Georgios Zacharopoulos Giovanni Ansaloni Laura Pozzi

DATA REUSE ANALYSIS FOR AUTOMATED SYNTHESIS OF CUSTOM INSTRUCTIONS IN SLIDING WINDOW APPLICATIONS

EuroLLVM 2017

Mar 28, 2017
Saarbrücken,
Germany

Università della Svizzera italiana (USI Lugano),
Faculty of Informatics

MOORE'S LAW – DENNARD SCALING

Expectation

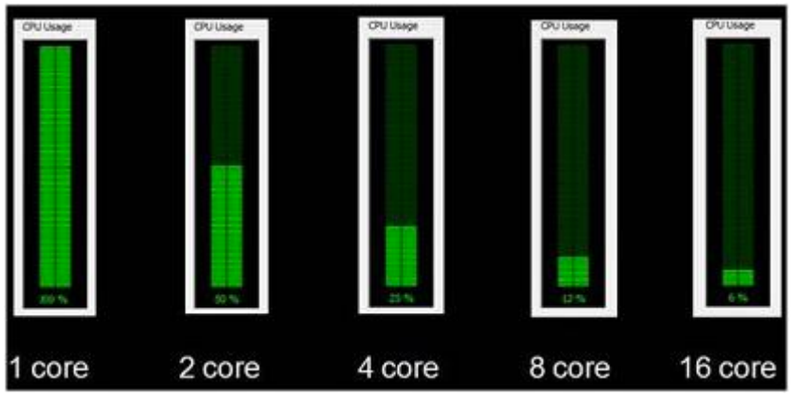
- Performance should keep increasing

Reality

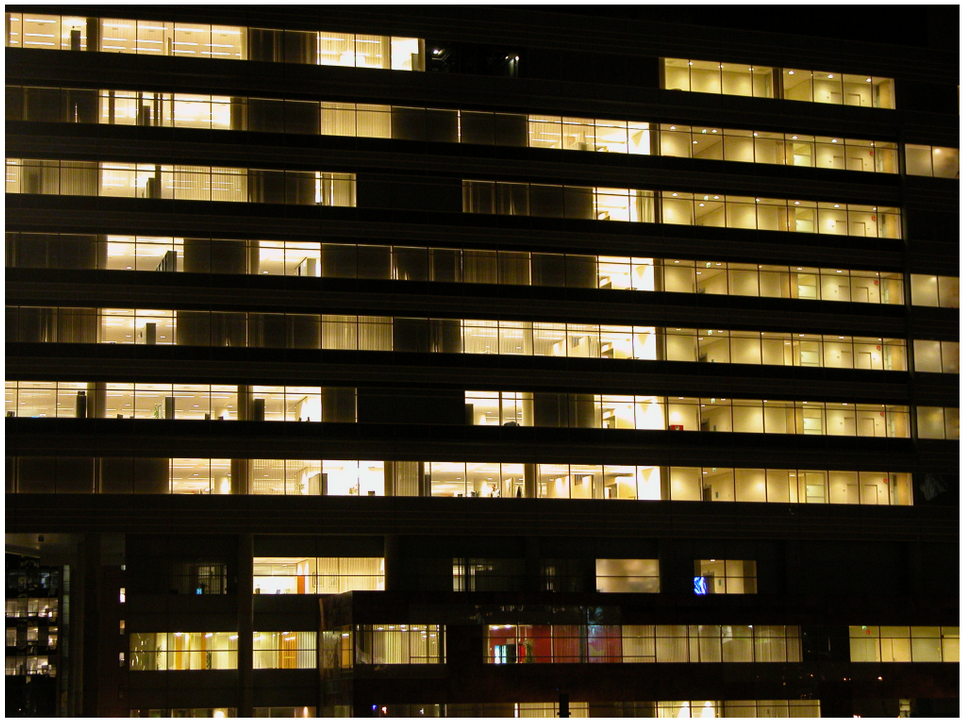
- Performance is NOT increasing anymore

BREAKDOWN OF DENNARD SCALING

- Single core –> Multiple cores

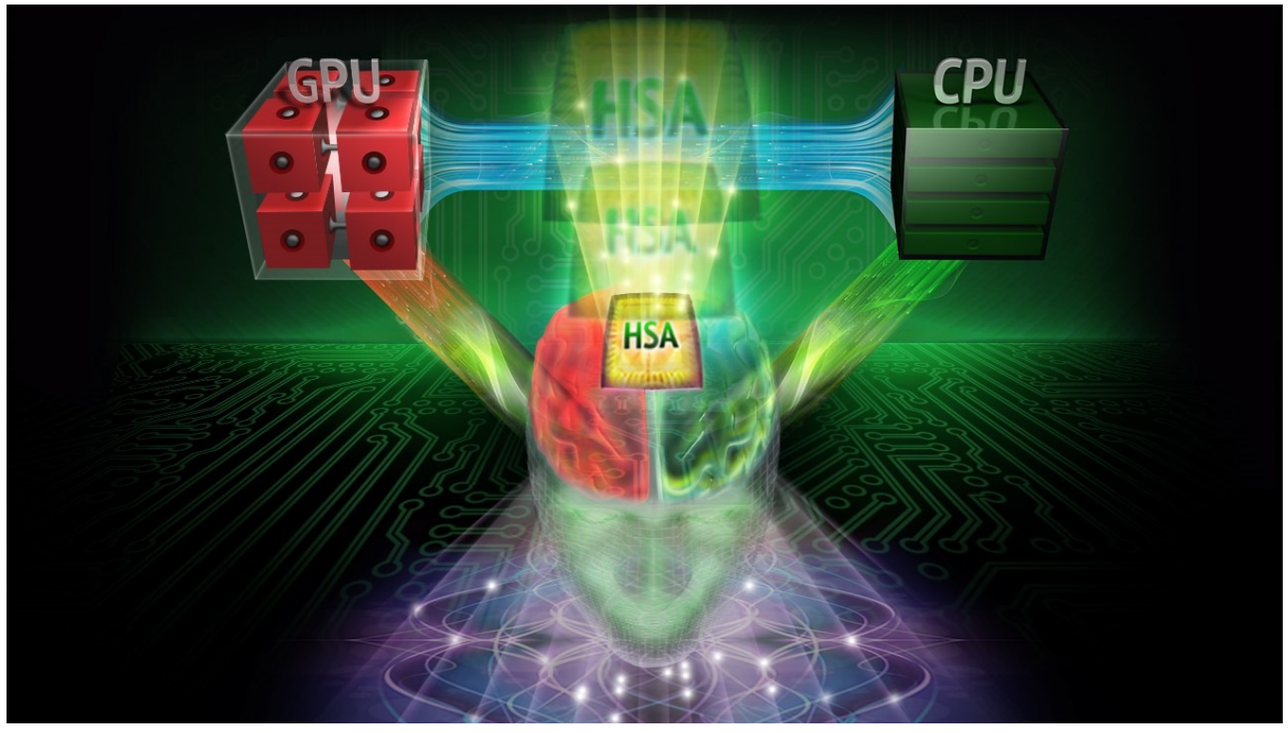


- Dark Silicon



HETEROGENEOUS ERA

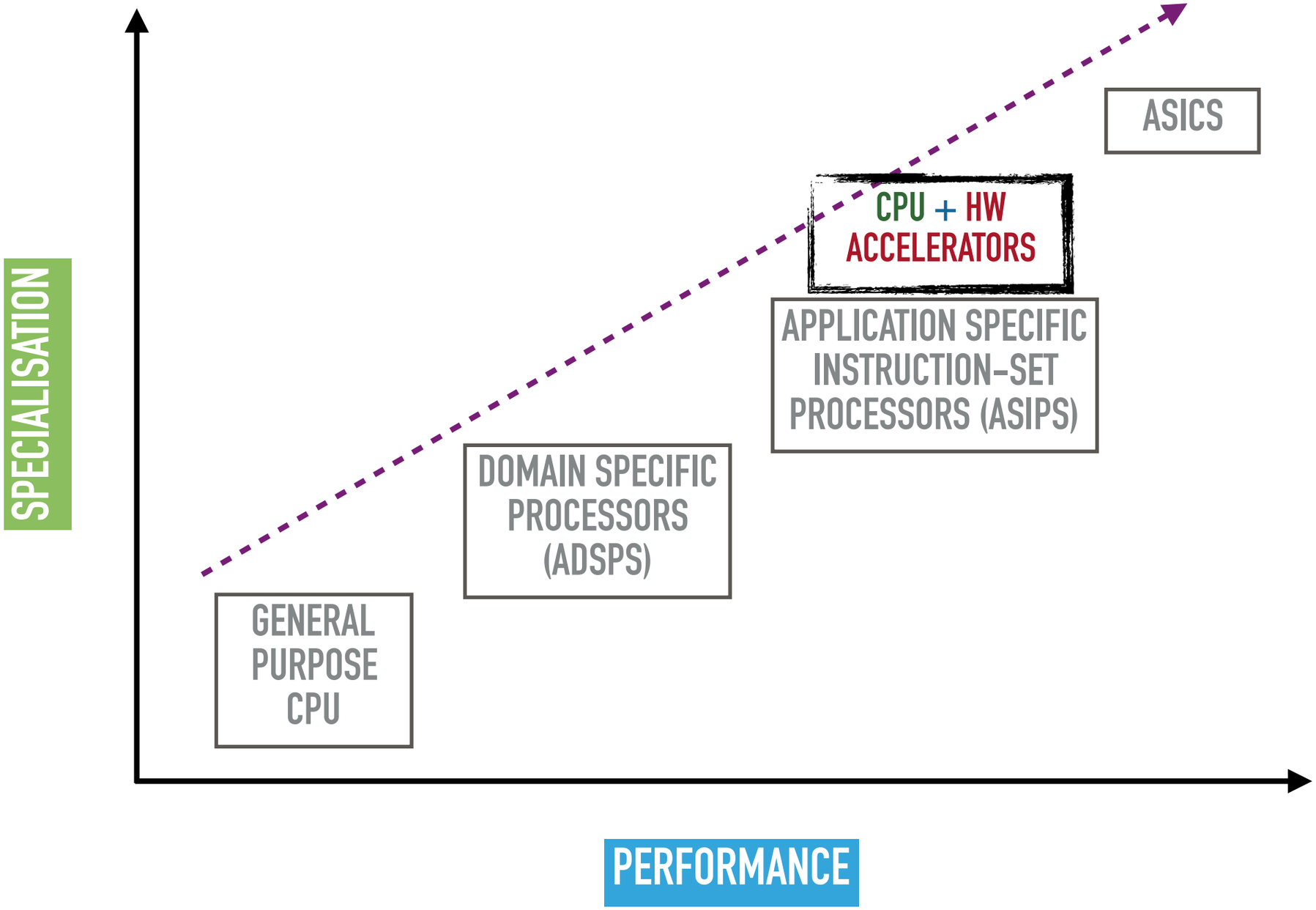
- Multiple cores –> Heterogeneous Computing
- Need for Specialised HW



[1]

[1] http://developer.amd.com/resources/heterogeneous-computing/what-is-heterogeneous-system-architecture-hsa/

SPECIALISATION AND PERFORMANCE



DATA TRANSFER IS THE BOTTLENECK

- Accelerate data transfer between accelerators and memory [4] [5]
- Custom storage as an optimisation [6] [7]

➡ Identify Data reuse for building custom storage

[4] G. Gutin, A. Johnstone, J. Reddington, E. Scott, and A. Yeo. An algorithm for finding input-output constrained convex sets in an acyclic digraph. J. Discrete Algorithms, 13:47–58, 2012.
[5] E. Giaquinta, A. Mishra, and L. Pozzi. Maximum convex subgraphs under I/O constraint for automatic identification of custom instructions. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 34(3):483–494, 2015.
[6] P. Biswas, N. Dutt, P. Ienne, and L. Pozzi. Automatic identification of application-specific functional units with architecturally visible storage. In *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition*, pages 212–217, Mar. 2006.
[7] M.Haaß,L.Bauer,andJ.Henkel.Automatic Custom Instruction Identification in Memory Streaming Algorithms. In Proceedings of the International Conference on Compilers, Architectures, and Synthesis for Embedded Systems, pages 1–9, Oct. 2014.

DATA TRANSFER IS THE BOTTLENECK

- Rely on source-to-source transformations [8] [9] [10]
  - Limited Parallelism [8] [9]
  - Large Internal Buffers [10]

[8] W. Meeus and D. Stroobandt. Automating data reuse in high-level synthesis. In Proceedings of the Design, Automation and Test in Europe Conference and Exhibition, pages 1–4, Mar. 2014.
[9] L.-N. Pouchet, P. Zhang, P. Sadayappan, and J. Cong. Polyhedral-based data reuse optimization for configurable computing. In Proceedings of the 2013 ACM/SIGDA 21st International Symposium on Field Programmable Gate Arrays, pages 29–38, Feb. 2013.
[10] M. Schmid, O. Reiche, F. Hannig, and J. Teich. Loop coarsening in C-based high-level synthesis. In Proceedings of the 26th International Conference on Application-specific Systems, Architectures and Processors, pages 166–173, July 2015.

DATA TRANSFER IS THE BOTTLENECK

- Rely on source-to-source transformations [8] [9] [10]
  - Limited Parallelism [8] [9]
  - Large Internal Buffers [10]

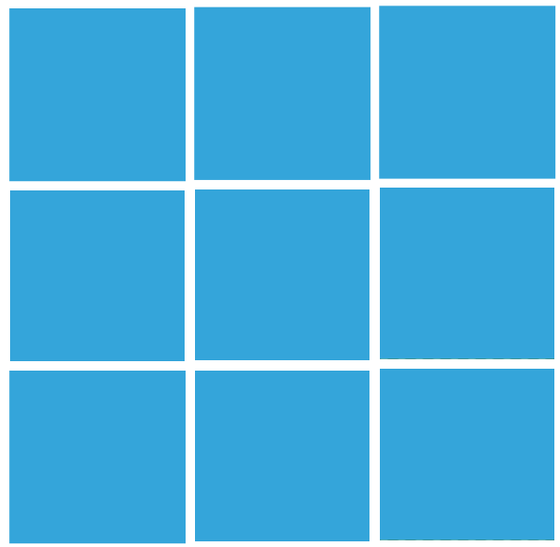
➡ Multiple Datapaths

➡ Area efficient Accelerators

[8] W. Meeus and D. Stroobandt. Automating data reuse in high-level synthesis. In Proceedings of the Design, Automation and Test in Europe Conference and Exhibition, pages 1–4, Mar. 2014.
[9] L.-N. Pouchet, P. Zhang, P. Sadayappan, and J. Cong. Polyhedral-based data reuse optimization for configurable computing. In Proceedings of the 2013 ACM/SIGDA 21st International Symposium on Field Programmable Gate Arrays, pages 29–38, Feb. 2013.
[10] M. Schmid, O. Reiche, F. Hannig, and J. Teich. Loop coarsening in C-based high-level synthesis. In Proceedings of the 26th International Conference on Application-specific Systems, Architectures and Processors, pages 166–173, July 2015.

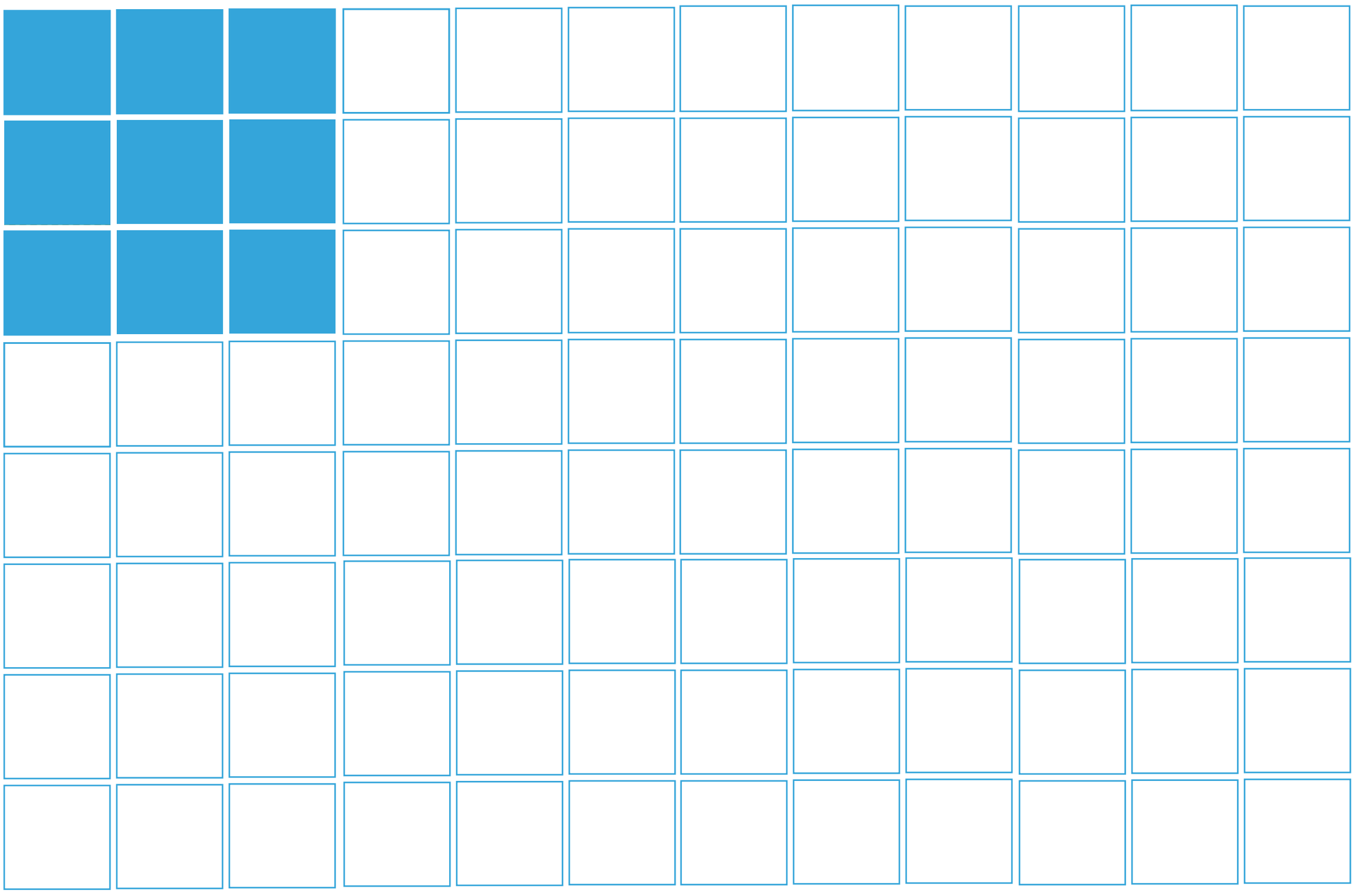
SW ANALYSIS FOR DATA REUSE

```
void Sobelsystem() {

  outer_loop:for(i = 1; i < 100; i++) {

    inner_loop:for(j = 1; j < 100; j++) {

      Sobelmodule(image[i-1][j-1], image[i][j-1],
        image[i+1][j+1], image[i-1][j], image[i+1][j],
        image[i-1][j+1], image[i][j+1], image[i+1][j-1],
        image[i][j]);

    }

  }

}
```

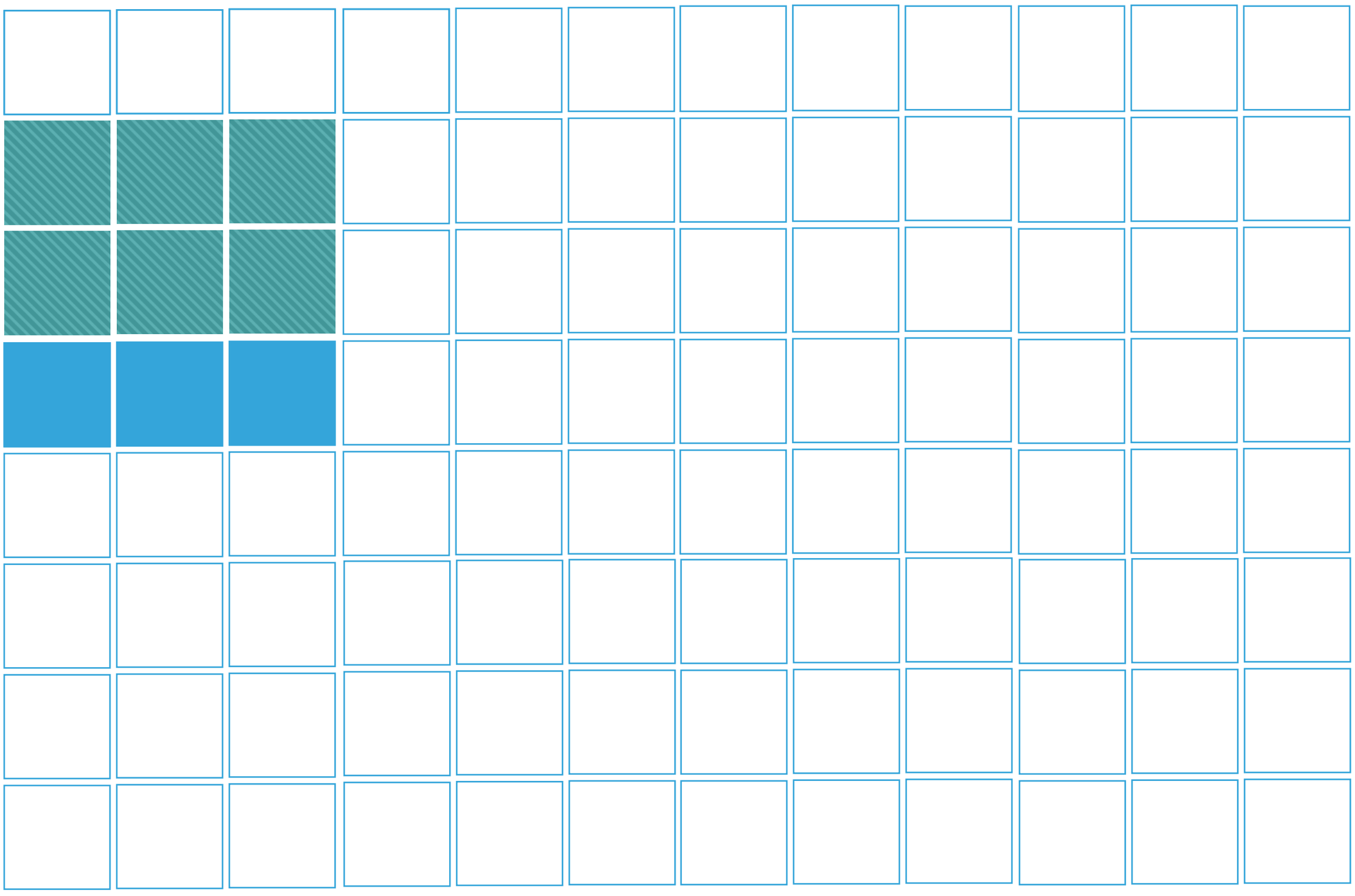


LOADS

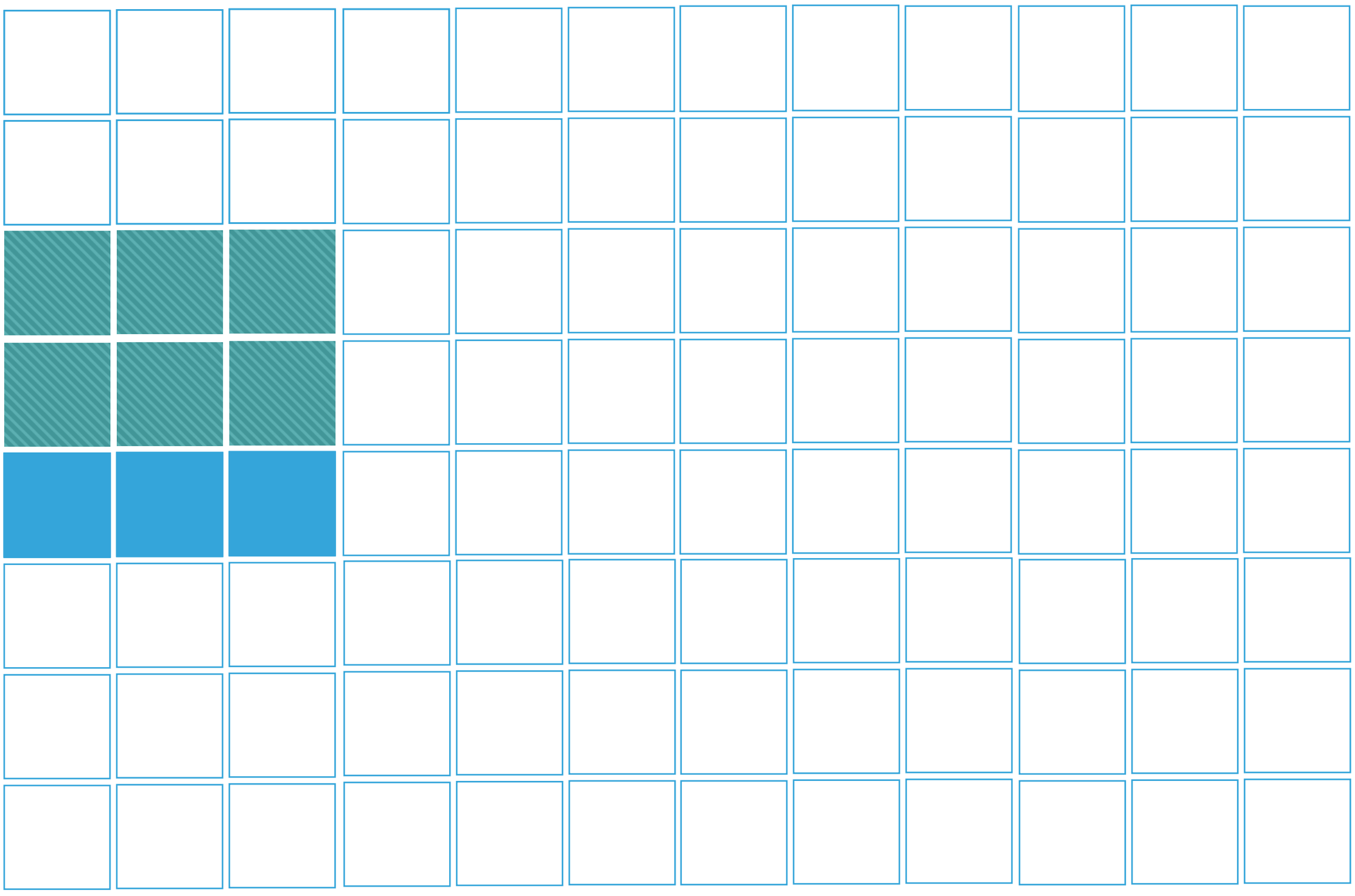
SW ANALYSIS FOR DATA REUSE



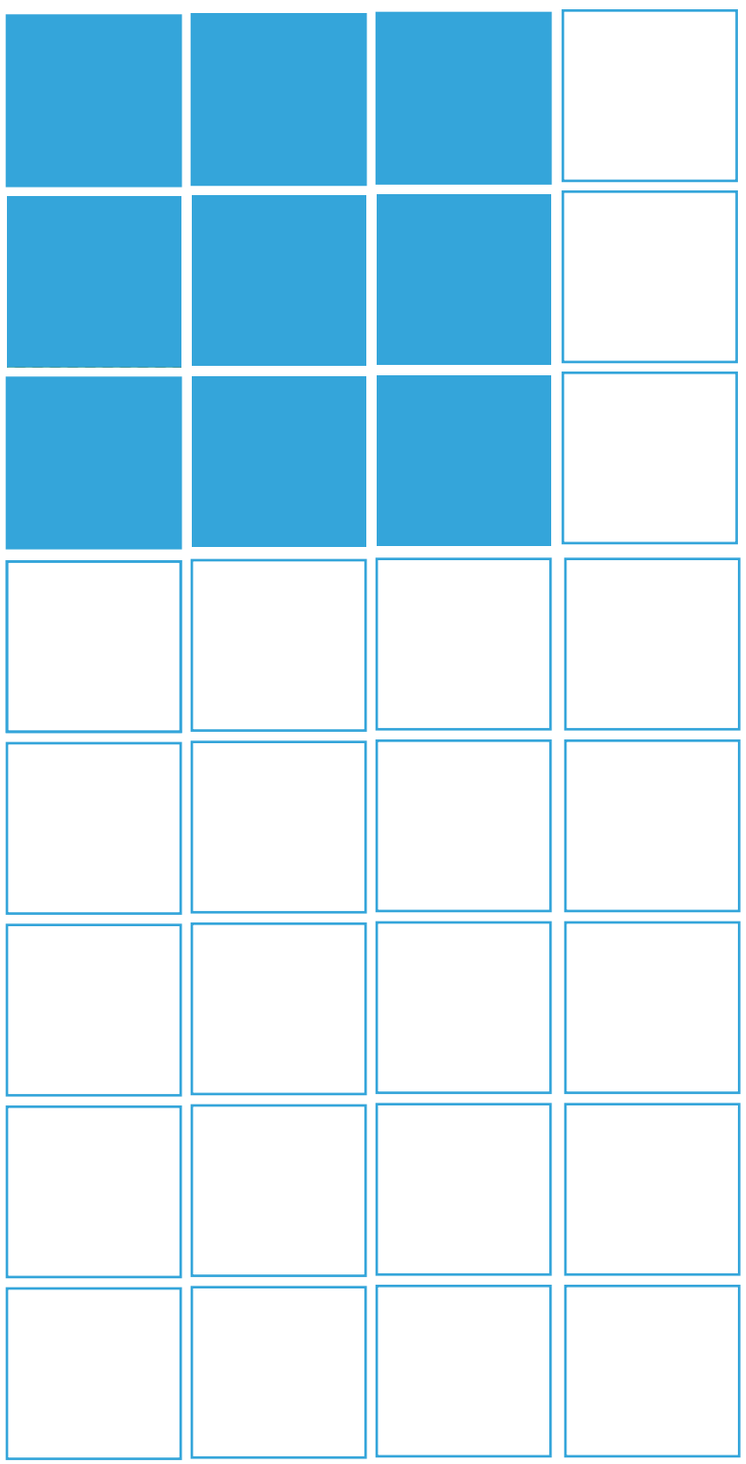
SW ANALYSIS FOR DATA REUSE

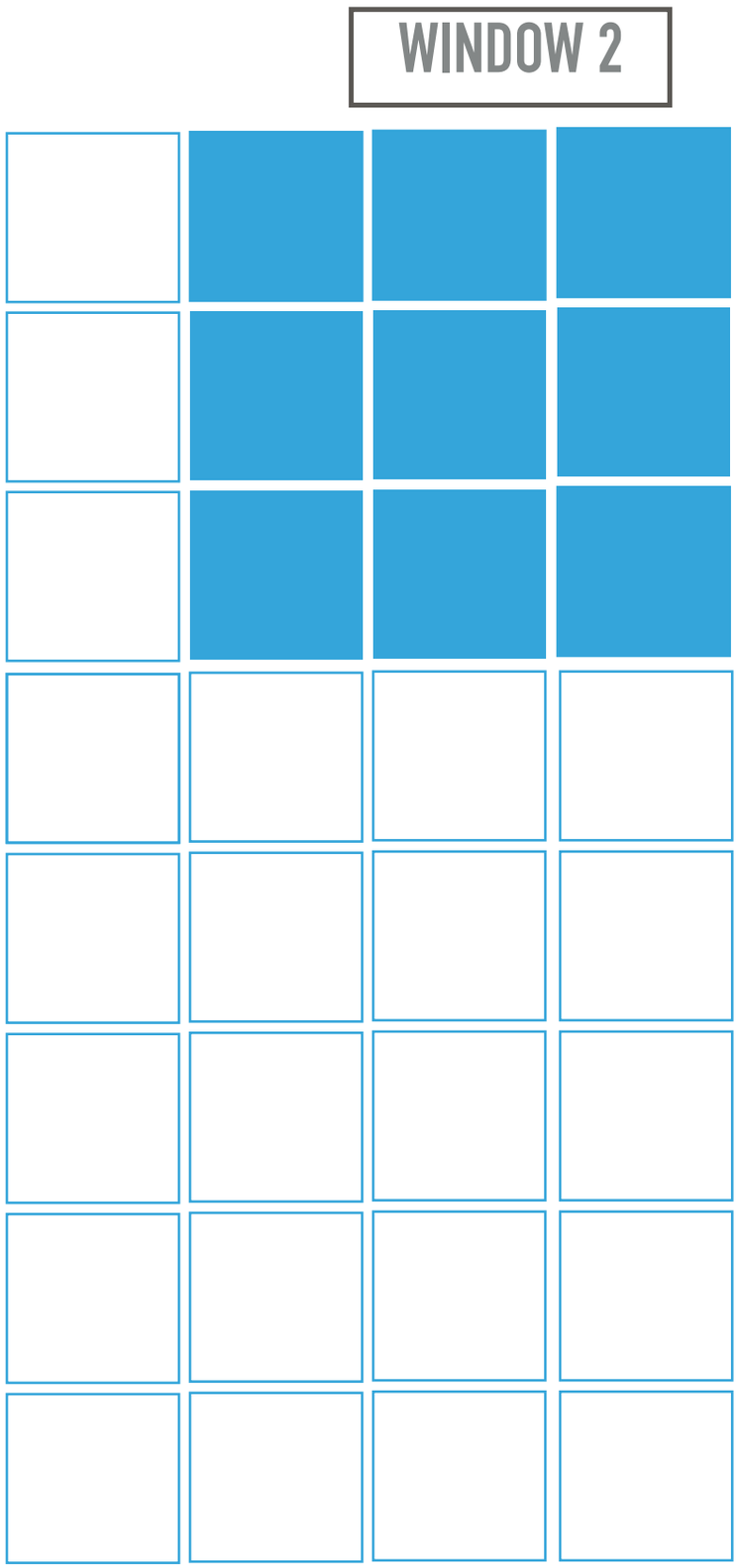


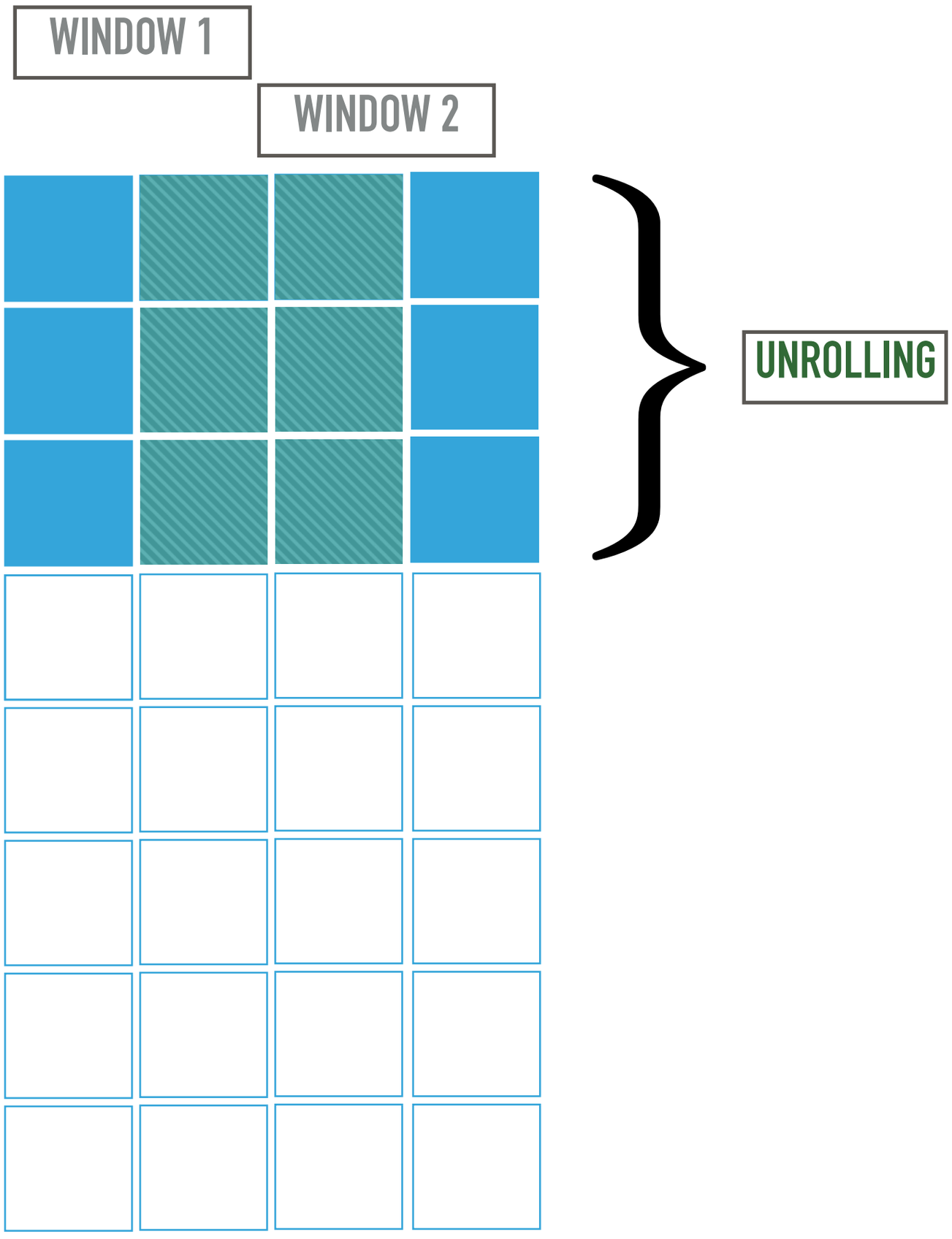
SW ANALYSIS FOR DATA REUSE

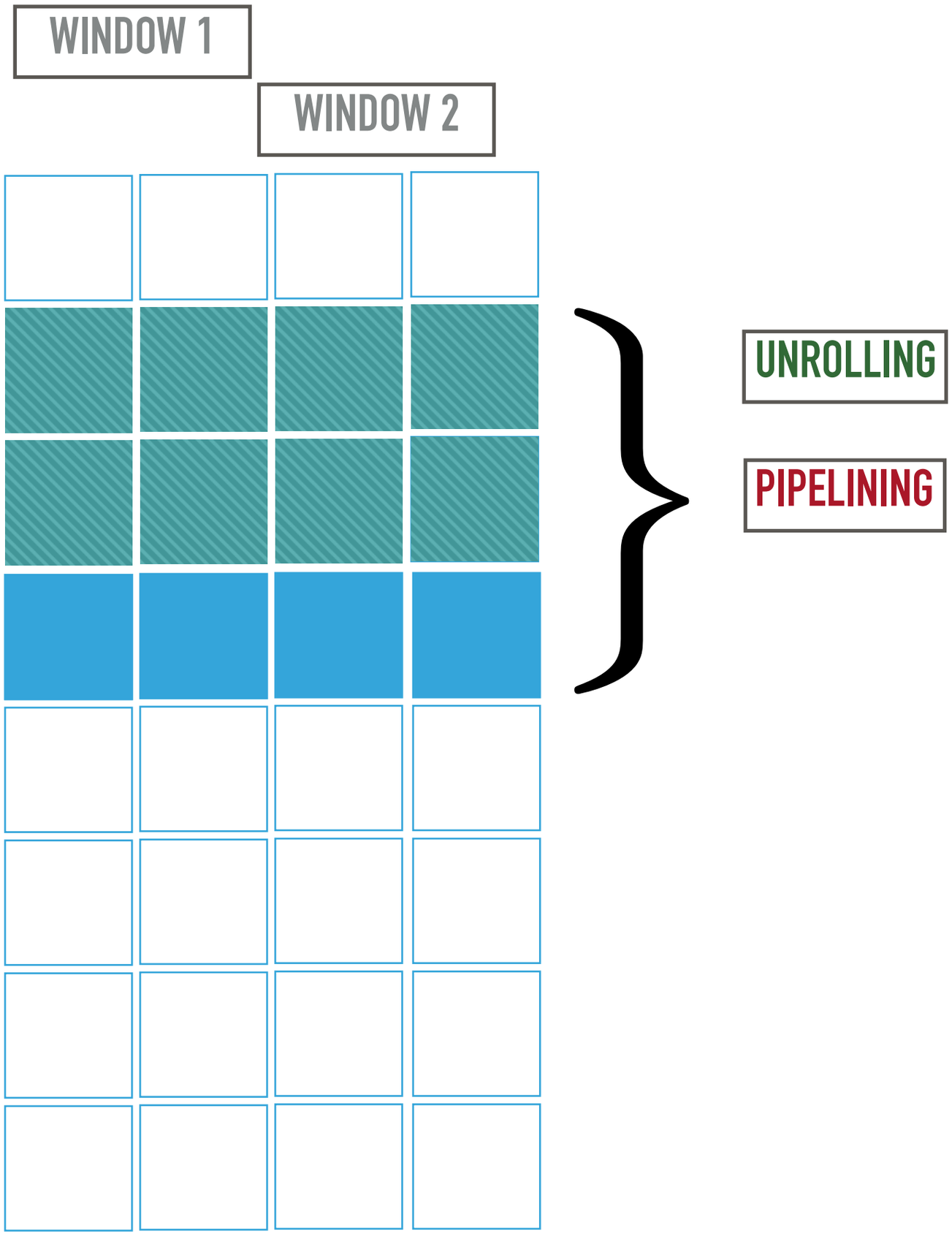


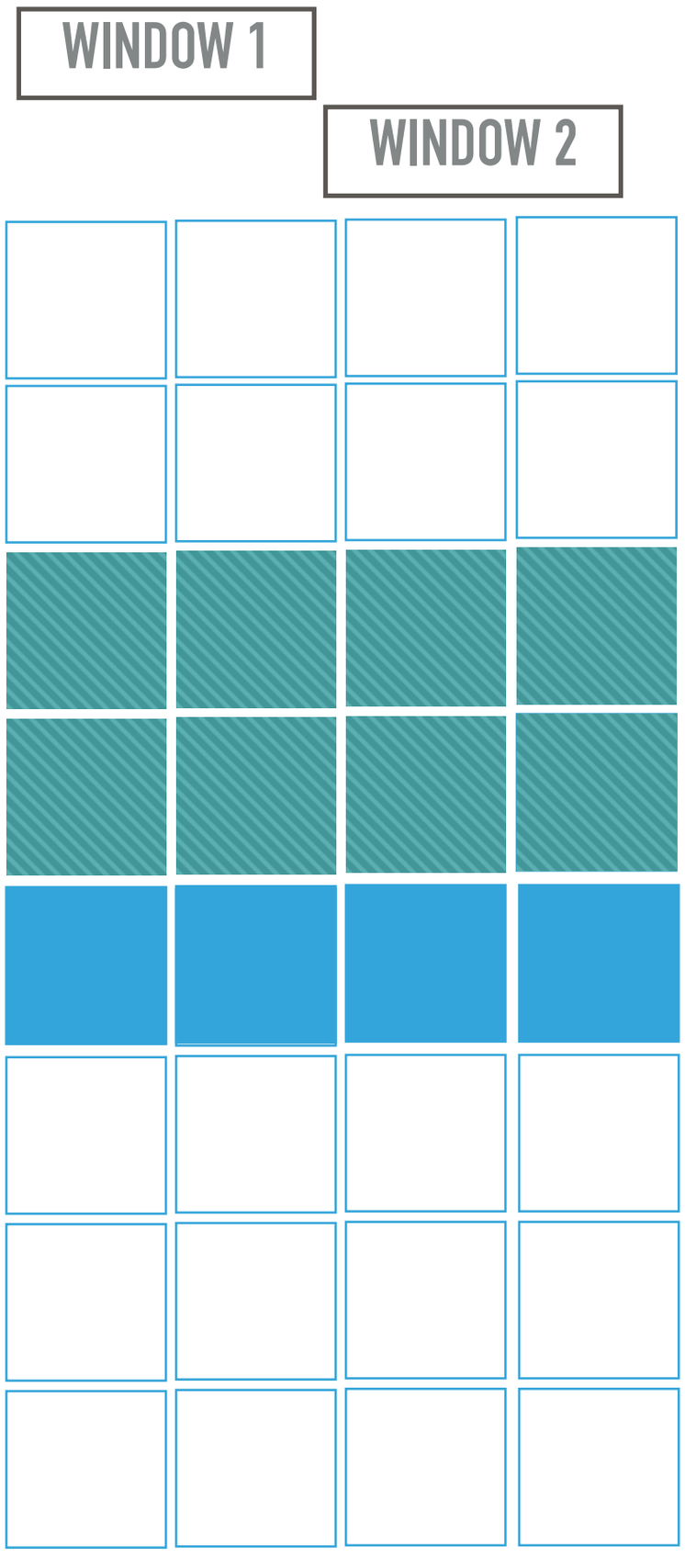
WINDOW 1




WINDOW 2


WINDOW 1
WINDOW 2
UNROLLING


WINDOW 1
WINDOW 2
UNROLLING
PIPELINING



UNROLLING

PIPELINING

COMPILERS

- Perform Source Code Analysis
- Identify Data Reuse in Sliding Window Applications

COMPILERS

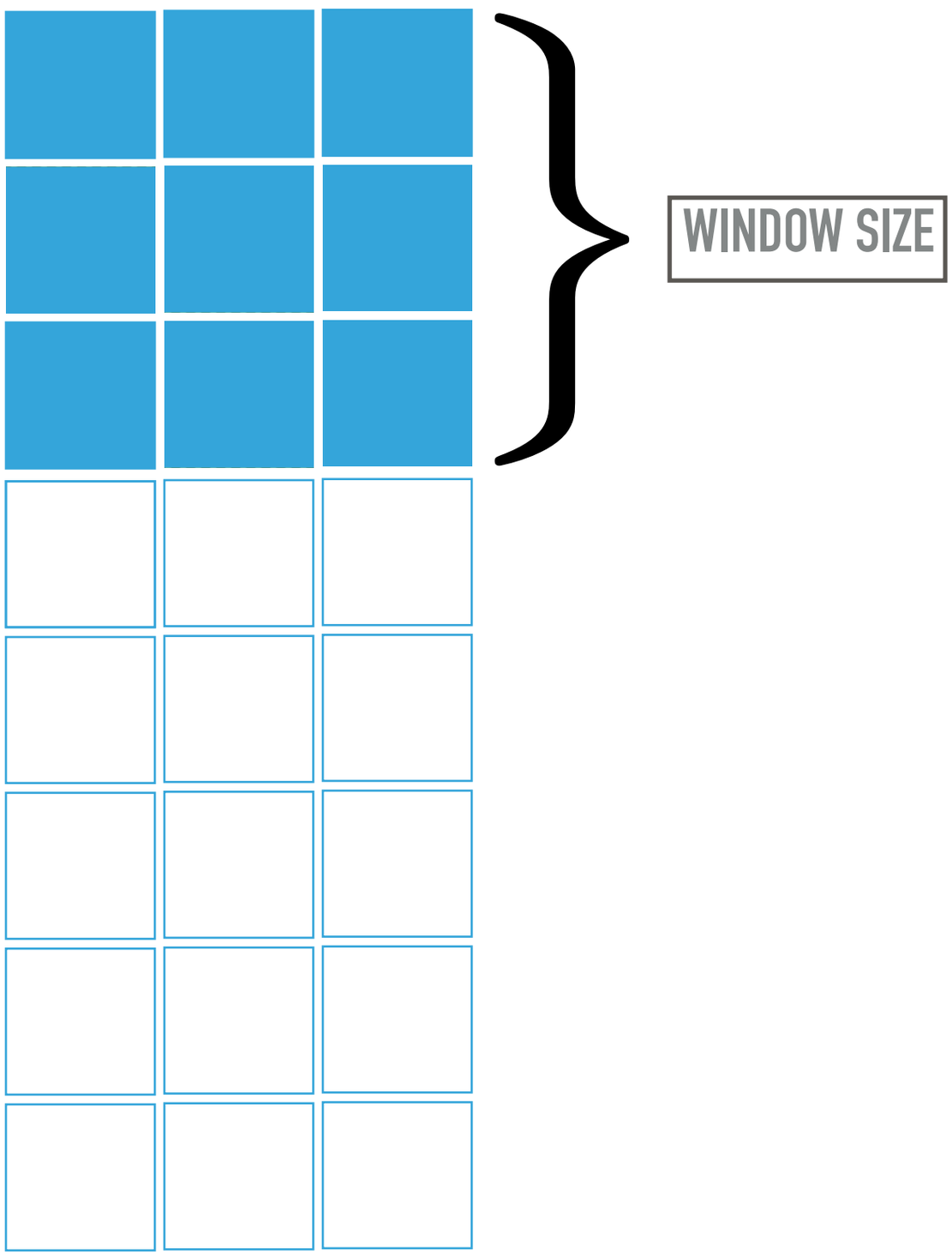
- Perform Source Code Analysis
- Identify Data Reuse in Sliding Window Applications

[2]

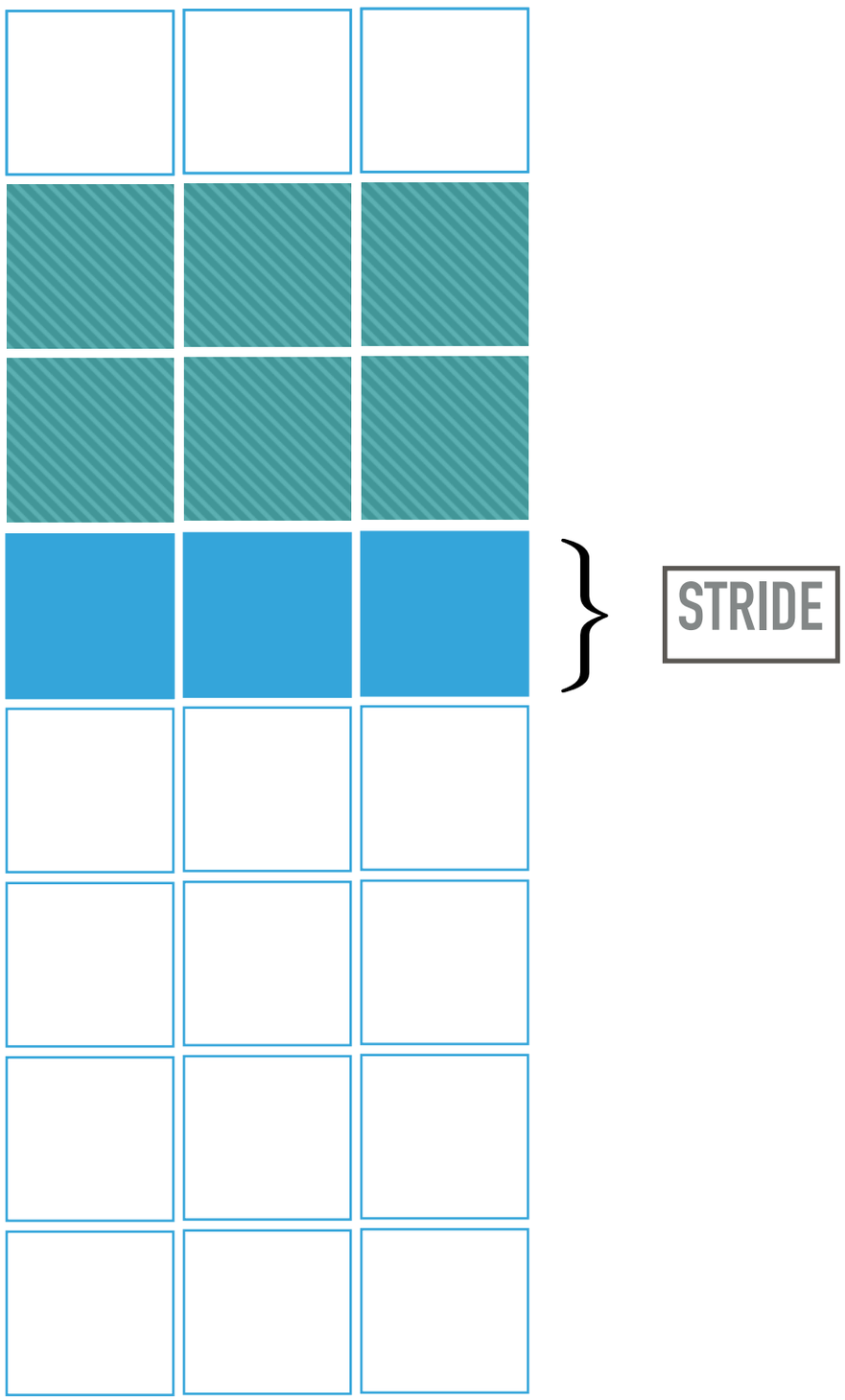
[3]

[2] C. Lattner and V. Adve. LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation. In Proceedings of the 2nd International Symposium on Code generation and optimization, Palo Alto, California, Mar 2004.
[3] T. Grosser, H. Zheng, R. Aloor, A. Simbürger, A. Größlinger, and L.-N. Pouchet. Polly-polyhedral optimization in llvm. In Proceedings of the First International Workshop on Polyhedral Compilation Techniques (IMPACT), volume 2011, 2011.

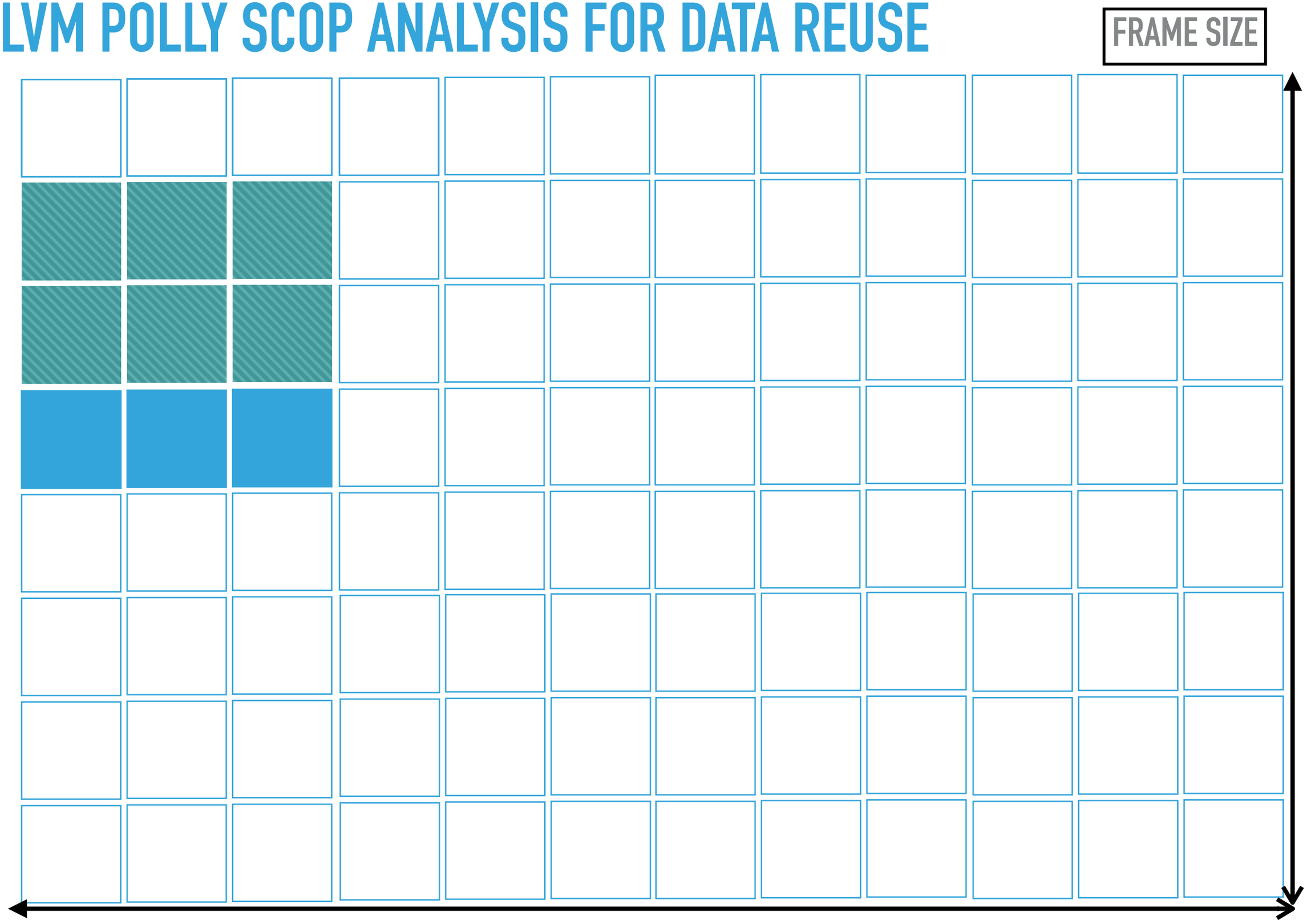
LLVM POLLY SCOP ANALYSIS FOR DATA REUSE



LLVM POLLY SCOP ANALYSIS FOR DATA REUSE



LLVM POLLY SCOP ANALYSIS FOR DATA REUSE

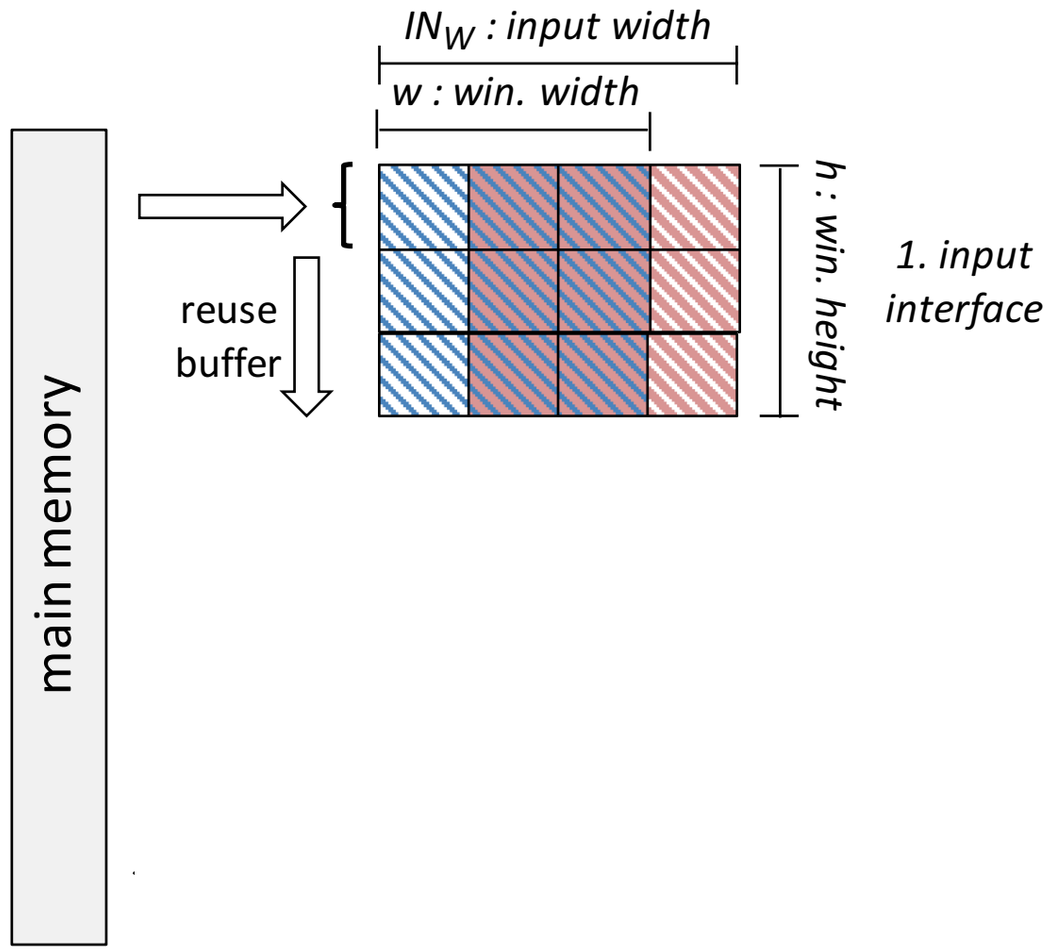


HW IMPLEMENTATION

Parameters retrieved with LLVM Polly Analysis Pass:

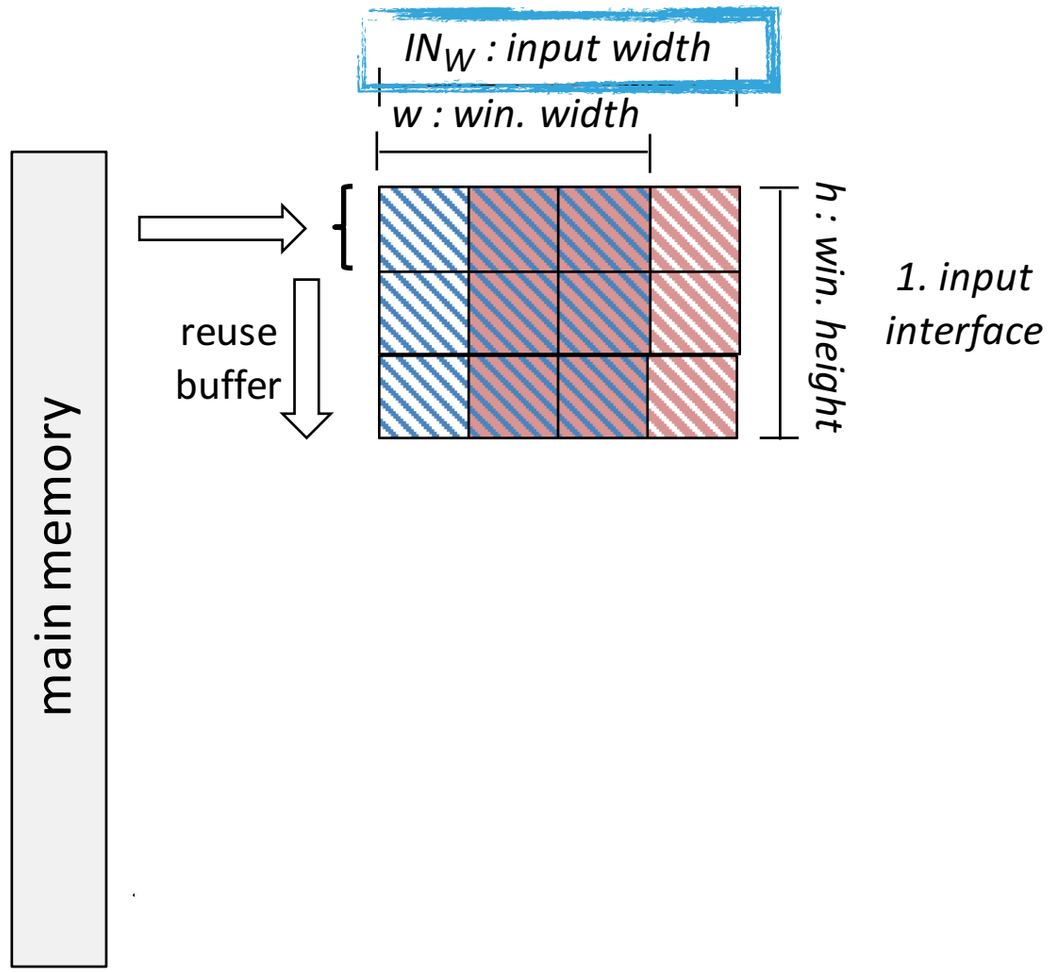
- Horizontal and Vertical Window Size
- Stride
- Iteration Domain (Frame Size)

HW IMPLEMENTATION



HW IMPLEMENTATION

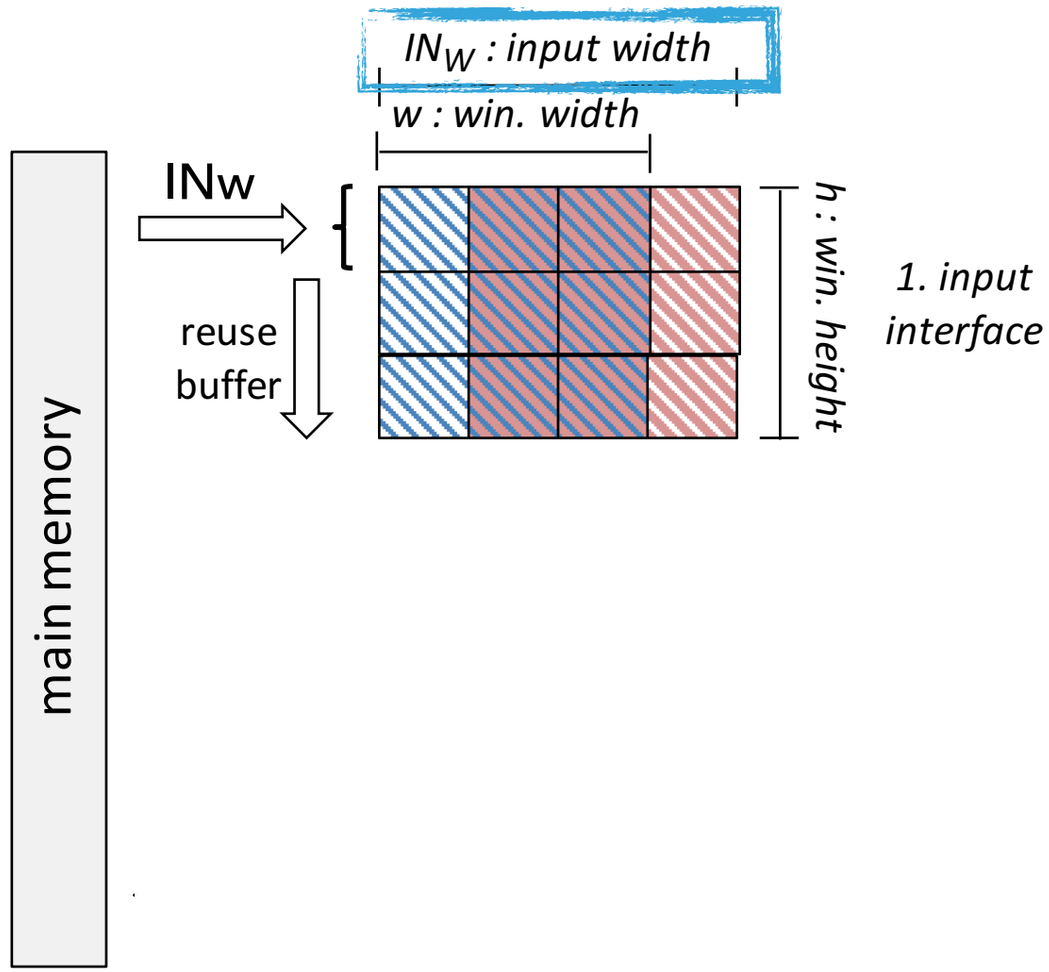
AVAILABLE INPUT DATA WIDTH



HW IMPLEMENTATION

Buffer Size:

Horizontal Width = Input Width (INw)



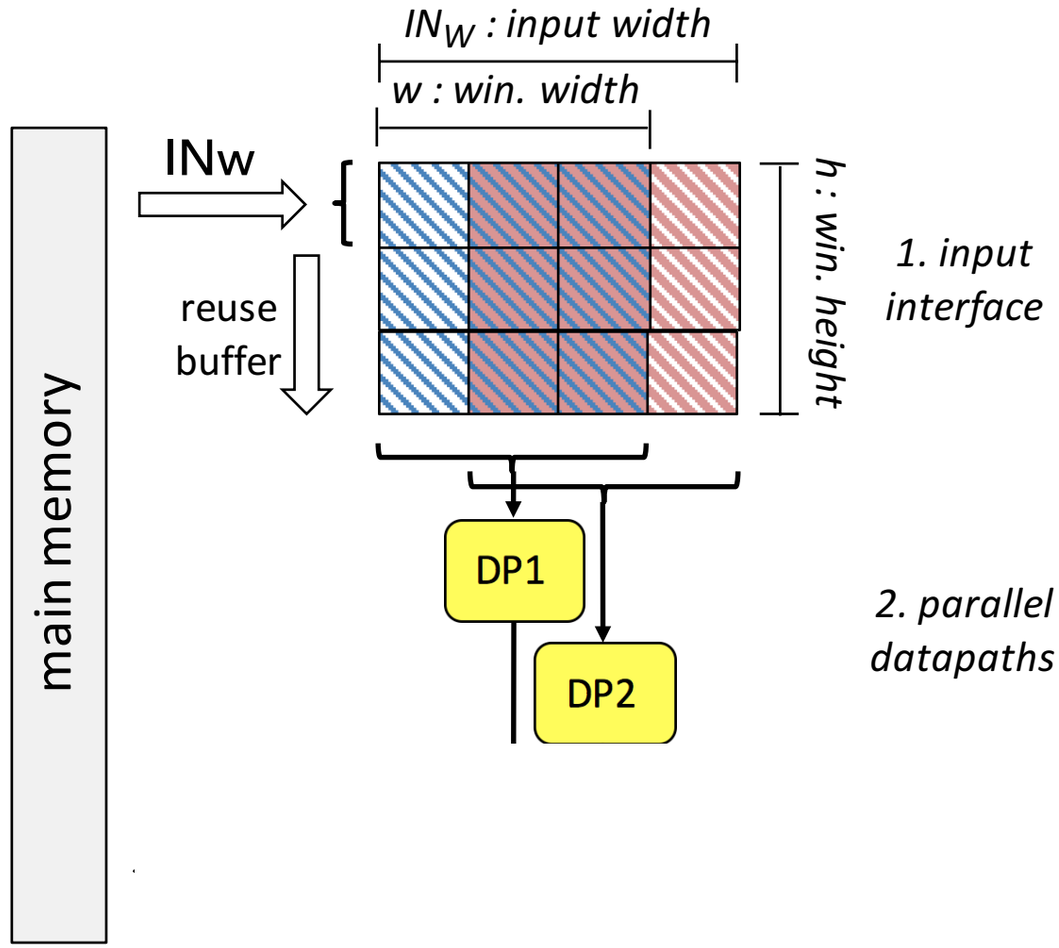
HW IMPLEMENTATION

Buffer Size:

Horizontal Width = Input Width (INw)

Example:

INw = Horizontal Win. Size + 1



HW IMPLEMENTATION

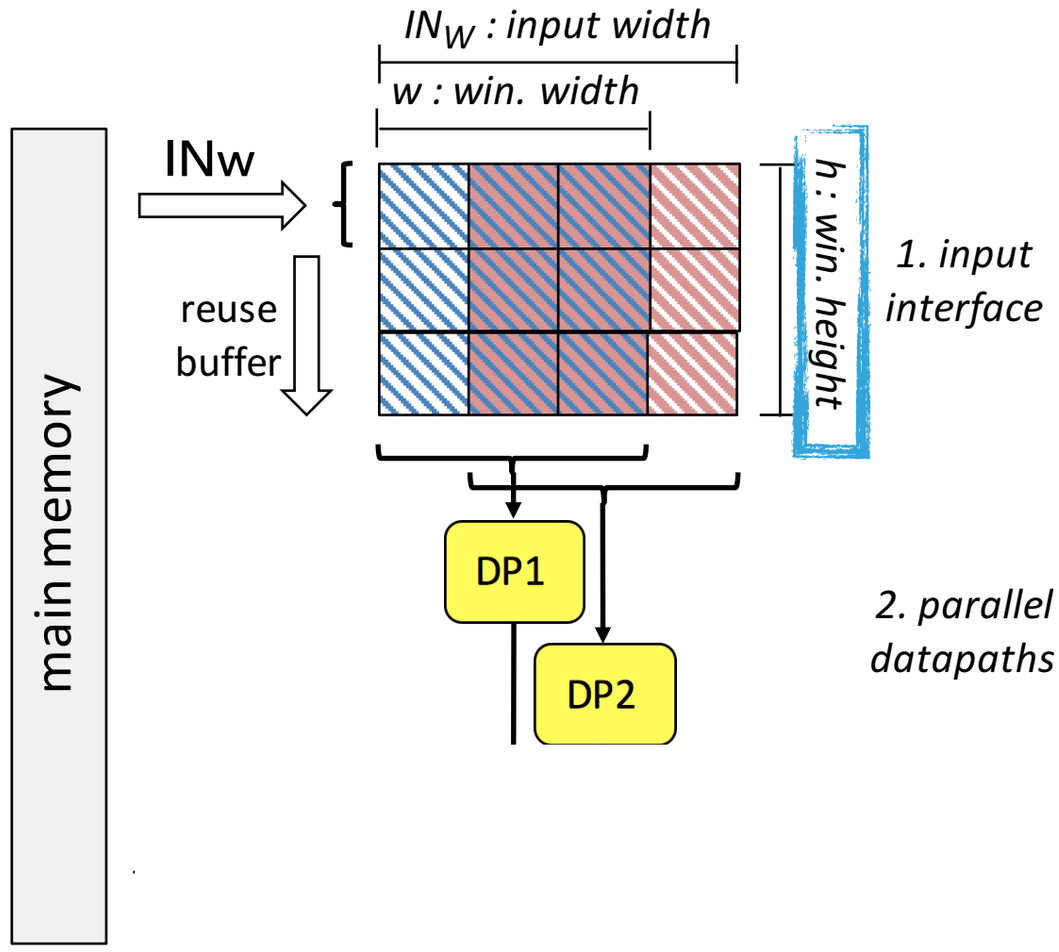
Buffer Size:

Horizontal Width = Input Width (INw)

Vertical Width = Vertical Win. Size

Example:

INw = Horizontal Win. Size + 1



HW IMPLEMENTATION

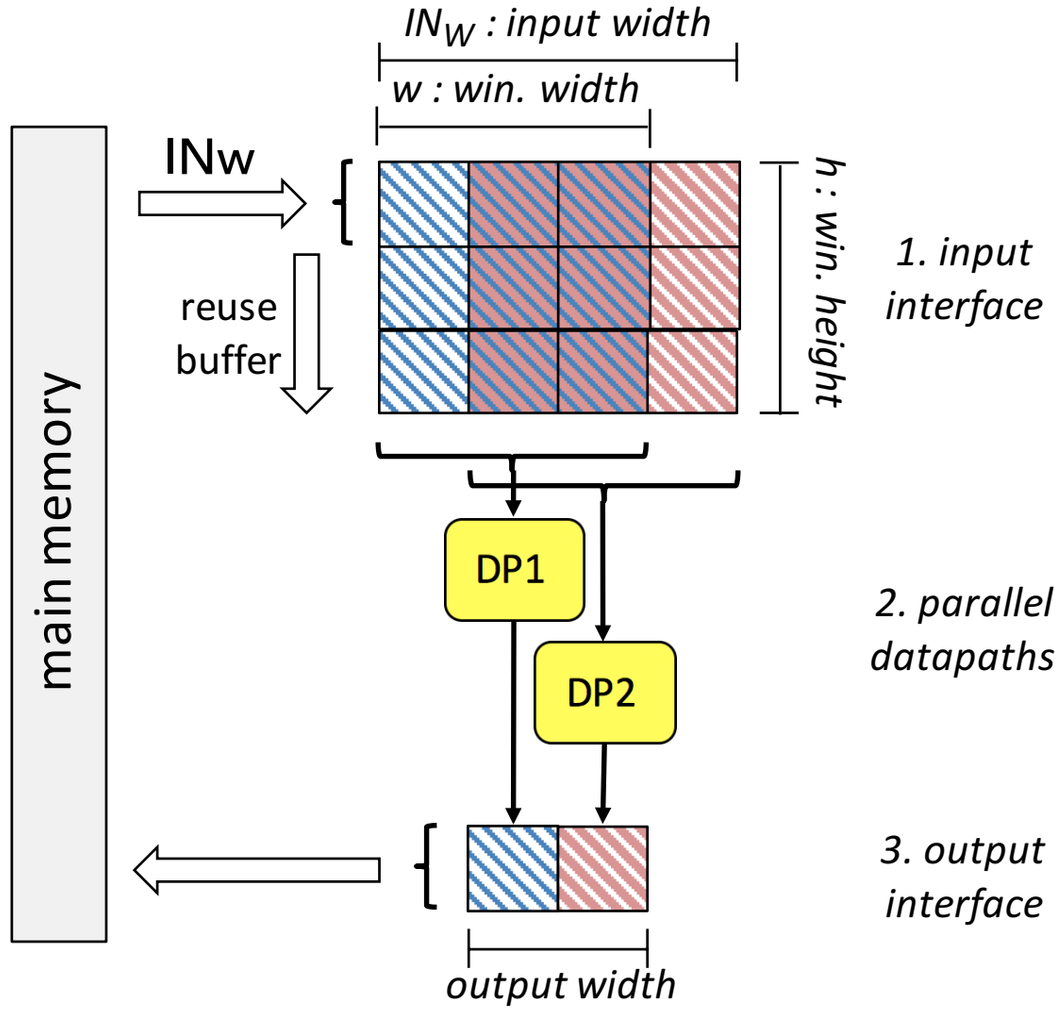
Buffer Size:

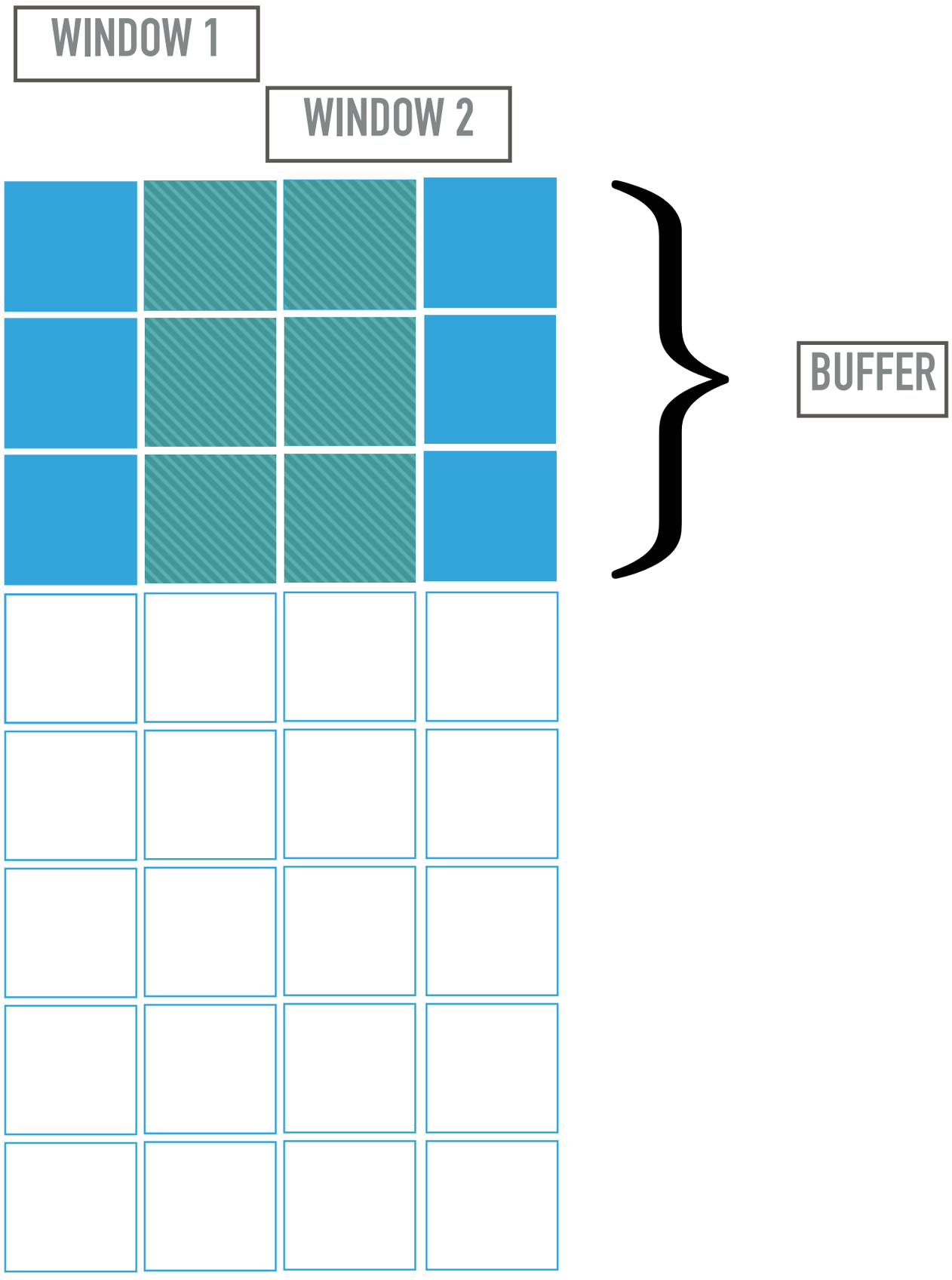
Horizontal Width = Input Width (INw)

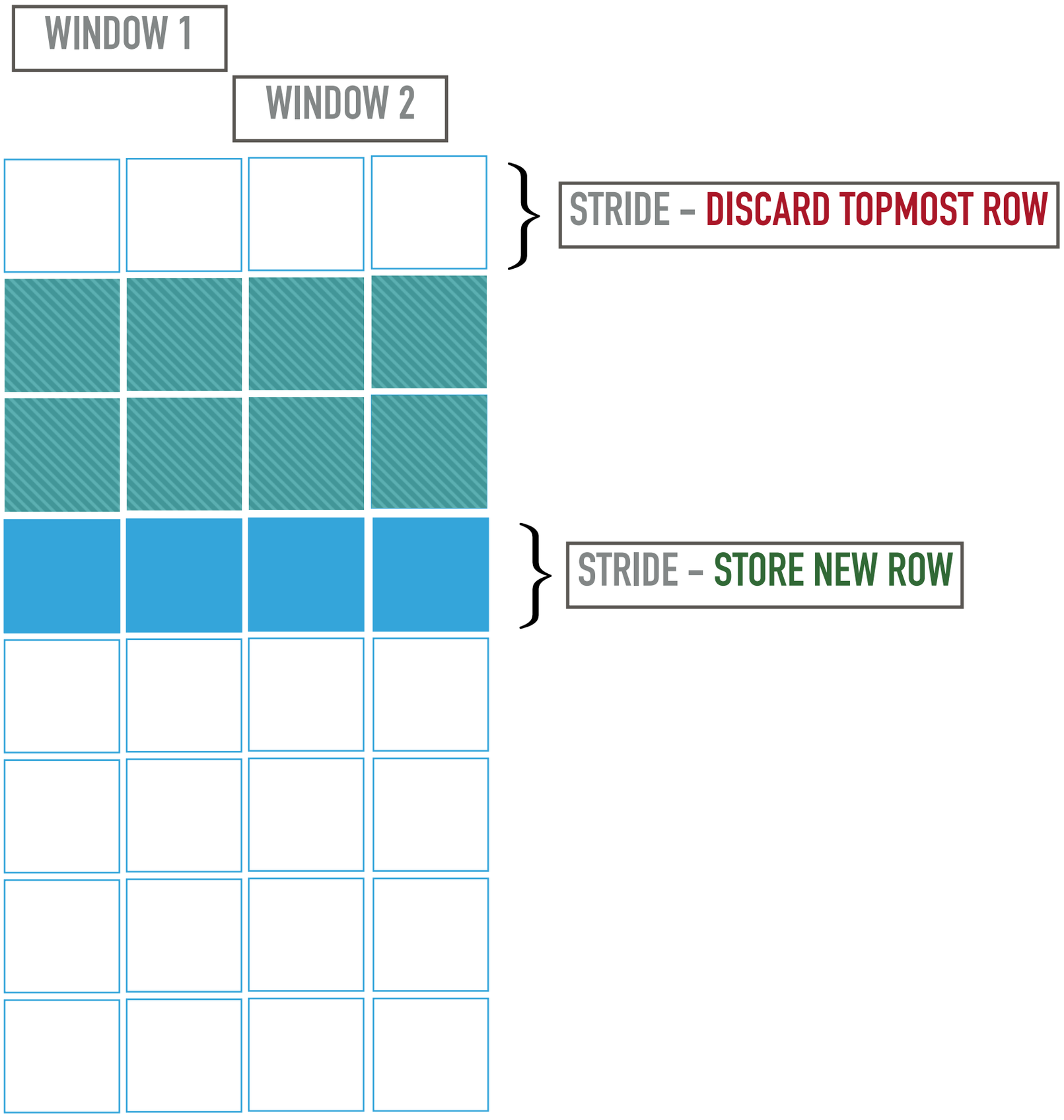
Vertical Width = Vertical Win. Size

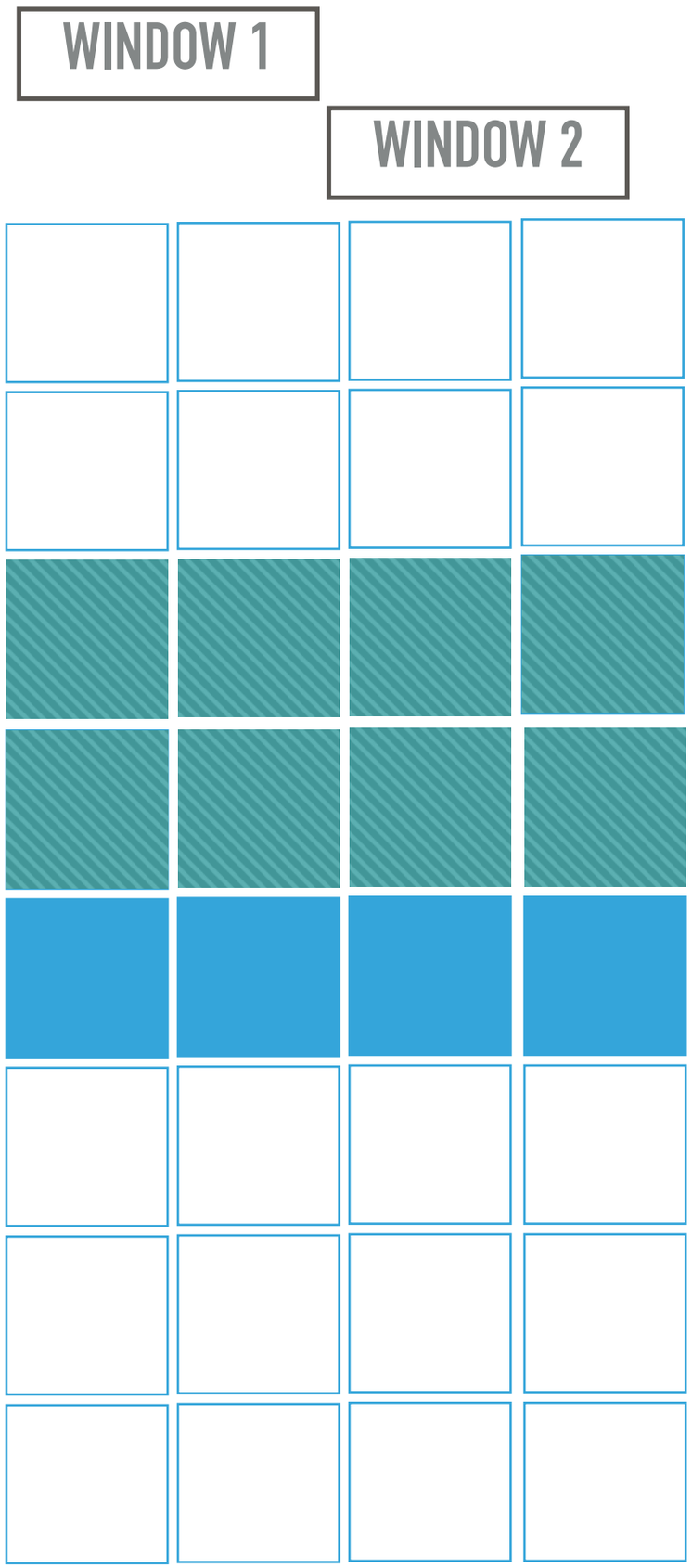
Example:

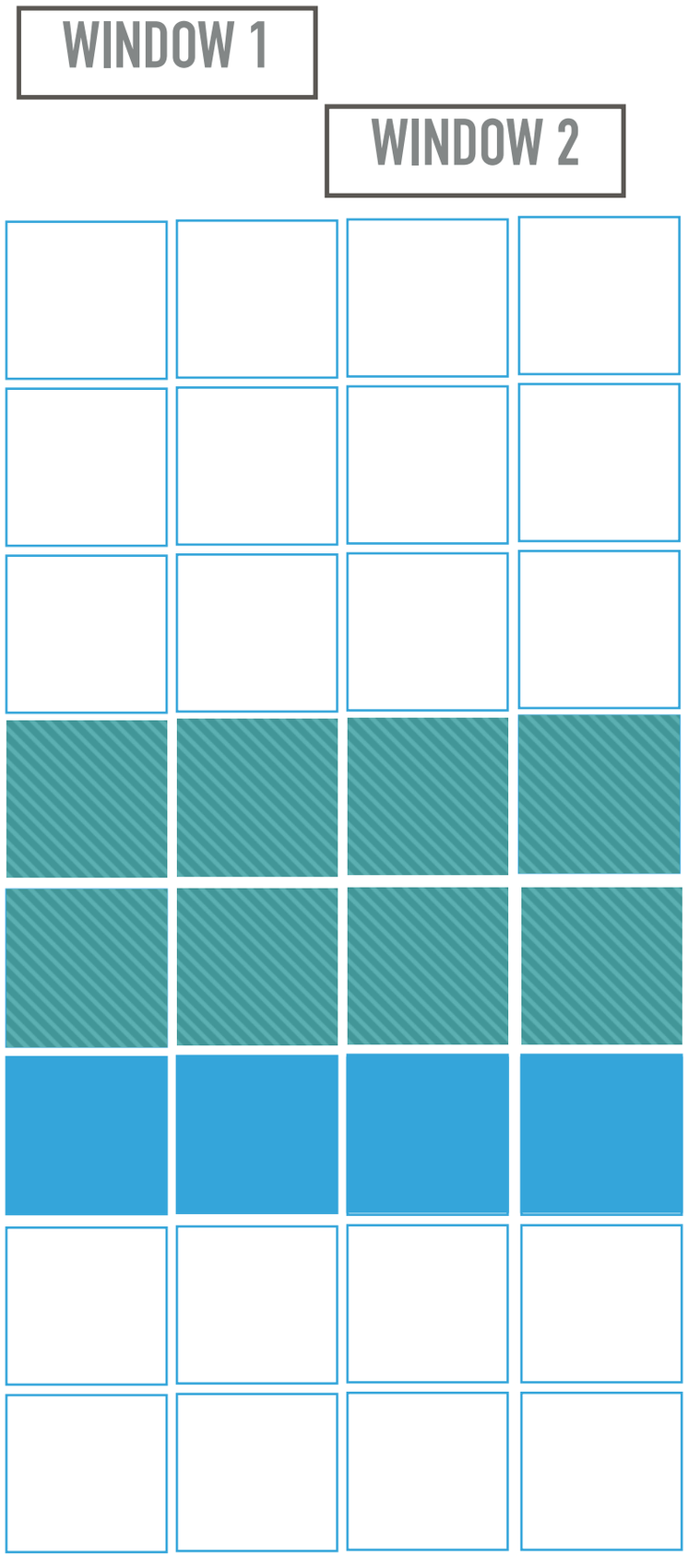
INw = Horizontal Win. Size + 1

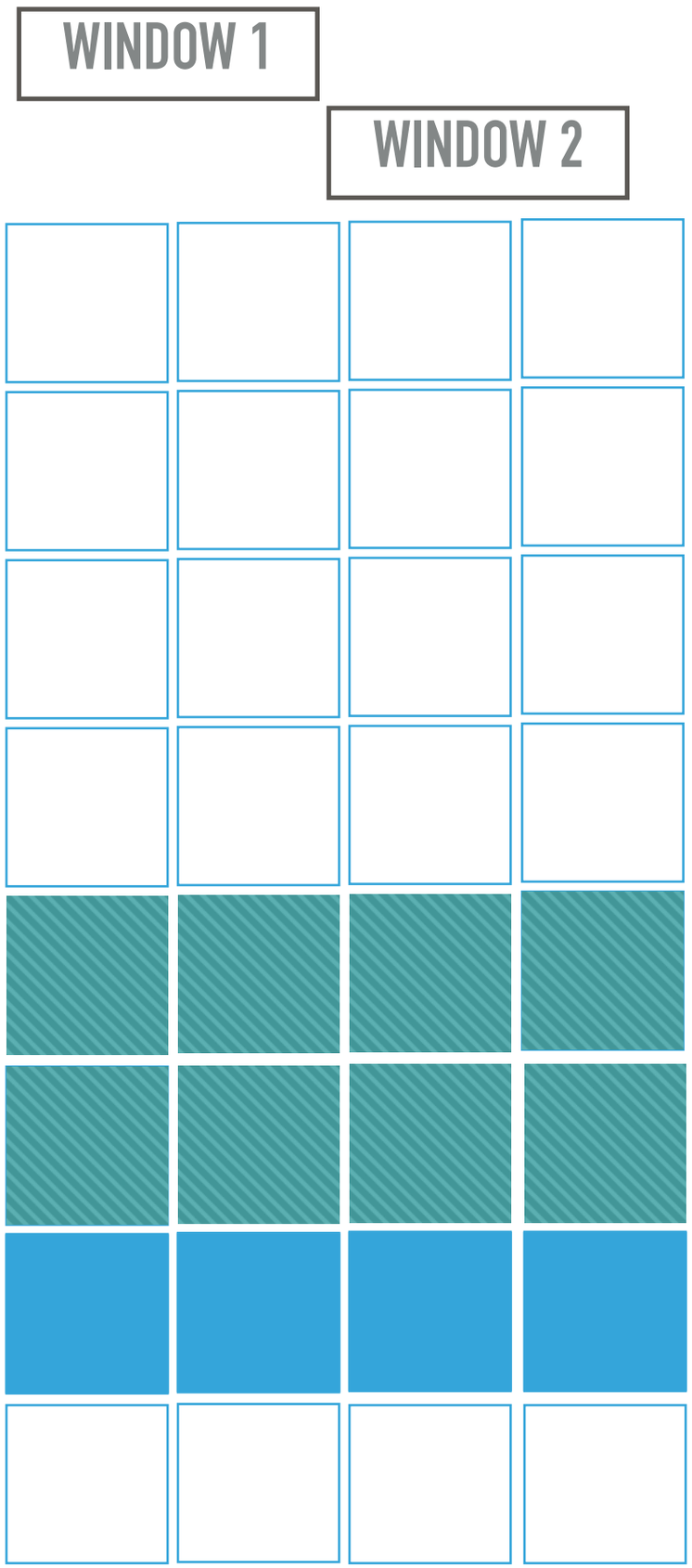


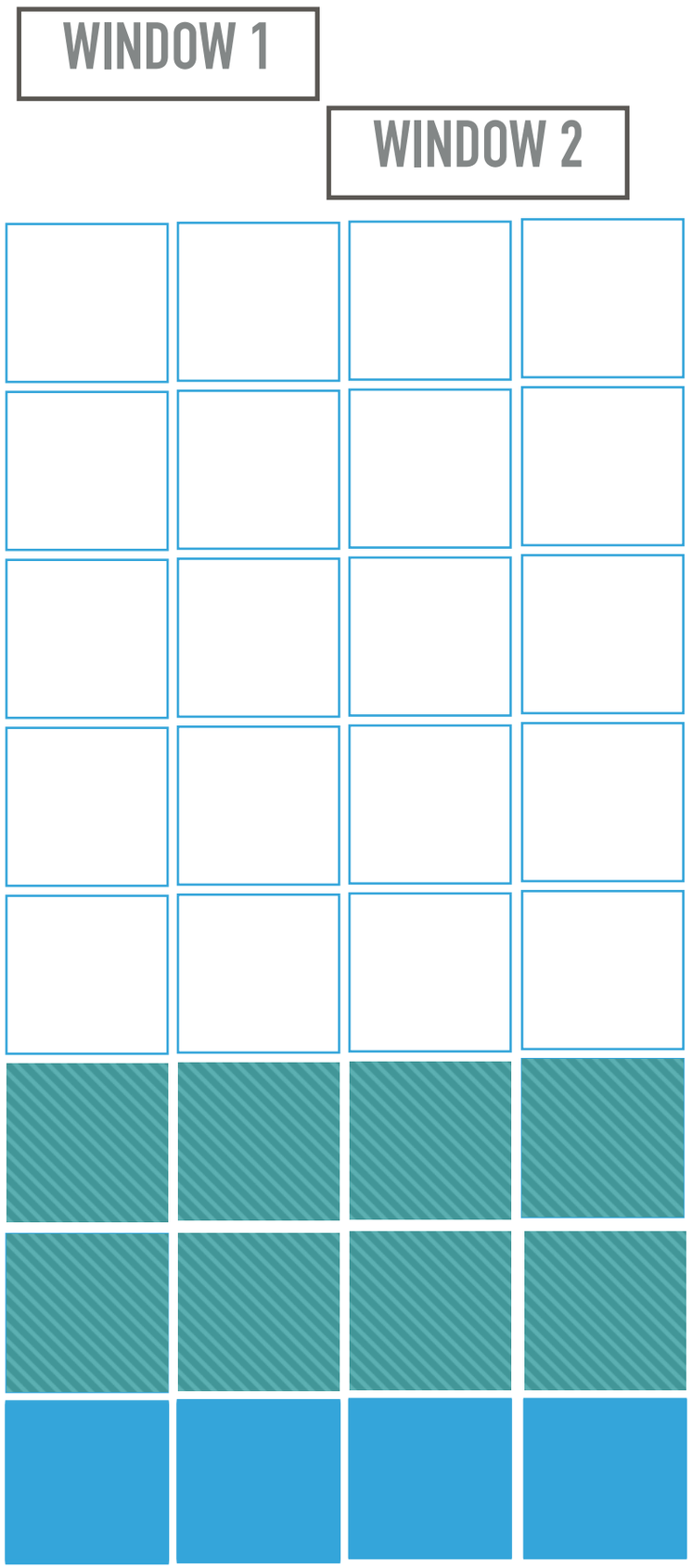

WINDOW 1
WINDOW 2
BUFFER

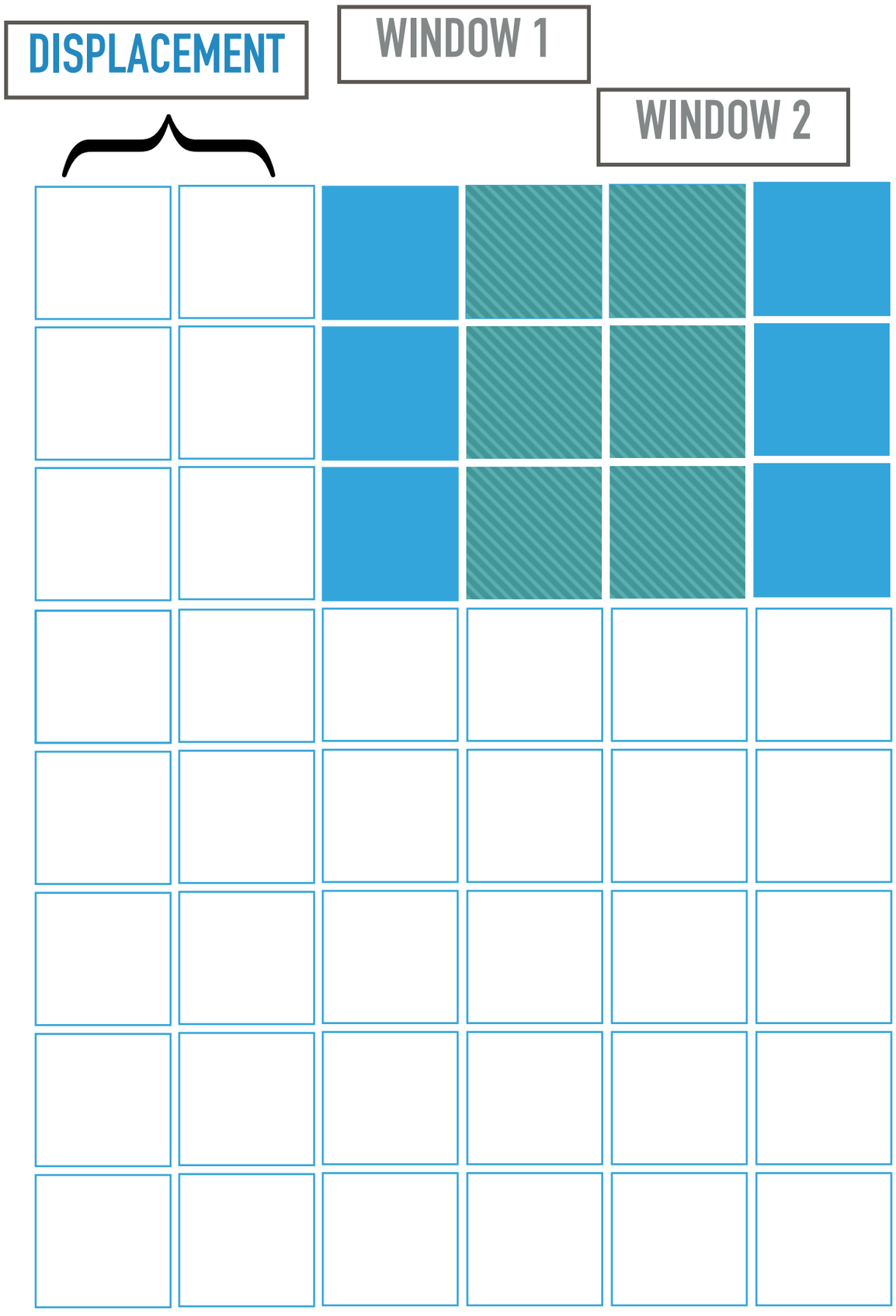

WINDOW 1
WINDOW 2
STRIDE – DISCARD TOPMOST ROW
STRIDE – STORE NEW ROW

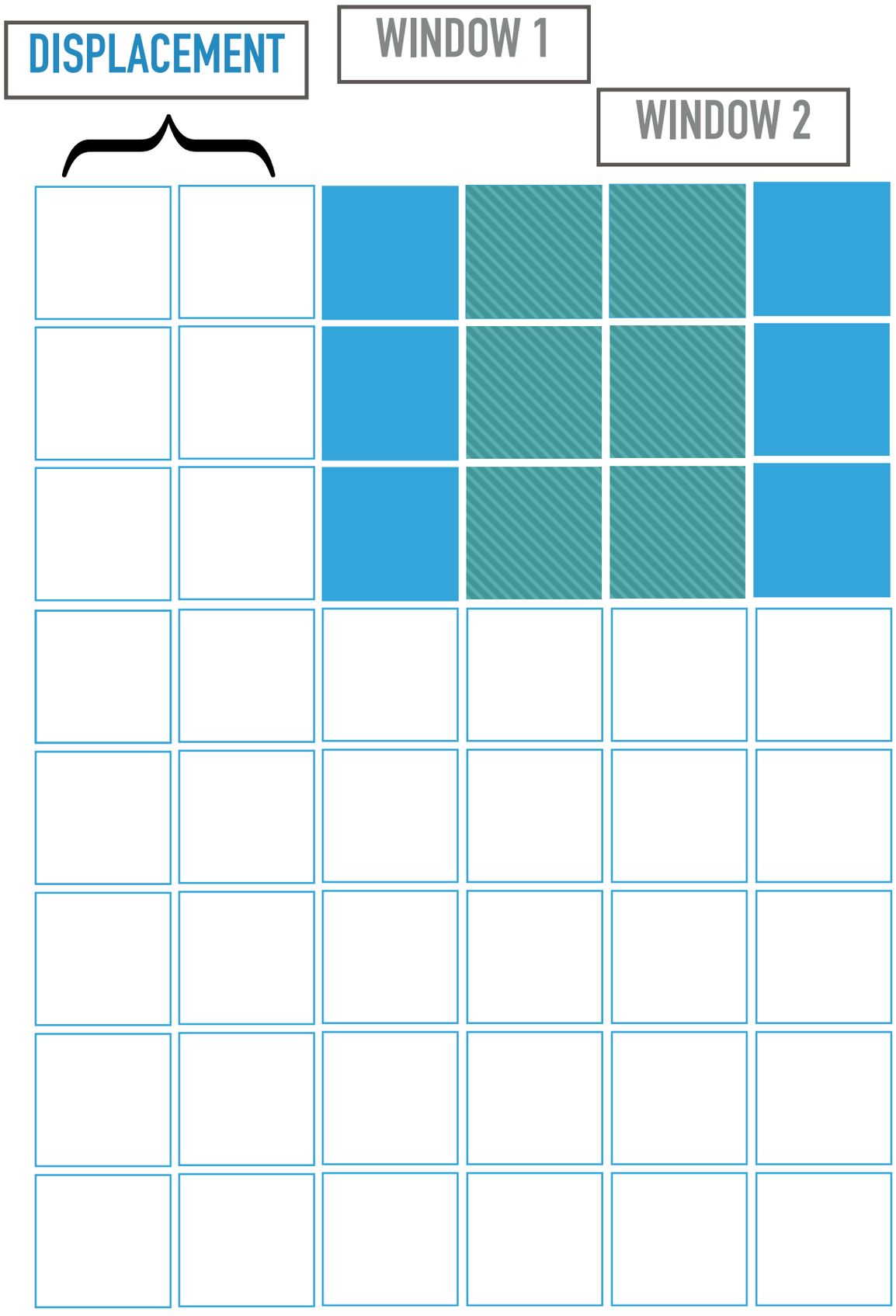

WINDOW 1
WINDOW 2


WINDOW 1
WINDOW 2

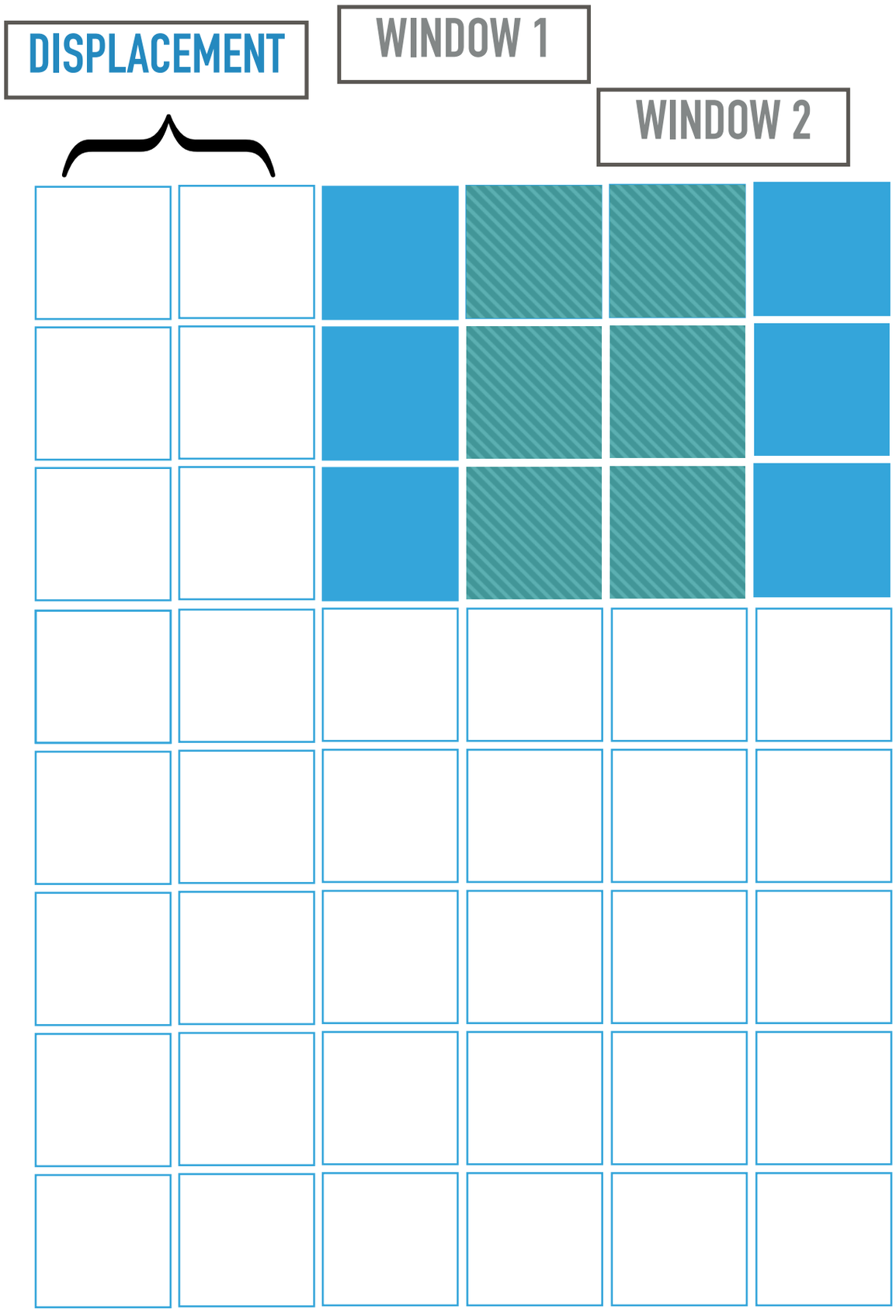

WINDOW 1
WINDOW 2


WINDOW 1
WINDOW 2


DISPLACEMENT
WINDOW 1
WINDOW 2



Horiz. Displacement = INw - Horiz. Win. Size + 1



Horiz. Displacement = INw - Horiz. Win. Size + 1

Example:

INw = Horizontal Win. Size + 1

Horiz. Displacement = 2

APPROACHES COMPARED

- ROCCC
- Vivado HLS
- Our Approach

APPROACHES COMPARED

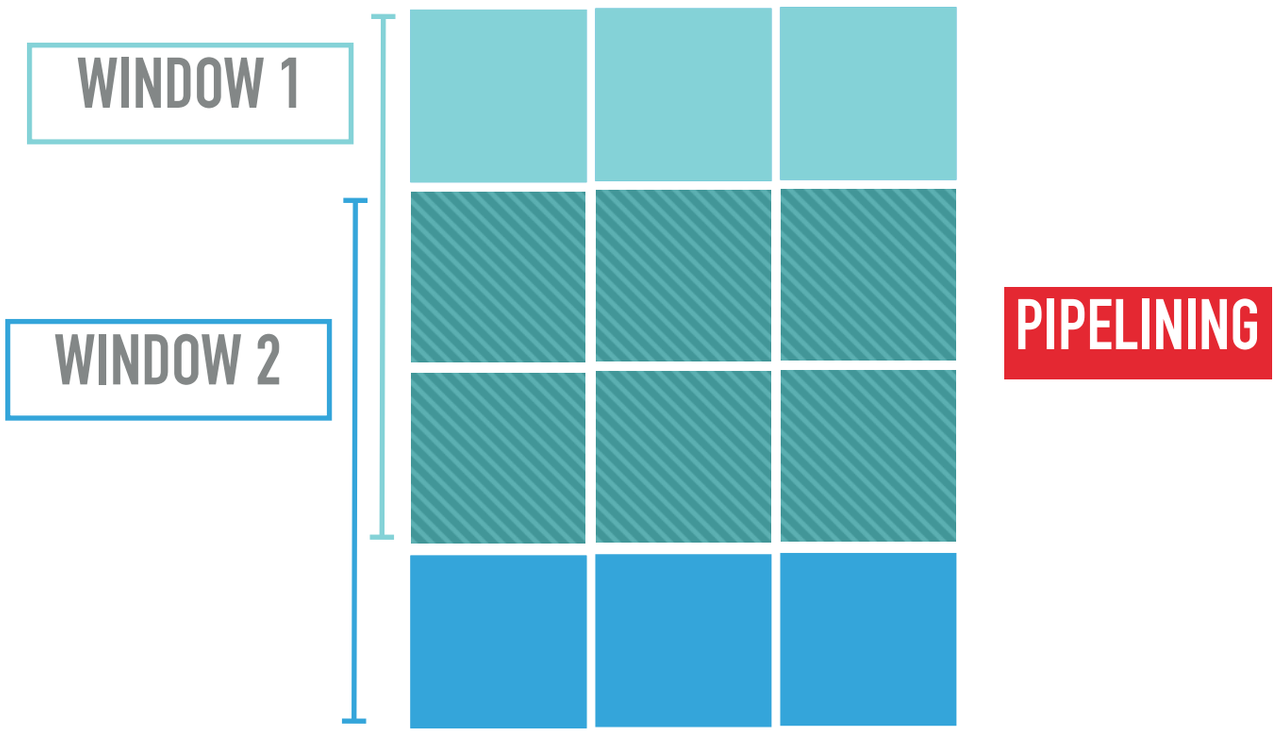
- ROCCC
- Vivado HLS
- Our Approach

PIPELINING
DATA REUSE

APPROACHES COMPARED

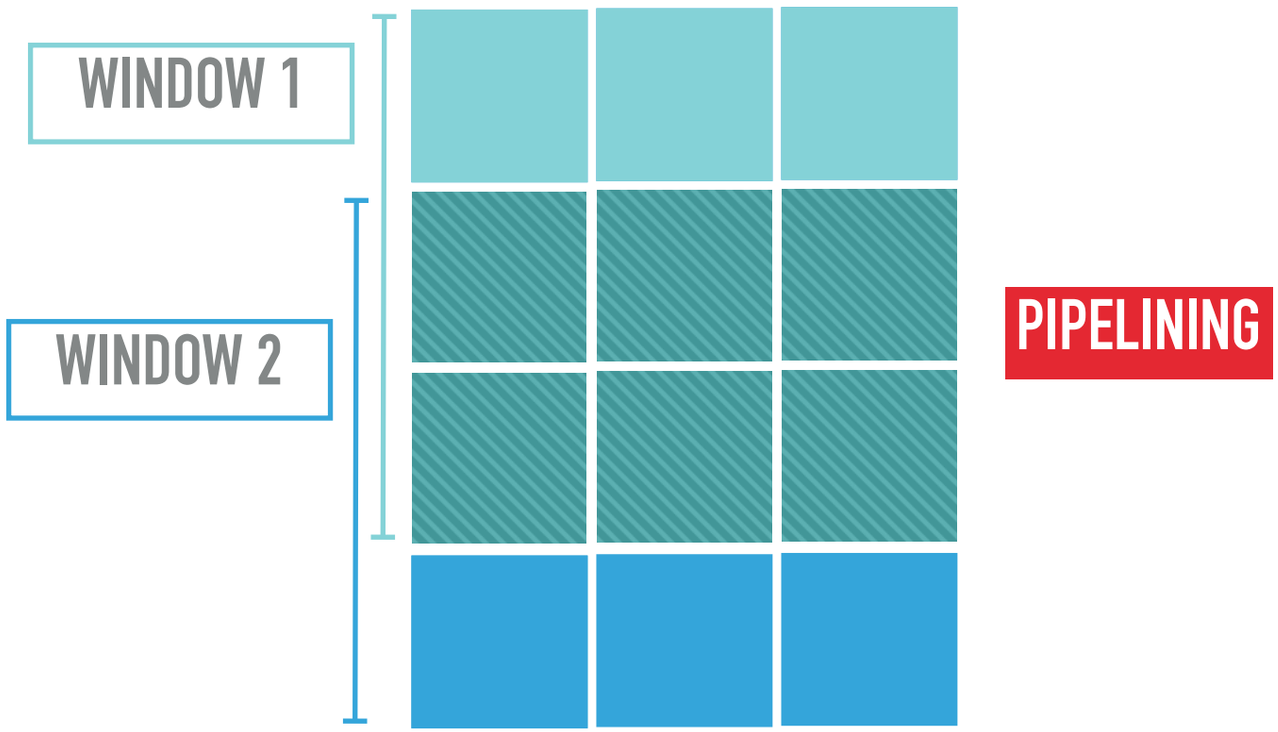
- ROCCC
- Vivado HLS
- Our Approach

PIPELINING
DATA REUSE



APPROACHES COMPARED

- ROCCC
- Vivado HLS
- Our Approach

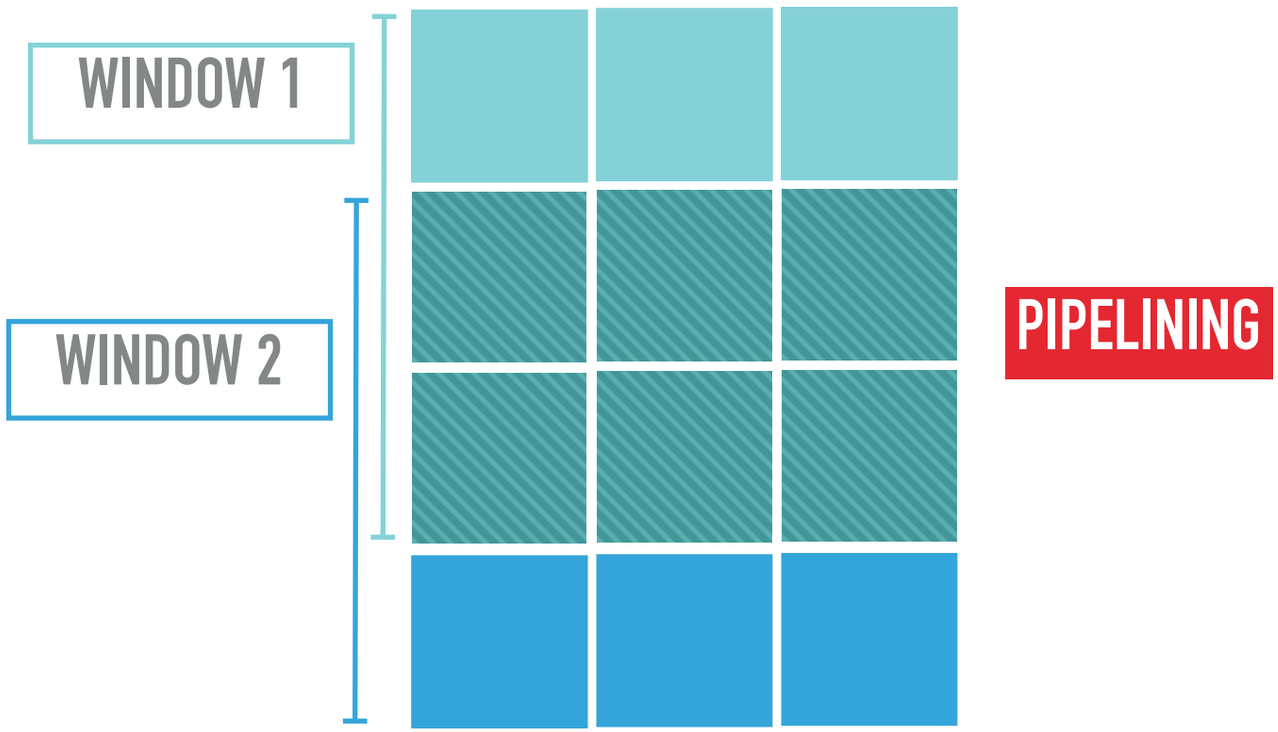


PIPELINING
DATA REUSE

ROCCC

APPROACHES COMPARED

- ROCCC
- Vivado HLS
- Our Approach



ROCCC

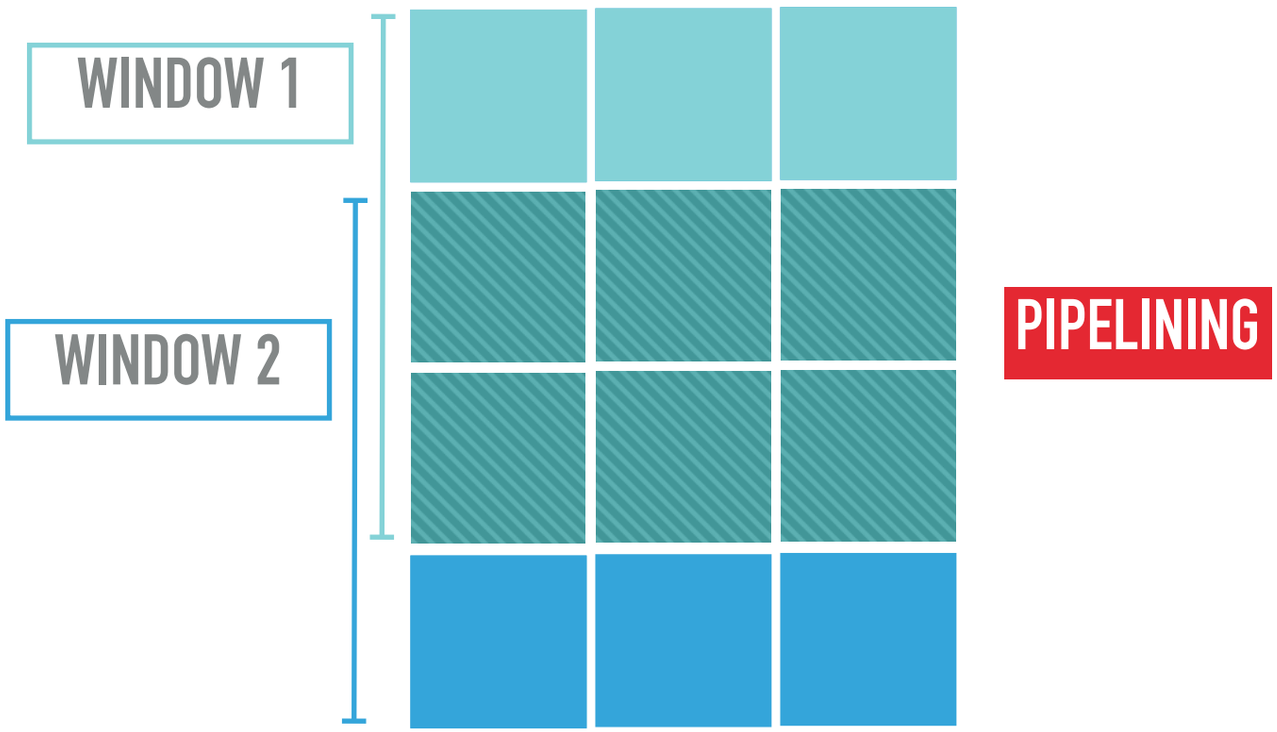
VIVADO

PIPELINING
DATA REUSE

Manual
Rewrite

APPROACHES COMPARED

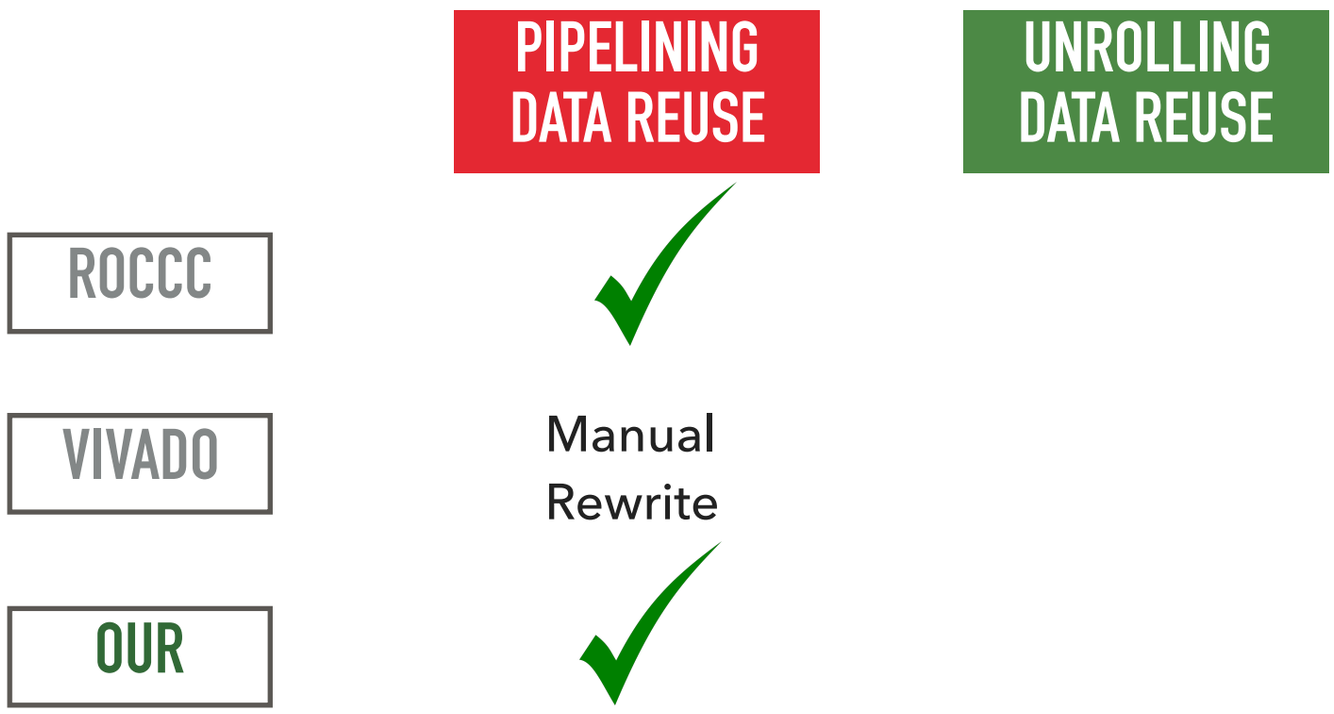
- ROCCC
- Vivado HLS
- Our Approach



	PIPELINING DATA REUSE
ROCCC	✓
VIVADO	Manual Rewrite
OUR	✓

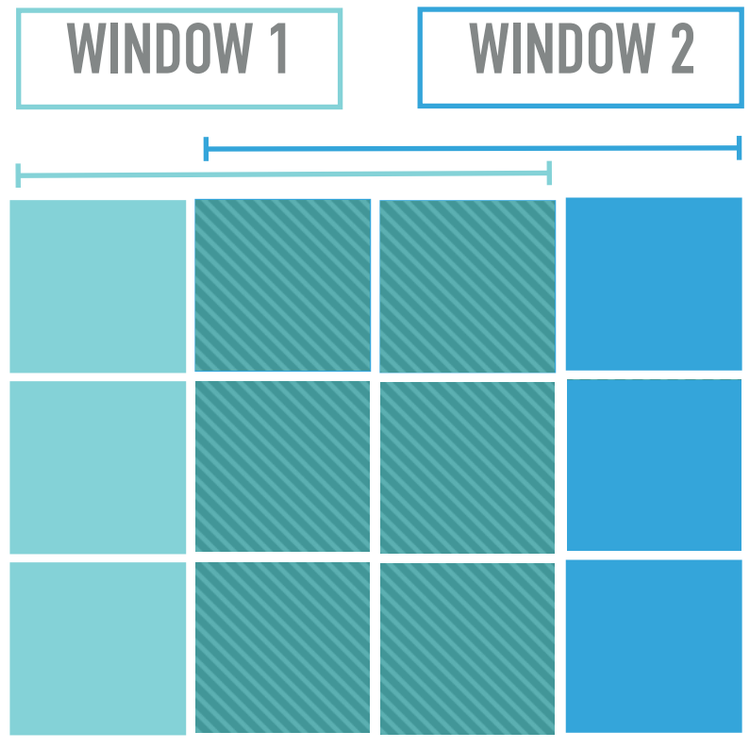
APPROACHES COMPARED

- ROCCC
- Vivado HLS
- Our Approach



APPROACHES COMPARED

- ROCCC
- Vivado HLS
- Our Approach

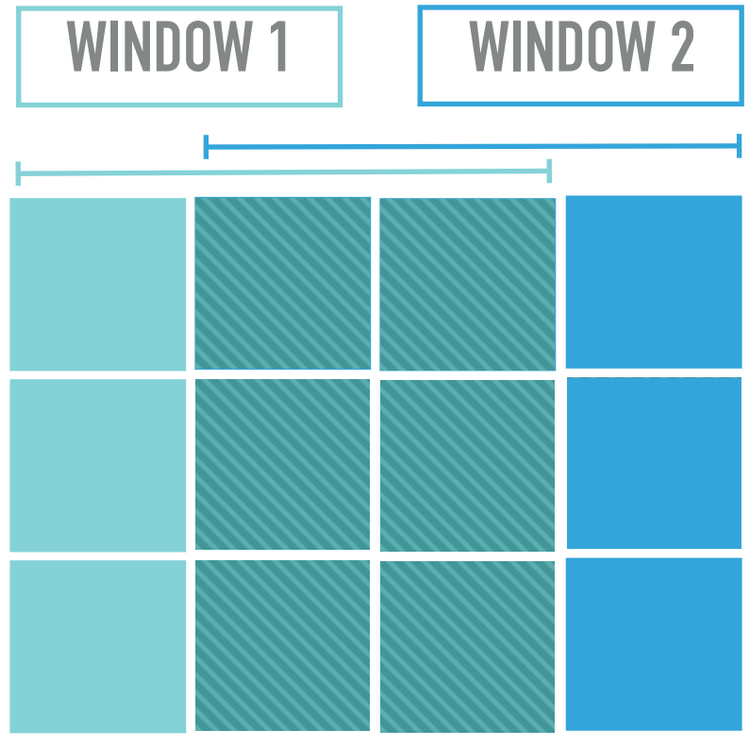


UNROLLING

	PIPELINING DATA REUSE	UNROLLING DATA REUSE
ROCCC	✓	
VIVADO	Manual Rewrite	
OUR	✓	

APPROACHES COMPARED

- ROCCC
- Vivado HLS
- Our Approach

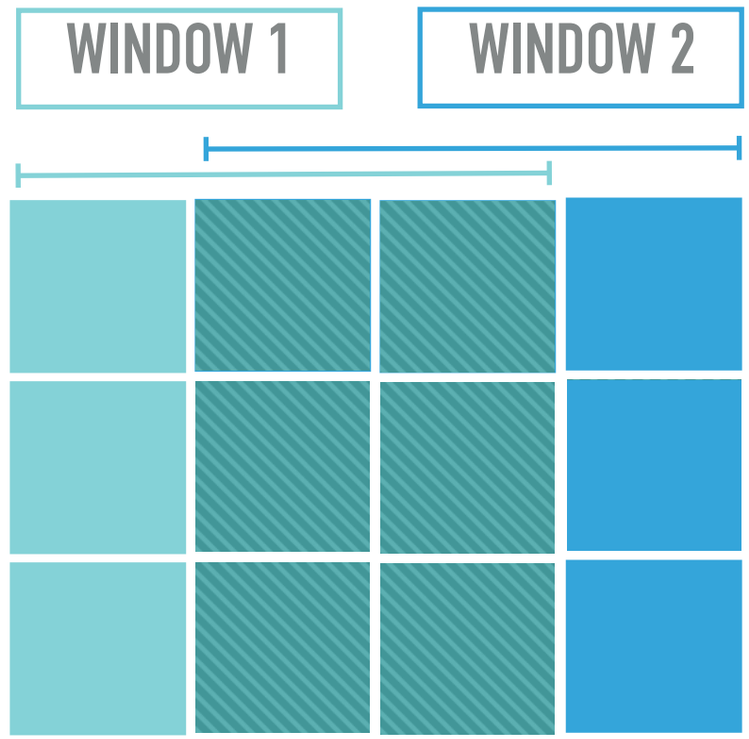


UNROLLING

	PIPELINING DATA REUSE	UNROLLING DATA REUSE
ROCCC	✓	✗
VIVADO	Manual Rewrite	
OUR	✓	

APPROACHES COMPARED

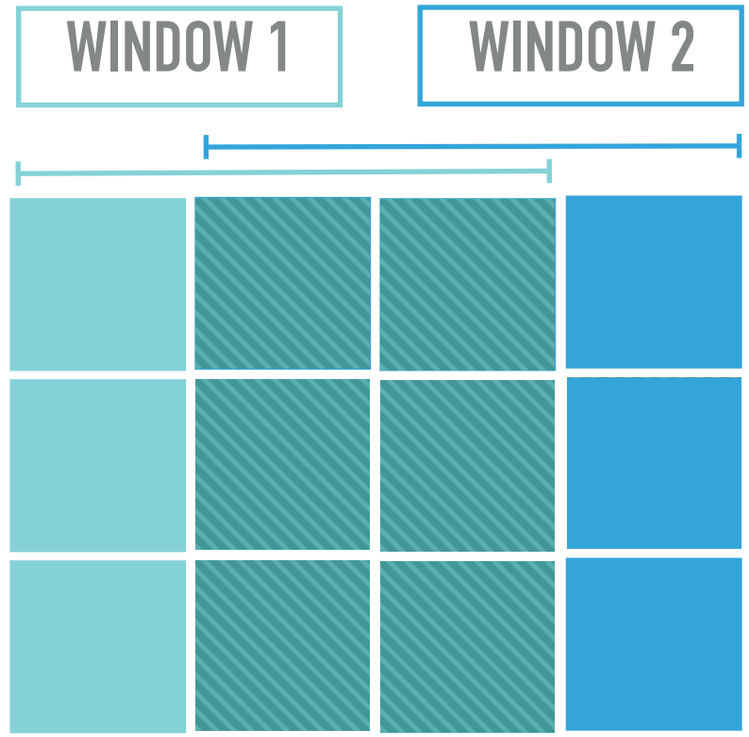
- ROCCC
- Vivado HLS
- Our Approach



	PIPELINING DATA REUSE	UNROLLING DATA REUSE
ROCCC	✓	✗
VIVADO	Manual Rewrite	✗
OUR	✓	

APPROACHES COMPARED

- ROCCC
- Vivado HLS
- Our Approach



	PIPELINING DATA REUSE	UNROLLING DATA REUSE
ROCCC	✓	✗
VIVADO	Manual Rewrite	✗
OUR	✓	✓

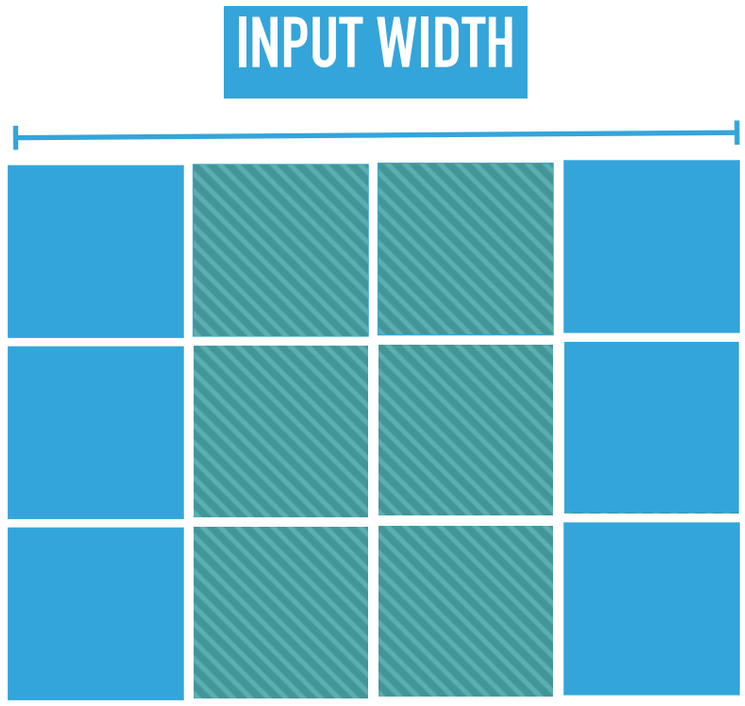
APPROACHES COMPARED

- ROCCC
- Vivado HLS
- Our Approach

	PIPELINING DATA REUSE	UNROLLING DATA REUSE	PARAMETRIC I/O WIDTH
ROCCC	✓	✗	
VIVADO	Manual Rewrite	✗	
OUR	✓	✓	

APPROACHES COMPARED

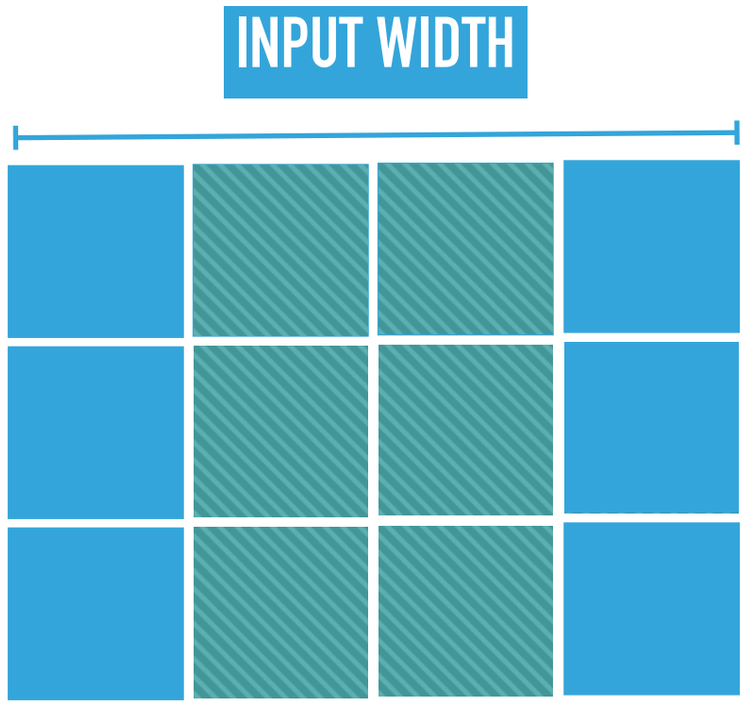
- ROCCC
- Vivado HLS
- Our Approach



	PIPELINING DATA REUSE	UNROLLING DATA REUSE	PARAMETRIC I/O WIDTH
ROCCC	✓	✗	
VIVADO	Manual Rewrite	✗	
OUR	✓	✓	

APPROACHES COMPARED

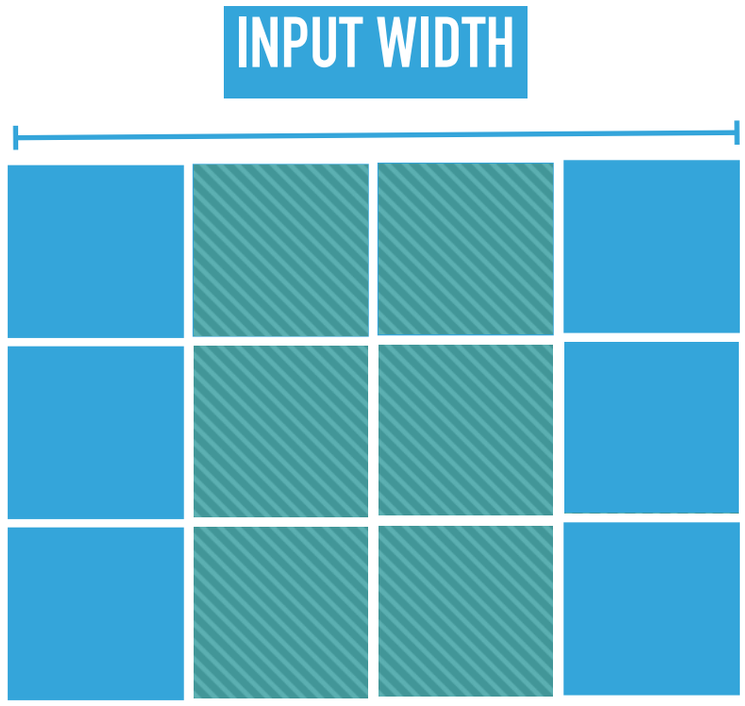
- ROCCC
- Vivado HLS
- Our Approach



	PIPELINING DATA REUSE	UNROLLING DATA REUSE	PARAMETRIC I/O WIDTH
ROCCC	✓	✗	✗
VIVADO	Manual Rewrite	✗	
OUR	✓	✓	

APPROACHES COMPARED

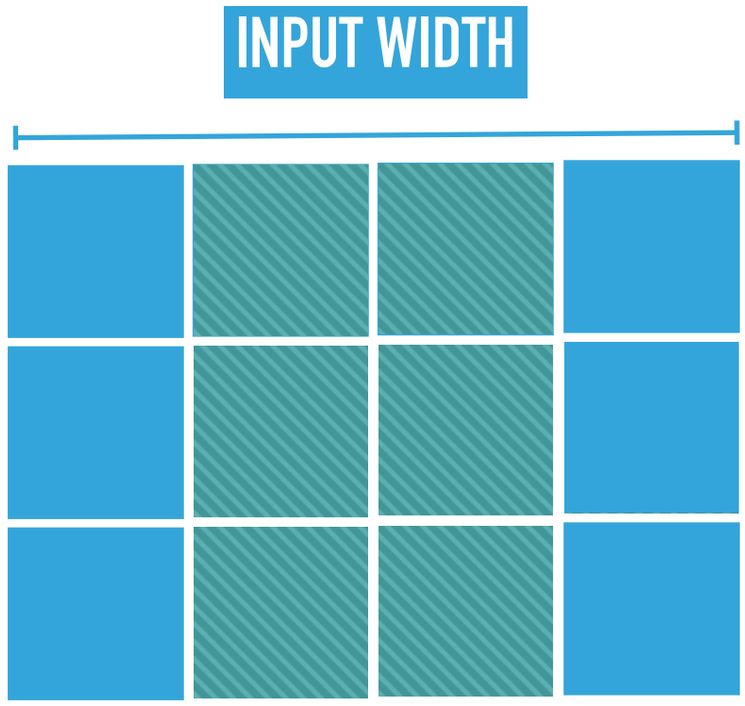
- ROCCC
- Vivado HLS
- Our Approach



	PIPELINING DATA REUSE	UNROLLING DATA REUSE	PARAMETRIC I/O WIDTH
ROCCC	✓	✗	✗
VIVADO	Manual Rewrite	✗	✓
OUR	✓	✓	

APPROACHES COMPARED

- ROCCC
- Vivado HLS
- Our Approach



	PIPELINING DATA REUSE	UNROLLING DATA REUSE	PARAMETRIC I/O WIDTH
ROCCC	✓	✗	✗
VIVADO	Manual Rewrite	✗	✓
OUR	✓	✓	✓

OUR APPROACH

- Xilinx Virtex7 FPGA (HDL Simulation)

OUR APPROACH

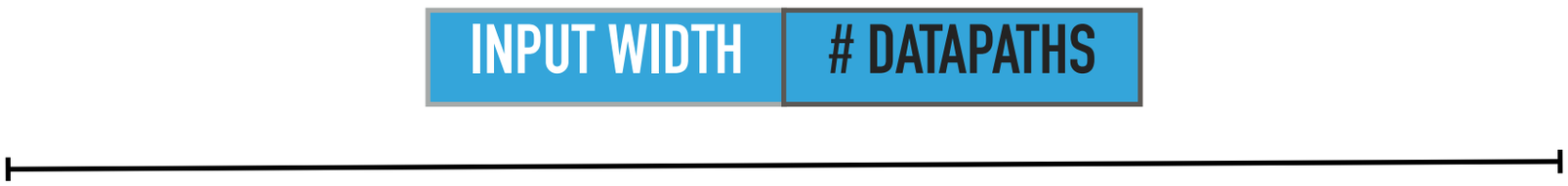
- Xilinx Virtex7 FPGA (HDL Simulation)

3 CONFIGURATIONS

OUR APPROACH

- Xilinx Virtex7 FPGA (HDL Simulation)

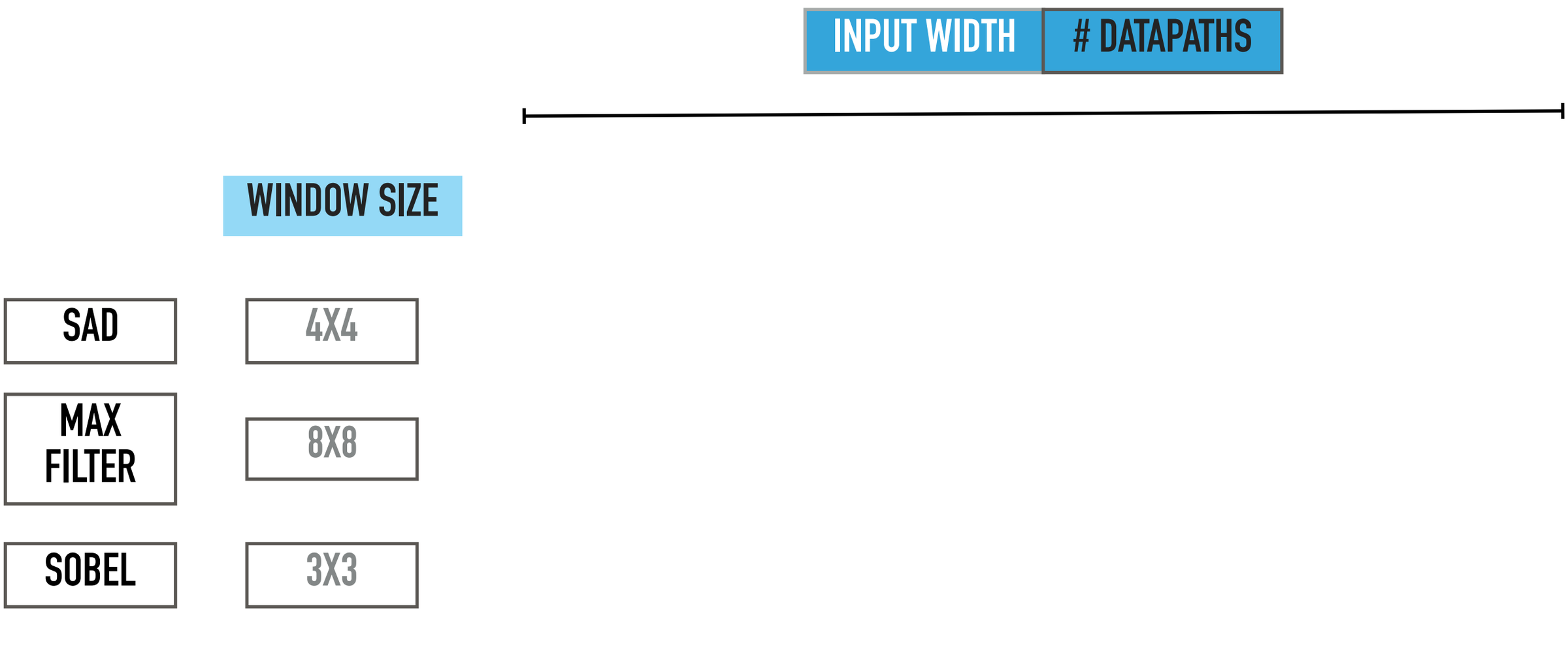
3 CONFIGURATIONS



OUR APPROACH

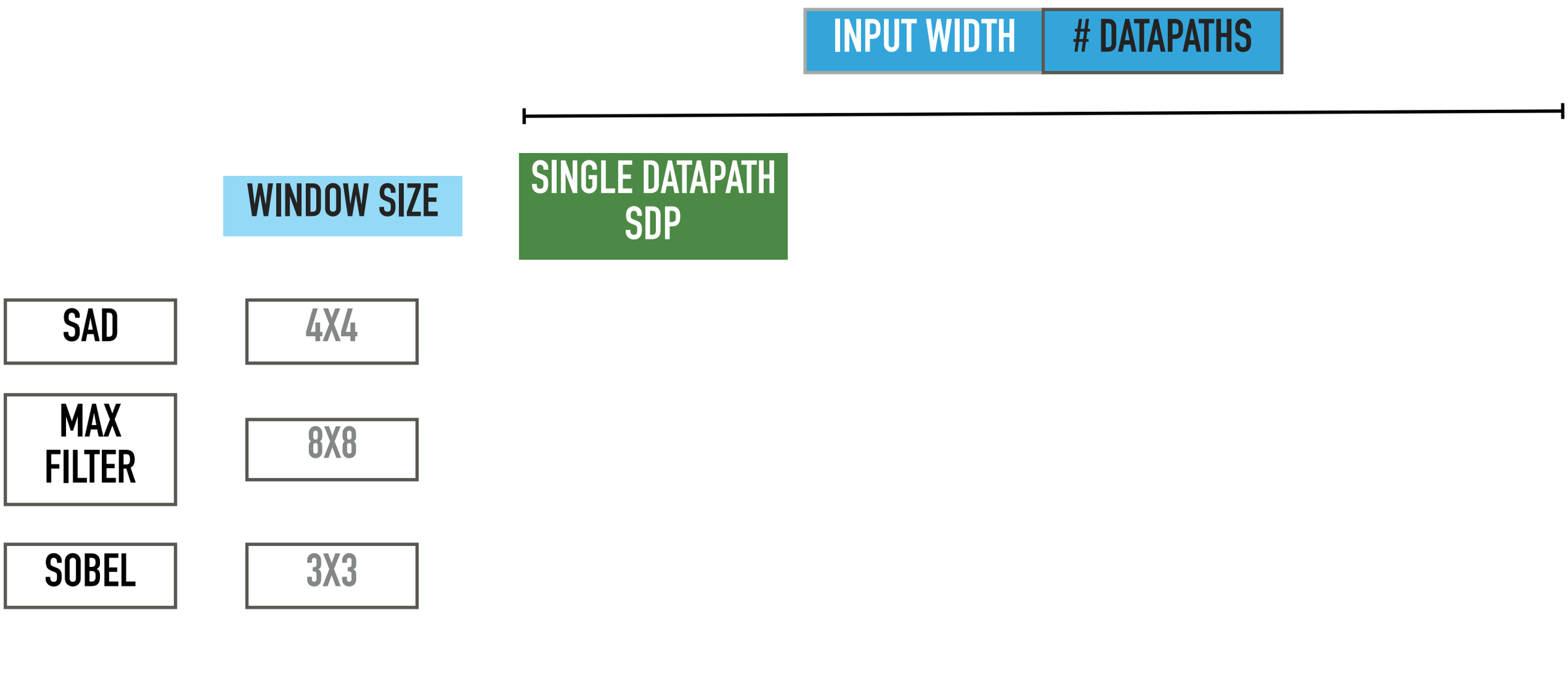
- Xilinx Virtex7 FPGA (HDL Simulation)

3 CONFIGURATIONS



OUR APPROACH

- Xilinx Virtex7 FPGA (HDL Simulation)



OUR APPROACH

- Xilinx Virtex7 FPGA (HDL Simulation)

	WINDOW SIZE	SINGLE DATAPATH SDP
SAD	4X4	4, 1
MAX FILTER	8X8	8, 1
SOBEL	3X3	3, 1

INPUT WIDTH → # DATAPATHS

OUR APPROACH

- Xilinx Virtex7 FPGA (HDL Simulation)

	WINDOW SIZE	SINGLE DATAPATH SDP	MULTIPLE DATAPATHS MDP
		INPUT WIDTH, # DATAPATHS	
SAD	4X4	4, 1	
MAX FILTER	8X8	8, 1	
SOBEL	3X3	3, 1	

OUR APPROACH

- Xilinx Virtex7 FPGA (HDL Simulation)

	WINDOW SIZE	SINGLE DATAPATH SDP	MULTIPLE DATAPATHS MDP
		INPUT WIDTH, # DATAPATHS	
SAD	4X4	4, 1	8, 5
MAX FILTER	8X8	8, 1	16, 9
SOBEL	3X3	3, 1	8, 6

OUR APPROACH

- Xilinx Virtex7 FPGA (HDL Simulation)

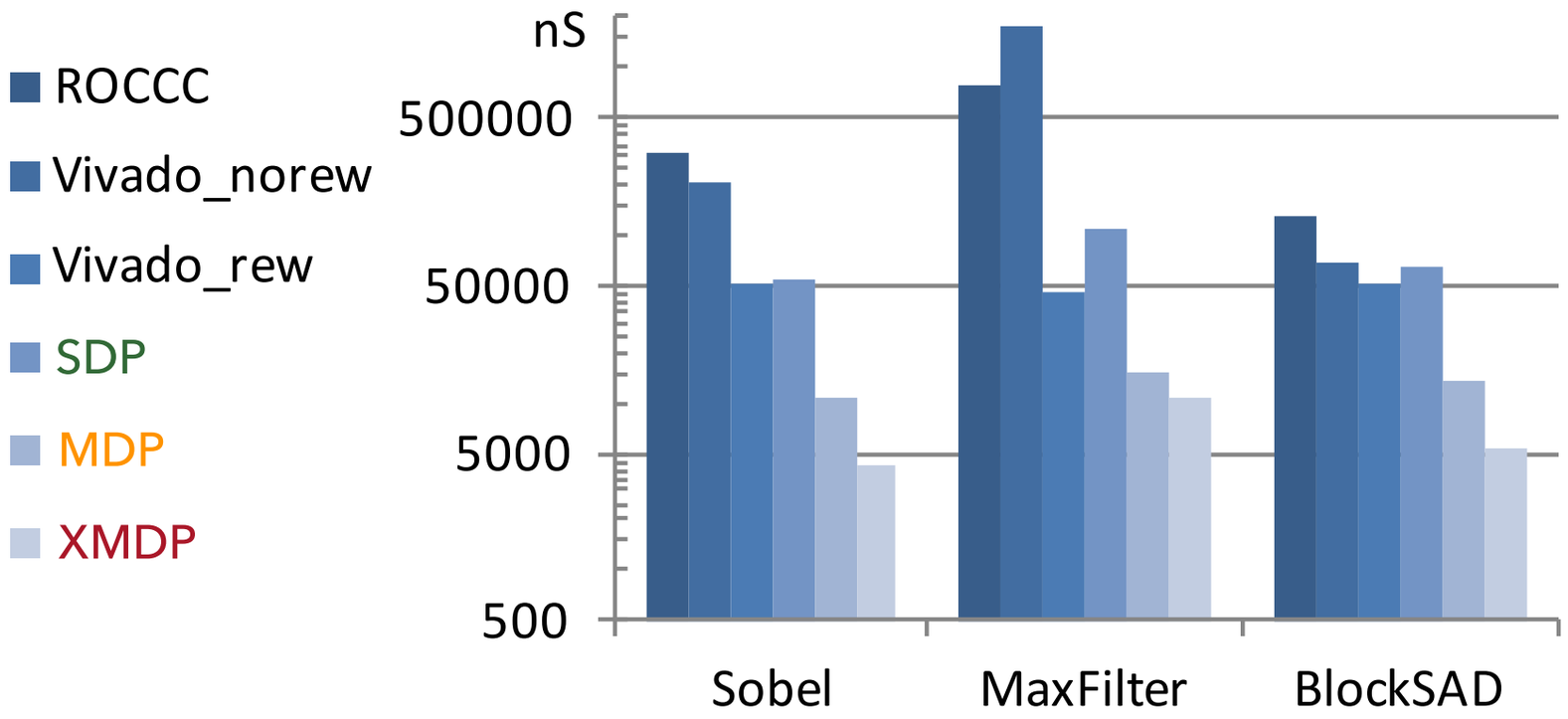
	INPUT WIDTH	# DATAPATHS	
	WINDOW SIZE	SINGLE DATAPATH SDP	MULTIPLE DATAPATHS MDP	EXTRA MULT. DATAPATHS XMDP
SAD	4X4	4, 1	8, 5	
MAX FILTER	8X8	8, 1	16, 9	
SOBEL	3X3	3, 1	8, 6	

OUR APPROACH

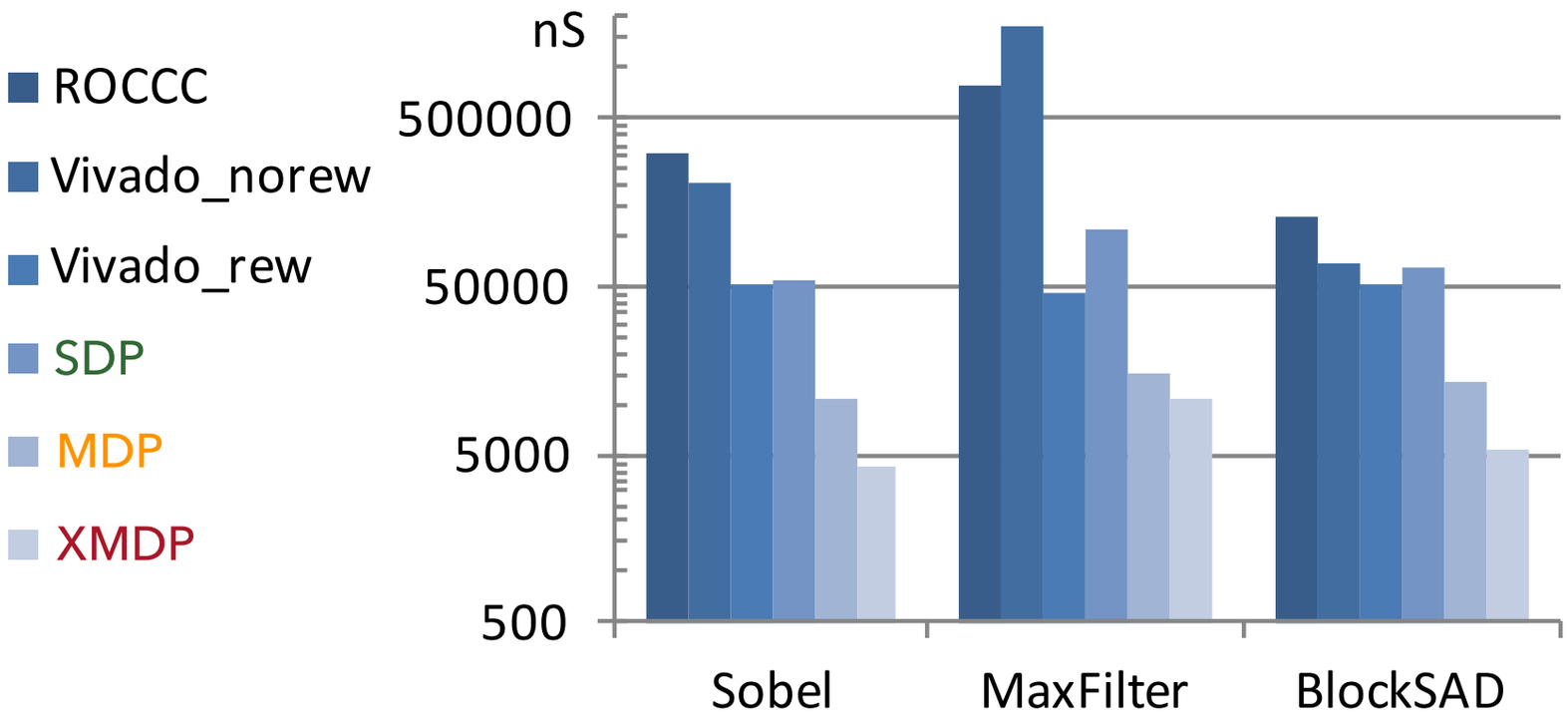
- Xilinx Virtex7 FPGA (HDL Simulation)

		INPUT WIDTH, # DATAPATHS		
	WINDOW SIZE	SINGLE DATAPATH SDP	MULTIPLE DATAPATHS MDP	EXTRA MULT. DATAPATHS XMDP
SAD	4X4	4, 1	8, 5	16, 13
MAX FILTER	8X8	8, 1	16, 9	24, 17
SOBEL	3X3	3, 1	8, 6	16, 14

EXECUTION TIME COMPARISON

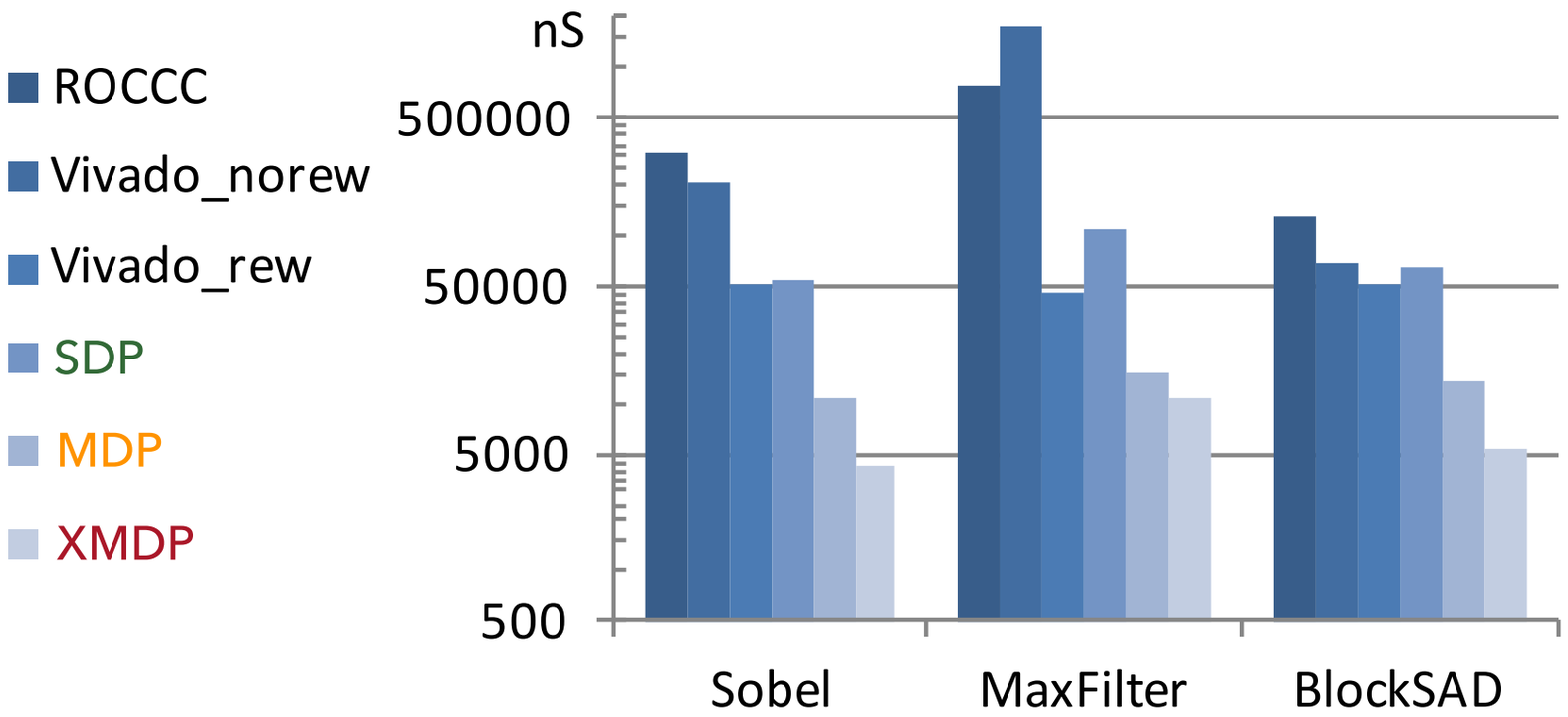


EXECUTION TIME COMPARISON



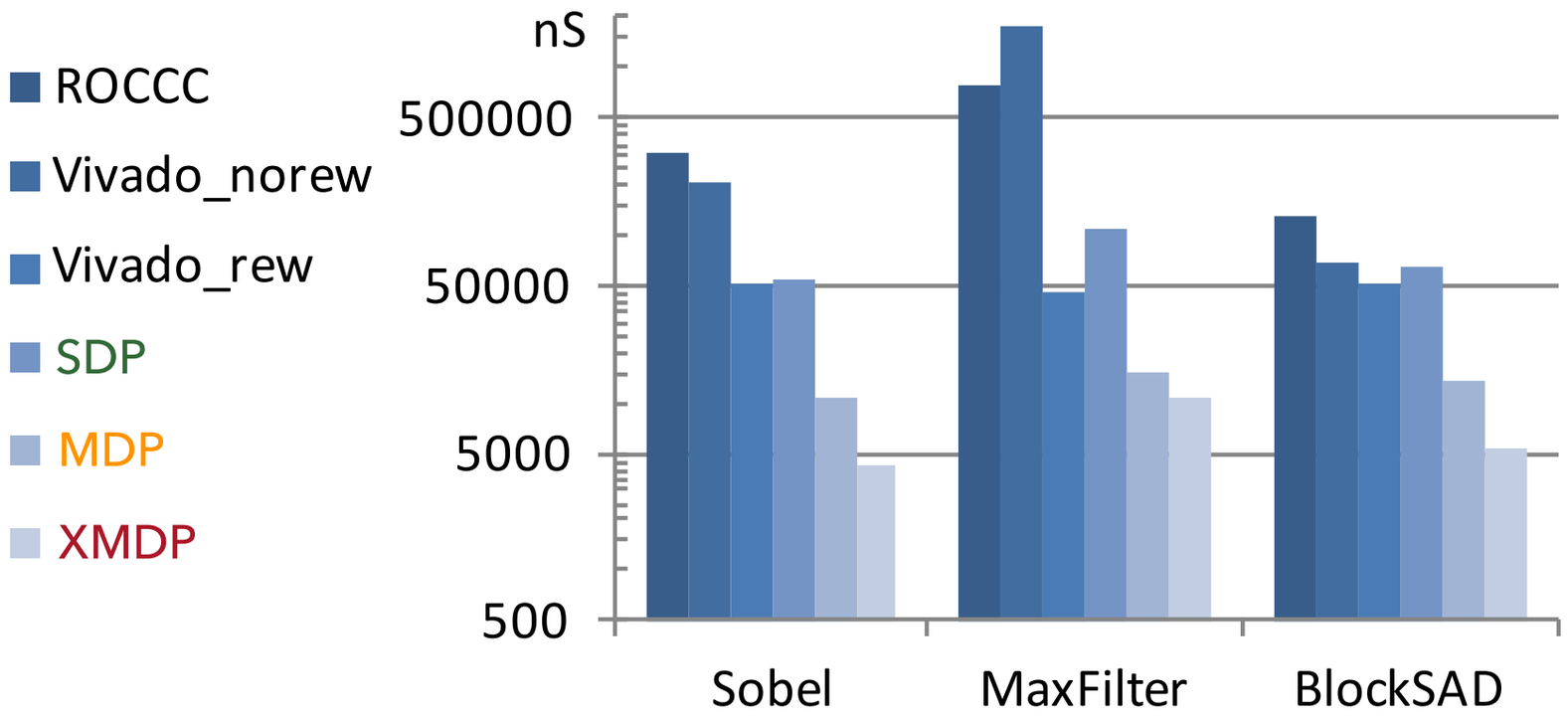
	PIPELINING DATA REUSE	UNROLLING DATA REUSE	PARAMETRIC I/O WIDTH
VIVADO_NOREW	✗	✗	✓

EXECUTION TIME COMPARISON



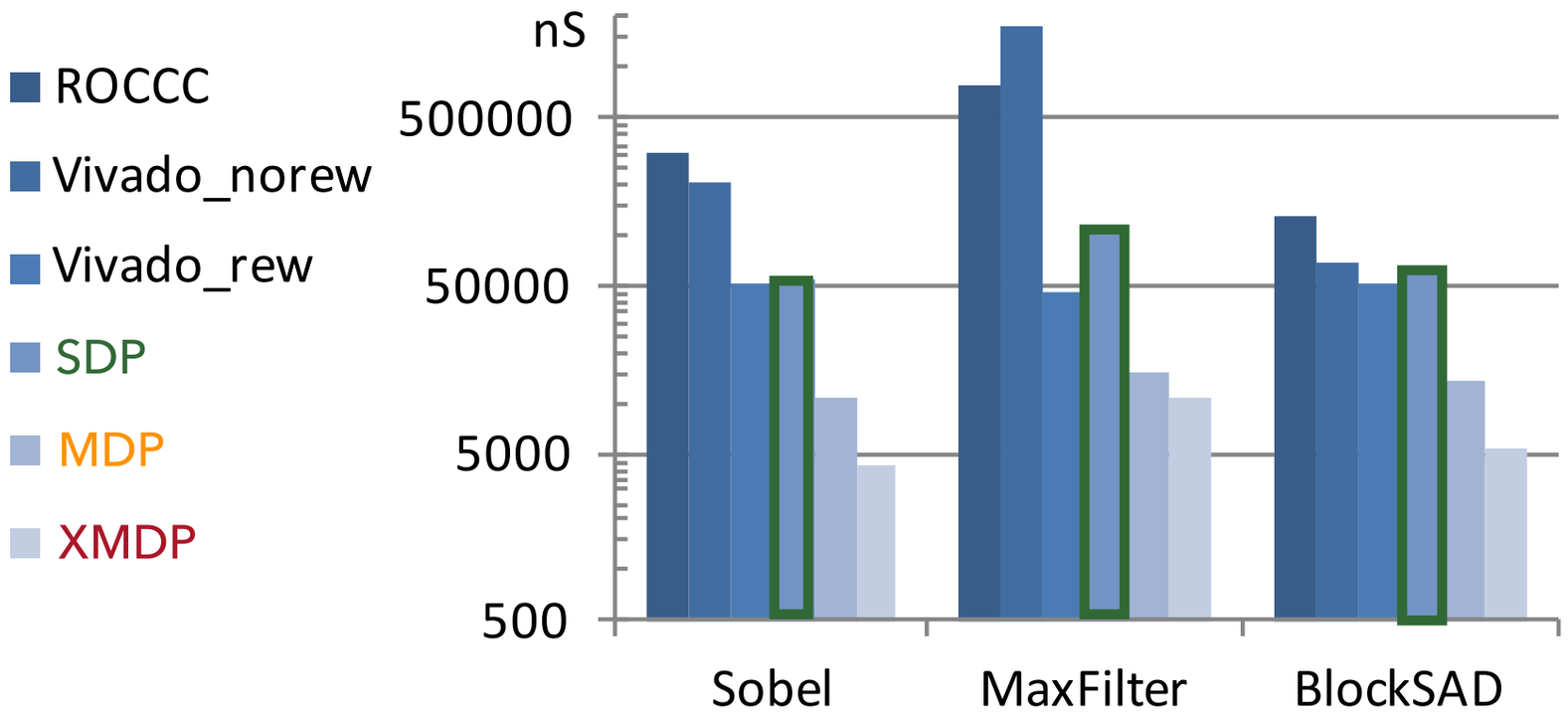
	PIPELINING DATA REUSE	UNROLLING DATA REUSE	PARAMETRIC I/O WIDTH
VIVADO_REW	Manual Rewrite	✗	✓

EXECUTION TIME COMPARISON



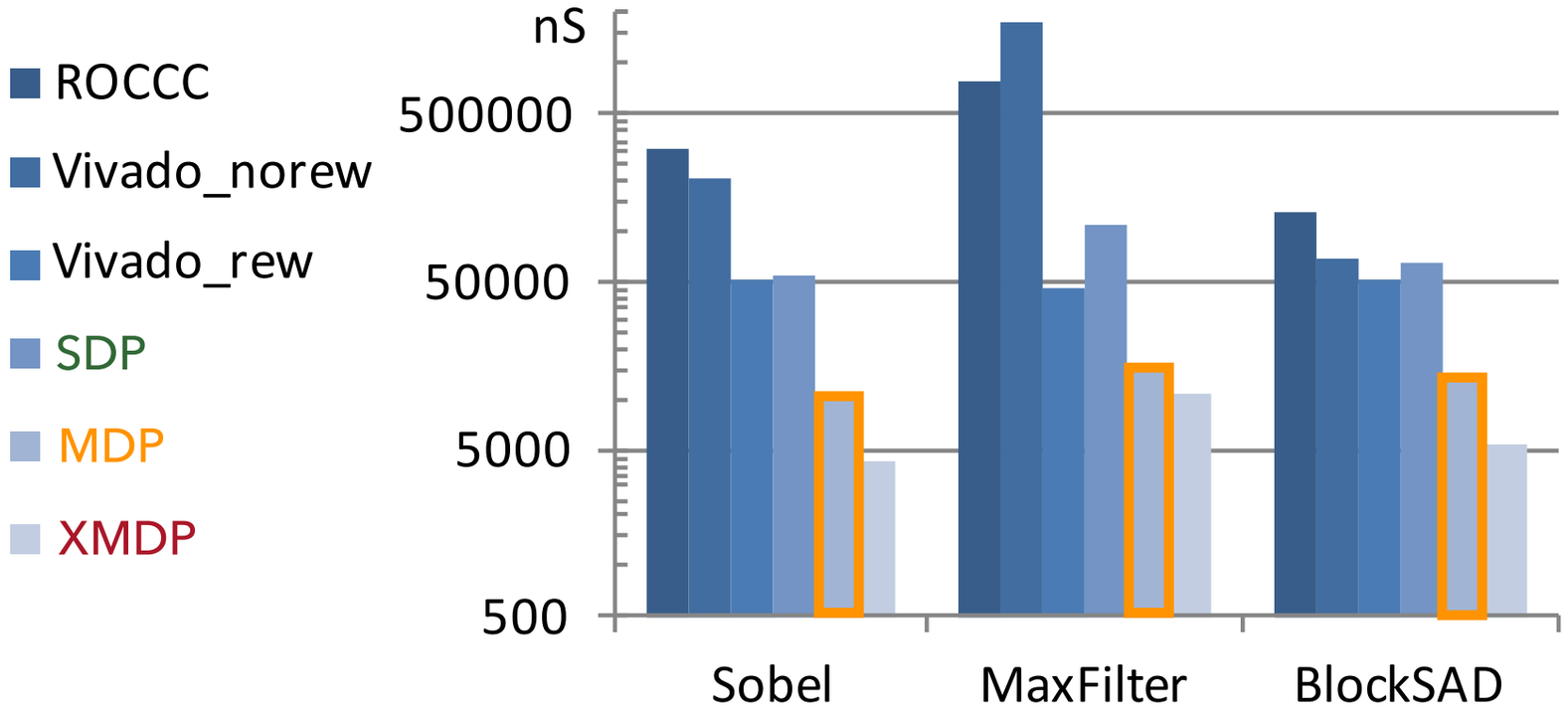
- Exec. Time to process a 100x100 frame

EXECUTION TIME COMPARISON



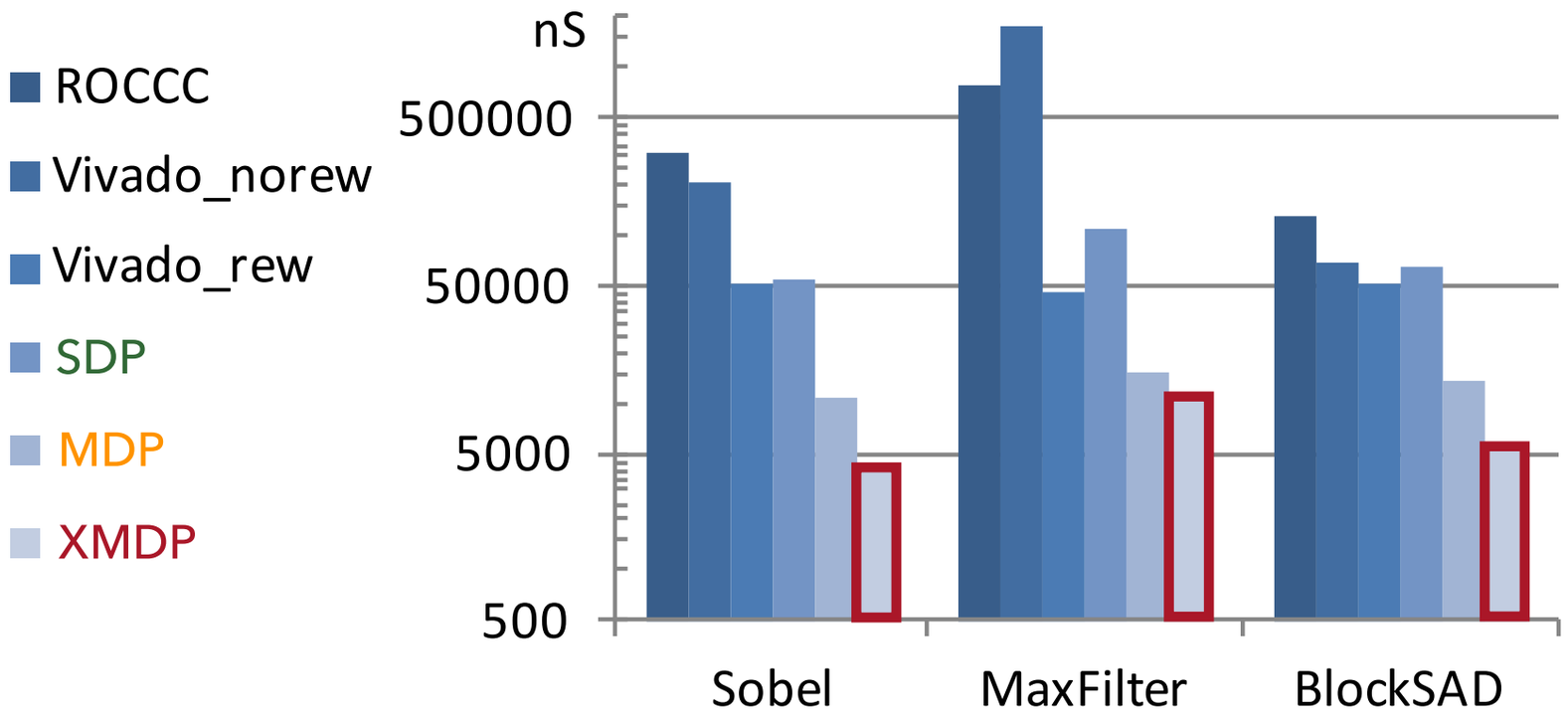
- Exec. Time to process a 100x100 frame

EXECUTION TIME COMPARISON



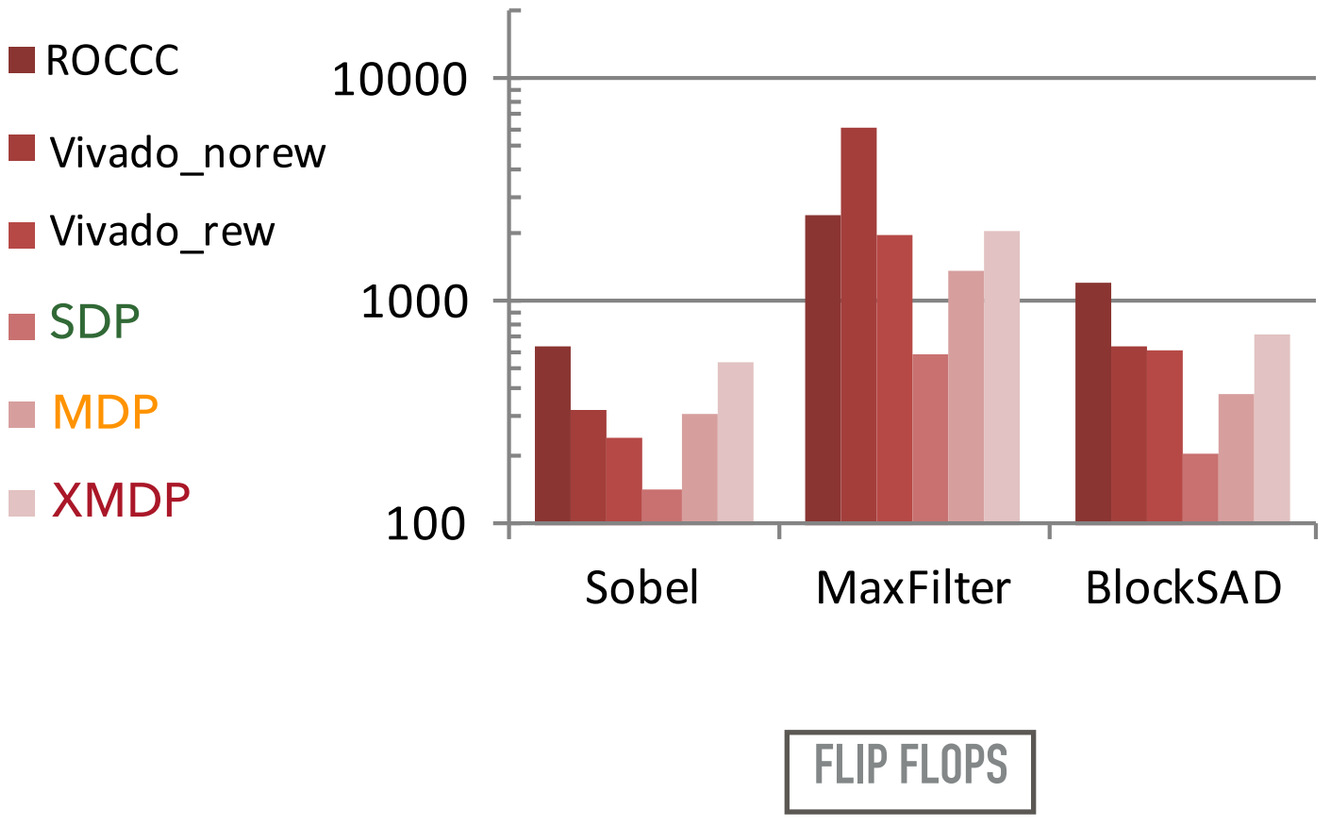
- Exec. Time to process a 100x100 frame

EXECUTION TIME COMPARISON

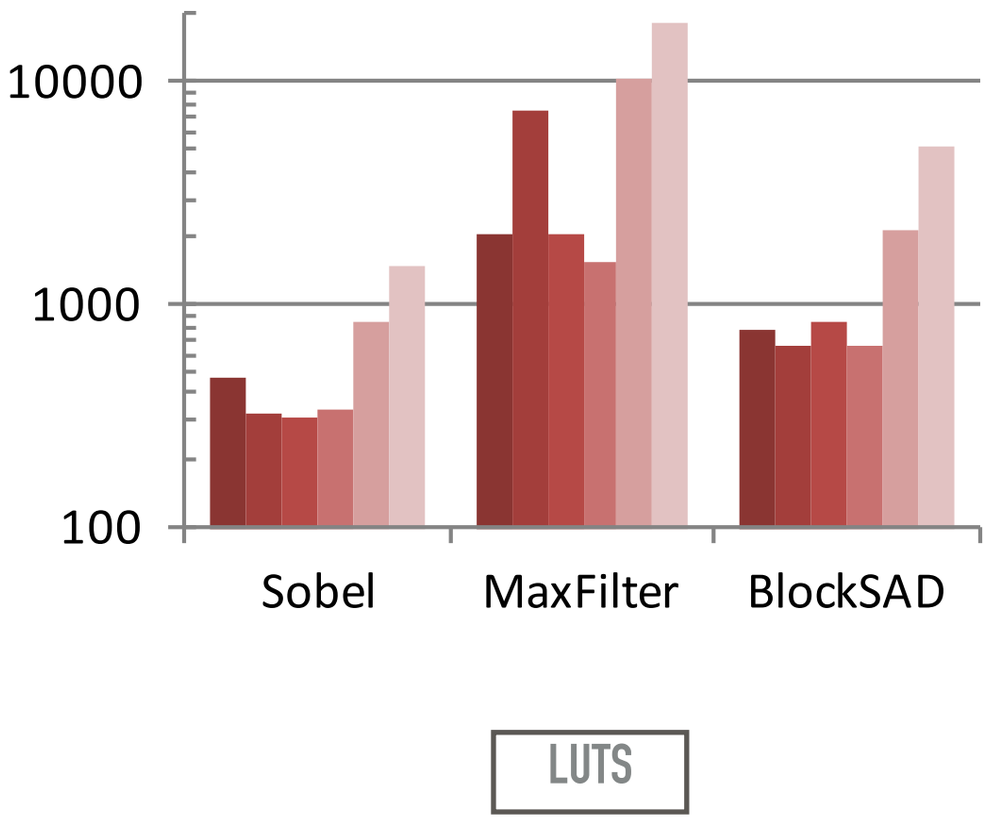


- Exec. Time to process a 100x100 frame

AREA COMPARISON

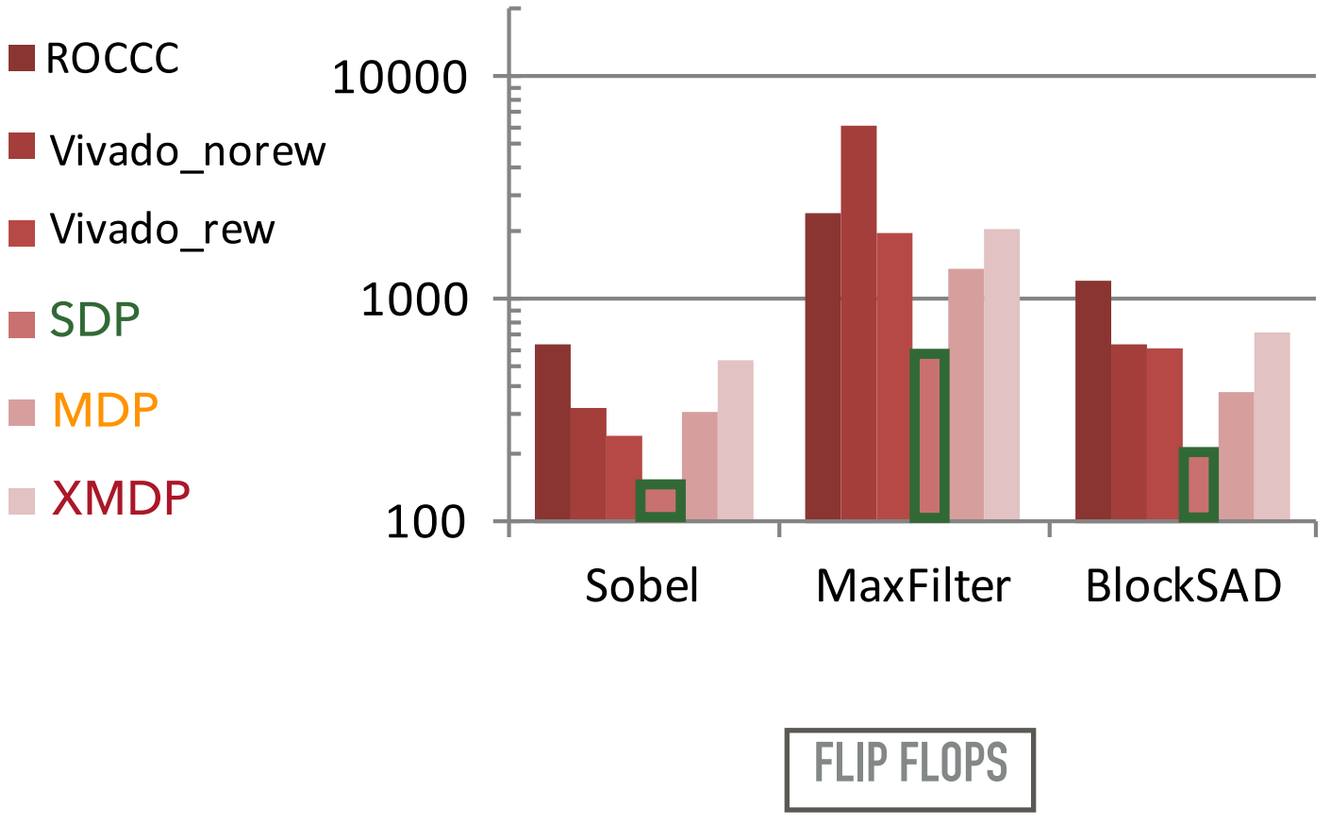


FLIP FLOPS

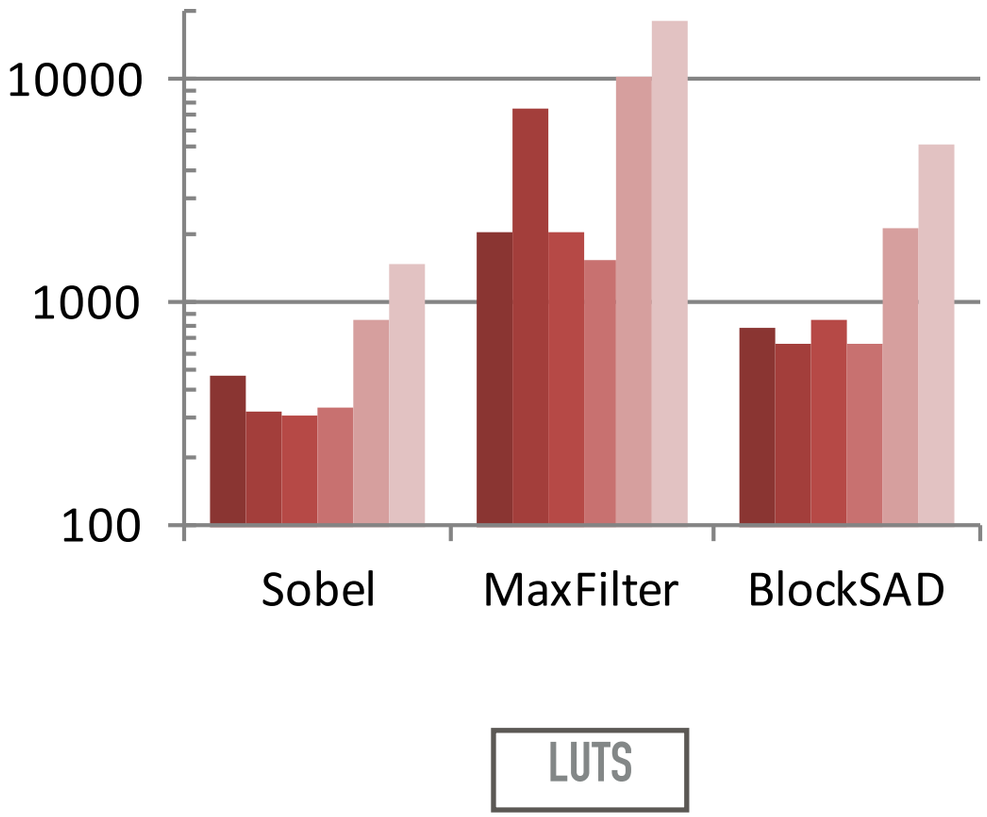


LUTS

AREA COMPARISON

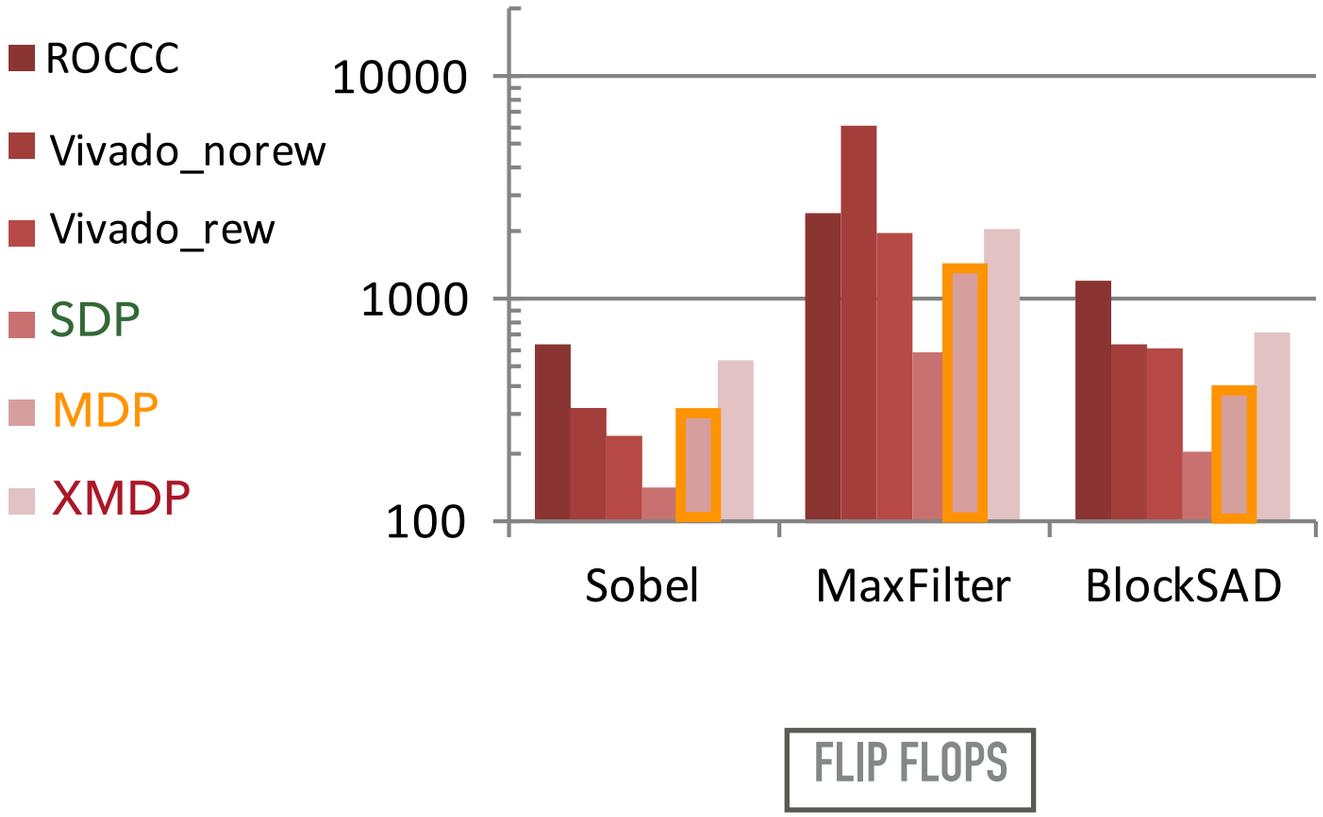


FLIP FLOPS

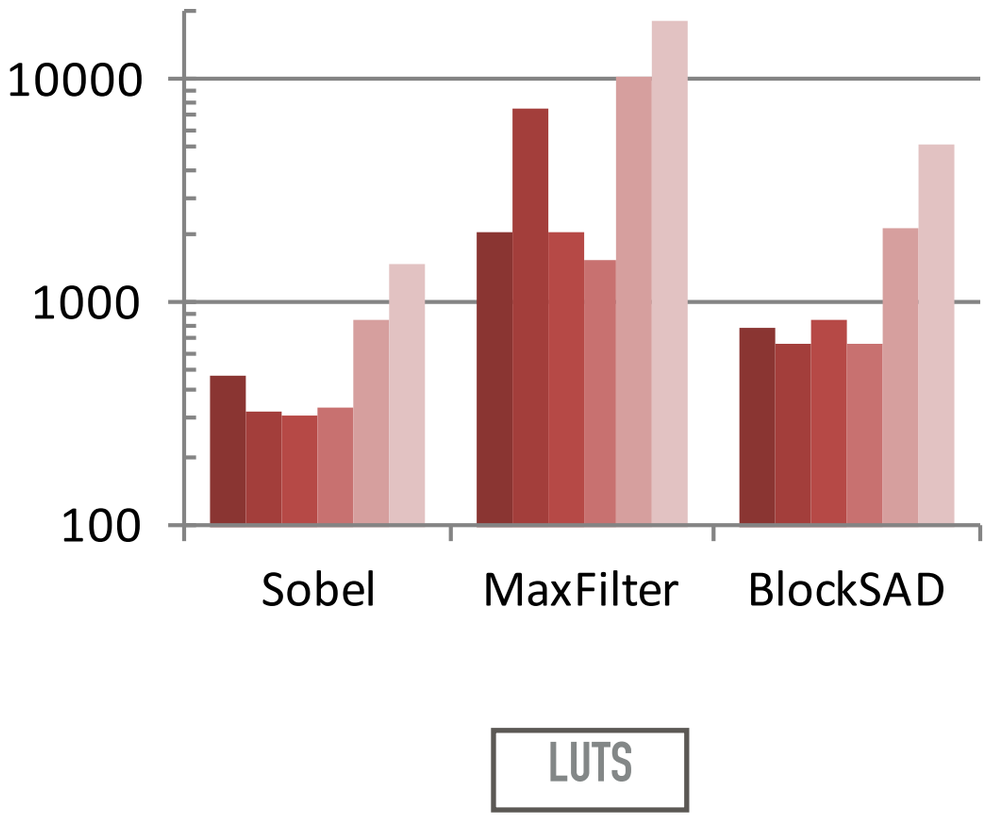


LUTS

AREA COMPARISON

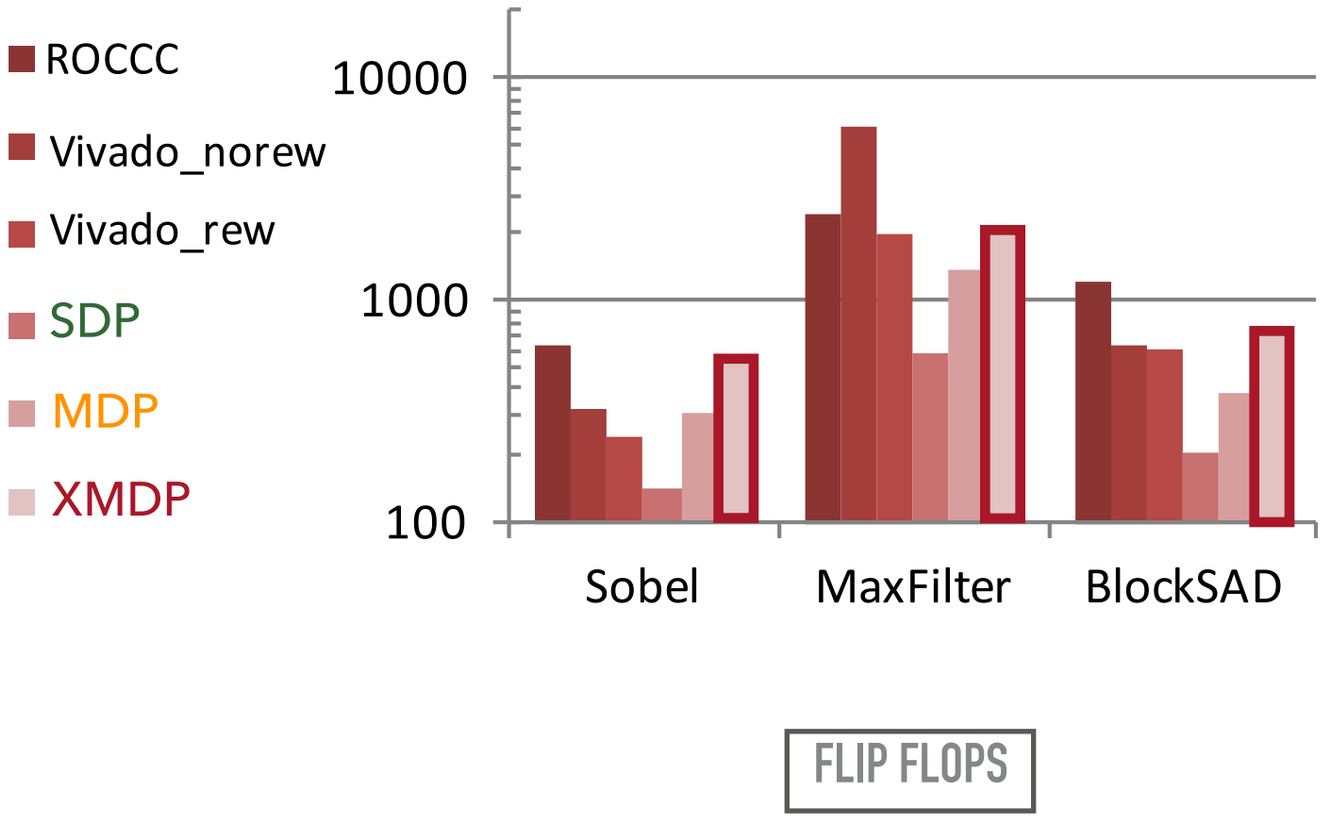


FLIP FLOPS

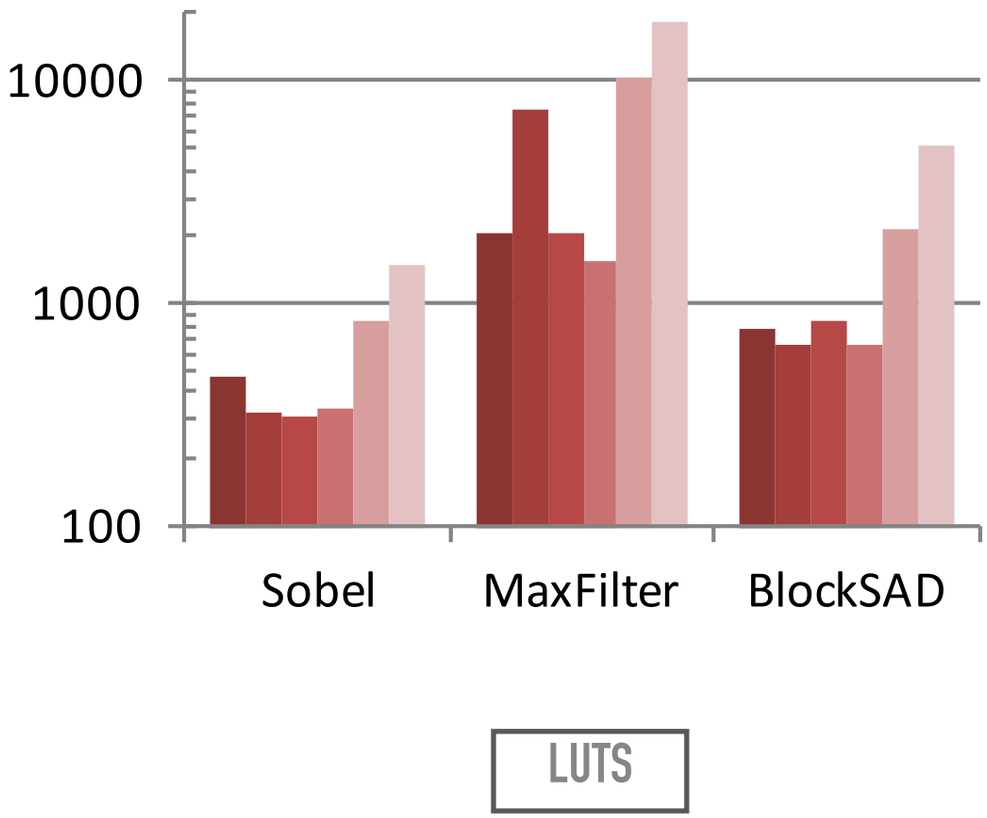


LUTS

AREA COMPARISON

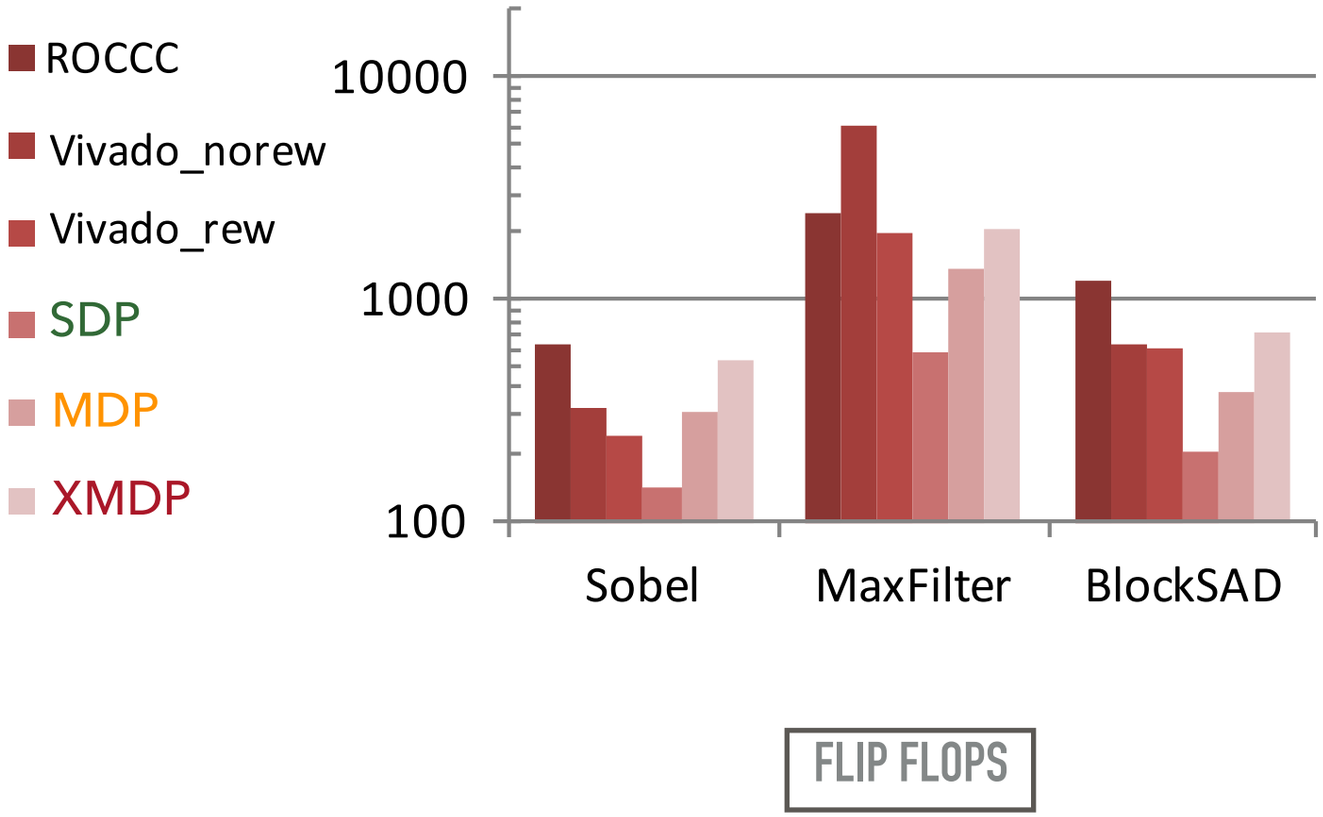


FLIP FLOPS

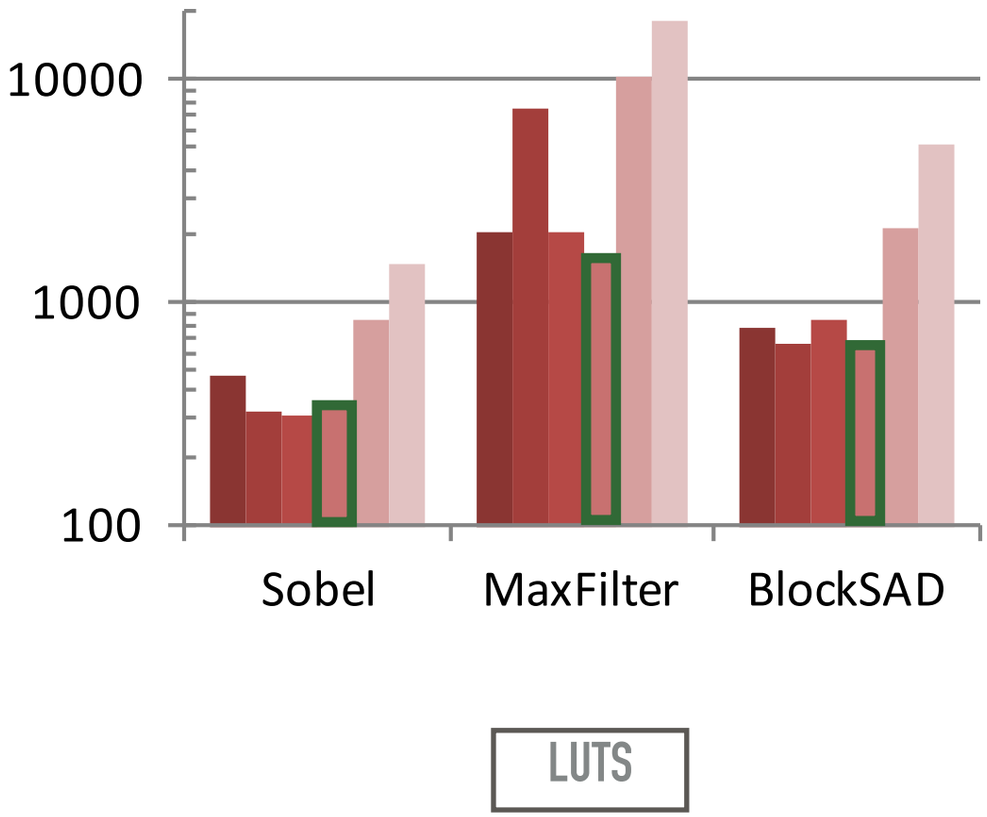


LUTS

AREA COMPARISON

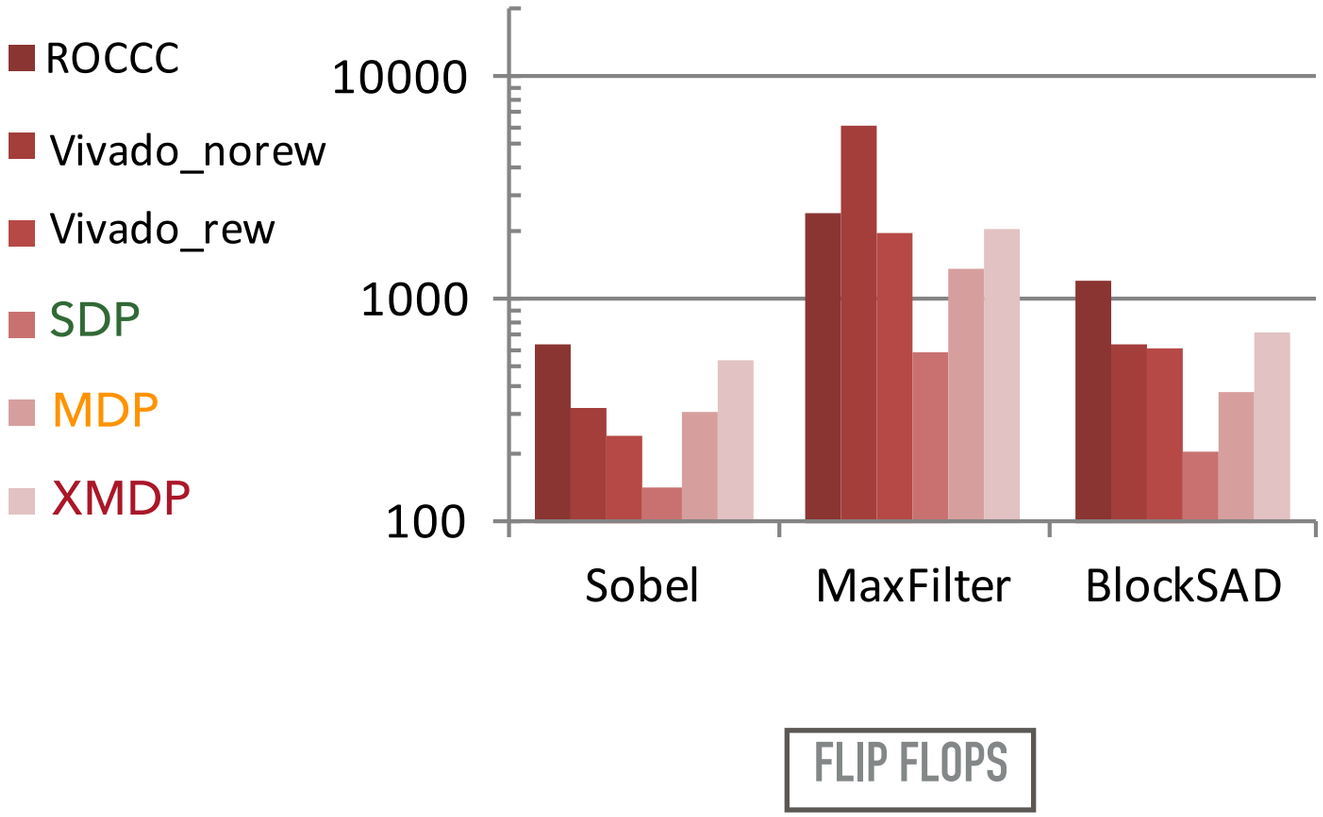


FLIP FLOPS

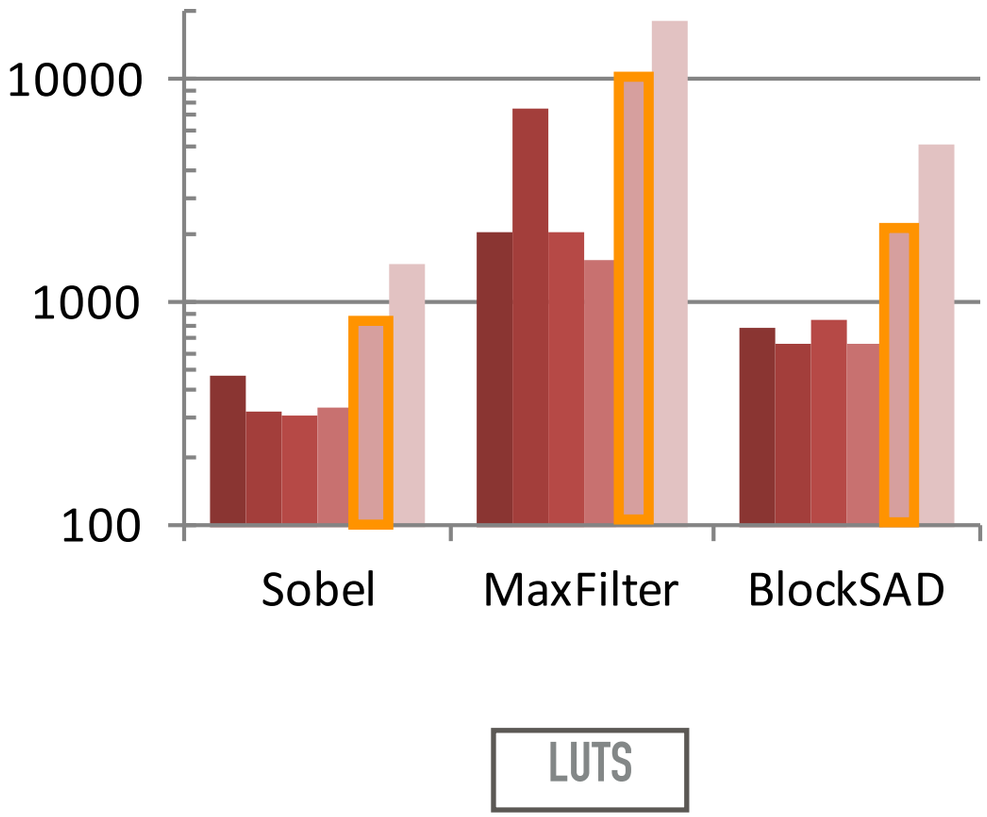


LUTS

AREA COMPARISON

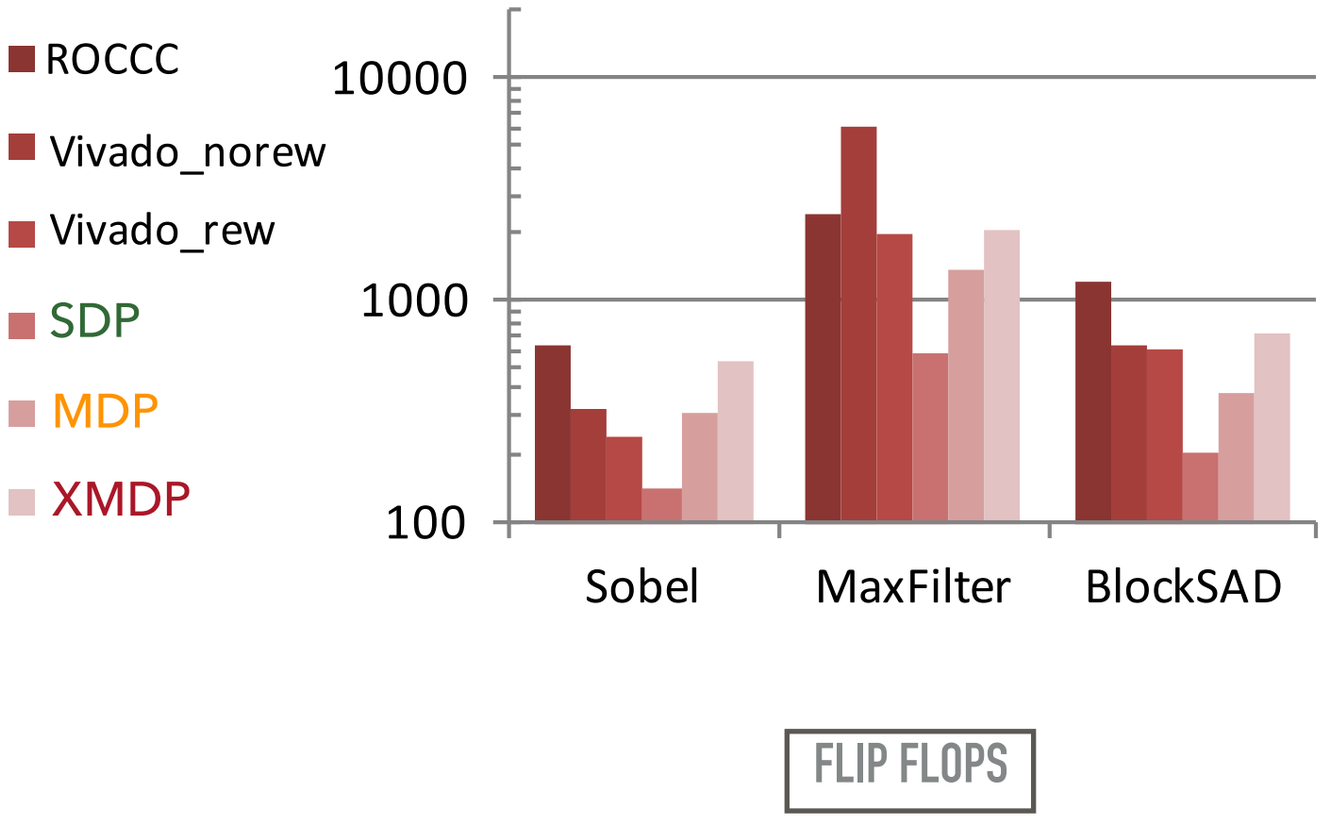


FLIP FLOPS

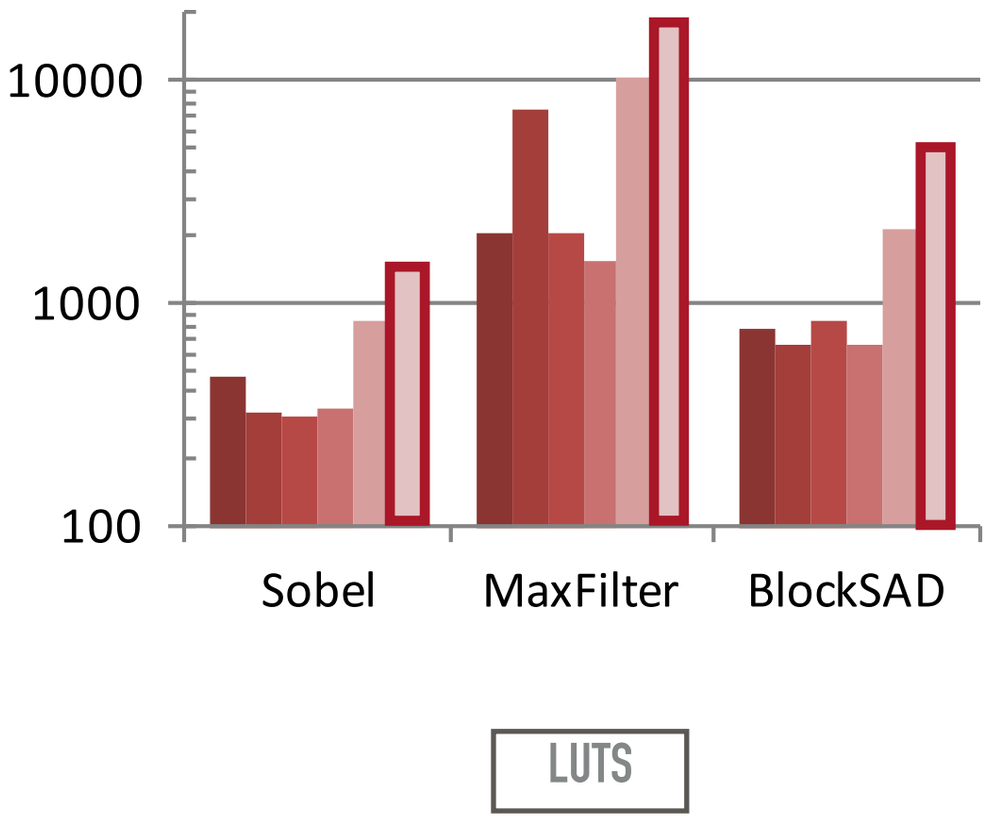


LUTS

AREA COMPARISON

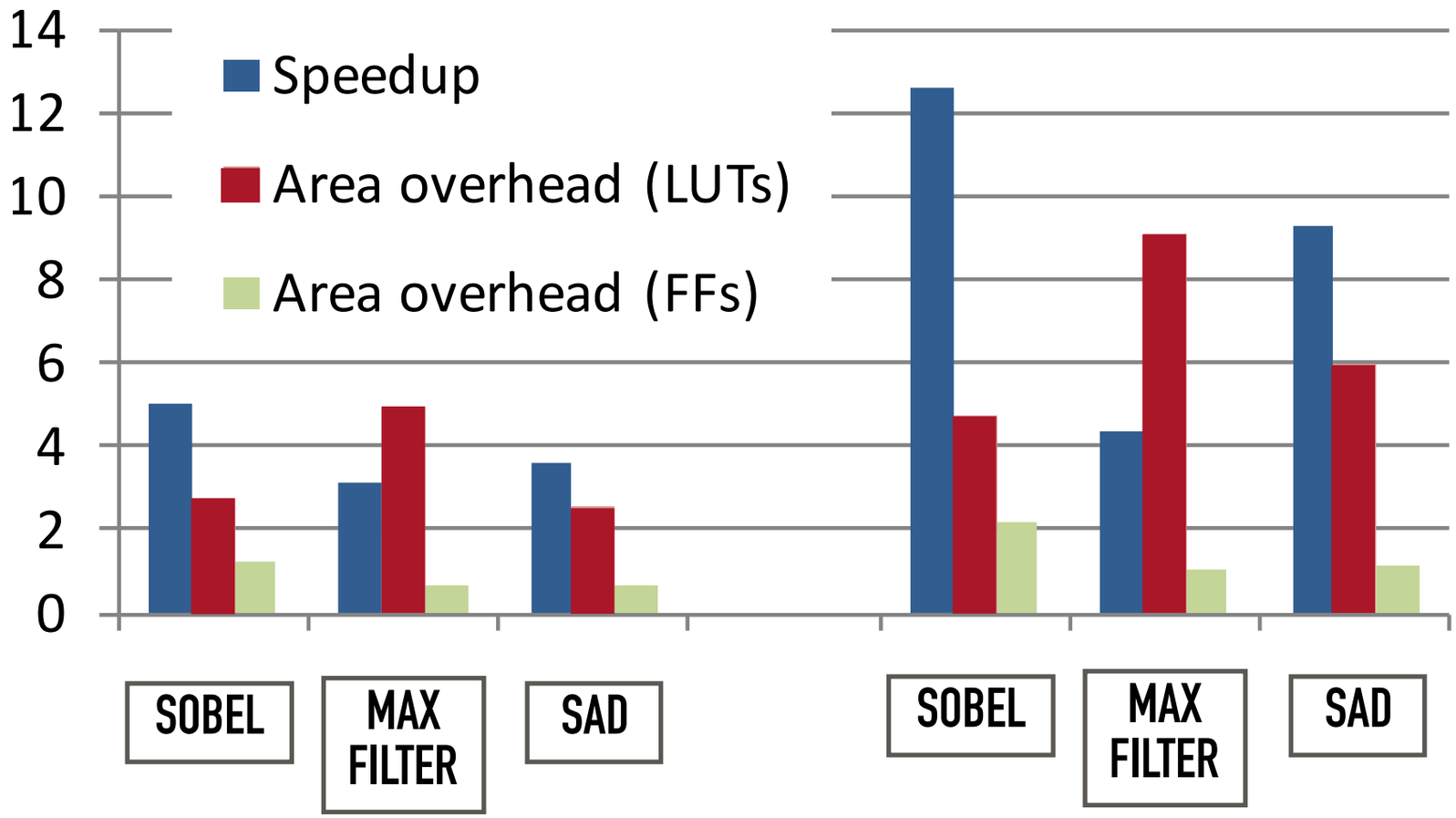


FLIP FLOPS



LUTS

SPEEDUP AND AREA



MDP vs Vivado_Rew

XMDP vs Vivado_Rew

DATA REUSE ANALYSIS BASED ON POLLY

Optimize HLS of Accelerators for Sliding Window Applications

Use Polly for Better HW Designs

DATA REUSE ANALYSIS BASED ON POLLY

Optimize HLS of Accelerators for Sliding Window Applications

Use Polly for Better HW Designs

- Exploit Data Locality

DATA REUSE ANALYSIS BASED ON POLLY

Optimize HLS of Accelerators for Sliding Window Applications

Use Polly for Better HW Designs

- Exploit Data Locality
- Design Efficient Buffers

DATA REUSE ANALYSIS BASED ON POLLY

Optimize HLS of Accelerators for Sliding Window Applications

Use Polly for Better HW Designs

- Exploit Data Locality
- Design Efficient Buffers
- Better Use of Resources

DATA REUSE ANALYSIS BASED ON POLLY

Optimize HLS of Accelerators for Sliding Window Applications

Use Polly for Better HW Designs

- Exploit Data Locality
- Design Efficient Buffers
- Better Use of Resources
- Better Performance

DATA REUSE ANALYSIS BASED ON POLLY

Optimize HLS of Accelerators for Sliding Window Applications

Use Polly for Better HW Designs

- Exploit Data Locality
- Design Efficient Buffers
- Better Use of Resources
- Better Performance

THANK YOU FOR YOUR ATTENTION!