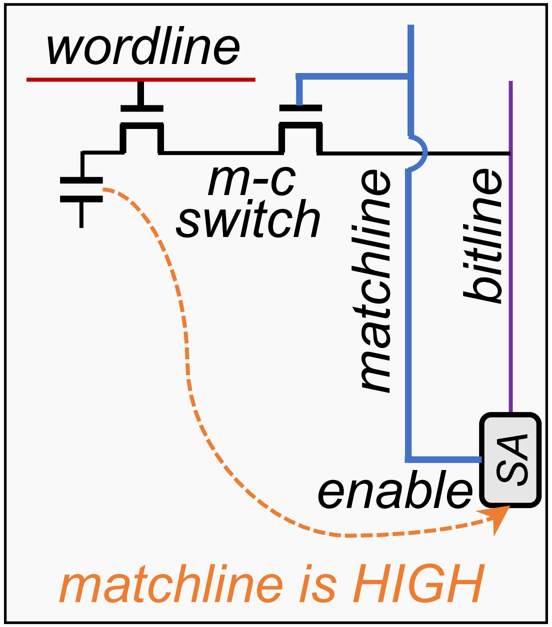
pLUTo Designs

- **Match Logic:** shared by the three pLUTo designs







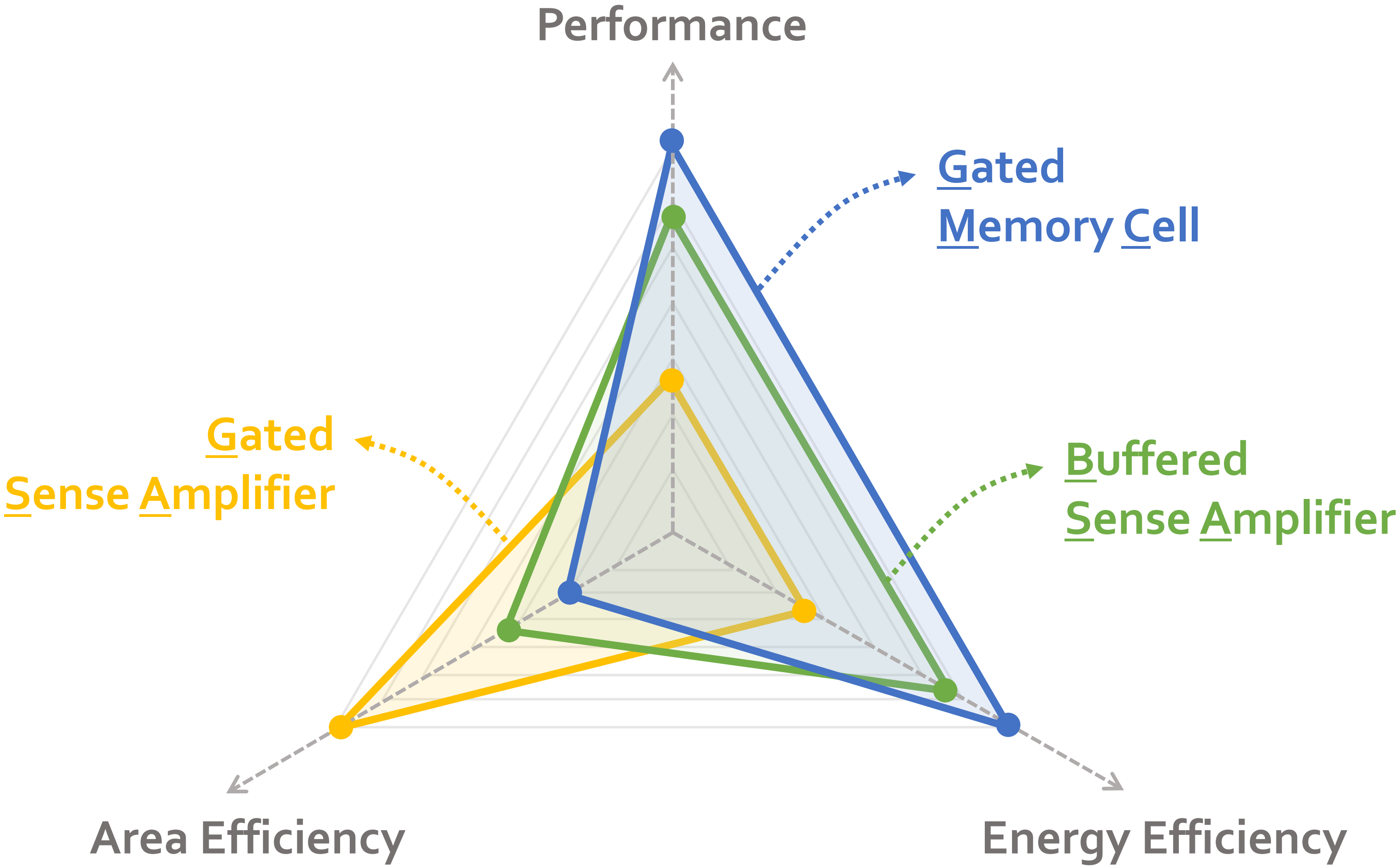


BSA
Buffered Sense Amplifier

GSA
Gated Sense Amplifier

GMC
Gated Memory Cell

pLUTo Designs: Tradeoff Space



pLUTo designs cover a *broad design space* and provide *different* performance, energy, and area efficiency

SAFARI

Contents

Introduction

pLUTo

Overview

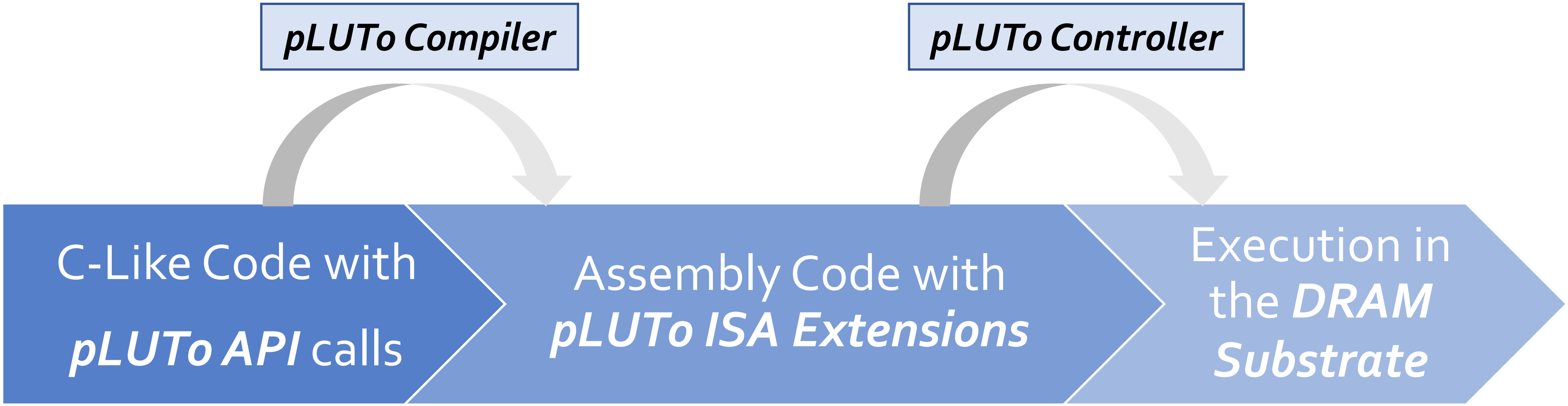
pLUTo Designs

System Integration

Evaluation

Conclusion

System Integration



pLUTo Compiler

pLUTo Controller

C-Like Code with *pLUTo API* calls

Assembly Code with ***pLUTo ISA Extensions***

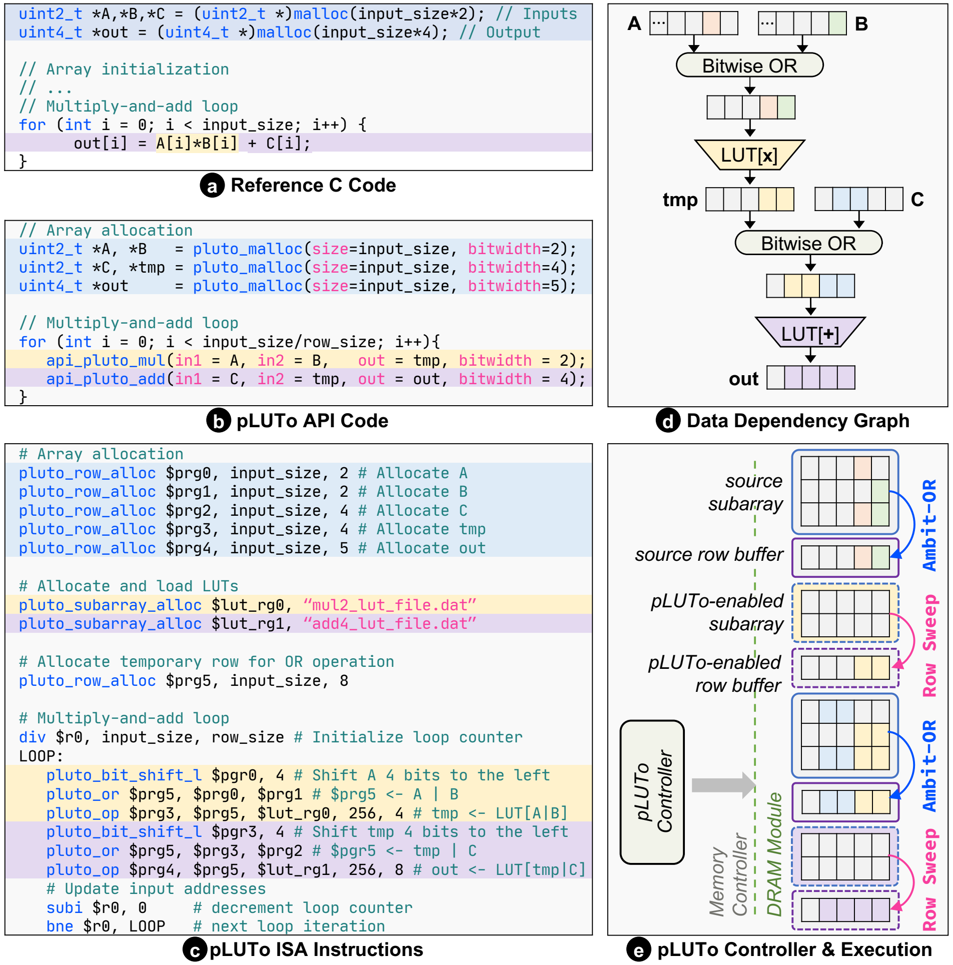
Execution in the ***DRAM Substrate***

```
api_pluto_mul
```

```
pluto_subarray_alloc
pluto_bit_shift_l
pluto_or
pluto_op
```

```
ACT
PRE
ACT
ACT
PRE
...
```

More in the Paper



```
uint2_t *A,*B,*C = (uint2_t *)malloc(input_size*2); // Inputs
uint4_t *out = (uint4_t *)malloc(input_size*4); // Output

// Array initialization
// ...
// Multiply-and-add loop
for (int i = 0; i < input_size; i++) {
      out[i] = A[i]*B[i] + C[i];
}
```

a Reference C Code

```
// Array allocation
uint2_t *A, *B   = pluto_malloc(size=input_size, bitwidth=2);
uint2_t *C, *tmp = pluto_malloc(size=input_size, bitwidth=4);
uint4_t *out     = pluto_malloc(size=input_size, bitwidth=5);

// Multiply-and-add loop
for (int i = 0; i < input_size/row_size; i++){
   api_pluto_mul(in1 = A, in2 = B,   out = tmp, bitwidth = 2);
   api_pluto_add(in1 = C, in2 = tmp, out = out, bitwidth = 4);
}
```

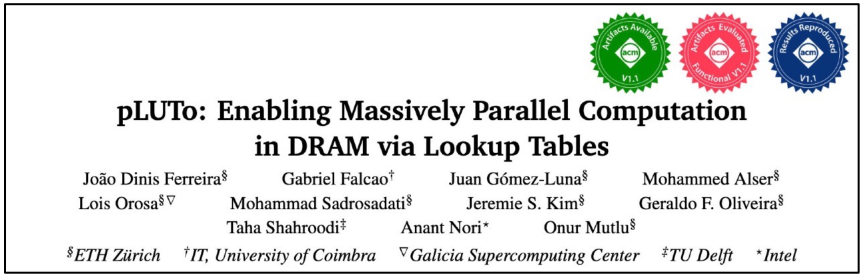
b pLUTo API Code

```
# Array allocation
pluto_row_alloc $prg0, input_size, 2 # Allocate A
pluto_row_alloc $prg1, input_size, 2 # Allocate B
pluto_row_alloc $prg2, input_size, 4 # Allocate C
pluto_row_alloc $prg3, input_size, 4 # Allocate tmp
pluto_row_alloc $prg4, input_size, 5 # Allocate out

# Allocate and load LUTs
pluto_subarray_alloc $lut_rg0, “mul2_lut_file.dat”
pluto_subarray_alloc $lut_rg1, “add4_lut_file.dat”

# Allocate temporary row for OR operation
pluto_row_alloc $prg5, input_size, 8

# Multiply-and-add loop
div $r0, input_size, row_size # Initialize loop counter
LOOP:
   pluto_bit_shift_l $pgr0, 4 # Shift A 4 bits to the left
   pluto_or $prg5, $prg0, $prg1 # $prg5 <- A | B
   pluto_op $prg3, $prg5, $lut_rg0, 256, 4 # tmp <- LUT[A|B]
   pluto_bit_shift_l $pgr3, 4 # Shift tmp 4 bits to the left
   pluto_or $prg5, $prg3, $prg2 # $pgr5 <- tmp | C
   pluto_op $prg4, $prg5, $lut_rg1, 256, 8 # out <- LUT[tmp|C]
   # Update input addresses
   subi $r0, 0       # decrement loop counter
   bne $r0, LOOP    # next loop iteration
```

c pLUTo ISA Instructions

d Data Dependency Graph

e pLUTo Controller & Execution



pLUTo: Enabling Massively Parallel Computation in DRAM via Lookup Tables

João Dinis Ferreira[§] Gabriel Falcao[†] Juan Gómez-Luna[§] Mohammed Alser[§]
Lois Orosa[§∇] Mohammad Sadrosadati[§] Jeremie S. Kim[§] Geraldo F. Oliveira[§]
Taha Shahroodi[‡] Anant Nori[*] Onur Mutlu[§]

[§]ETH Zürich [†]IT, University of Coimbra [∇]Galicia Supercomputing Center [‡]TU Delft [*]Intel

https://arxiv.org/pdf/2104.07699.pdf

Contents

Introduction

pLUTo

Overview

pLUTo Designs

System Integration

Evaluation

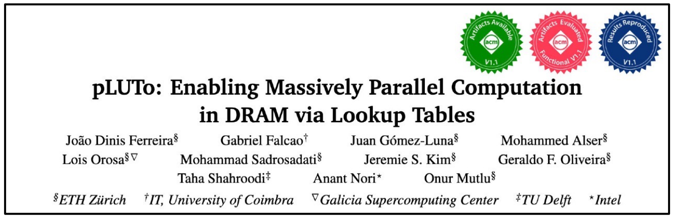
Conclusion

Methodology: Experimental Setup

- **Baselines**
  - **CPU** (Intel® Xeon Gold 5118)
  - **GPU** (NVIDIA® GeForce RTX 3080 Ti)
  - **Processing-near-Memory (PnM)**
- **pLUTo**
  - In-house simulator, open-sourced
  - https://github.com/CMU-SAFARI/pLUTo
- **We evaluate:**
  - Performance
  - Energy consumption
  - Area overhead
  - Circuit-level reliability and correctness
  - …

Methodology: Experimental Setup

- **Baselines**
  - **CPU** (Intel® Xeon Gold 5118)
  - **GPU** (NVIDIA® GeForce RTX 3080 Ti)
  - **Processing-near-Memory (PnM)**
- **pLUTo**
  - In-house simulator, open-sourced
  - https://github.com/CMU-SAFARI/pLUTo
- **We evaluate:**
  - **Performance**
  - **Energy consumption**
  - Area overhead
  - Circuit-level reliability and correctness
  - …



pLUTo: Enabling Massively Parallel Computation in DRAM via Lookup Tables

João Dinis Ferreira[§] Gabriel Falcao[†] Juan Gómez-Luna[§] Mohammed Alser[§]
Lois Orosa[§∇] Mohammad Sadrosadati[§] Jeremie S. Kim[§] Geraldo F. Oliveira[§]
Taha Shahroodi[‡] Anant Nori[*] Onur Mutlu[§]

[§]ETH Zürich [†]IT, University of Coimbra [∇]Galicia Supercomputing Center [‡]TU Delft [*]Intel

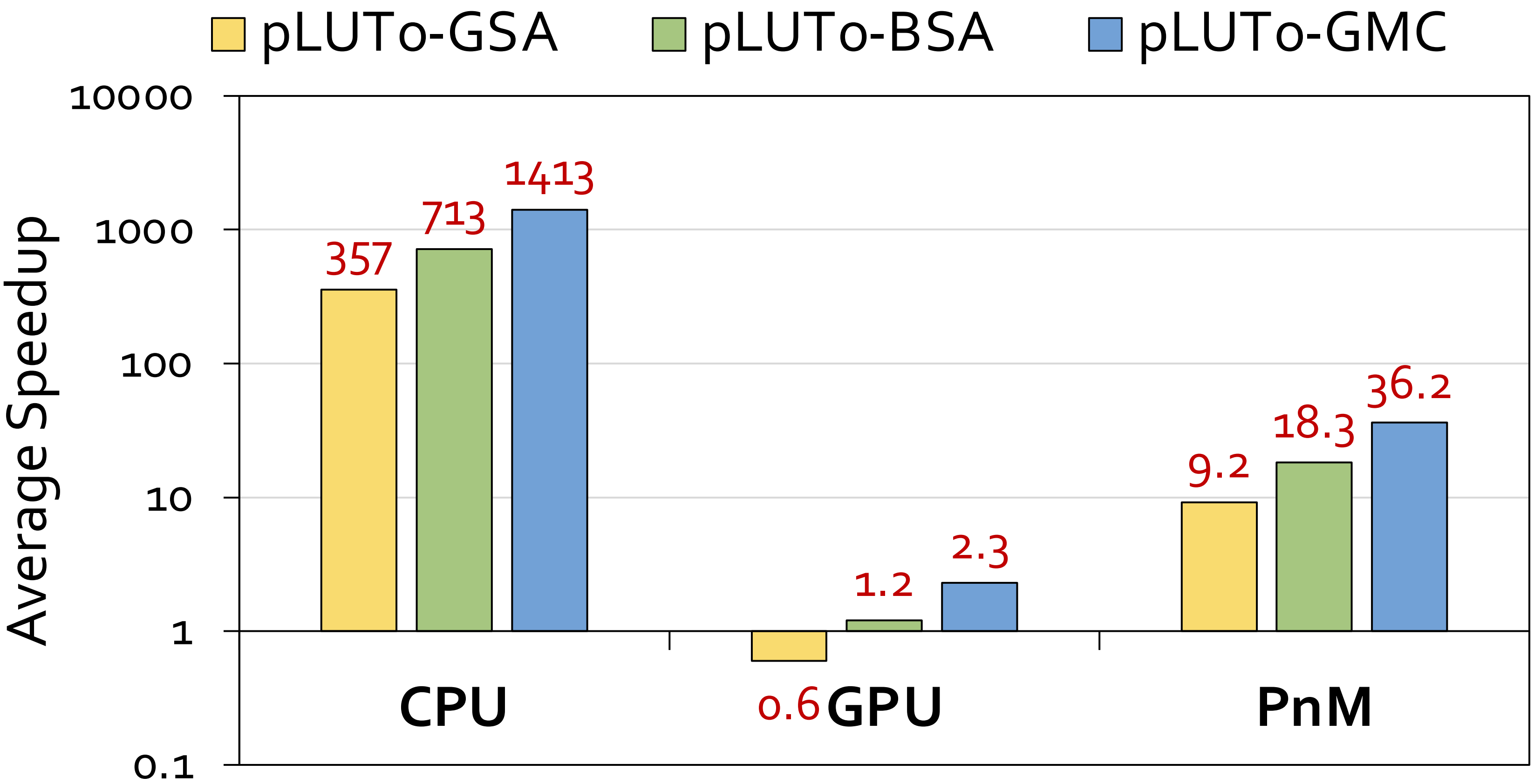
Methodology: Workloads

- **7 real-world workloads (not well supported by prior work)**
  - CRC-8/16/32
  - Salsa20
  - VMPC
  - Image Binarization
  - Color Grading

- **4 synthetic workloads (supported by prior work)**
  - Vector Addition
  - Vector Point-Wise Multiplication
  - Row-Level Bitwise Logic Operations
  - Bit Counting

Performance

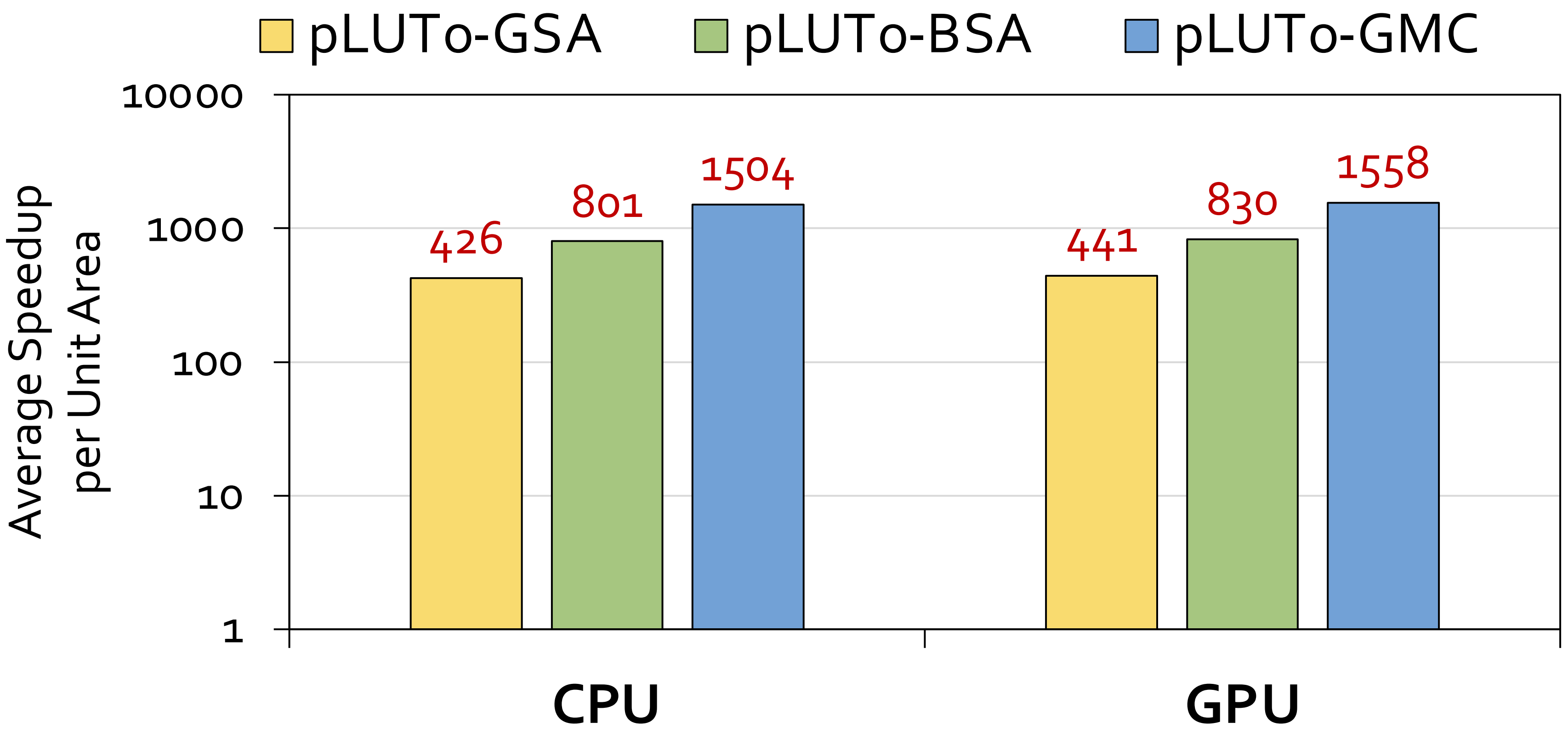
Average speedup across 7 real-world workloads



pLUTo *significantly outperforms* CPU, GPU and PnM baselines

Performance (normalized to area)

Average speedup normalized to area across 7 real-world workloads

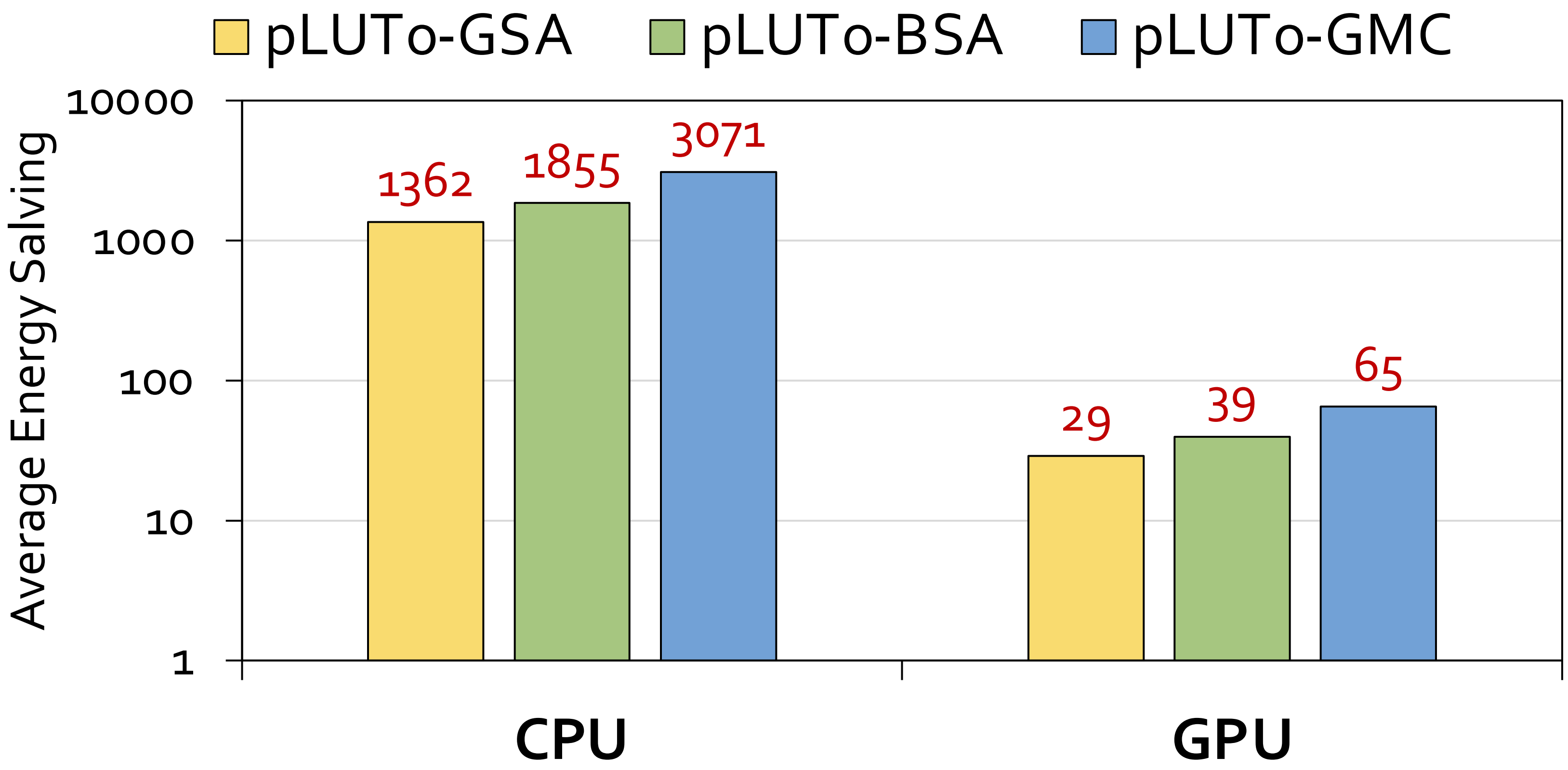


pLUTo provides *substantially higher* performance per unit area than *both* the CPU and the GPU

SAFARI

Energy Consumption

Average energy consumption across 7 real-world workloads



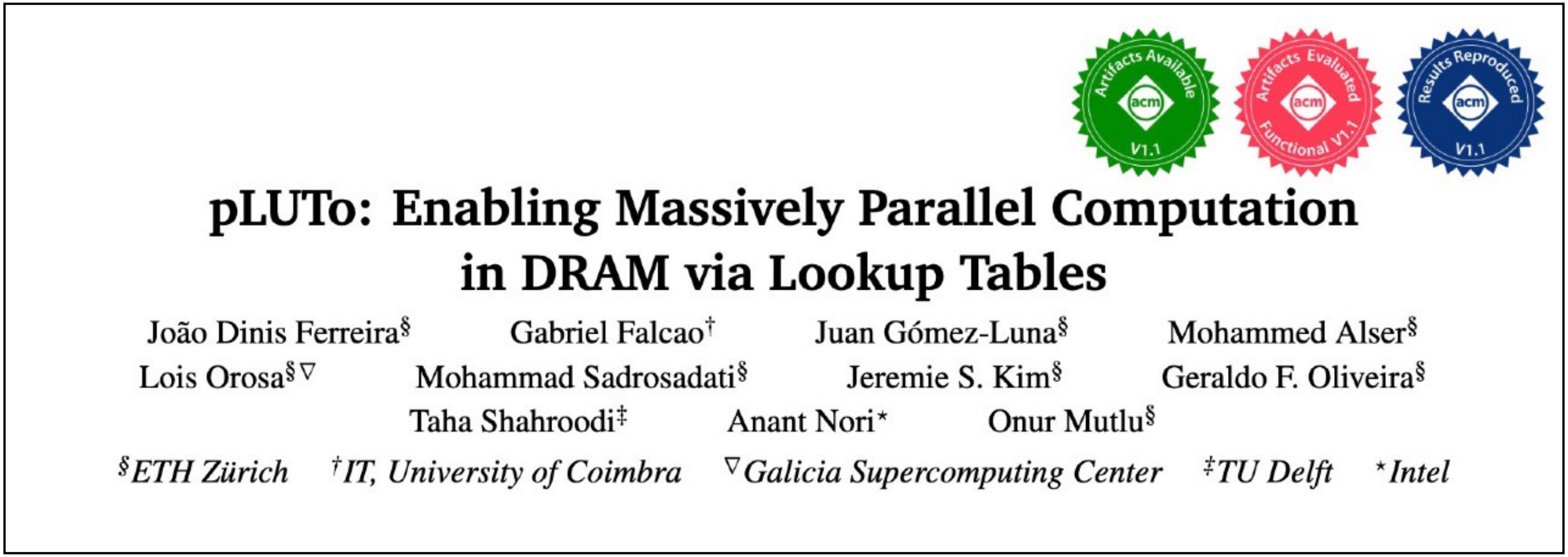
pLUTo *significantly reduces energy consumption* compared to processor-centric architectures for various workloads

SAFARI

More Results in the Paper

- Comparison with FPGA
- Area Overhead Analysis
- Circuit-Level Reliability & Correctness

- Subarray-Level Parallelism
- LUT Loading Overhead
- Range of Supported Operations



pLUTo: Enabling Massively Parallel Computation in DRAM via Lookup Tables

João Dinis Ferreira[§] Gabriel Falcao[†] Juan Gómez-Luna[§] Mohammed Alser[§]
Lois Orosa[§▽] Mohammad Sadrosadati[§] Jeremie S. Kim[§] Geraldo F. Oliveira[§]
Taha Shahroodi[‡] Anant Nori[*] Onur Mutlu[§]

[§]*ETH Zürich* [†]*IT, University of Coimbra* [▽]*Galicia Supercomputing Center* [‡]*TU Delft* [*]*Intel*

Contents

Introduction

pLUTo

Overview

pLUTo Designs

System Integration

Evaluation

Conclusion

pLUTo Summary

The ***pLUTo Lookup Table Query*** introduces support for **the execution of arbitrarily complex operations** in DRAM

pLUTo Summary

The *pLUTo Lookup Table Query* introduces support for **the execution of arbitrarily complex operations** in DRAM

Buffered Sense Amplifier, Gated Sense Amplifier & Gated Memory Cell target different **performance/energy/area tradeoffs**

pLUTo Summary

The ***pLUTo Lookup Table Query*** introduces support for **the execution of arbitrarily complex operations** in DRAM

Buffered Sense Amplifier, Gated Sense Amplifier & Gated Memory Cell target different **performance/energy/area tradeoffs**

pLUTo API, pLUTo Compiler & pLUTo Controller **facilitate programmer adoption** of **pLUTo**

pLUTo Summary

The *pLUTo Lookup Table Query* introduces support for **the execution of arbitrarily complex operations** in DRAM

Buffered Sense Amplifier, Gated Sense Amplifier & Gated Memory Cell target different **performance/energy/area tradeoffs**

pLUTo API, pLUTo Compiler & pLUTo Controller **facilitate programmer adoption** of **pLUTo**

pLUTo greatly outperforms the CPU/GPU/PnM baselines in **performance and energy** while incurring **small area overheads**

pLUTo

Enabling Massively Parallel Computation in DRAM via Lookup Tables

João Dinis Ferreira   Gabriel Falcao   Juan Gómez-Luna   Mohammed Alser
Lois Orosa   Mohammad Sadrosadati   Jeremie S. Kim   Geraldo F. Oliveira
Taha Shahroodi   Anant Nori   Onur Mutlu

SAFARI   ETH Zürich   TUDelft
instituto de telecomunicações   CESGA GALICIA SUPERCOMPUTING CENTER   intel

pLUTo Summary

The ***pLUTo Lookup Table Query*** introduces support for **the execution of arbitrarily complex operations** in DRAM

Buffered Sense Amplifier, Gated Sense Amplifier & Gated Memory Cell target different **performance/energy/area tradeoffs**

pLUTo API, pLUTo Compiler & pLUTo Controller
facilitate programmer adoption of **pLUTo**

pLUTo greatly outperforms the CPU/GPU/PnM baselines in **performance and energy** while incurring **small area overheads**

arXiv

GitHub

Artifacts Available V1.1 · Artifacts Evaluated Functional V1.1 · Results Reproduced V1.1

pLUTo

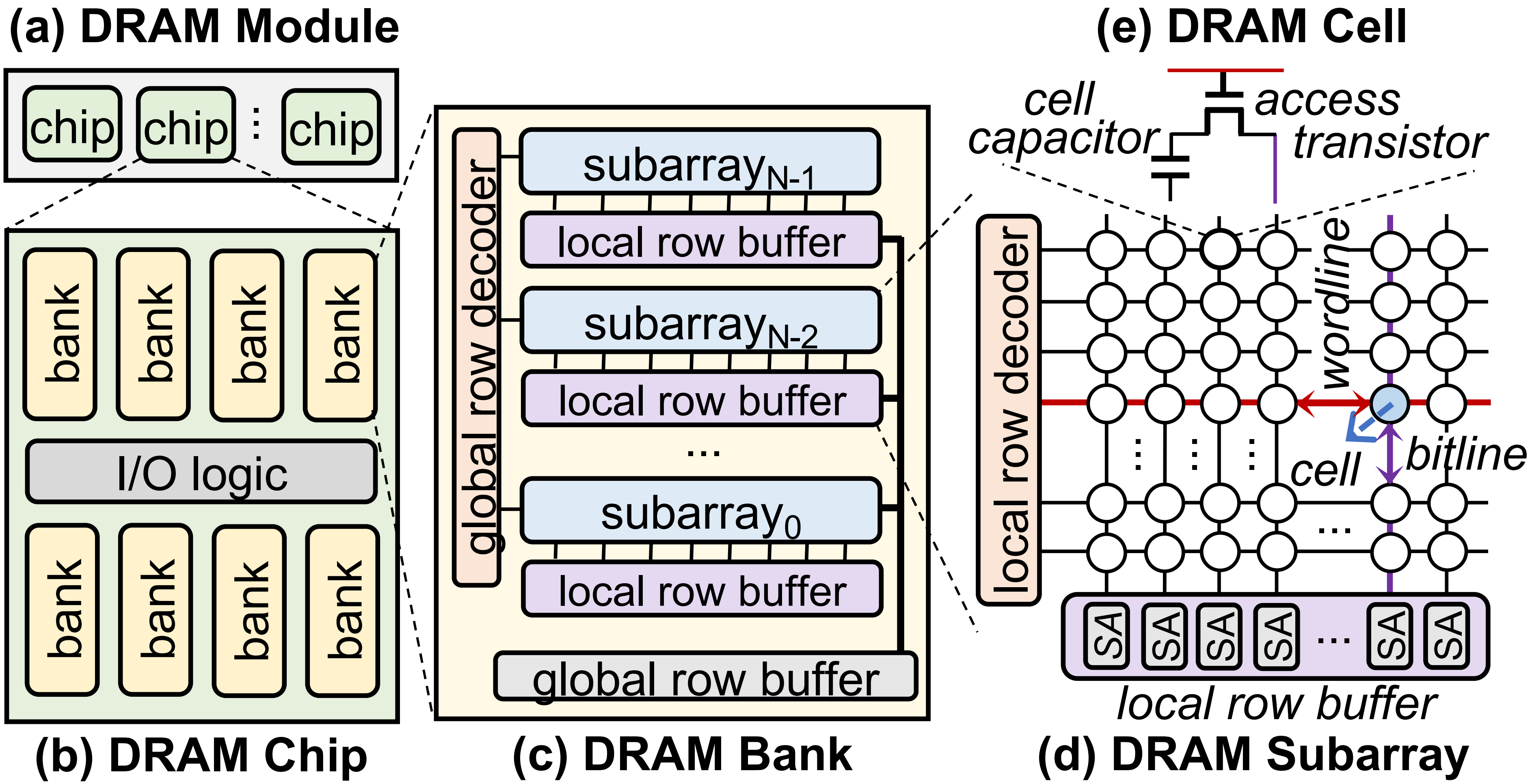
Enabling Massively Parallel Computation in DRAM via Lookup Tables

João Dinis Ferreira Gabriel Falcao Juan Gómez-Luna Mohammed Alser
Lois Orosa Mohammad Sadrosadati Jeremie S. Kim Geraldo F. Oliveira
Taha Shahroodi Anant Nori Onur Mutlu

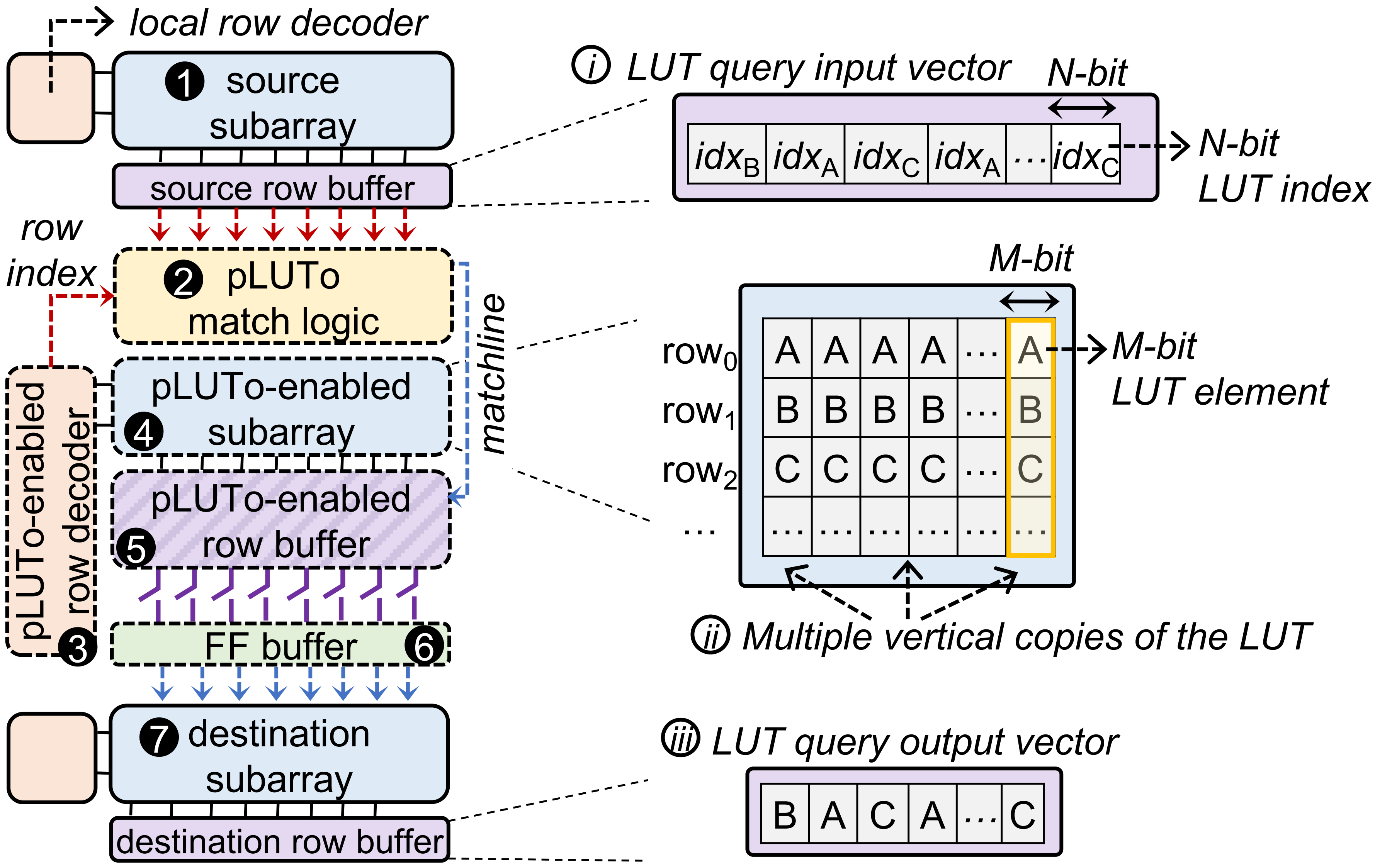
SAFARI ETH Zürich TU Delft
instituto de telecomunicações CESGA GALICIA SUPERCOMPUTING CENTER intel

Backup Slides

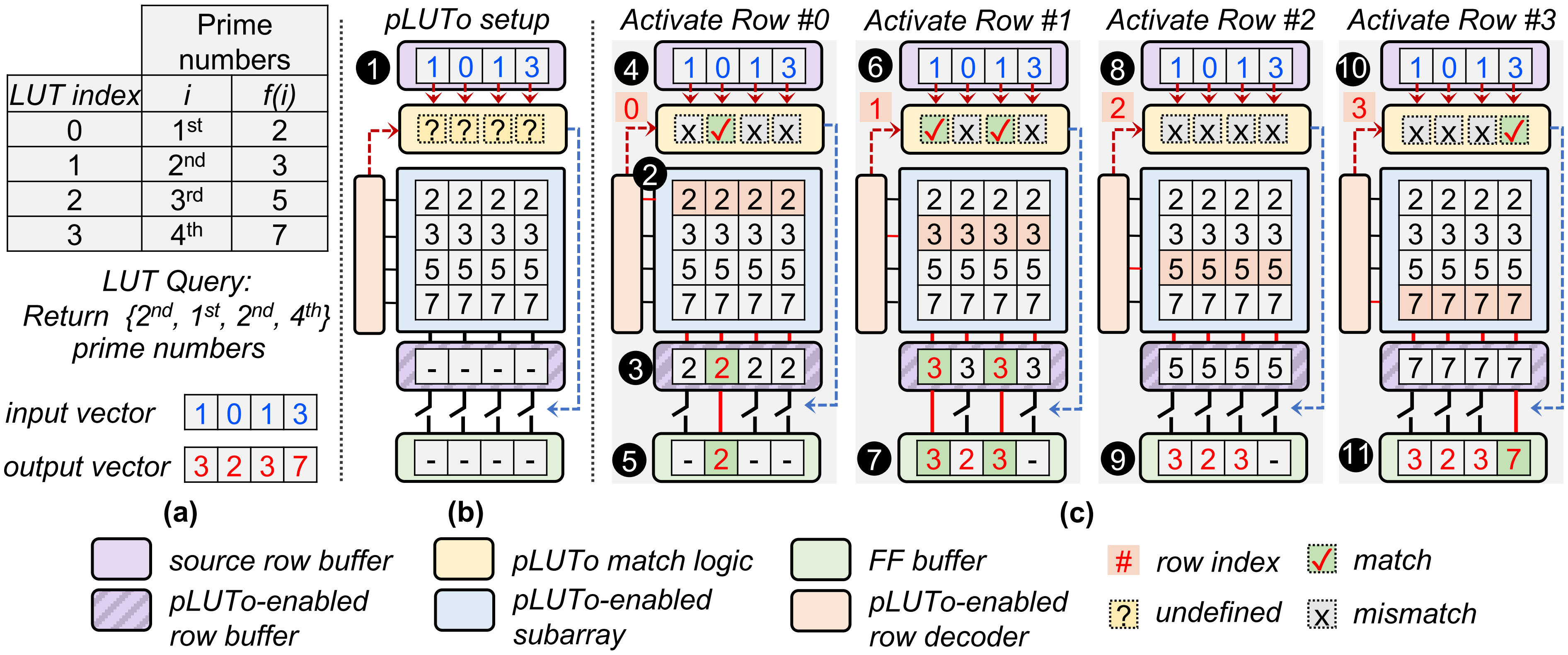
DRAM Organization



pLUTo Components



pLUTo Example Operation



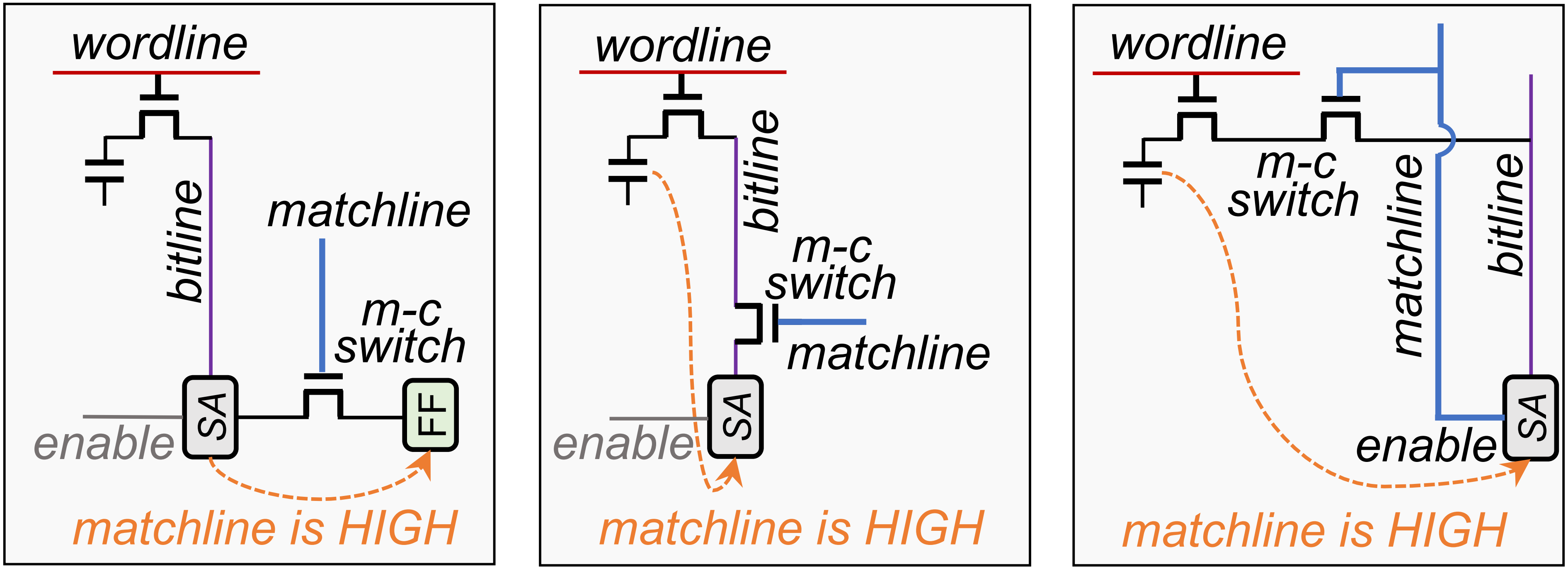
	Prime numbers	
LUT index	*i*	*f(i)*
0	1st	2
1	2nd	3
2	3rd	5
3	4th	7

LUT Query:
Return {2nd, 1st, 2nd, 4th} prime numbers

input vector 1 0 1 3

output vector 3 2 3 7

pLUTo Cell Designs



(a) pLUTo-BSA **(b) pLUTo-GSA** **(c) pLUTo-GMC**

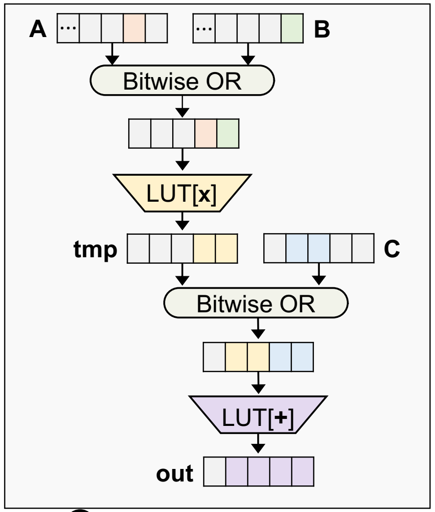
System Integration: End-to-End

```
uint2_t *A,*B,*C = (uint2_t *)malloc(input_size*2); // Inputs
uint4_t *out = (uint4_t *)malloc(input_size*4); // Output

// Array initialization
// ...
// Multiply-and-add loop
for (int i = 0; i < input_size; i++) {
      out[i] = A[i]*B[i] + C[i];
}
```

a Reference C Code

```
// Array allocation
uint2_t *A, *B   = pluto_malloc(size=input_size, bitwidth=2);
uint2_t *C, *tmp = pluto_malloc(size=input_size, bitwidth=4);
uint4_t *out     = pluto_malloc(size=input_size, bitwidth=5);

// Multiply-and-add loop
for (int i = 0; i < input_size/row_size; i++){
   api_pluto_mul(in1 = A, in2 = B,   out = tmp, bitwidth = 2);
   api_pluto_add(in1 = C, in2 = tmp, out = out, bitwidth = 4);
}
```

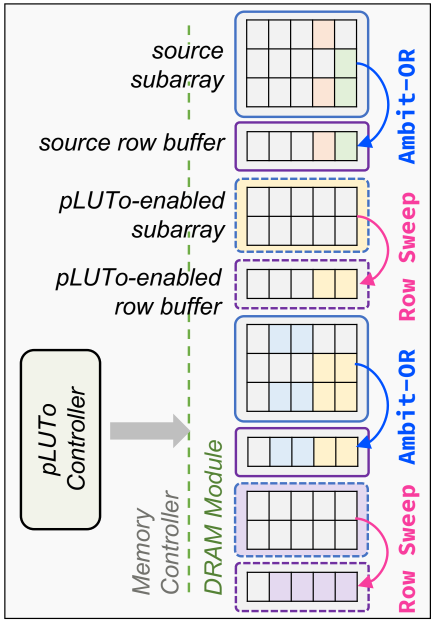
b pLUTo API Code



d Data Dependency Graph

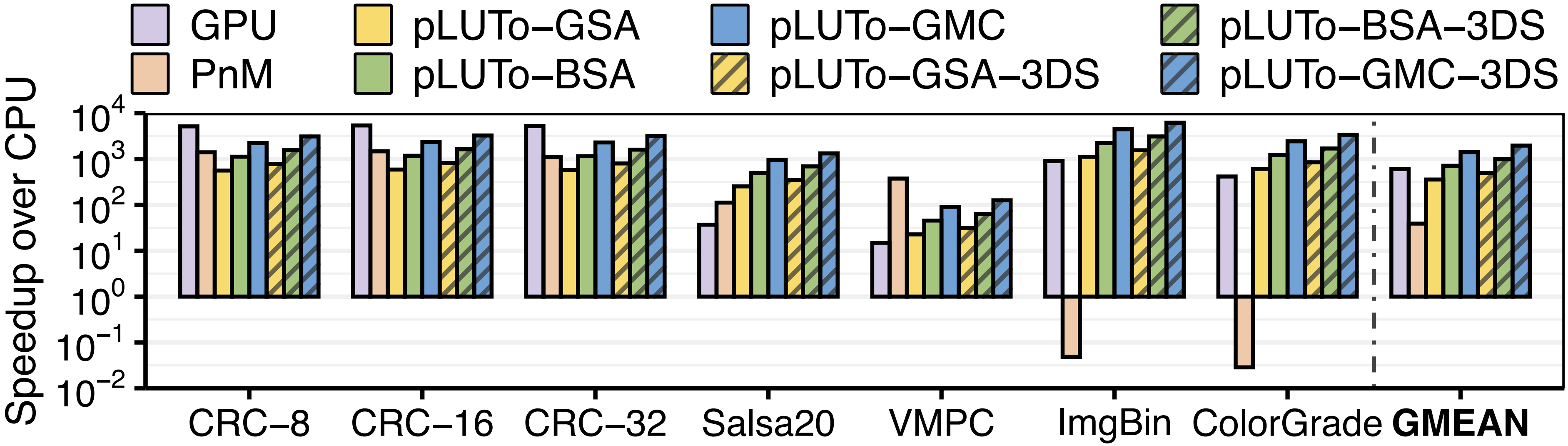
```
# Array allocation
pluto_row_alloc $prg0, input_size, 2 # Allocate A
pluto_row_alloc $prg1, input_size, 2 # Allocate B
pluto_row_alloc $prg2, input_size, 4 # Allocate C
pluto_row_alloc $prg3, input_size, 4 # Allocate tmp
pluto_row_alloc $prg4, input_size, 5 # Allocate out

# Allocate and load LUTs
pluto_subarray_alloc $lut_rg0, “mul2_lut_file.dat”
pluto_subarray_alloc $lut_rg1, “add4_lut_file.dat”

# Allocate temporary row for OR operation
pluto_row_alloc $prg5, input_size, 8

# Multiply-and-add loop
div $r0, input_size, row_size # Initialize loop counter
LOOP:
   pluto_bit_shift_l $pgr0, 4 # Shift A 4 bits to the left
   pluto_or $prg5, $prg0, $prg1 # $prg5 <- A | B
   pluto_op $prg3, $prg5, $lut_rg0, 256, 4 # tmp <- LUT[A|B]
   pluto_bit_shift_l $pgr3, 4 # Shift tmp 4 bits to the left
   pluto_or $prg5, $prg3, $prg2 # $pgr5 <- tmp | C
   pluto_op $prg4, $prg5, $lut_rg1, 256, 8 # out <- LUT[tmp|C]
   # Update input addresses
   subi $r0, 0      # decrement loop counter
   bne $r0, LOOP    # next loop iteration
```

c pLUTo ISA Instructions



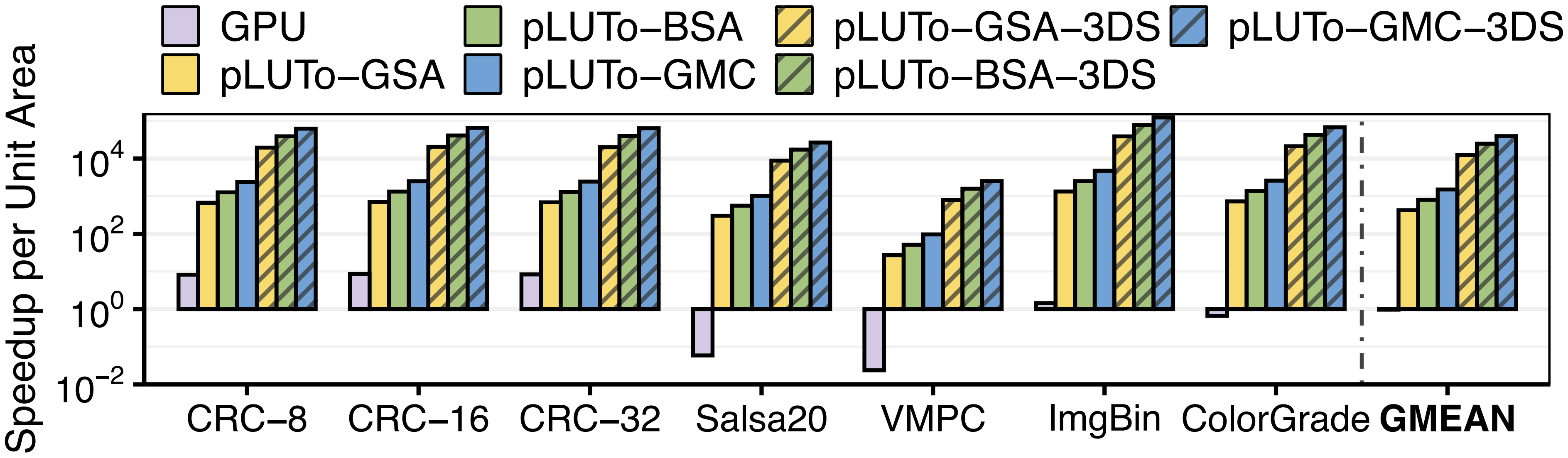
e pLUTo Controller & Execution

Speedup

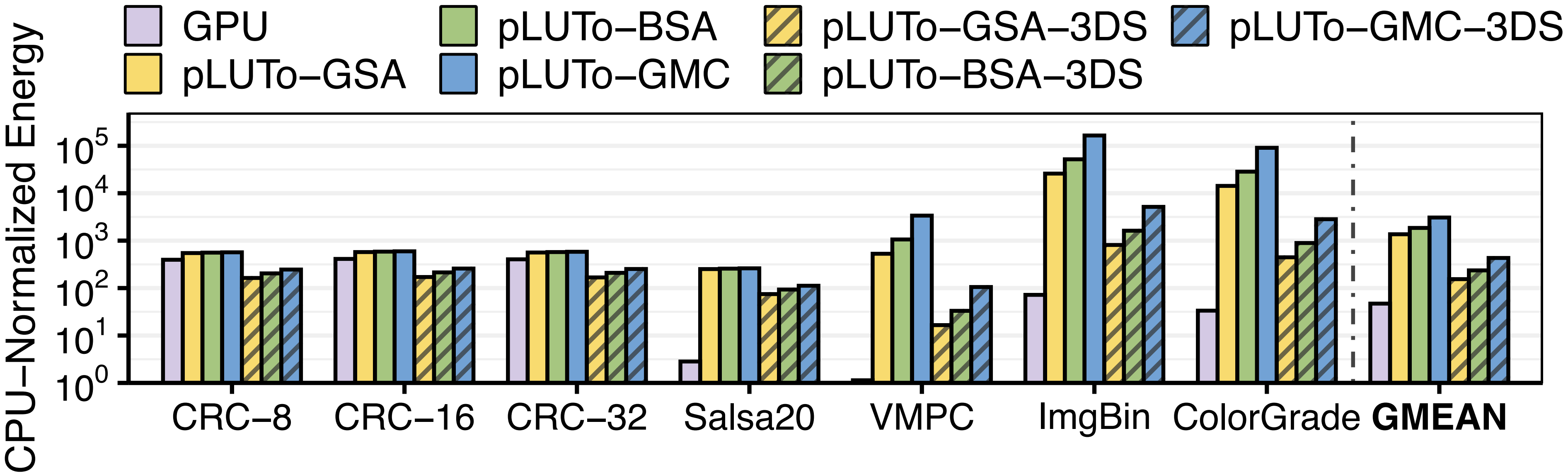


SAFARI

Speedup Normalized to Unit Area

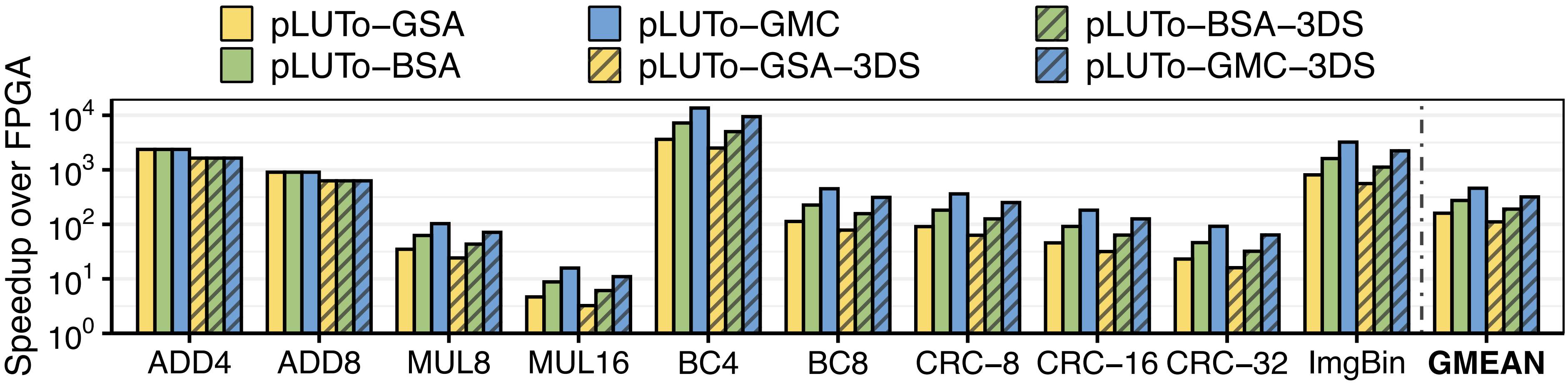


Energy, Normalized

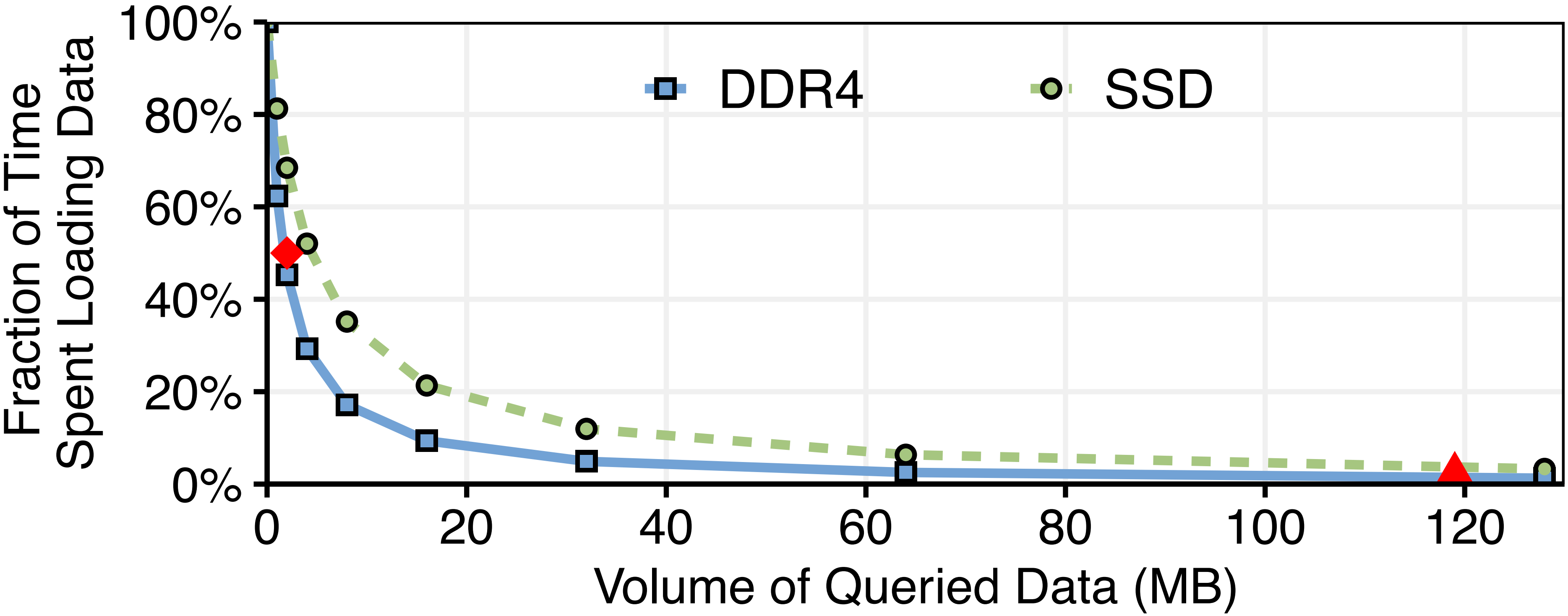


SAFARI

Speedup Over FPGA



Fraction of Time Spent Loading LUTs (DDR4, SSD) vs. Volume of Queried Data (MB)



Comparison of Operations Supported by pLUTo vs. Prior Work

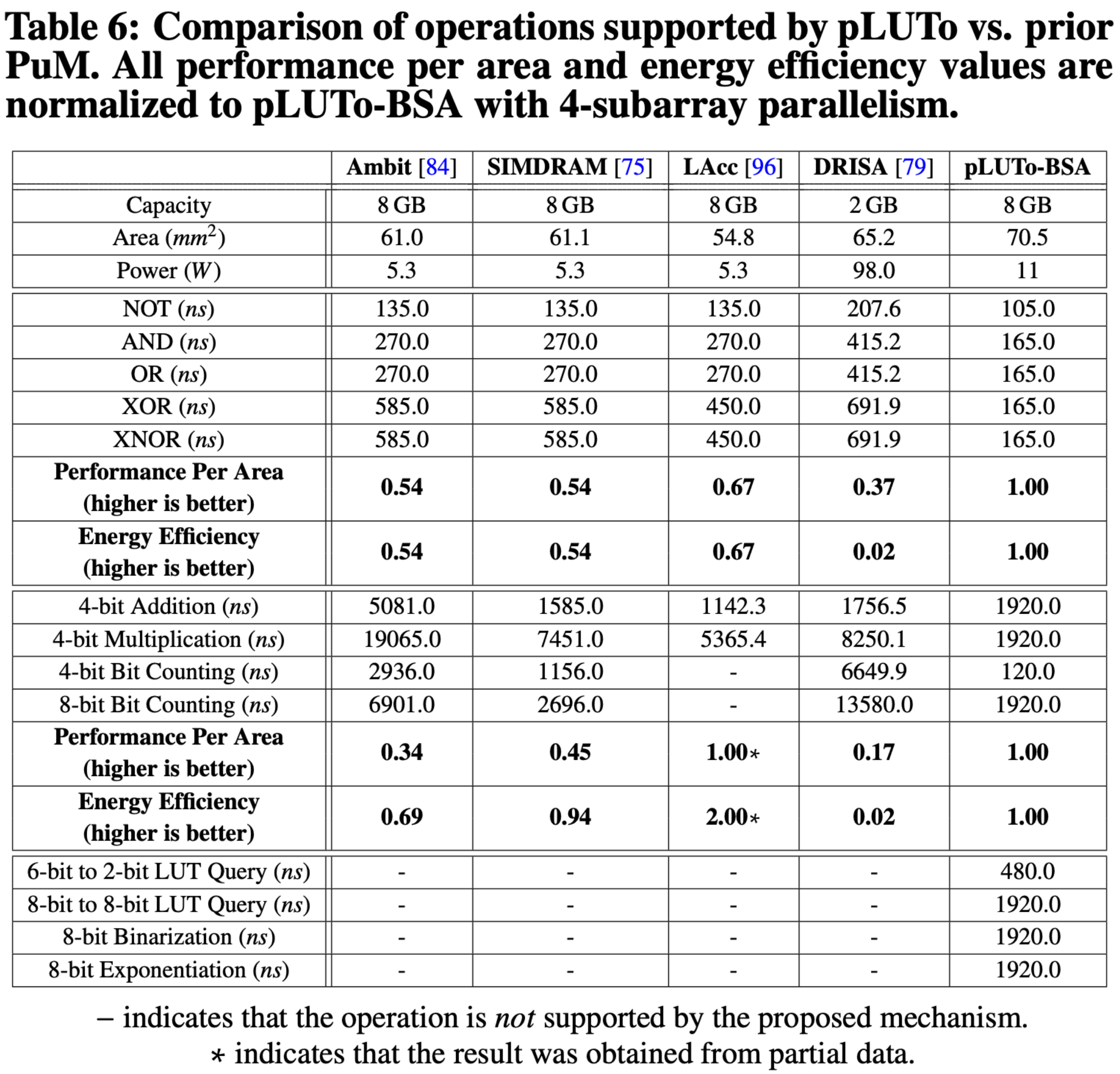


Table 6: Comparison of operations supported by pLUTo vs. prior PuM. All performance per area and energy efficiency values are normalized to pLUTo-BSA with 4-subarray parallelism.

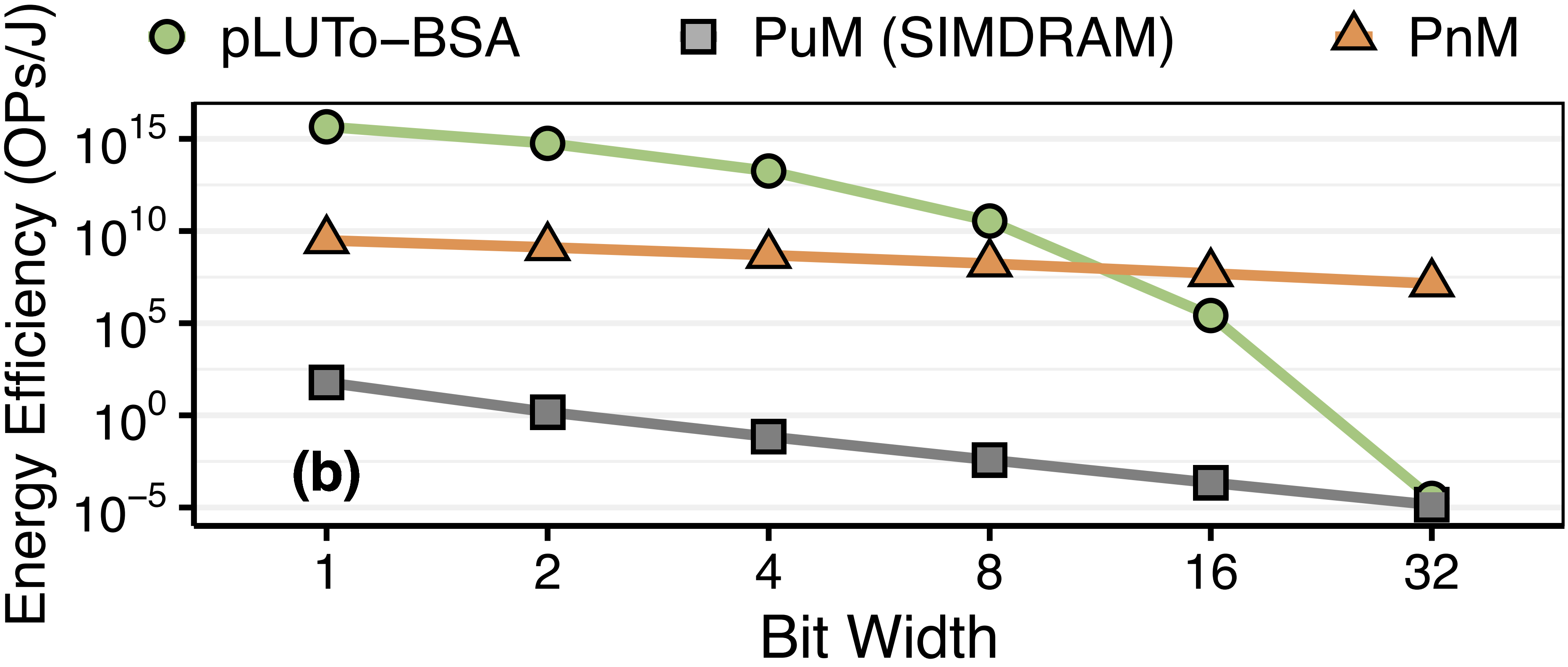
	Ambit [84]	SIMDRAM [75]	LAcc [96]	DRISA [79]	pLUTo-BSA
Capacity	8 GB	8 GB	8 GB	2 GB	8 GB
Area (mm^2)	61.0	61.1	54.8	65.2	70.5
Power (W)	5.3	5.3	5.3	98.0	11
NOT (ns)	135.0	135.0	135.0	207.6	105.0
AND (ns)	270.0	270.0	270.0	415.2	165.0
OR (ns)	270.0	270.0	270.0	415.2	165.0
XOR (ns)	585.0	585.0	450.0	691.9	165.0
XNOR (ns)	585.0	585.0	450.0	691.9	165.0
Performance Per Area (higher is better)	**0.54**	**0.54**	**0.67**	**0.37**	**1.00**
Energy Efficiency (higher is better)	**0.54**	**0.54**	**0.67**	**0.02**	**1.00**
4-bit Addition (ns)	5081.0	1585.0	1142.3	1756.5	1920.0
4-bit Multiplication (ns)	19065.0	7451.0	5365.4	8250.1	1920.0
4-bit Bit Counting (ns)	2936.0	1156.0	-	6649.9	120.0
8-bit Bit Counting (ns)	6901.0	2696.0	-	13580.0	1920.0
Performance Per Area (higher is better)	**0.34**	**0.45**	**1.00**$\ast$	**0.17**	**1.00**
Energy Efficiency (higher is better)	**0.69**	**0.94**	**2.00**$\ast$	**0.02**	**1.00**
6-bit to 2-bit LUT Query (ns)	-	-	-	-	480.0
8-bit to 8-bit LUT Query (ns)	-	-	-	-	1920.0
8-bit Binarization (ns)	-	-	-	-	1920.0
8-bit Exponentiation (ns)	-	-	-	-	1920.0

– indicates that the operation is *not* supported by the proposed mechanism.

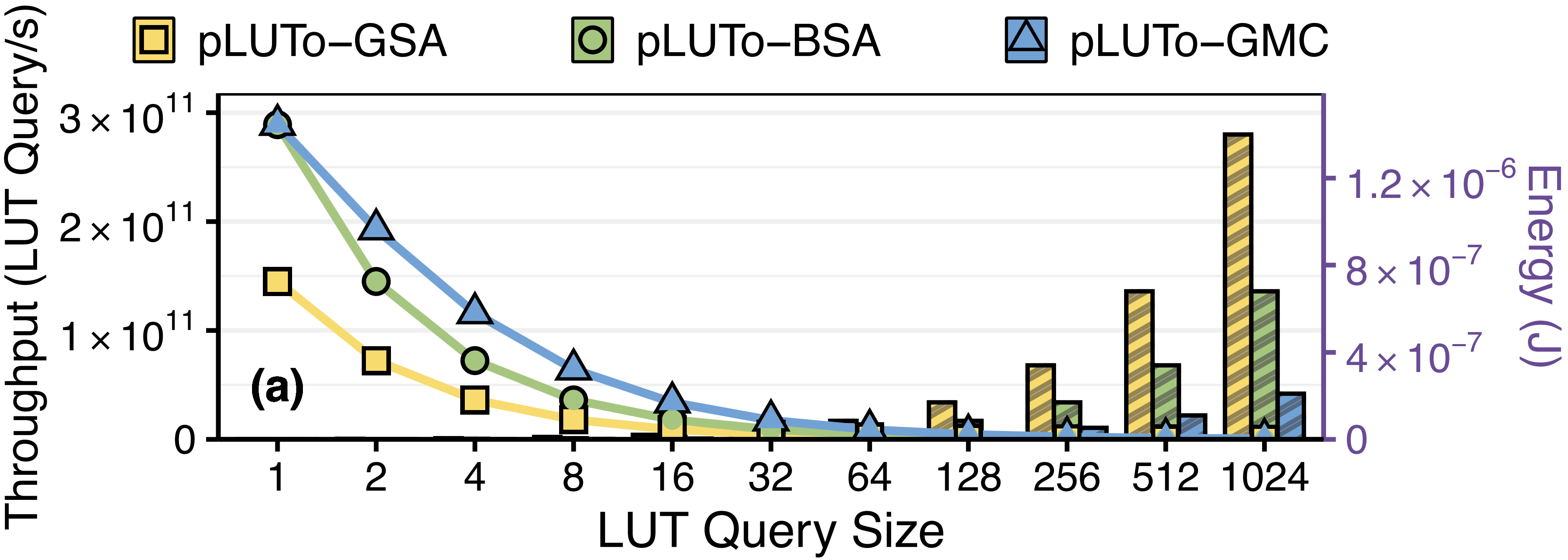
$\ast$ indicates that the result was obtained from partial data.

SAFARI

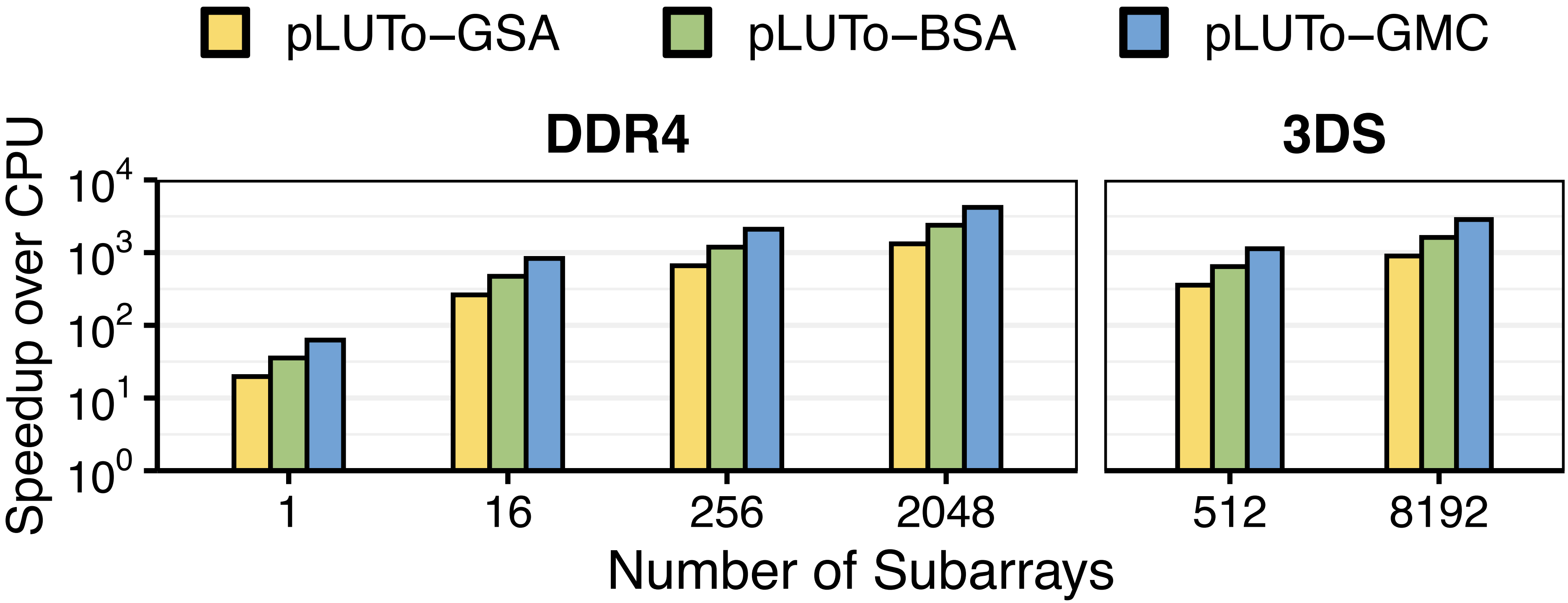
Energy Efficiency (pLUTo vs. PuM vs. PnM) vs. Bit Width, one operation



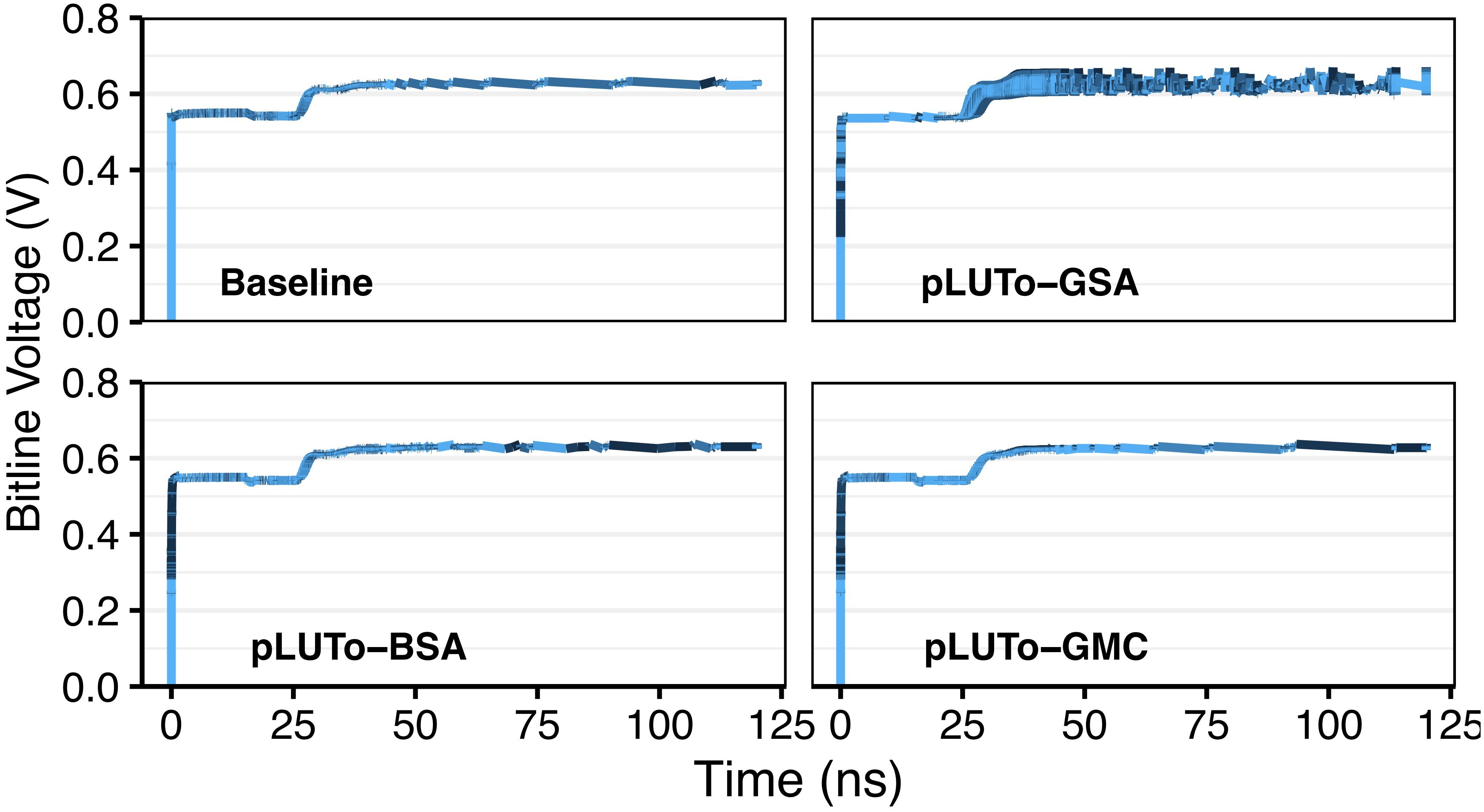
Throughput & Energy Consumption of pLUTo vs. LUT Query Size



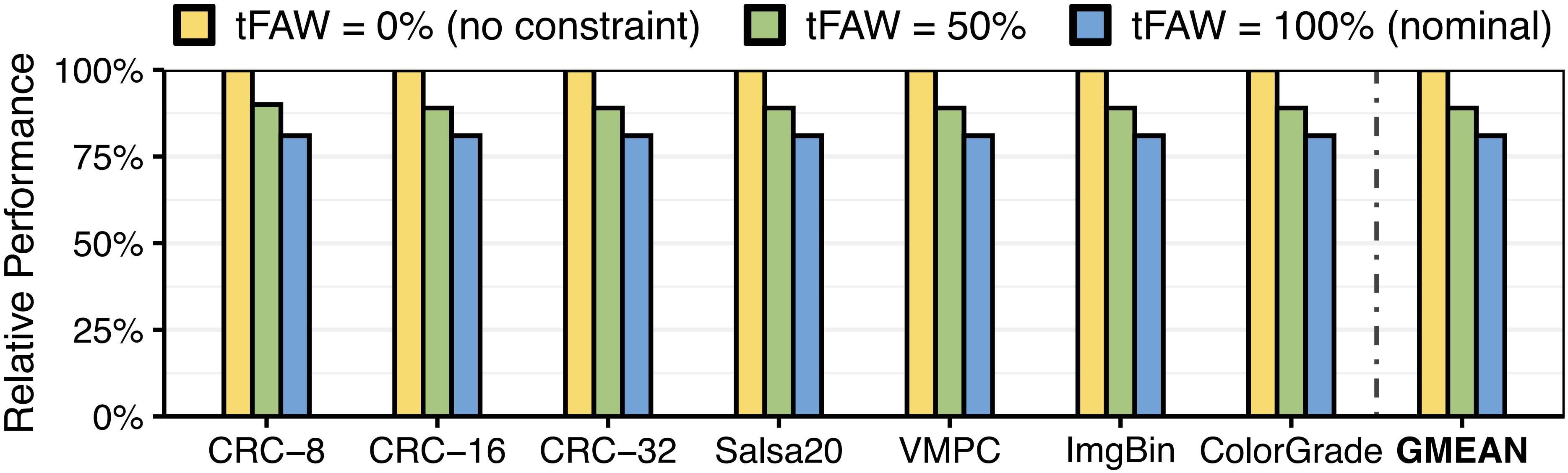
Speedup vs. Subarray-Level Parallelism



Circuit-Level Reliability and Correctness



Impact of tFAW


tFAW = 0% (no constraint)
tFAW = 50%
tFAW = 100% (nominal)
Relative Performance
CRC−8
CRC−16
CRC−32
Salsa20
VMPC
ImgBin
ColorGrade
GMEAN