STRUCTURAL AGNOSTIC SPMV: ADAPTING CSR-ADAPTIVE FOR IRREGULAR MATRICES

MAYANK DAGA AND JOSEPH L. GREATHOUSE
AMD RESEARCH
ADVANCED MICRO DEVICES, INC.

THIS TALK IN ONE SLIDE

AMD

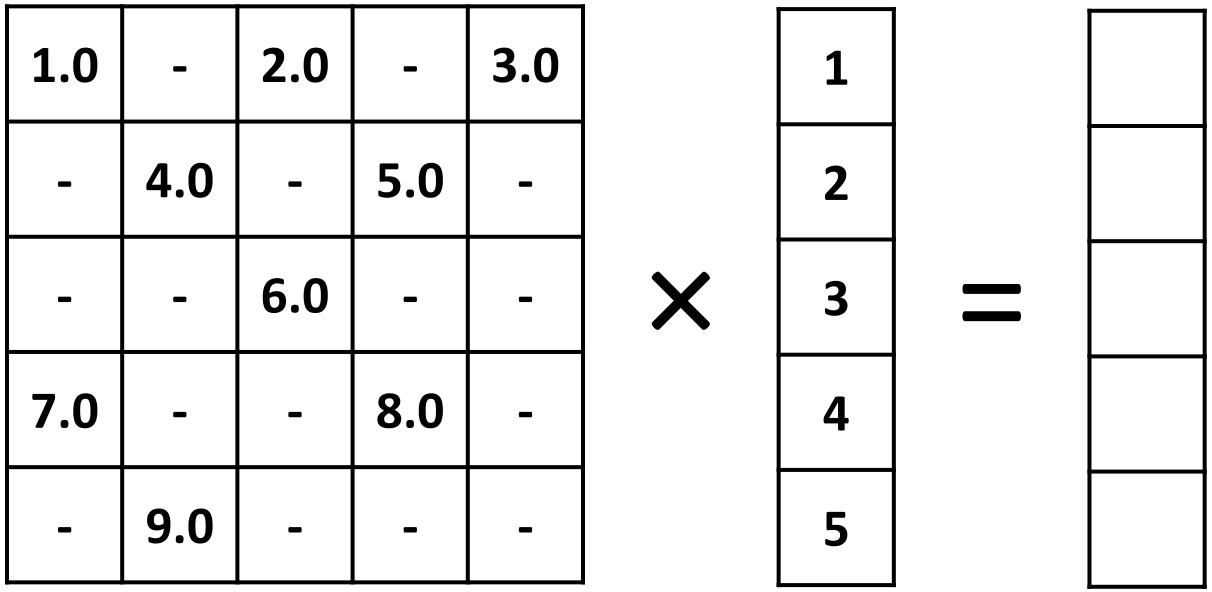
Demonstrate how to save space and time by using the CSR format for GPU-based SpMV

Dynamic GPU-based SpMV algorithm that is efficient for regular and irregular matrices

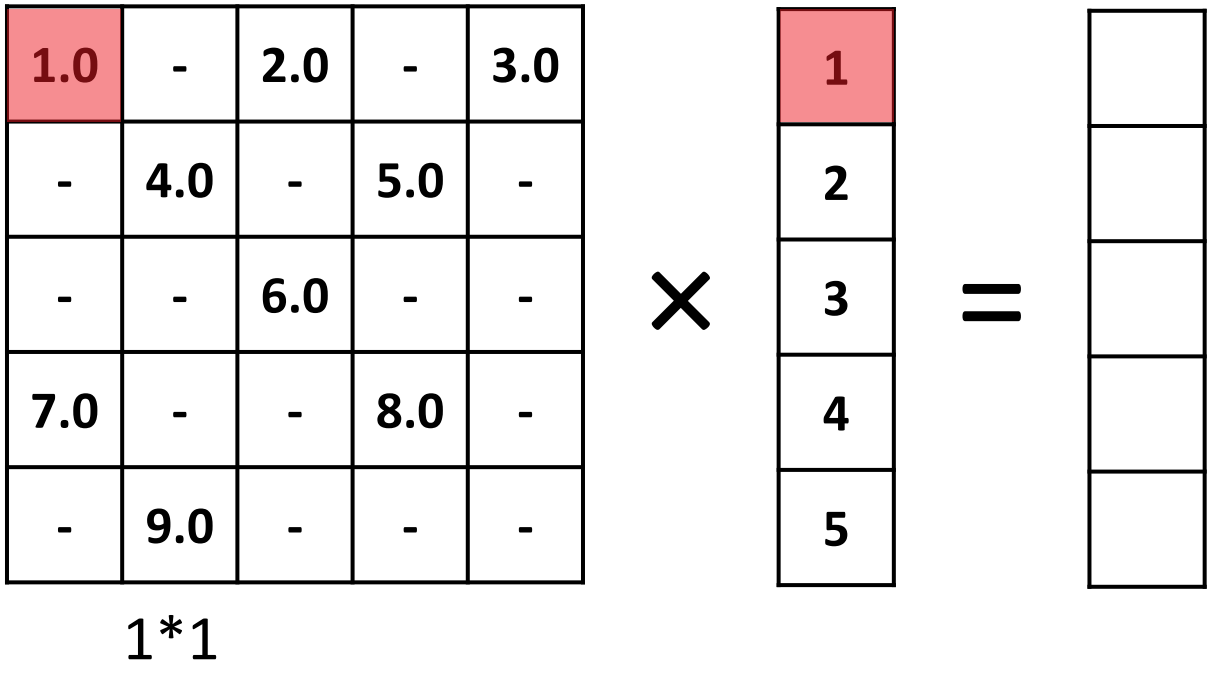
2x faster than CSR-Adaptive
36% faster than CSR5 at 10x less overheads

SPARSE MATRIX-VECTOR MULTIPLICATION (SPMV)

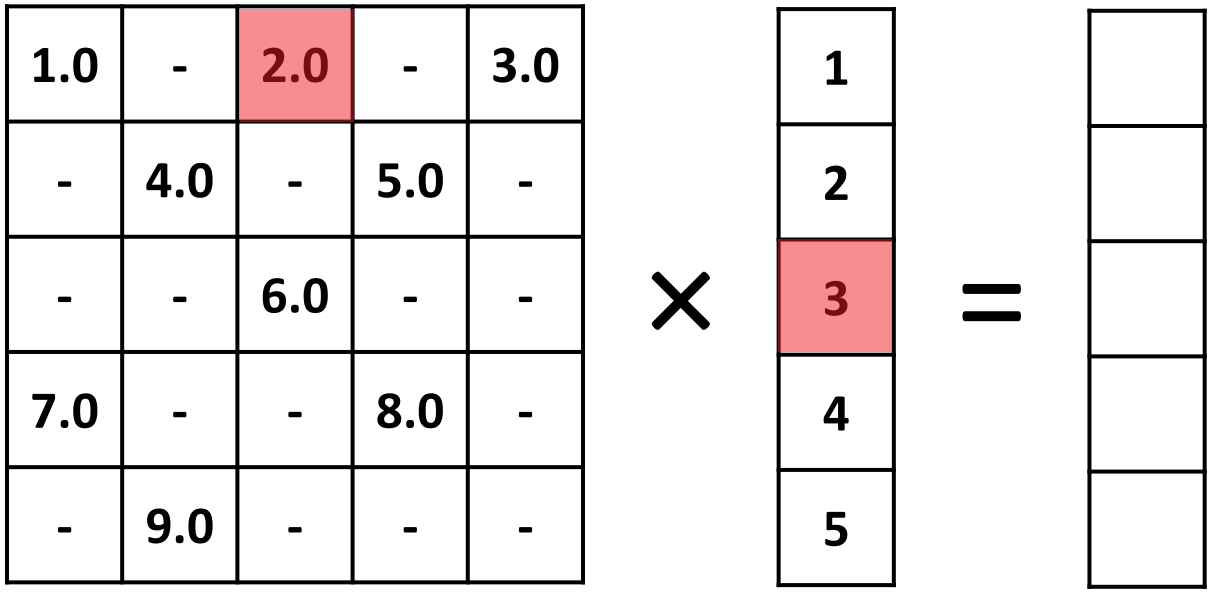
AMD



SPARSE MATRIX-VECTOR MULTIPLICATION (SPMV)

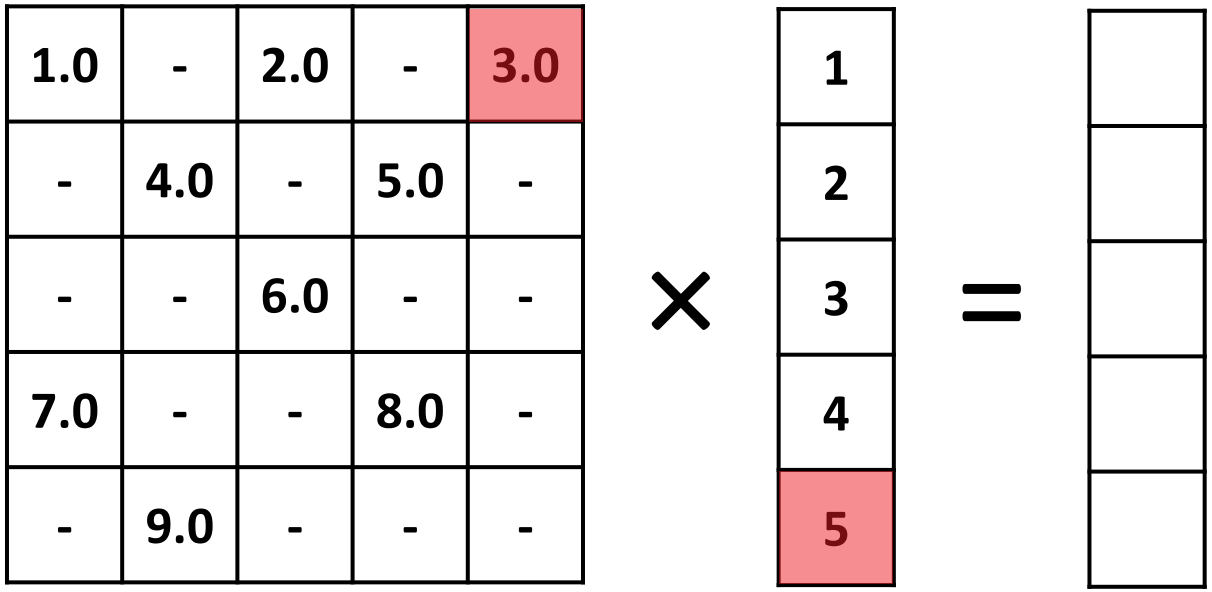


SPARSE MATRIX-VECTOR MULTIPLICATION (SPMV)

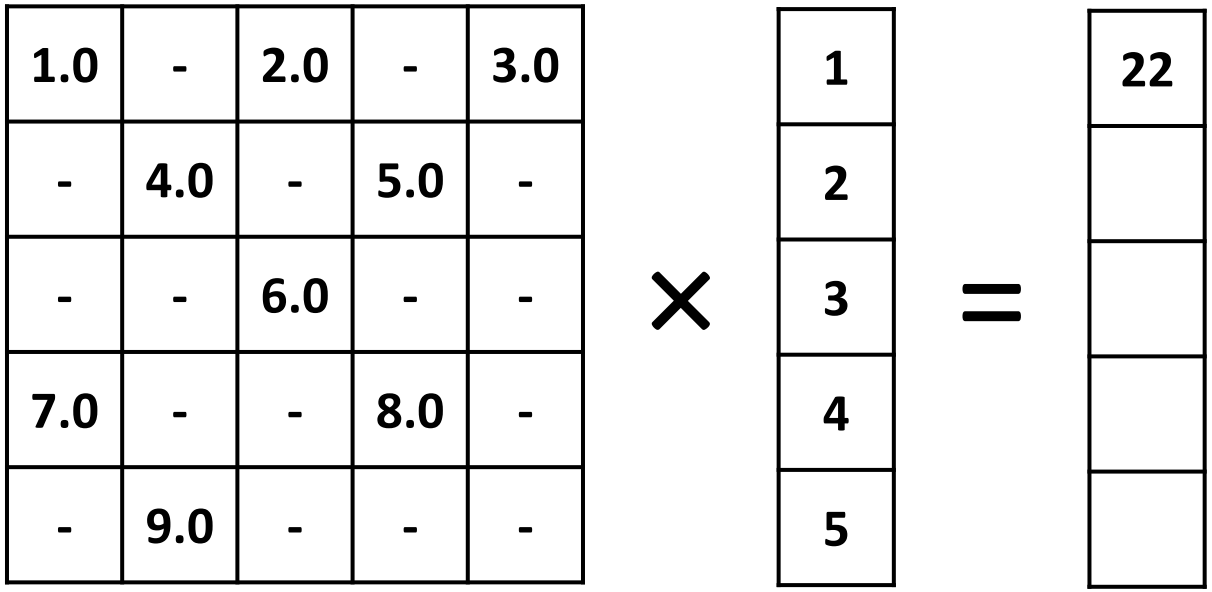


1*1 + 2*3

SPARSE MATRIX-VECTOR MULTIPLICATION (SPMV)

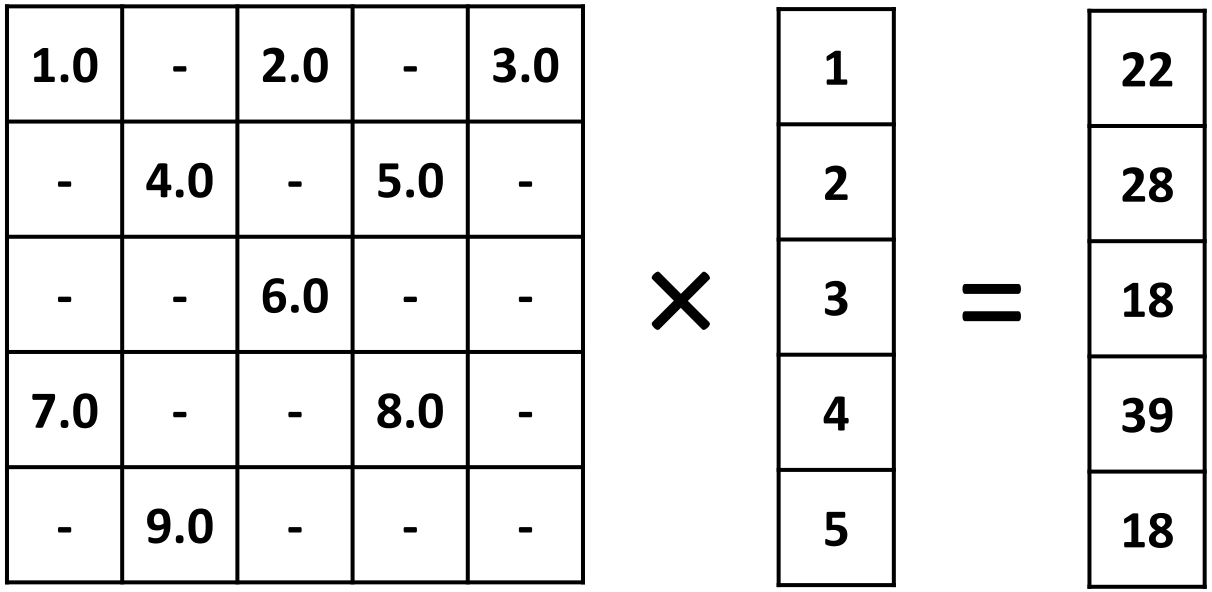


SPARSE MATRIX-VECTOR MULTIPLICATION (SPMV)



SPARSE MATRIX-VECTOR MULTIPLICATION (SPMV)

AMD



SPARSE MATRIX-VECTOR MULTIPLICATION (SPMV)

AMD

1.0	-	2.0	-	3.0
-	4.0	-	5.0	-
-	-	6.0	-	-
7.0	-	-	8.0	-
-	9.0	-	-	-

×

1
2
3
4
5

=

22
28
18
39
18

Traditionally Bandwidth Limited

Performance of SpMV is dominated by

how

the sparse matrix is stored in memory

COMPRESSED SPARSE ROW (CSR)

1.0	-	2.0	-	3.0
-	4.0	-	5.0	-
-	-	6.0	-	-
7.0	-	-	8.0	-
-	9.0	-	-	-

values:

COMPRESSED SPARSE ROW (CSR)

1.0	-	2.0	-	3.0
-	4.0	-	5.0	-
-	-	6.0	-	-
7.0	-	-	8.0	-
-	9.0	-	-	-

values:

1.0	2.0	3.0	4.0	5.0	6.0	7.0	8.0	9.0

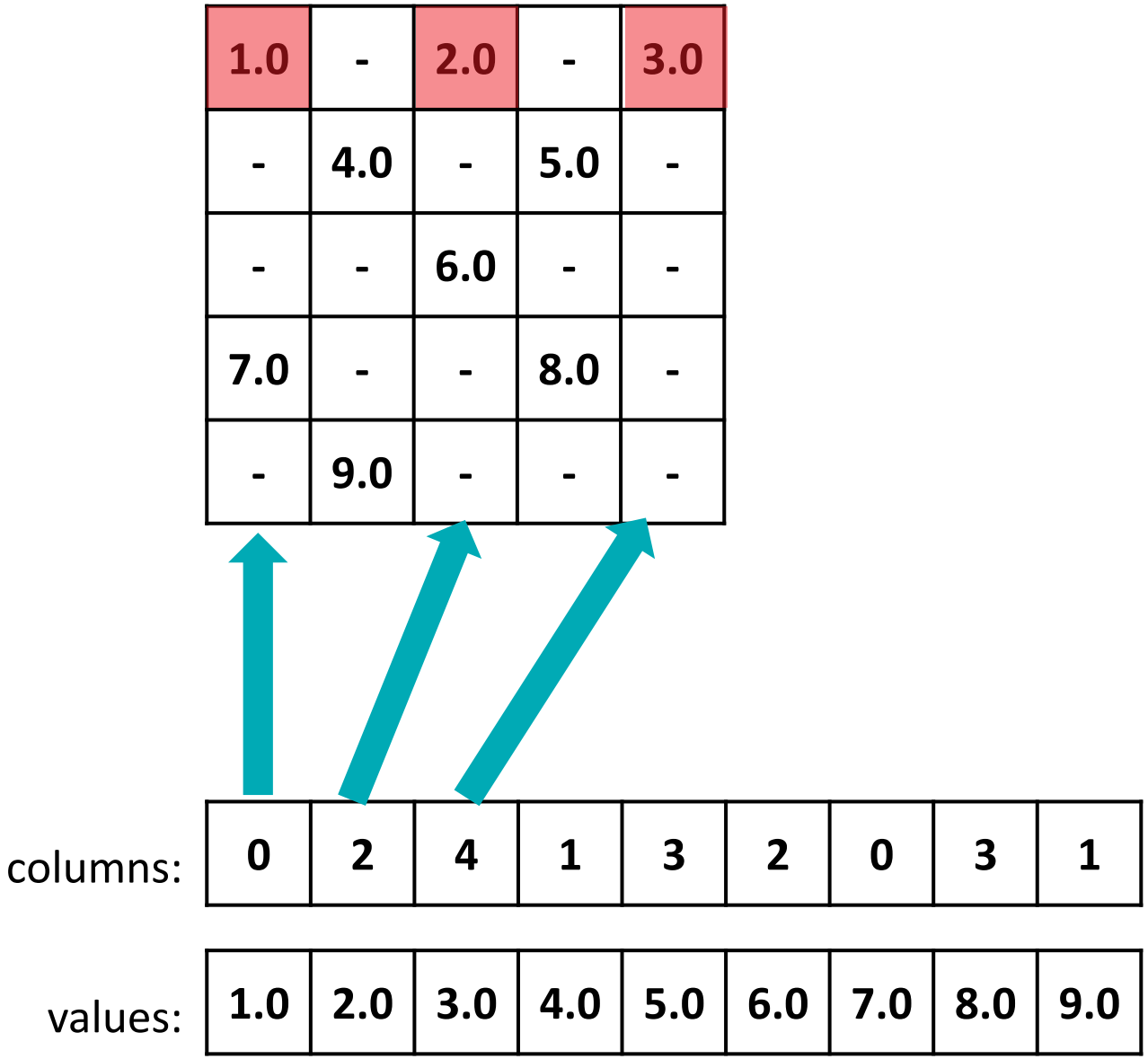
COMPRESSED SPARSE ROW (CSR)

1.0	-	2.0	-	3.0
-	4.0	-	5.0	-
-	-	6.0	-	-
7.0	-	-	8.0	-
-	9.0	-	-	-

columns:	0	2	4	1	3	2	0	3	1

values:	1.0	2.0	3.0	4.0	5.0	6.0	7.0	8.0	9.0

COMPRESSED SPARSE ROW (CSR)



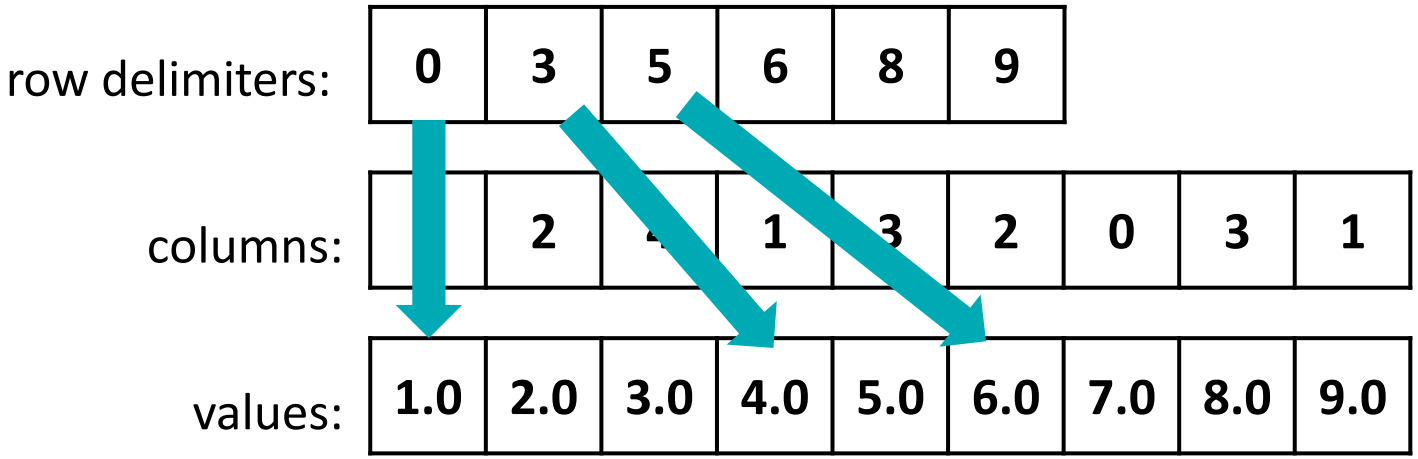
COMPRESSED SPARSE ROW (CSR)

1.0	-	2.0	-	3.0
-	4.0	-	5.0	-
-	-	6.0	-	-
7.0	-	-	8.0	-
-	9.0	-	-	-

row delimiters:	0	3	5	6	8	9		
columns:	0	2	4	1	3	2	0	3	1
values:	1.0	2.0	3.0	4.0	5.0	6.0	7.0	8.0	9.0

COMPRESSED SPARSE ROW (CSR)

1.0	-	2.0	-	3.0
-	4.0	-	5.0	-
-	-	6.0	-	-
7.0	-	-	8.0	-
-	9.0	-	-	-



COMPRESSED SPARSE ROW (CSR)

1.0	-	2.0	-	3.0
-	4.0	-	5.0	-
-	-	6.0	-	-
7.0	-	-	8.0	-
-	9.0	-	-	-

row delimiters:

0	3	5	6	8	9

columns:

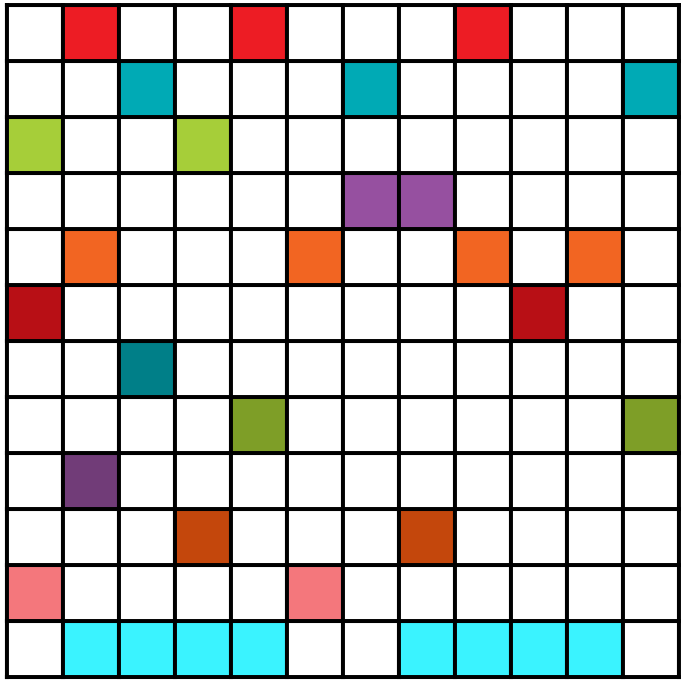
0	2	4	1	3	2	0	3	1

values:

1.0	2.0	3.0	4.0	5.0	6.0	7.0	8.0	9.0

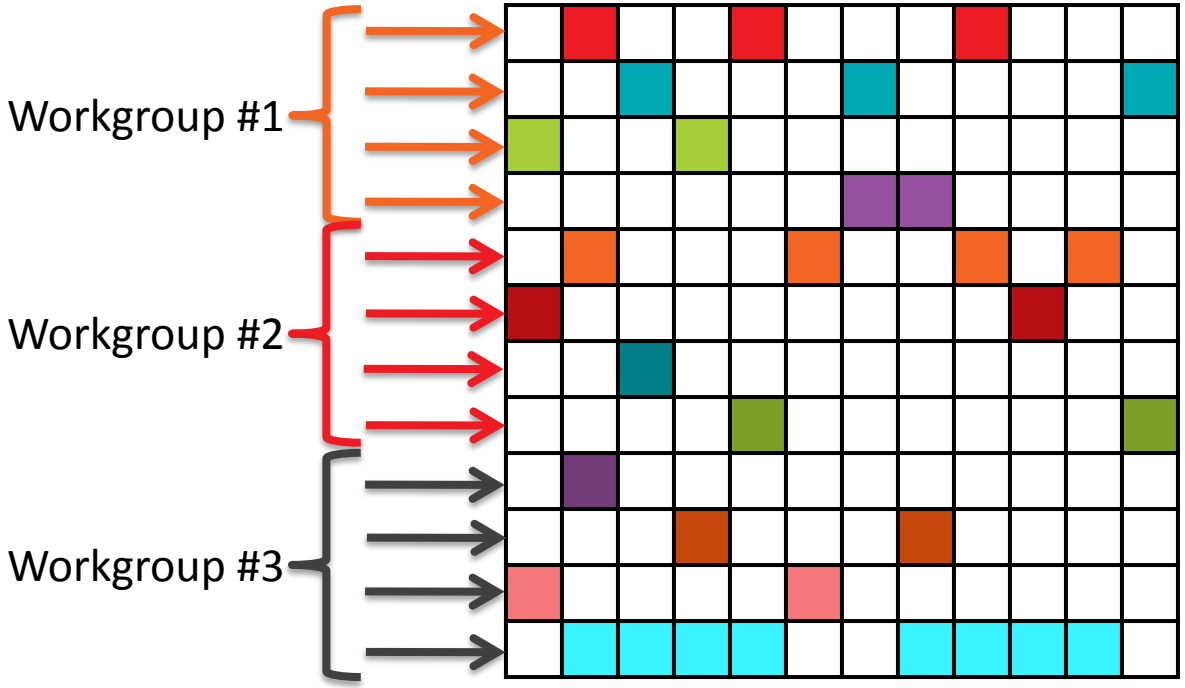
CSR-SCALAR

- One thread/work-item per matrix row



CSR-SCALAR

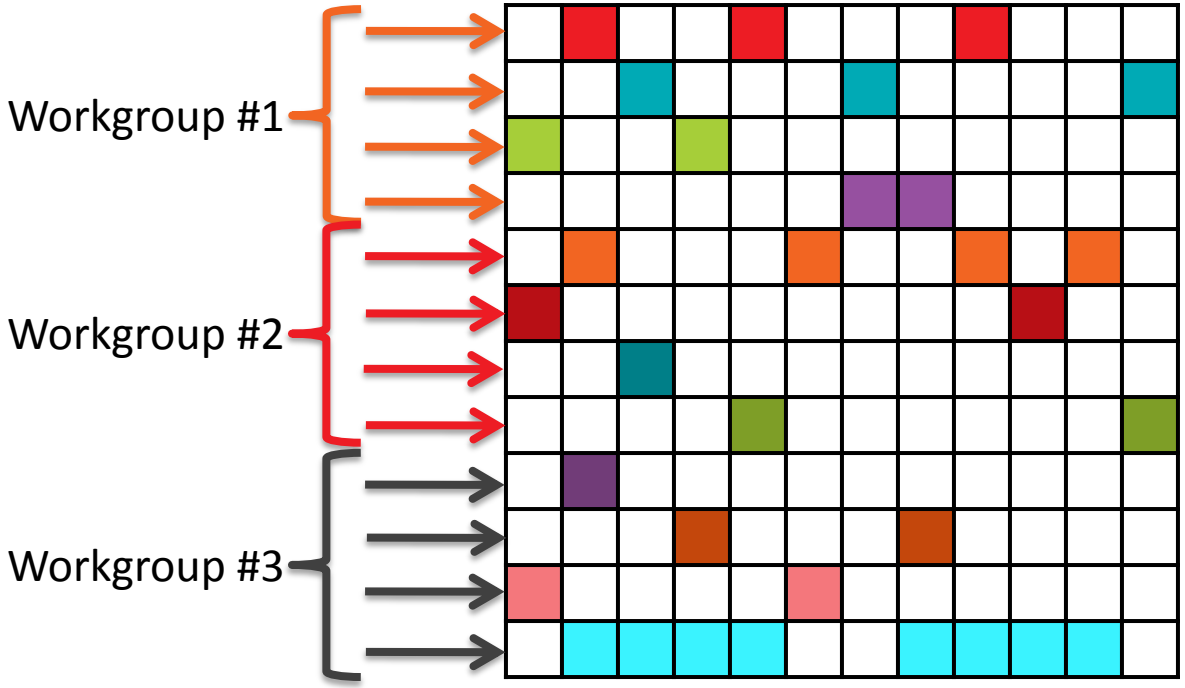
- One thread/work-item per matrix row



CSR-SCALAR

AMD

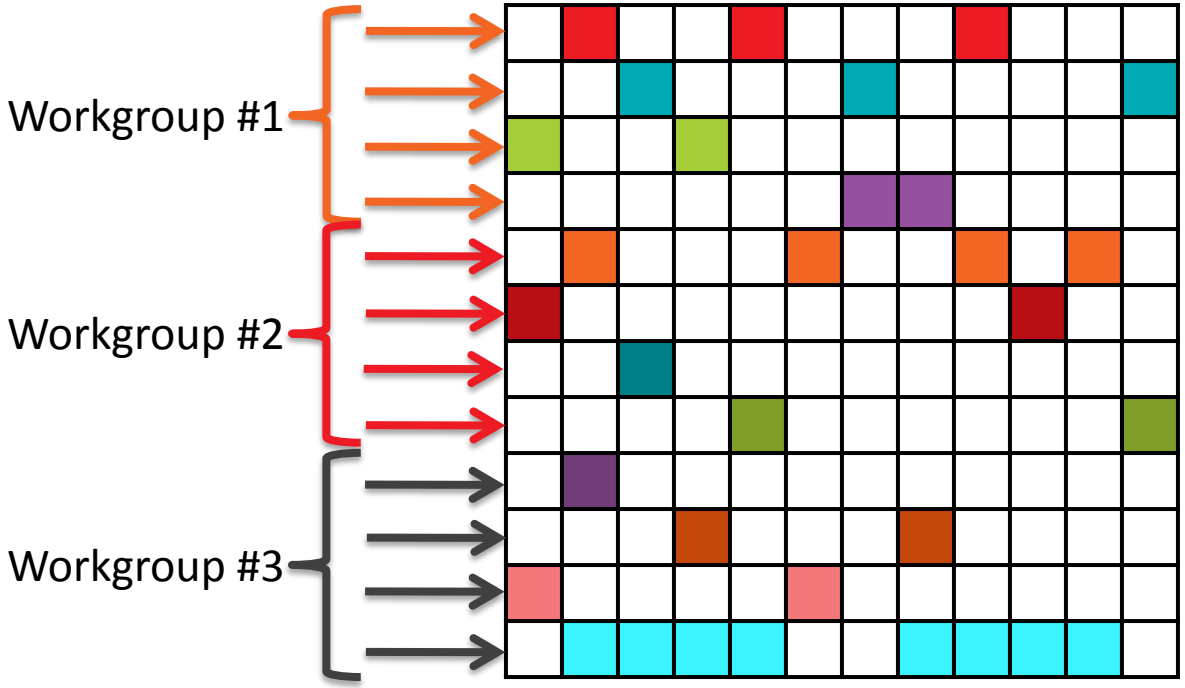
- One thread/work-item per matrix row



CSR-SCALAR

AMD

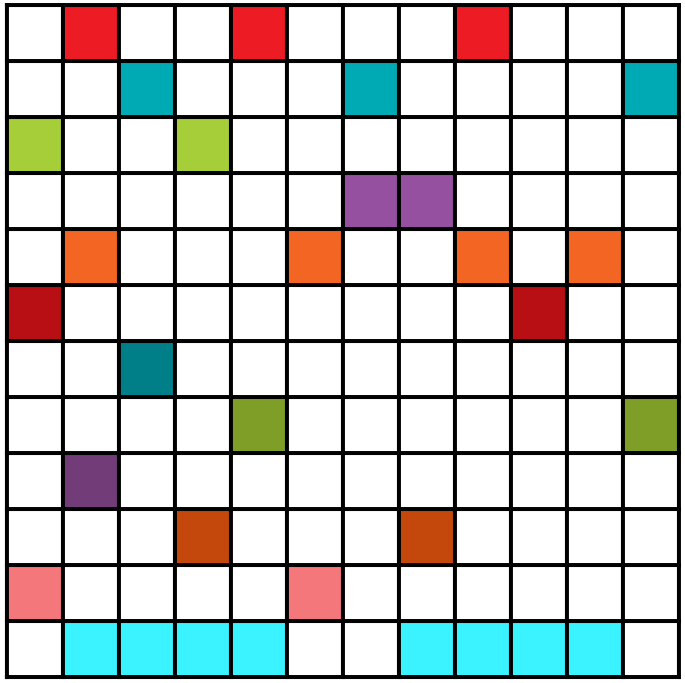
- One thread/work-item per matrix row



Uncoalesced accesses

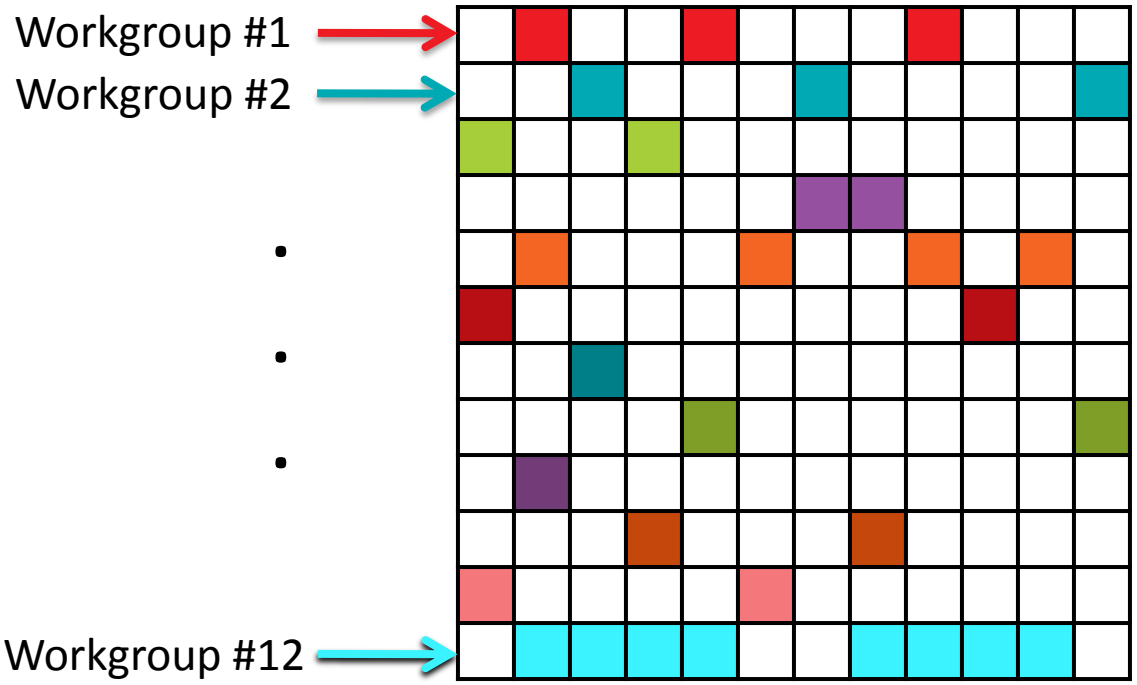
CSR-VECTOR

- One workgroup (multiple threads) per matrix row



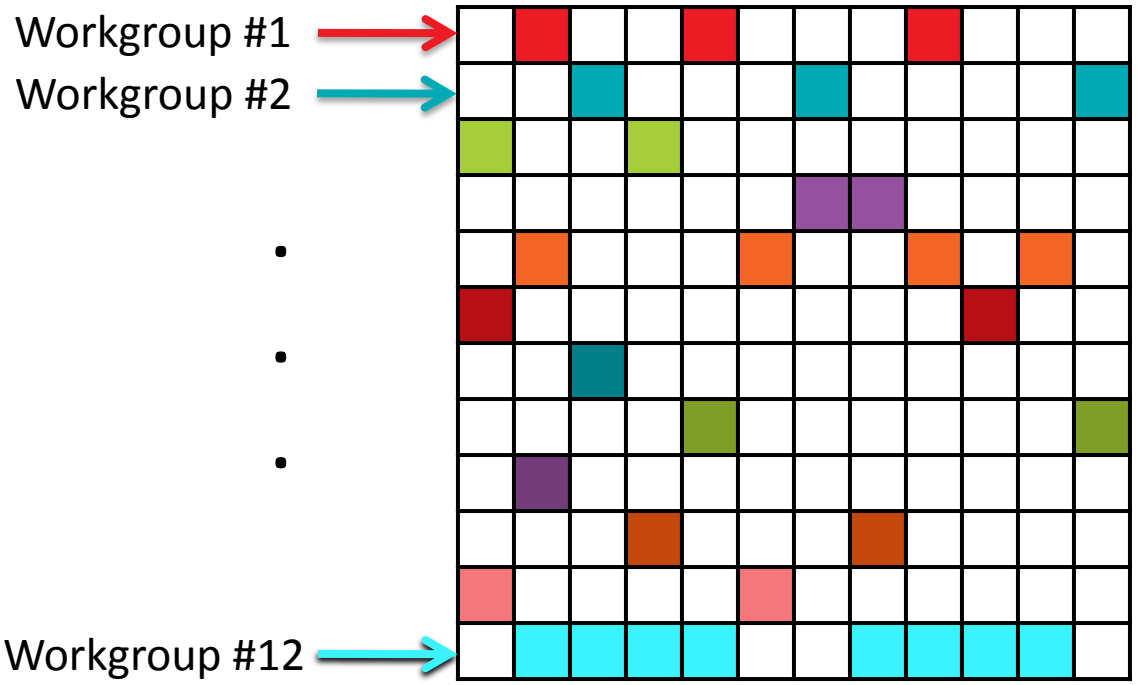
CSR-VECTOR

- One workgroup (multiple threads) per matrix row



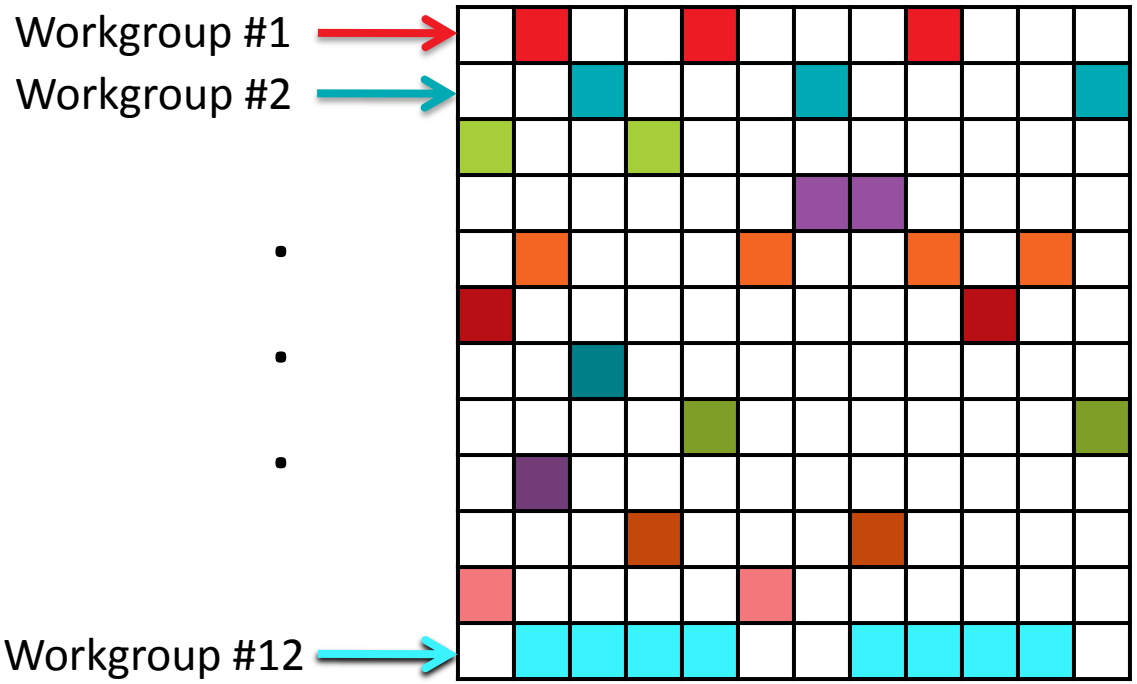
CSR-VECTOR

- One workgroup (multiple threads) per matrix row



CSR-VECTOR

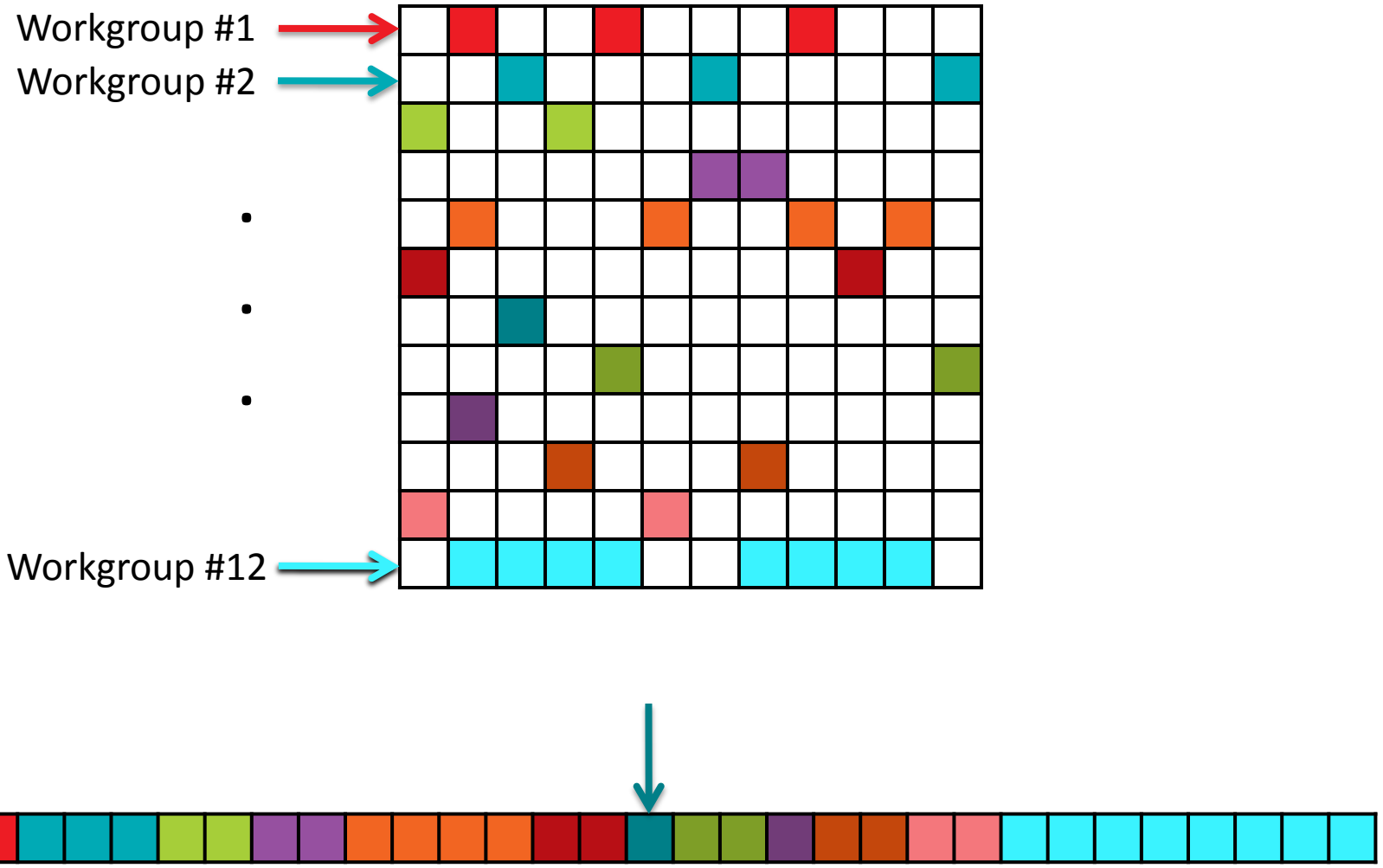
- One workgroup (multiple threads) per matrix row



Good: coalesced accesses

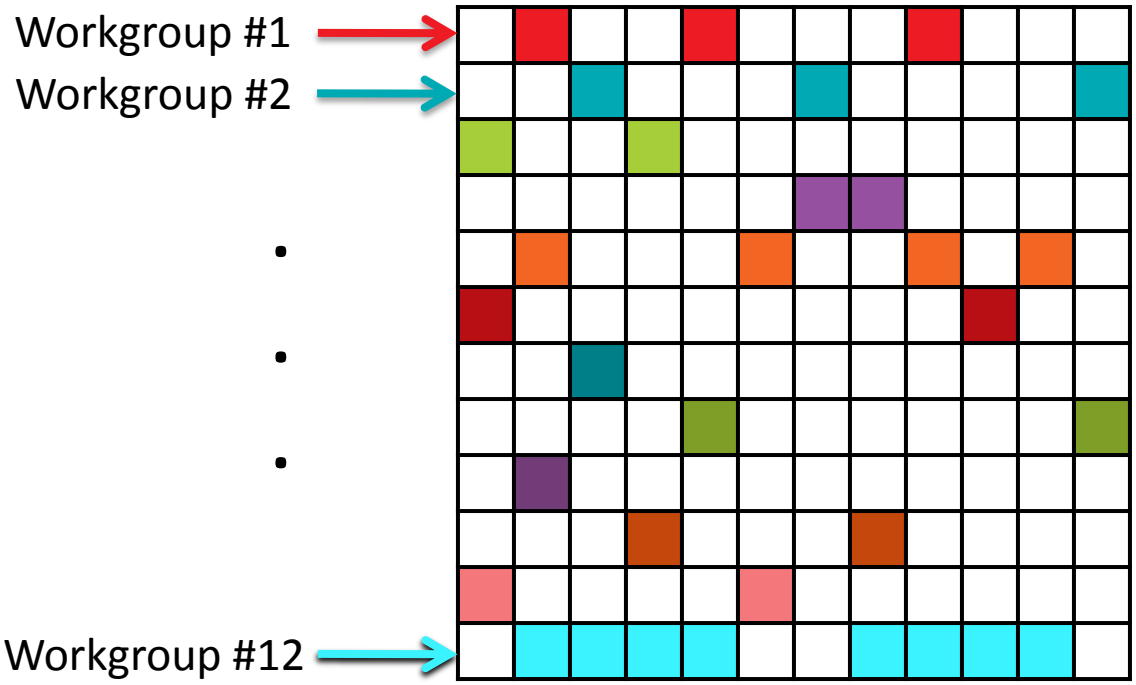
CSR-VECTOR

- One workgroup (multiple threads) per matrix row

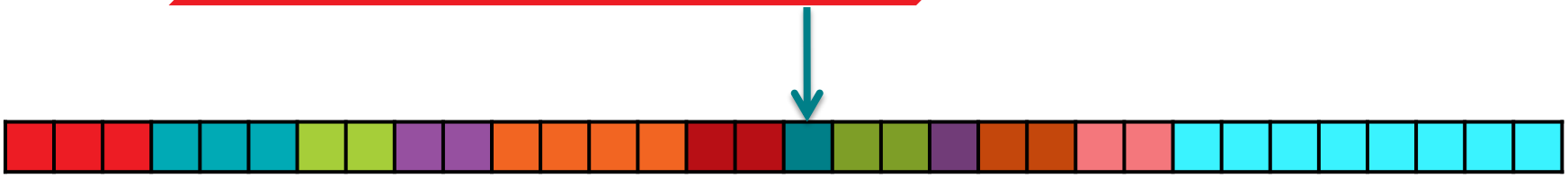


CSR-VECTOR

- One workgroup (multiple threads) per matrix row



Bad: poor parallelism



MAKING SPMV FAST ON GPUs

AMD

- Neither CSR-scalar nor CSR-vector offer ideal performance

MAKING SPMV FAST ON GPUs

AMD

- Neither CSR-scalar nor CSR-vector offer ideal performance
- Traditional solution: new matrix storage format + new algorithms
  - More than *fifty* new storage formats have been proposed

MAKING SPMV FAST ON GPUs

AMD

- Neither CSR-scalar nor CSR-vector offer ideal performance

- Traditional solution: new matrix storage format + new algorithms
  - More than *fifty* new storage formats have been proposed

rewrite software

space and time overhead

MAKING SPMV FAST ON GPUs

AMD

- Neither CSR-scalar nor CSR-vector offer ideal performance
- Traditional solution: new matrix storage format + new algorithms
  - More than *fifty* new storage formats have been proposed

rewrite software

space and time overhead

- Can we find a good GPU algorithm that **does not modify the original CSR data**?

AMD

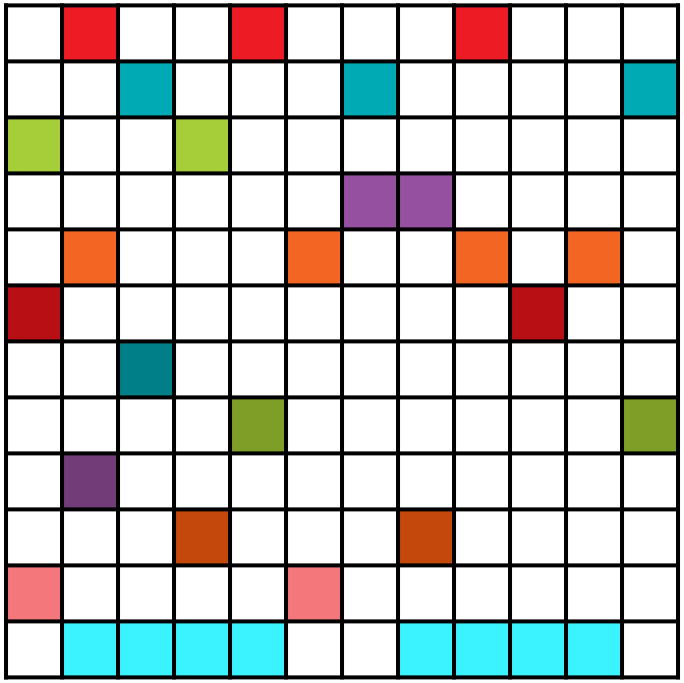
CSR-Adaptive

AMD

Automatically adapts
to *different* problems
in *different* domains

CSR-Adaptive

INCREASING BANDWIDTH EFFICIENCY



INCREASING BANDWIDTH EFFICIENCY

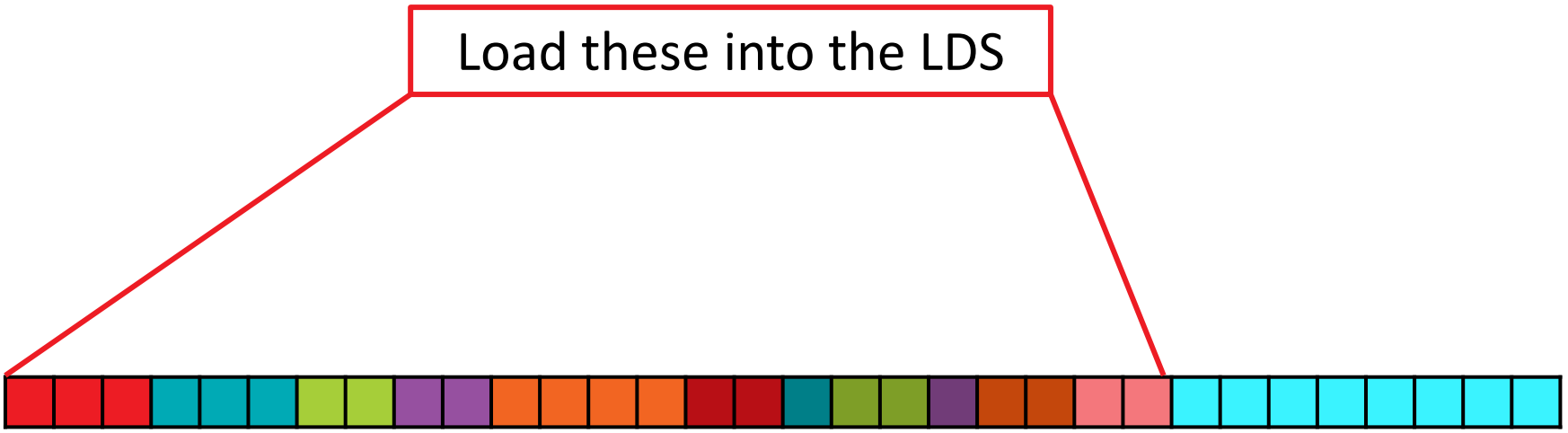
INCREASING BANDWIDTH EFFICIENCY

INCREASING BANDWIDTH EFFICIENCY

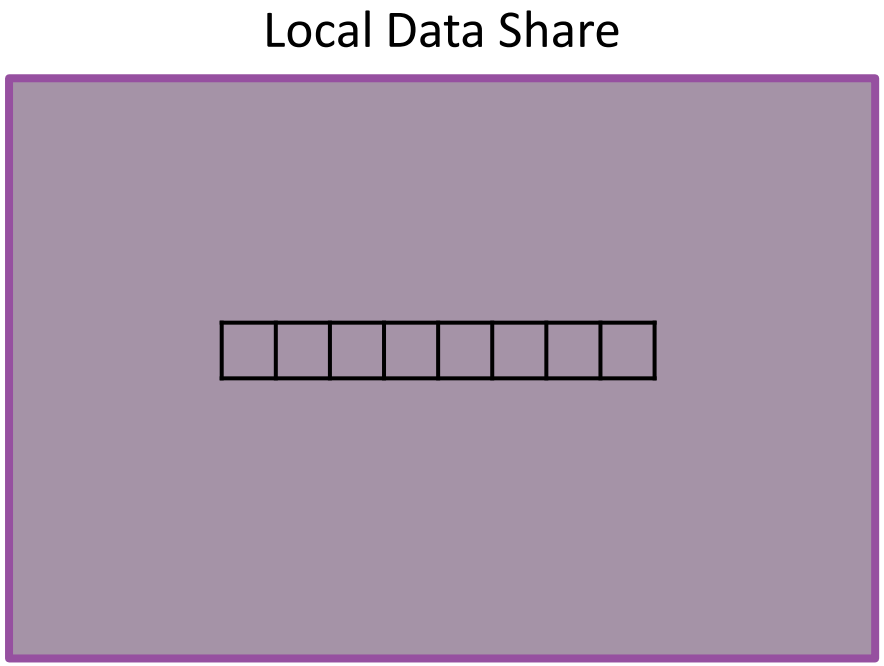
- How can we get good bandwidth for other “short” rows?

INCREASING BANDWIDTH EFFICIENCY

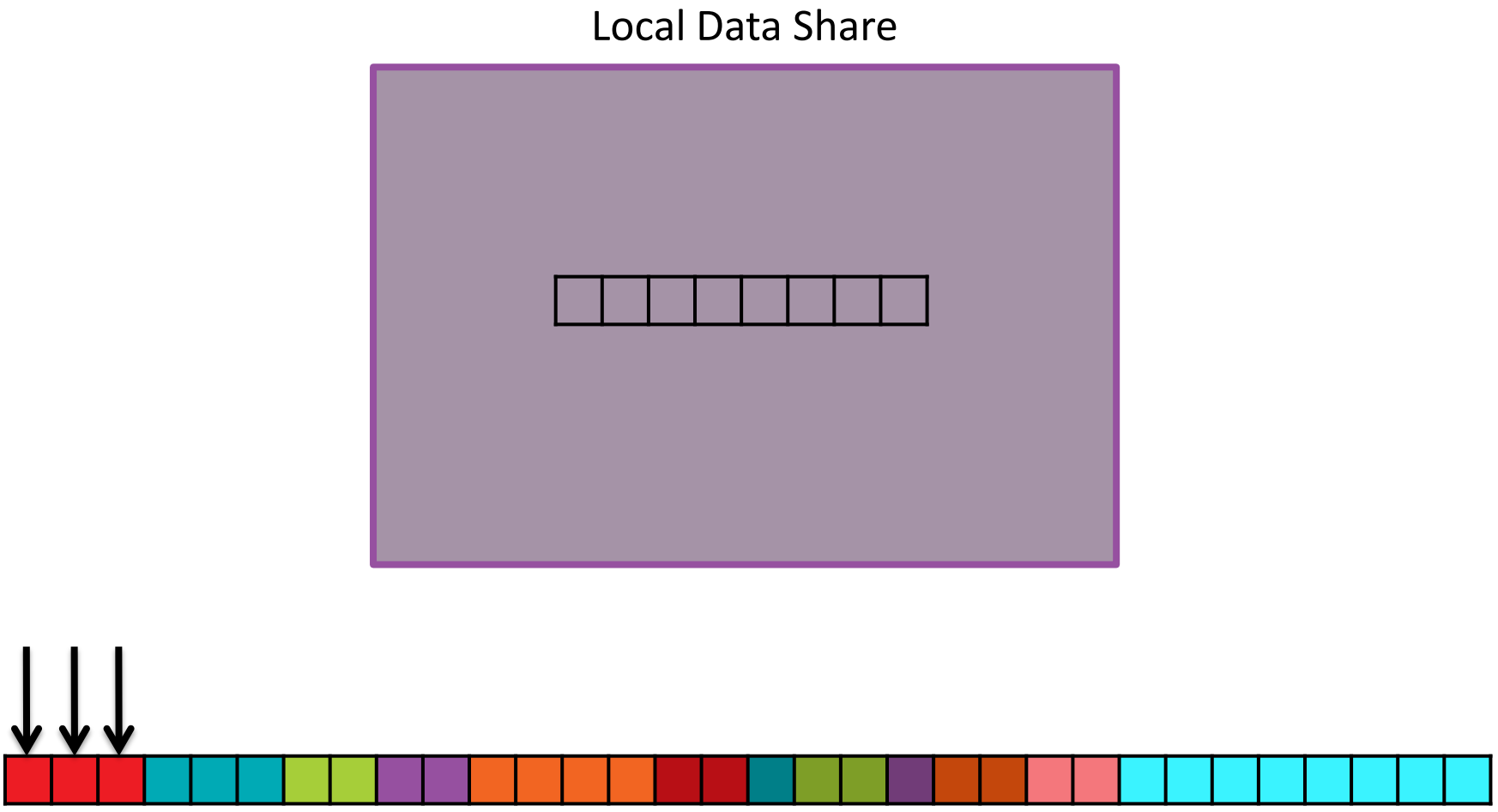
- How can we get good bandwidth for other “short” rows?



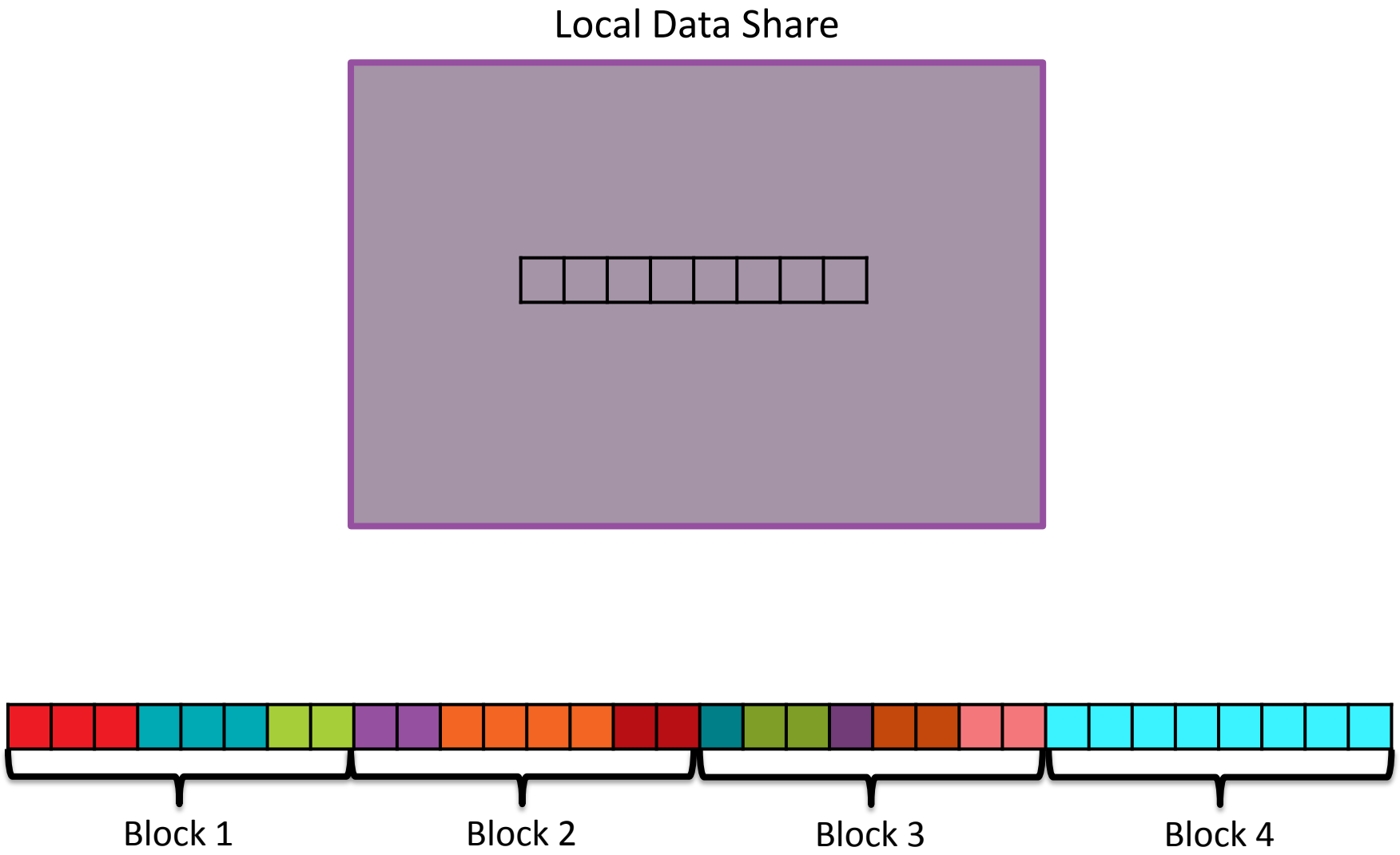
CSR-STREAM: STREAM MATRIX INTO THE LDS



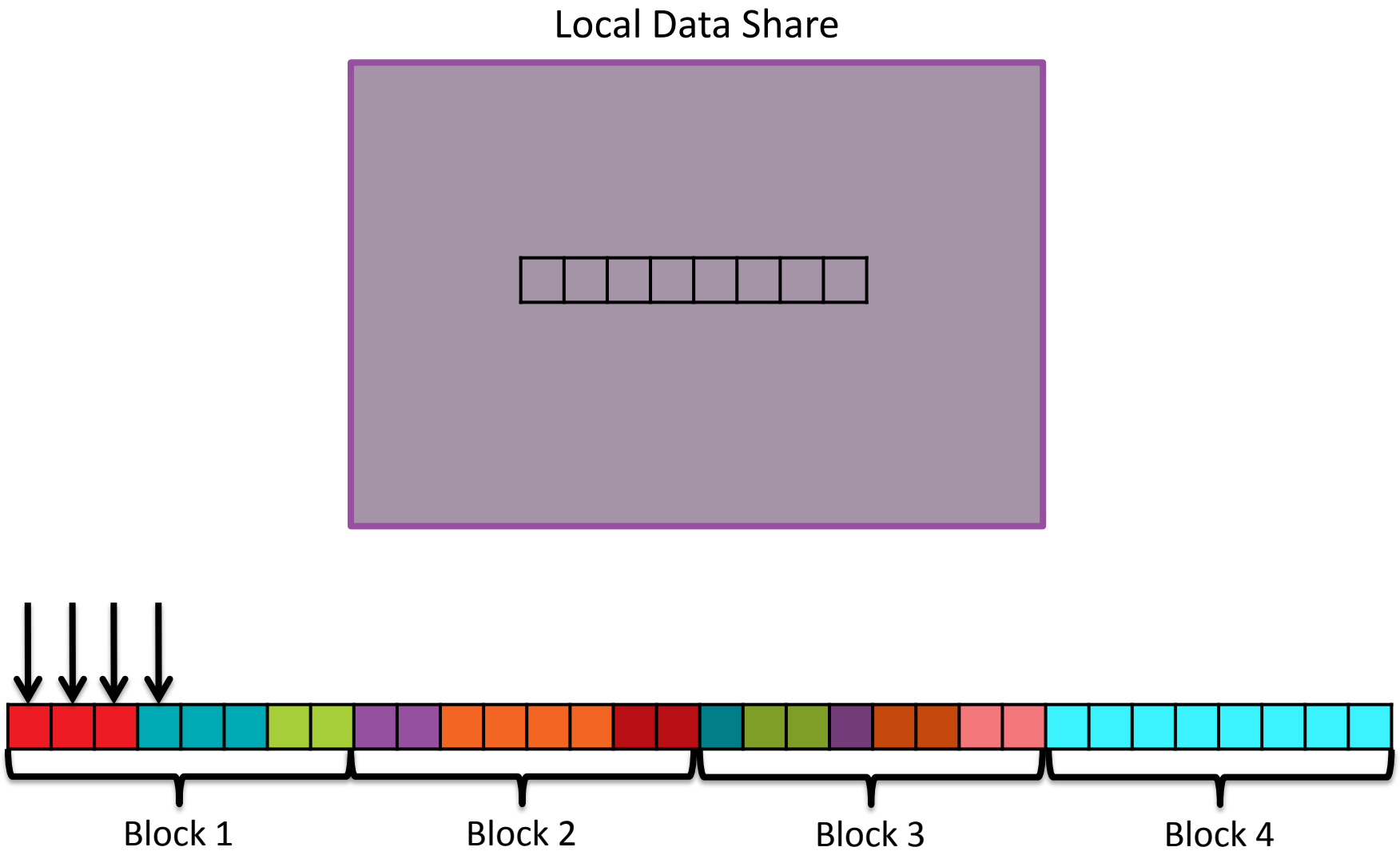
CSR-STREAM: STREAM MATRIX INTO THE LDS



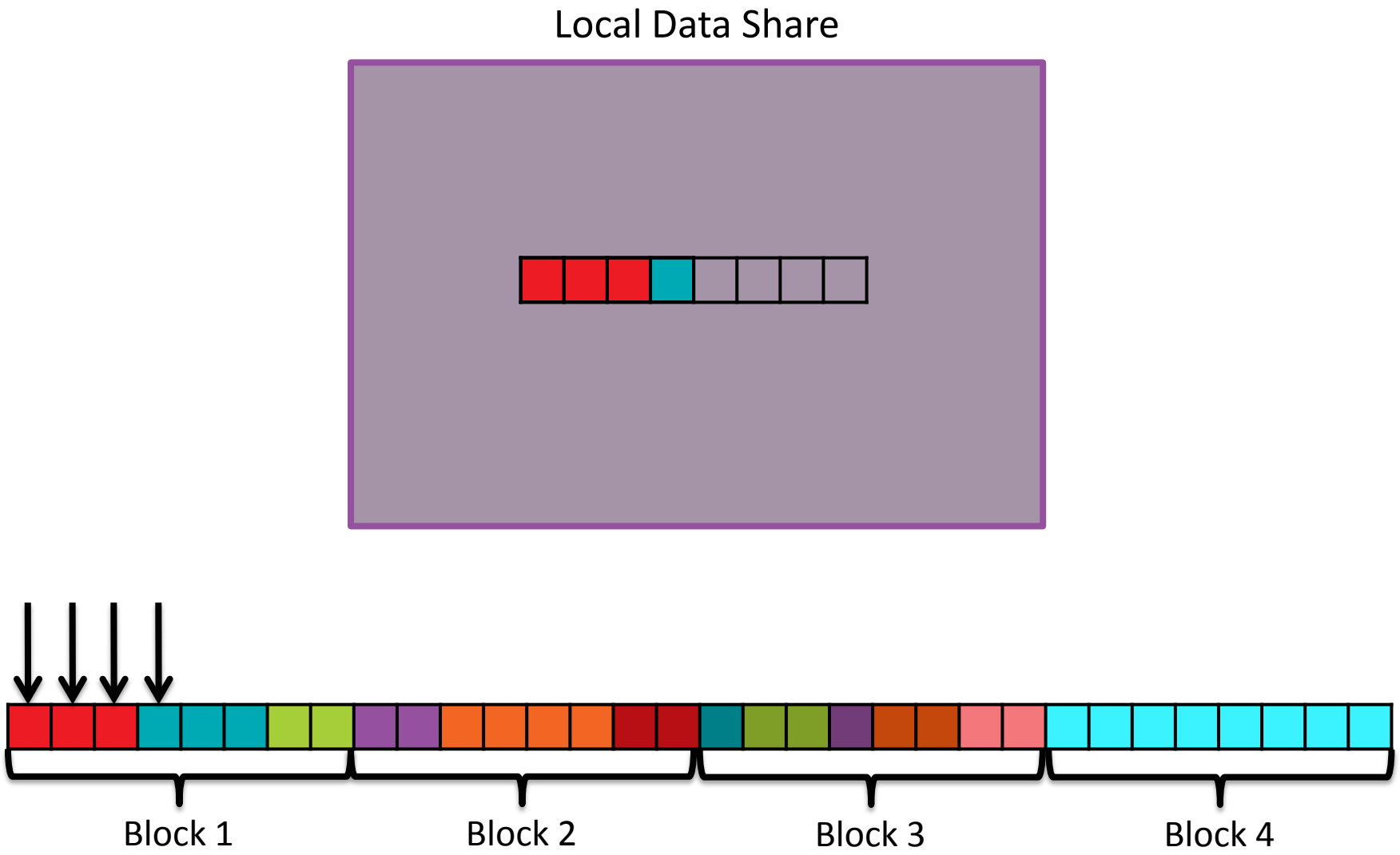
CSR-STREAM: STREAM MATRIX INTO THE LDS



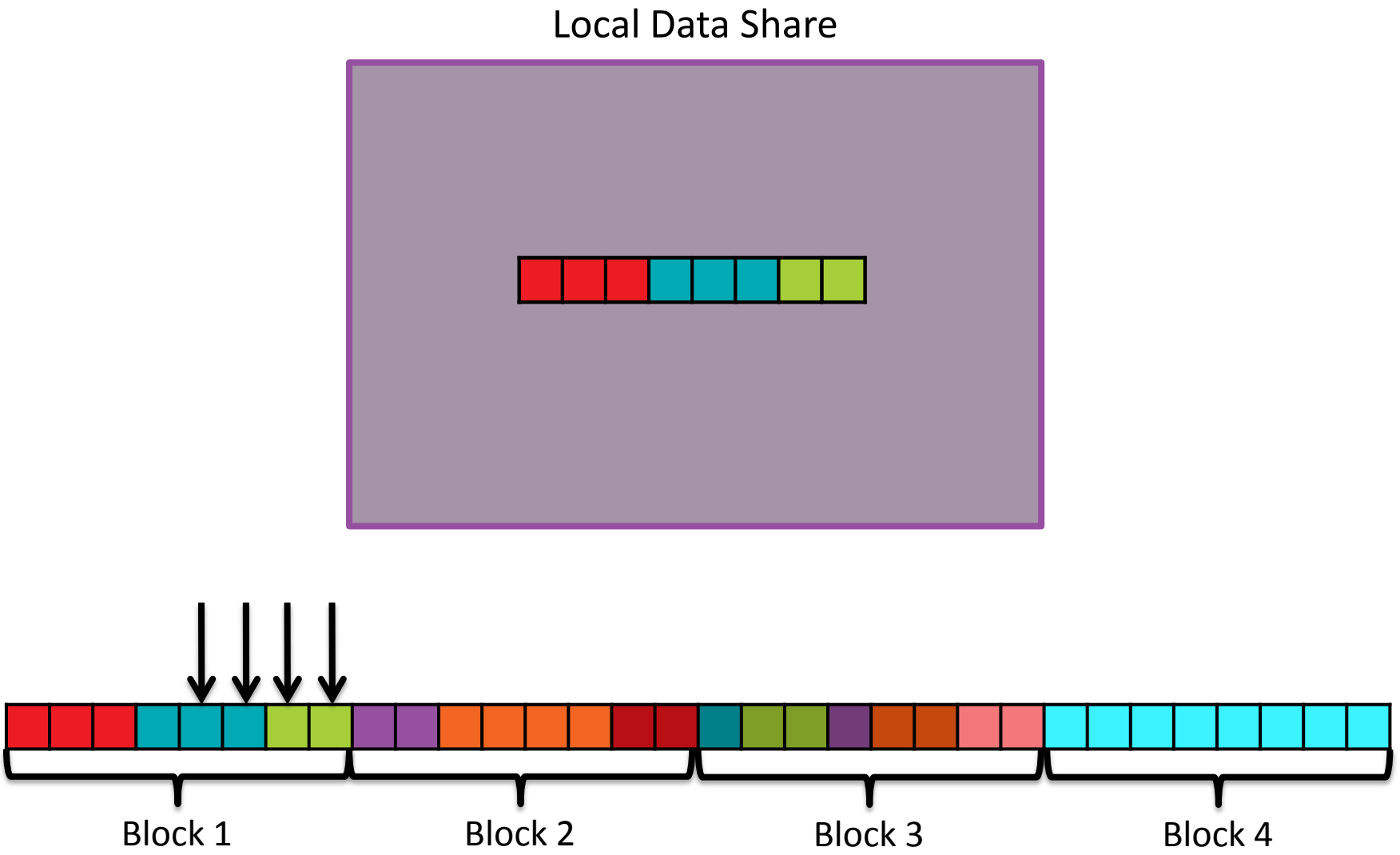
CSR-STREAM: STREAM MATRIX INTO THE LDS



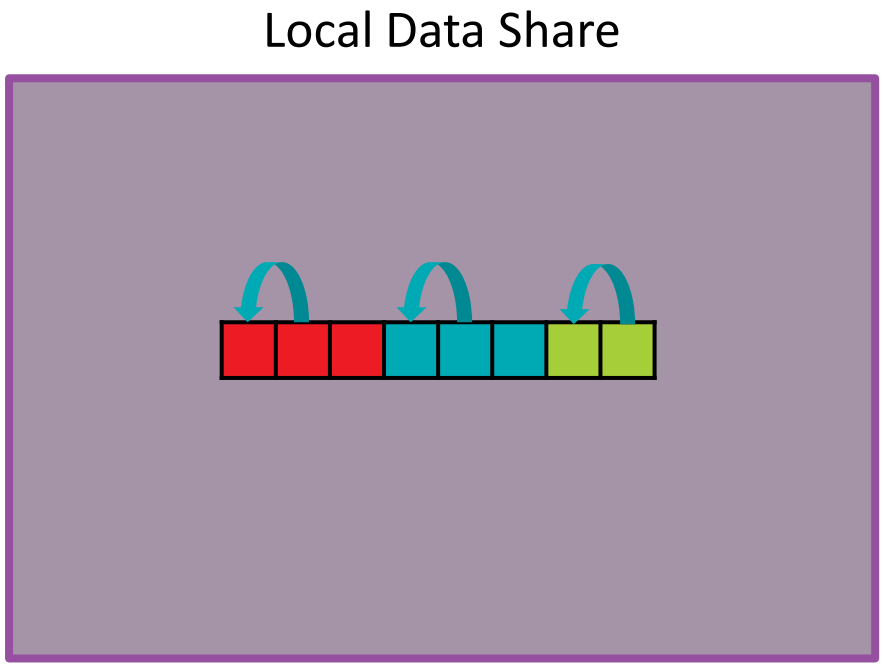
CSR-STREAM: STREAM MATRIX INTO THE LDS

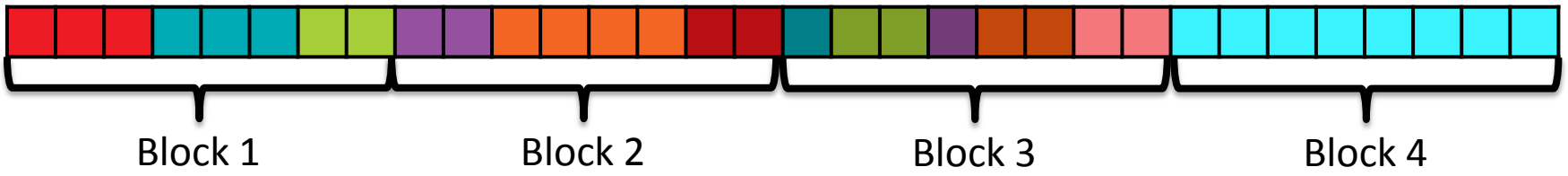


CSR-STREAM: STREAM MATRIX INTO THE LDS

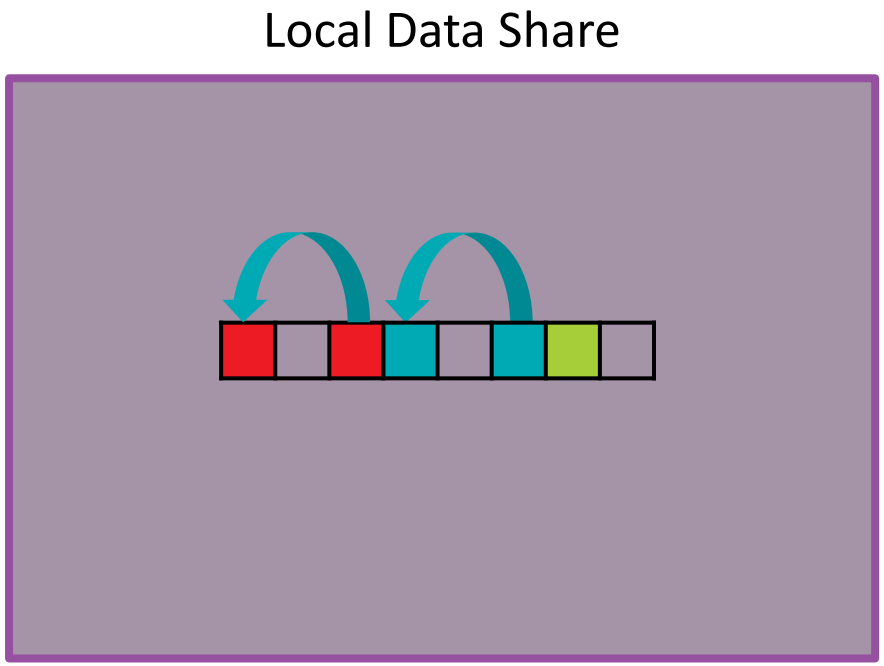


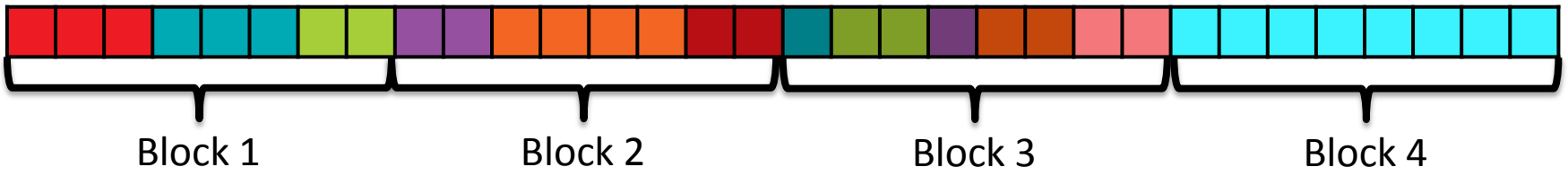
CSR-STREAM: STREAM MATRIX INTO THE LDS



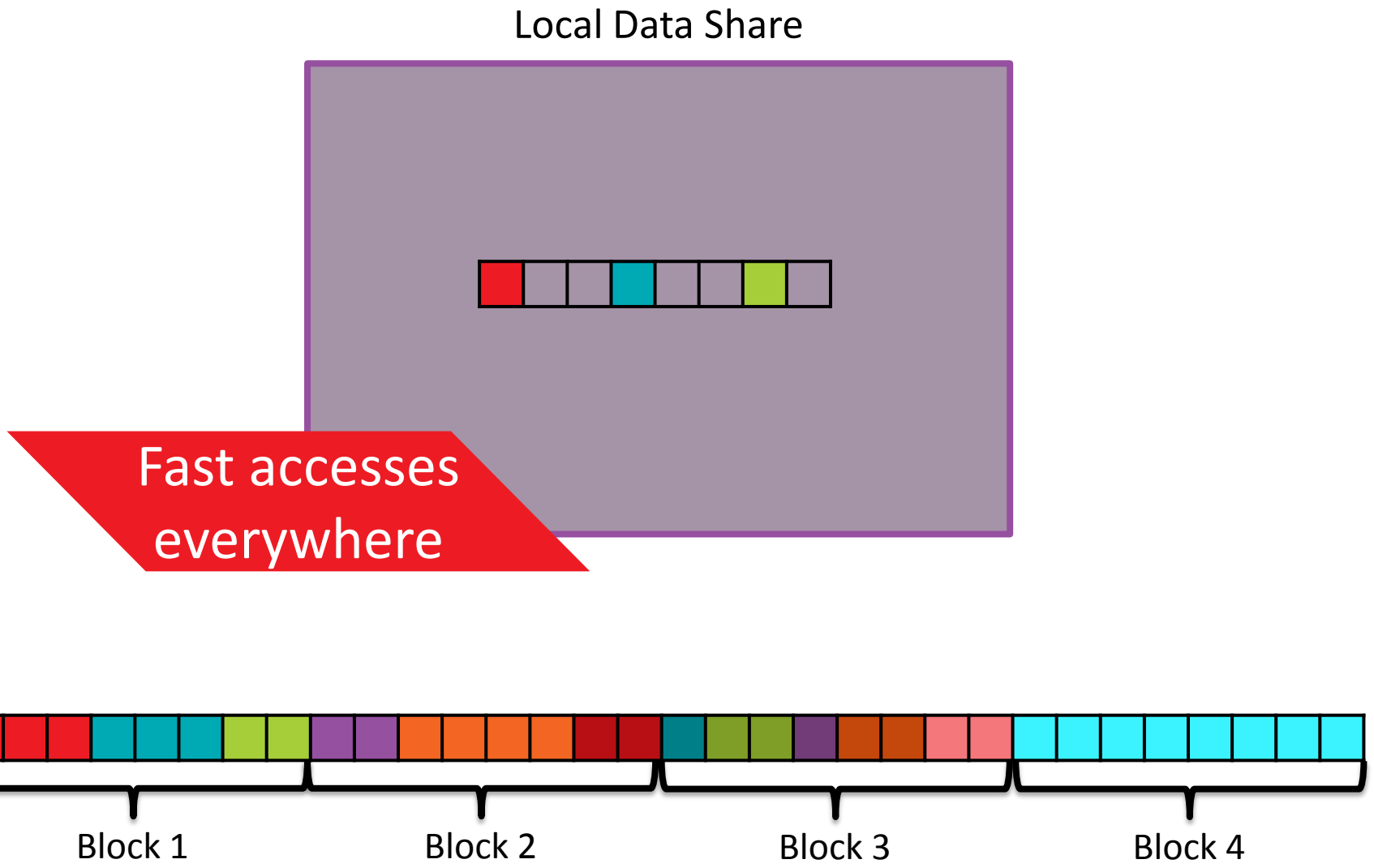


CSR-STREAM: STREAM MATRIX INTO THE LDS

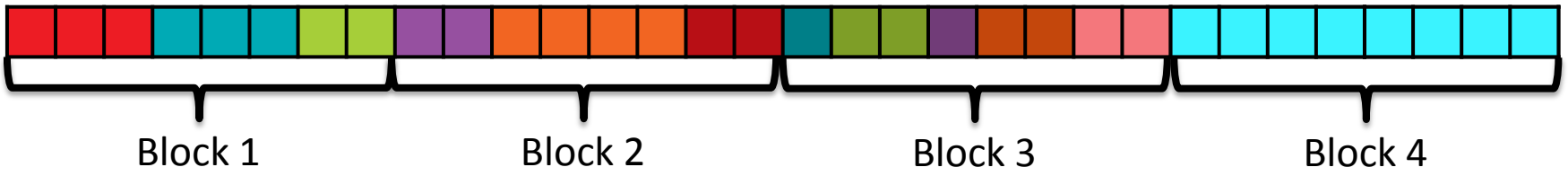




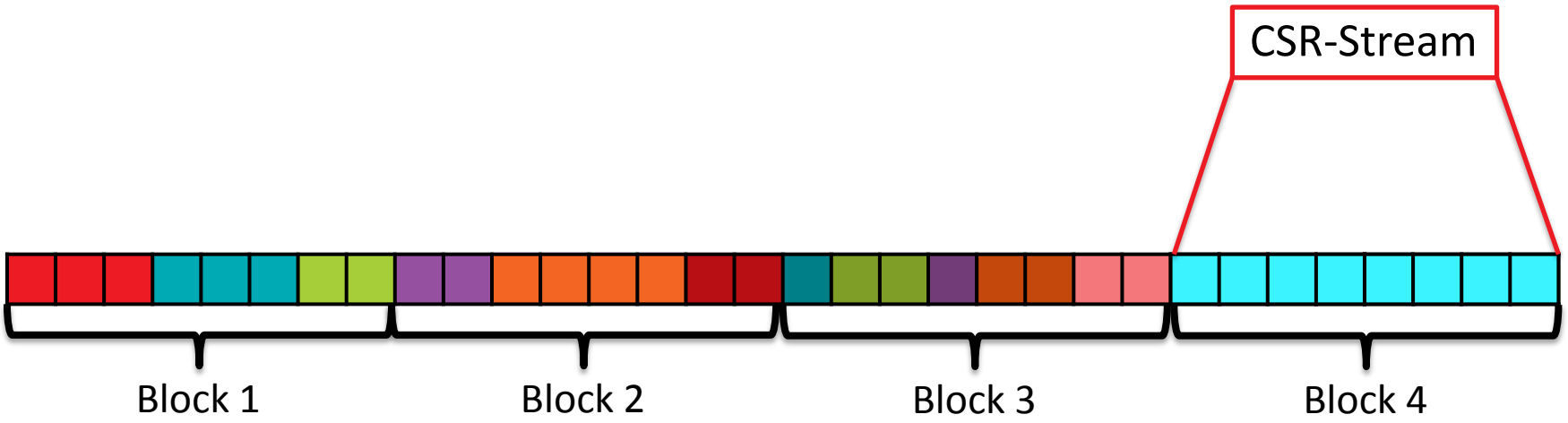
CSR-STREAM: STREAM MATRIX INTO THE LDS



- How can we get good bandwidth for “medium-sized” rows?

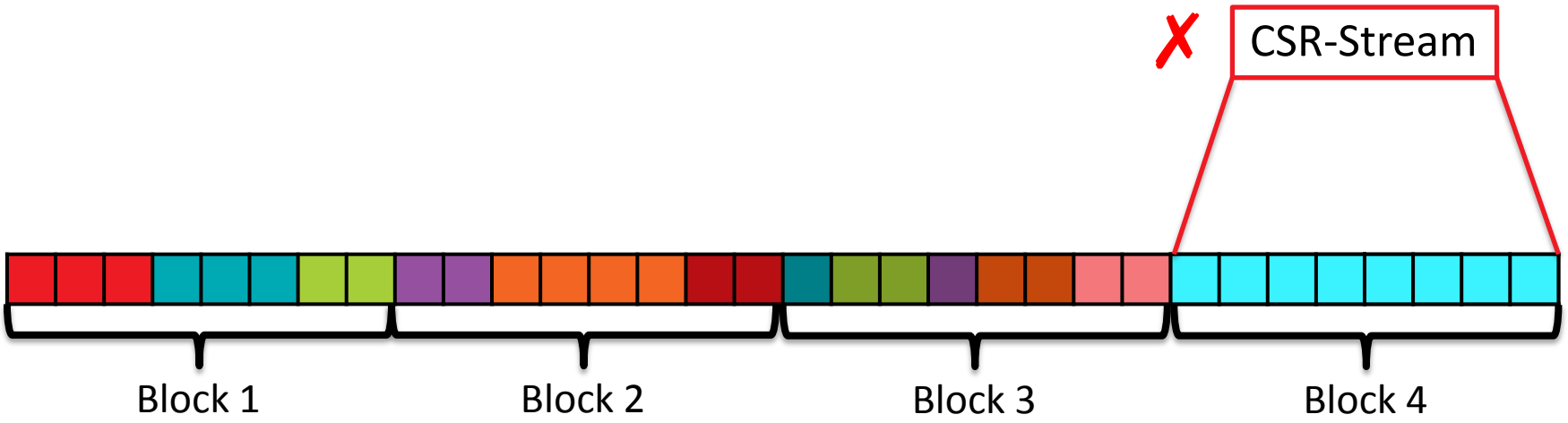


- How can we get good bandwidth for “medium-sized” rows?



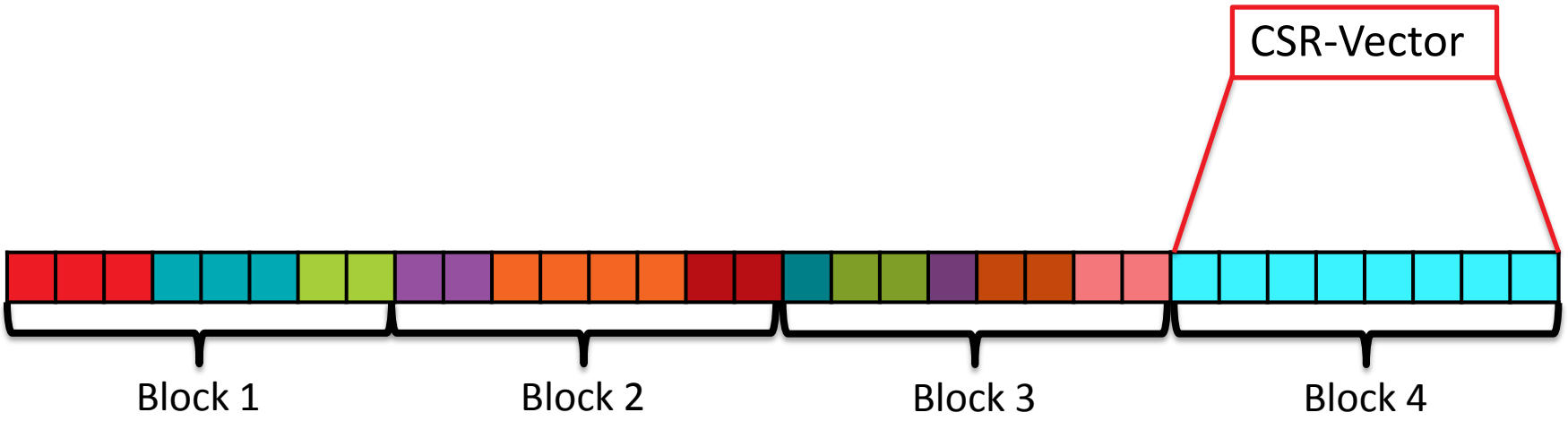
AMD

- How can we get good bandwidth for “medium-sized” rows?



AMD

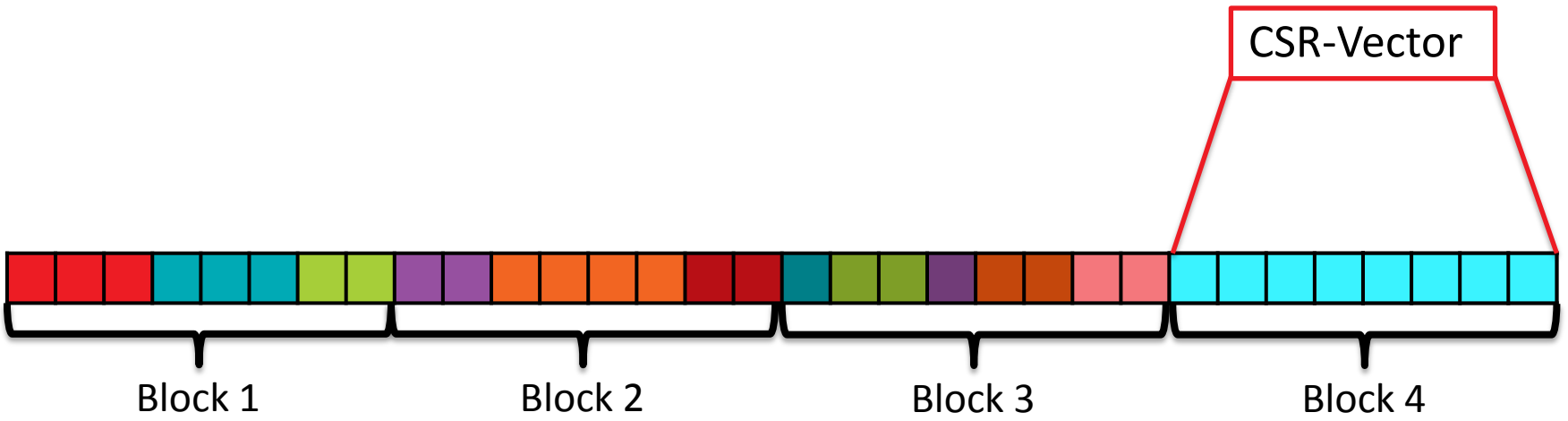
- How can we get good bandwidth for “medium-sized” rows?



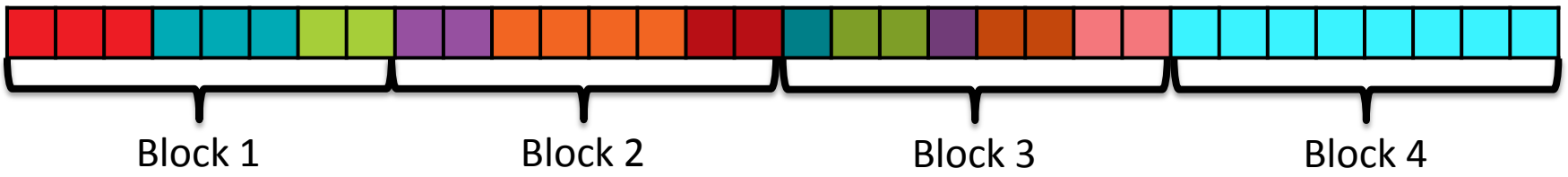
- How can we get good bandwidth for “medium-sized” rows?

Improved reduction technique

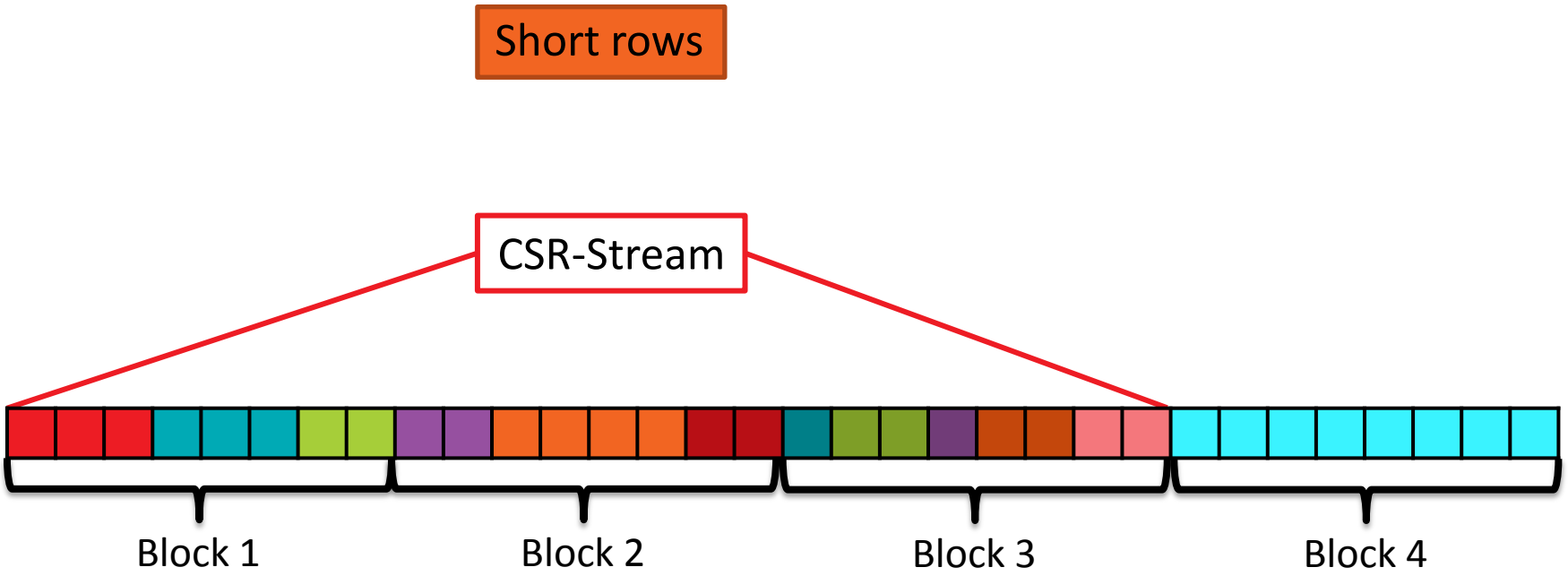
Efficient use of memory subsystem



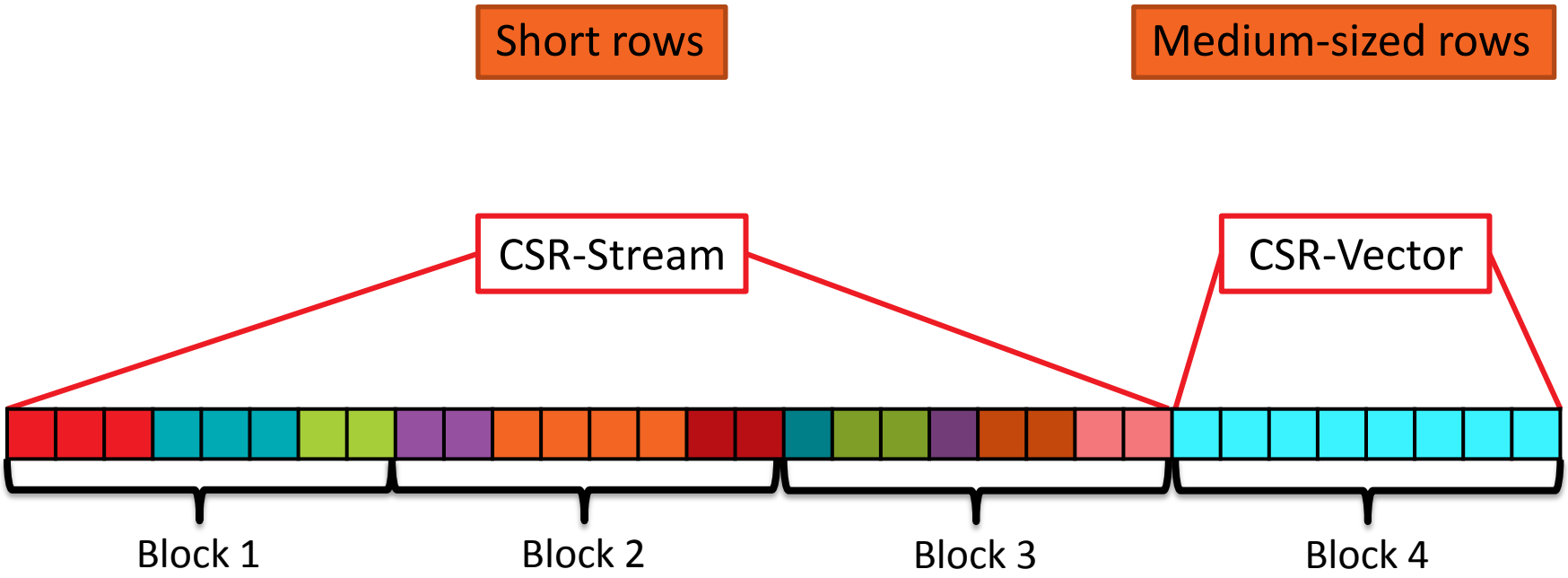
CSR-ADAPTIVE: SHORT AND MEDIUM ROWS



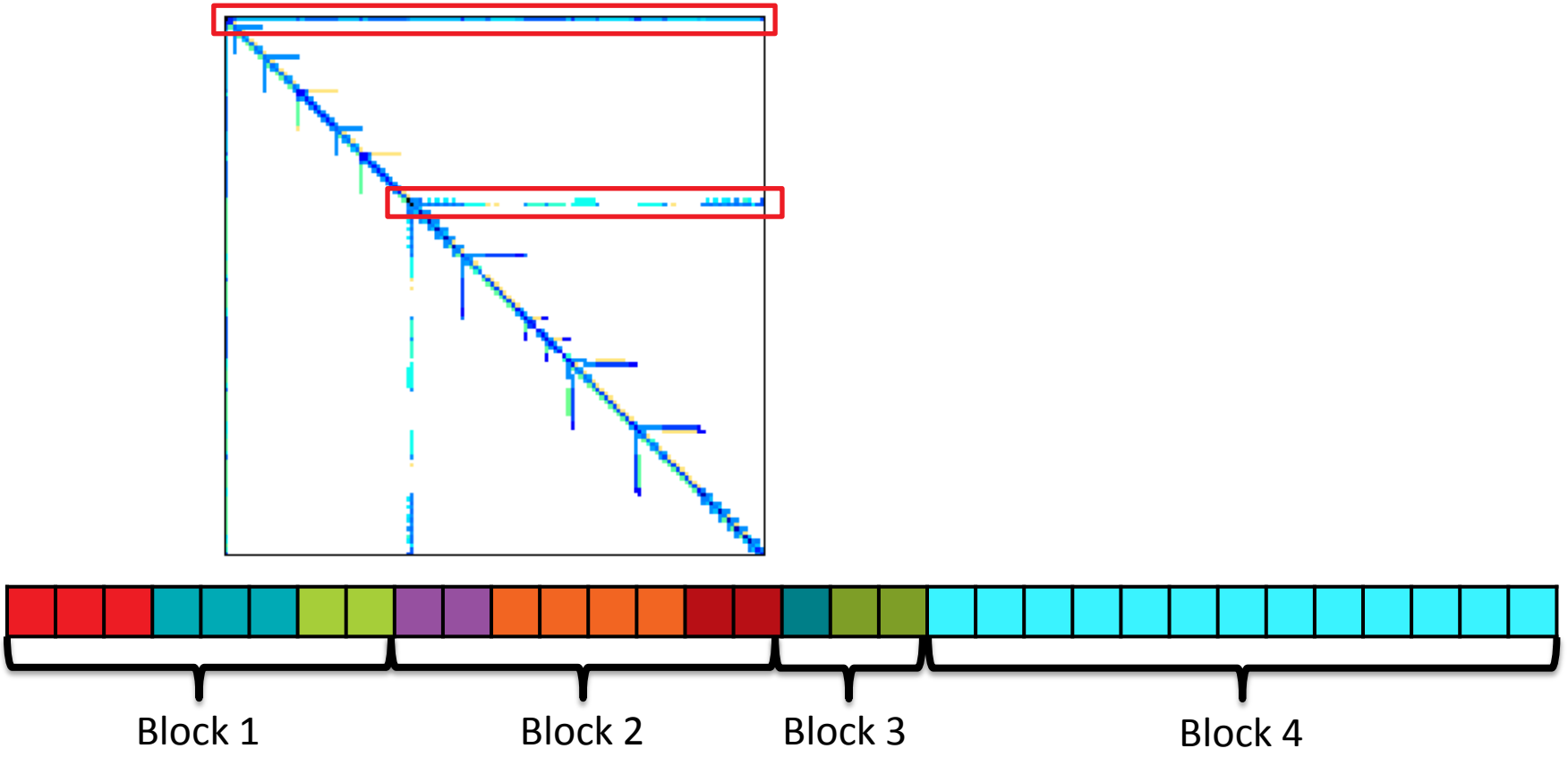
CSR-ADAPTIVE: SHORT AND MEDIUM ROWS



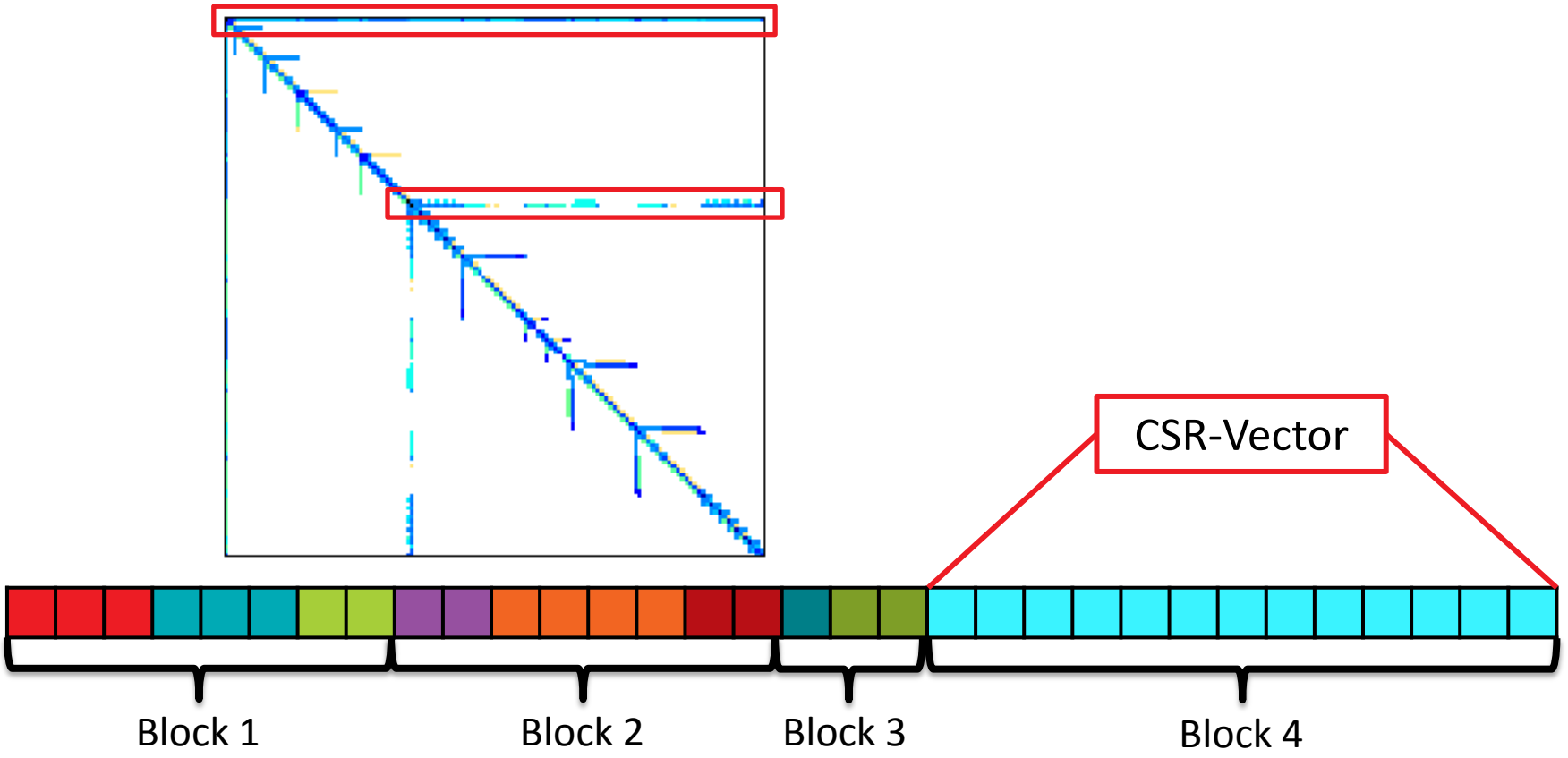
CSR-ADAPTIVE: SHORT AND MEDIUM ROWS



WHAT ABOUT VERY LONG ROWS?

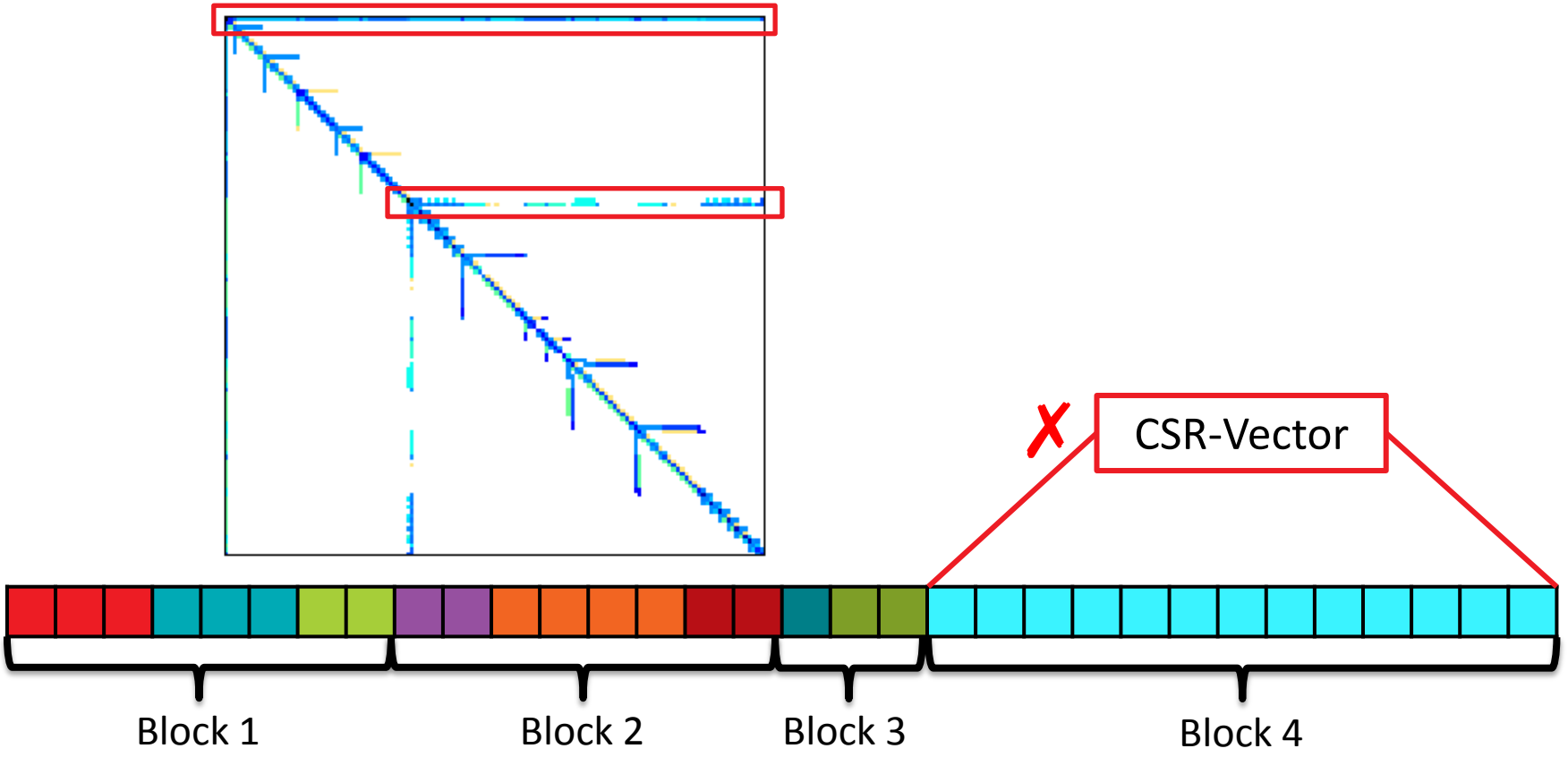


WHAT ABOUT VERY LONG ROWS?



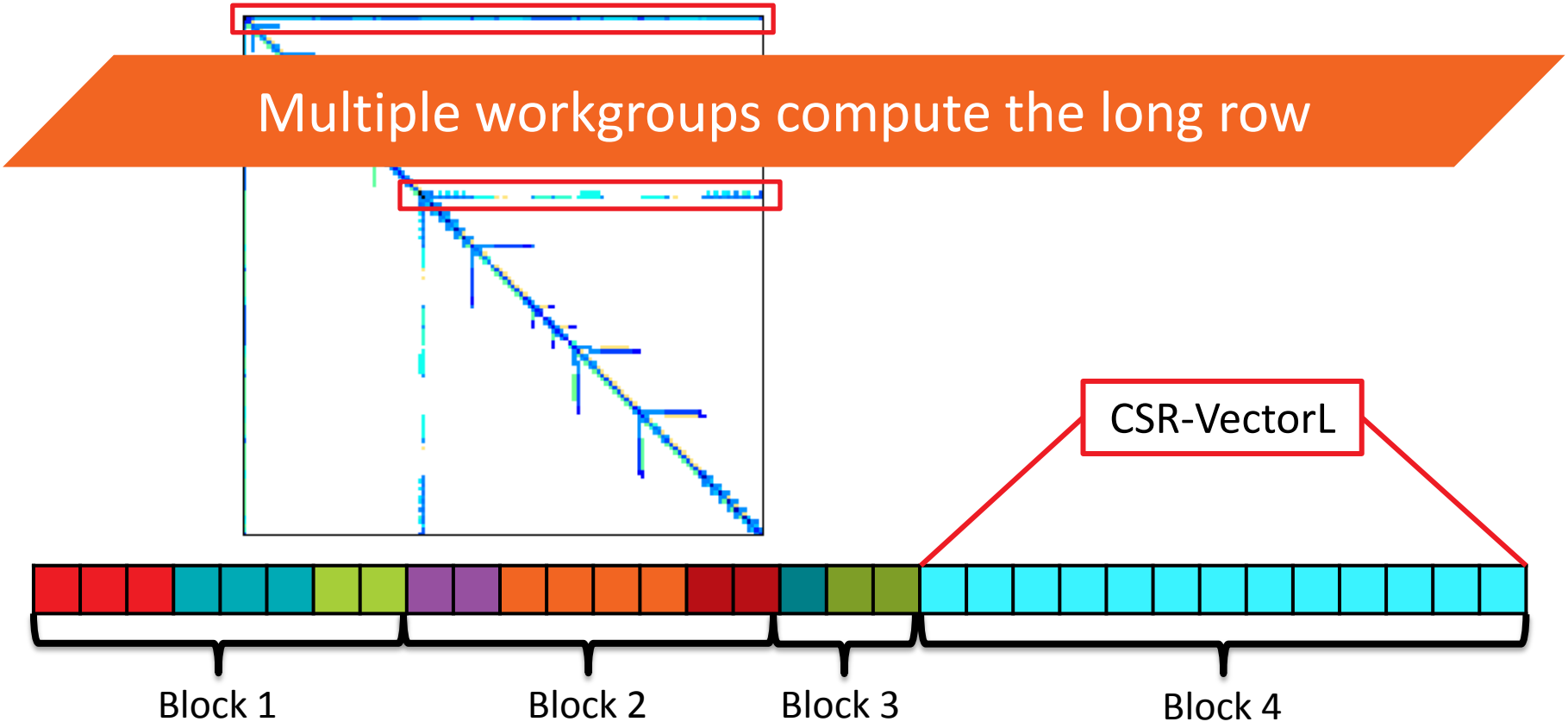
WHAT ABOUT VERY LONG ROWS?

- CSR-Vector: One workgroup can become a performance bottle-neck



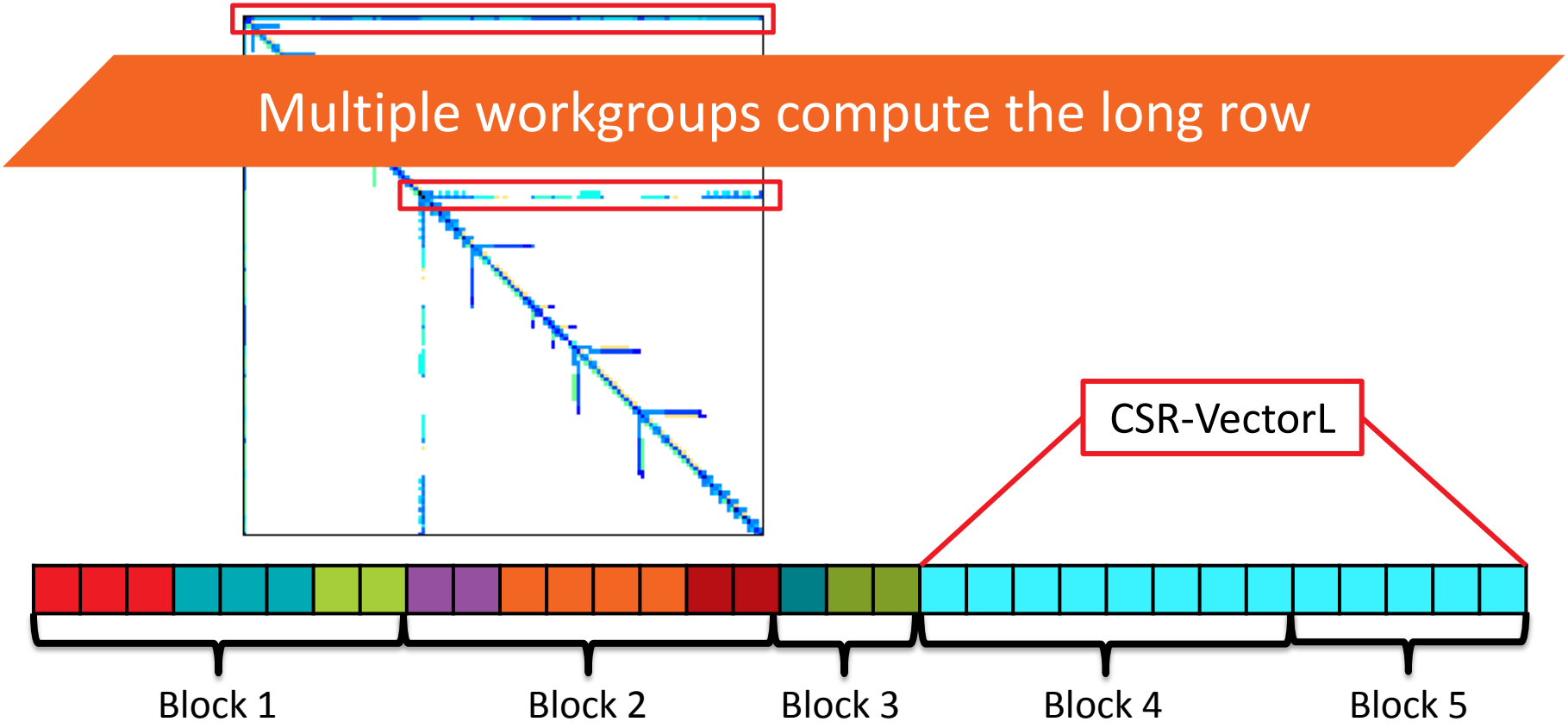
WHAT ABOUT VERY LONG ROWS?

- CSR-Vector: One workgroup can become a performance bottle-neck

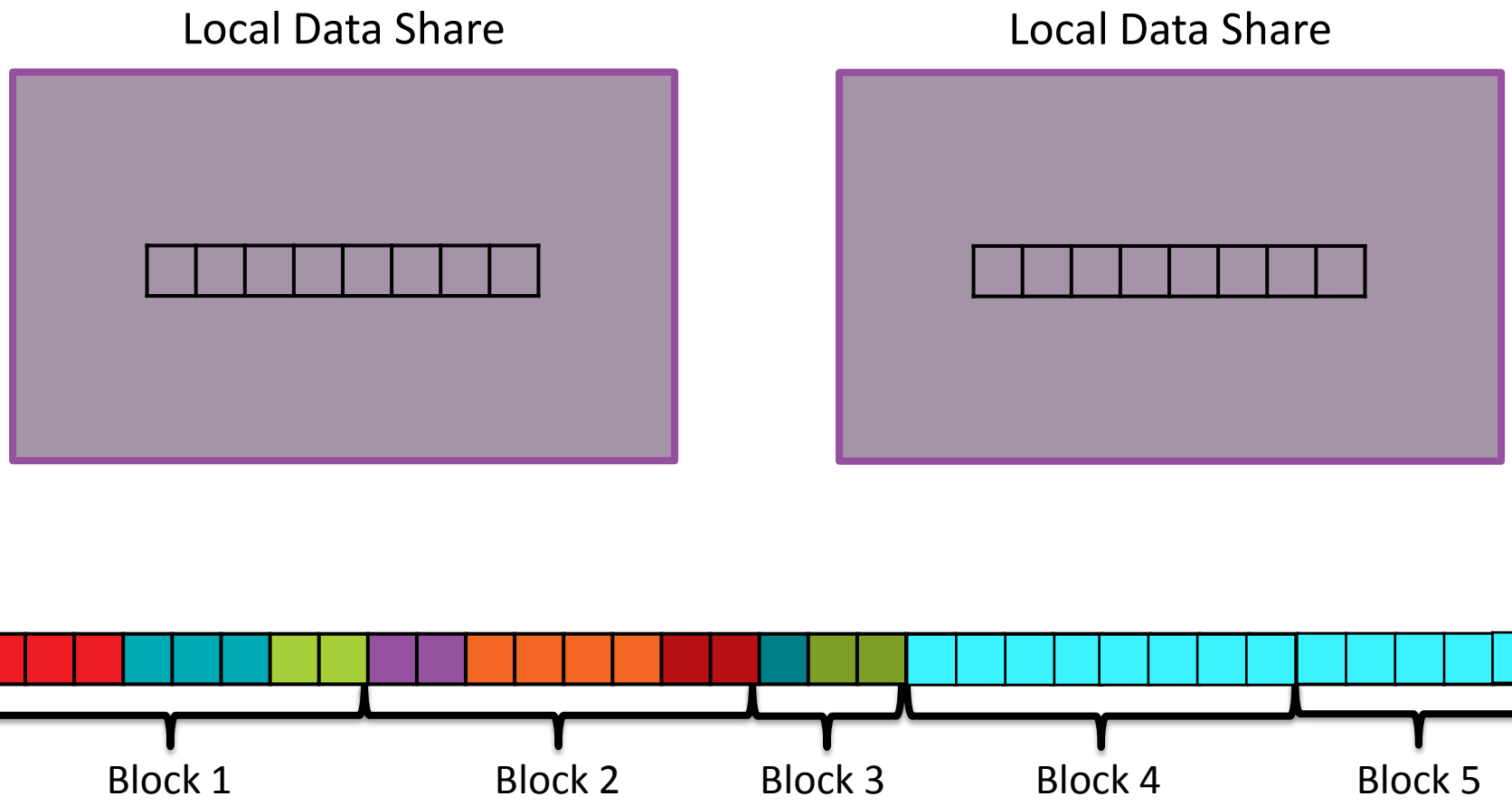


WHAT ABOUT VERY LONG ROWS?

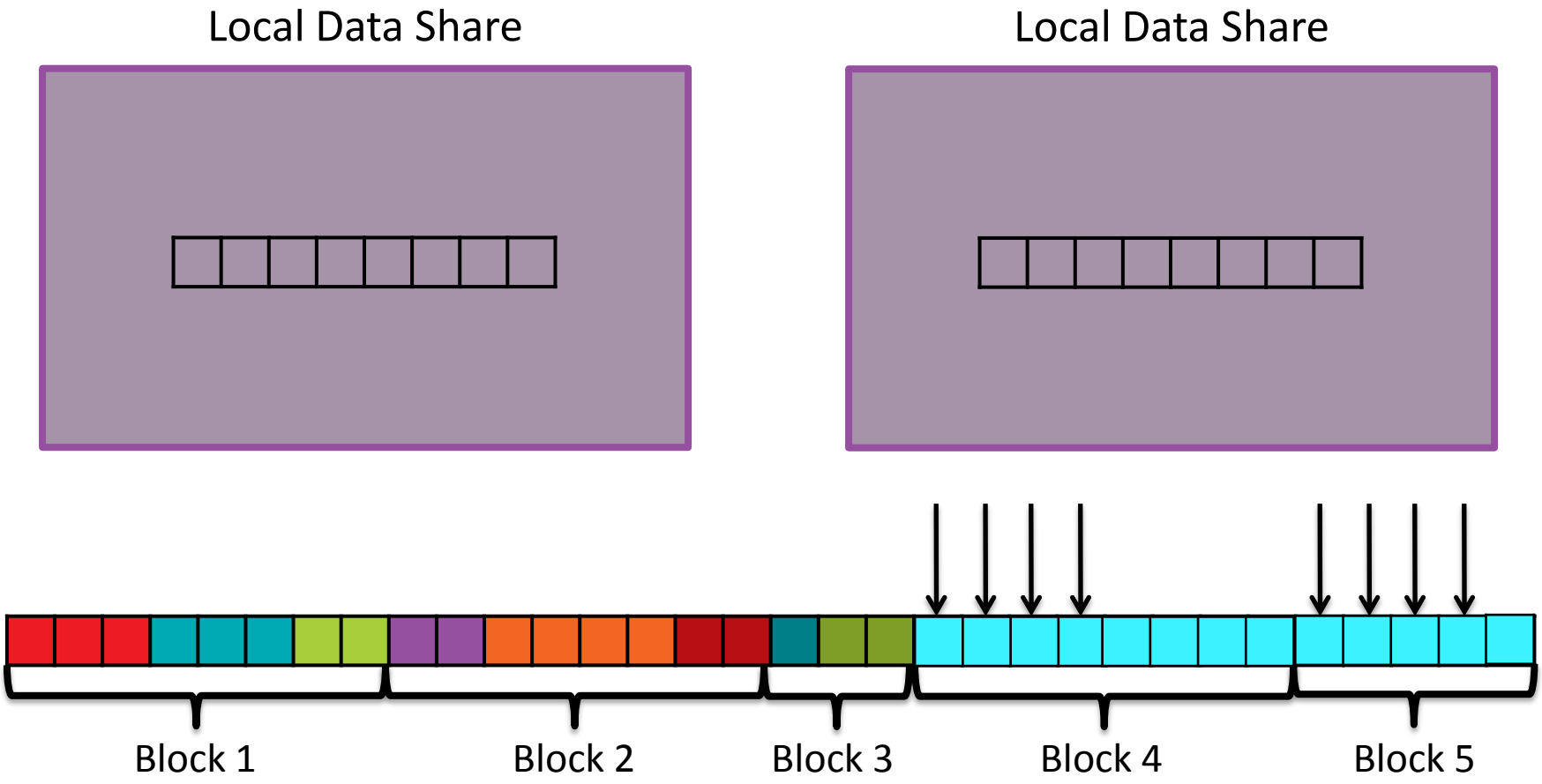
- CSR-Vector: One workgroup can become a performance bottle-neck



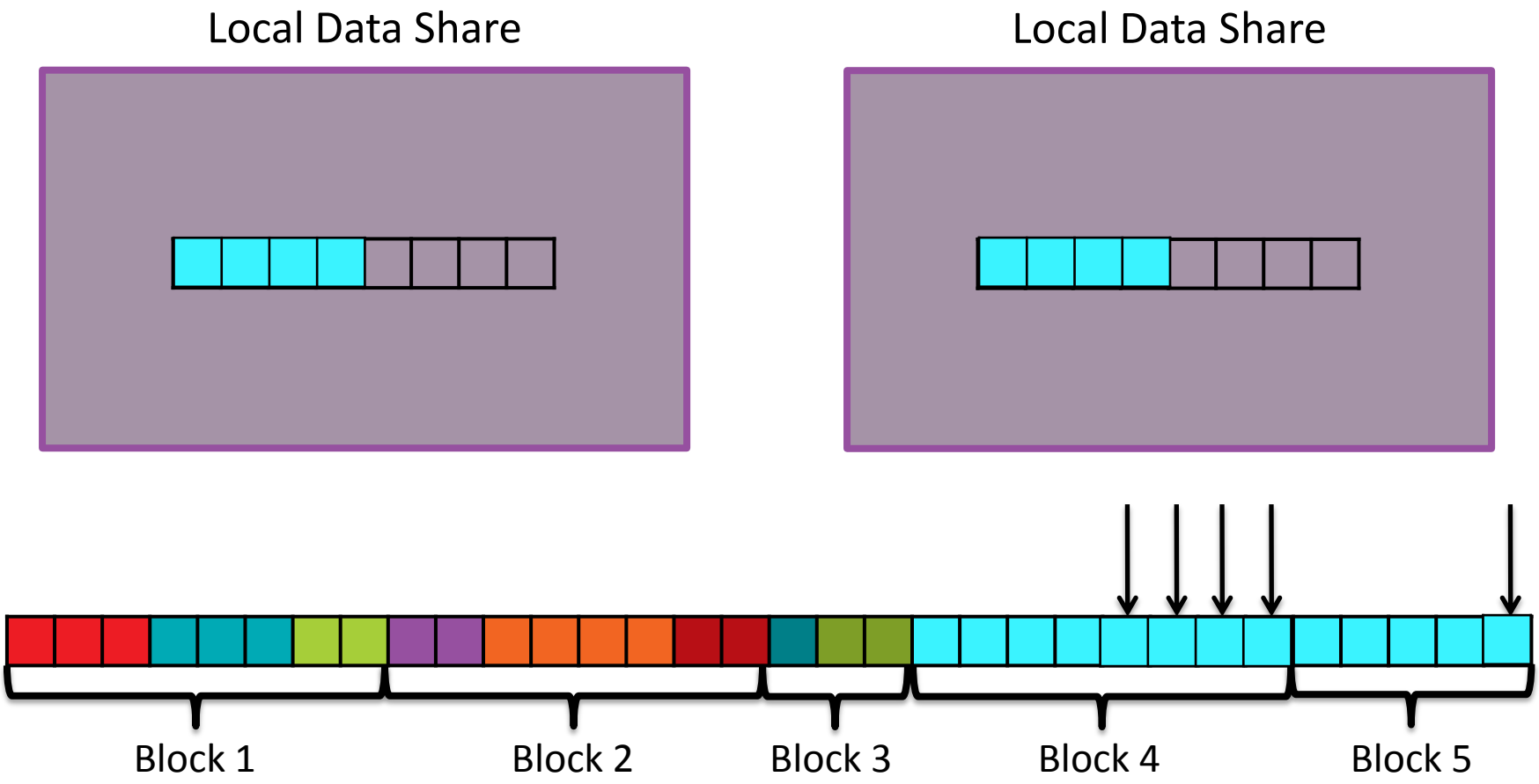
CSR-VECTORL (LONG ROWS)



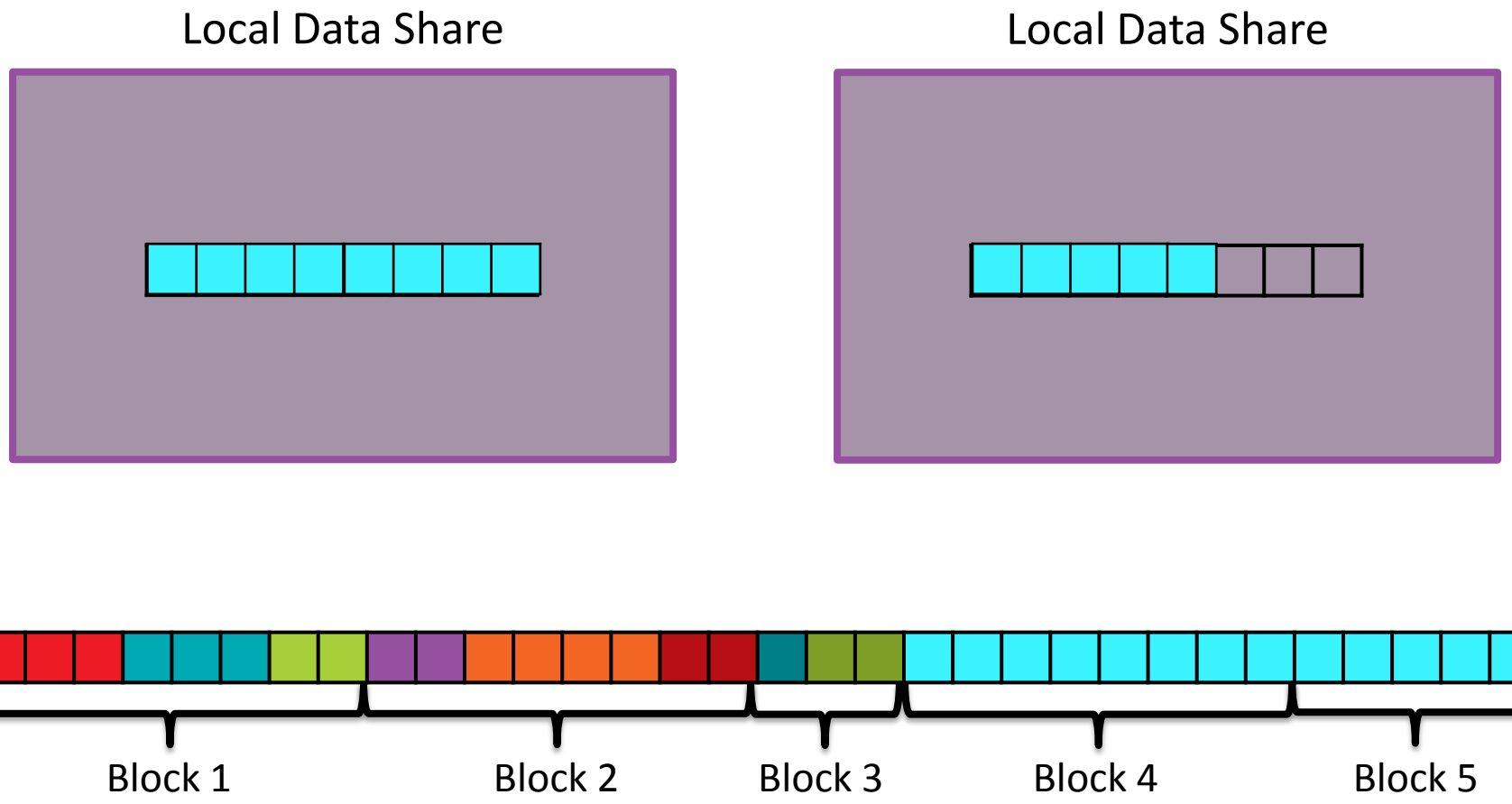
CSR-VECTORL (LONG ROWS)



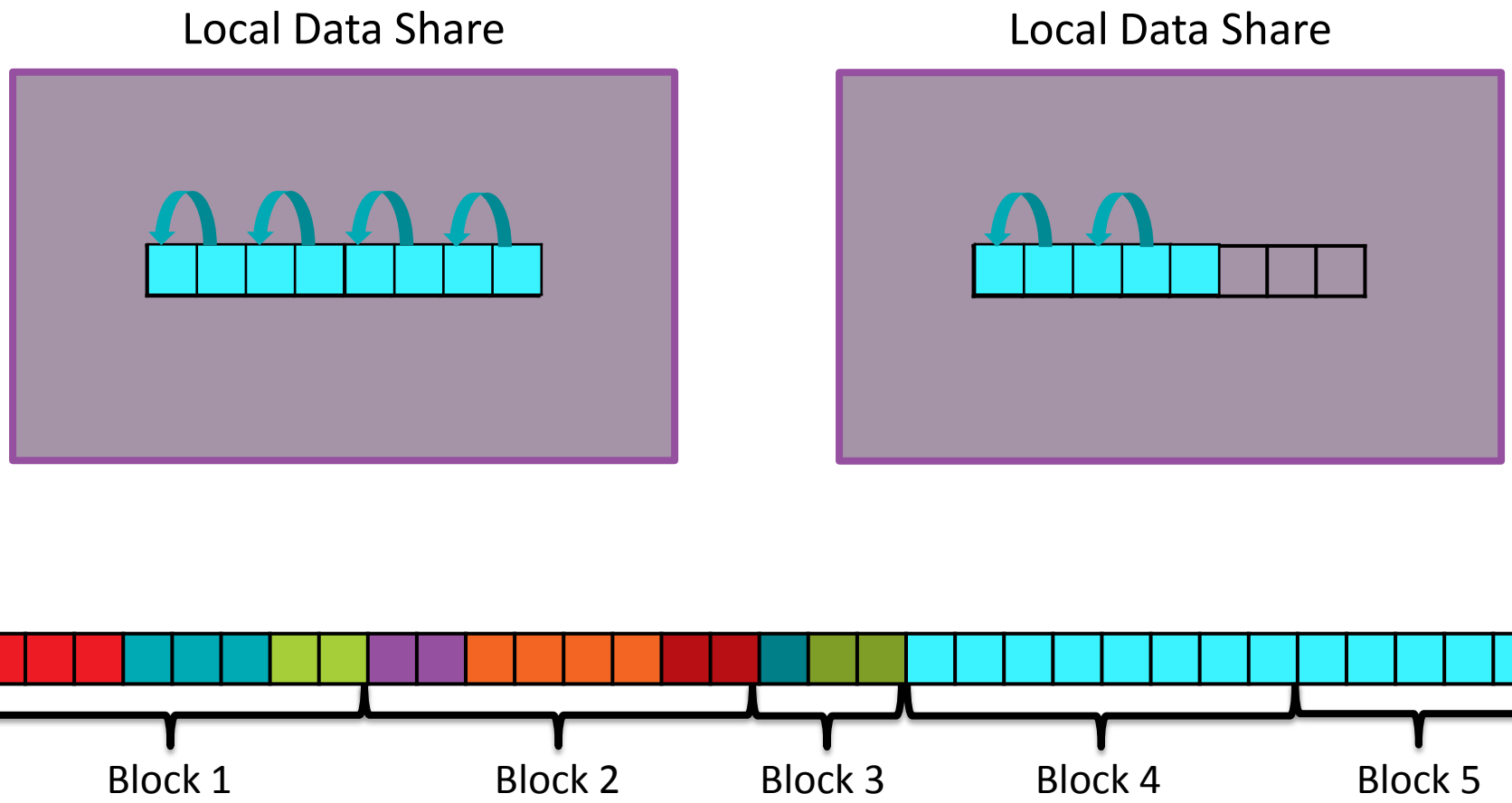
CSR-VECTORL (LONG ROWS)



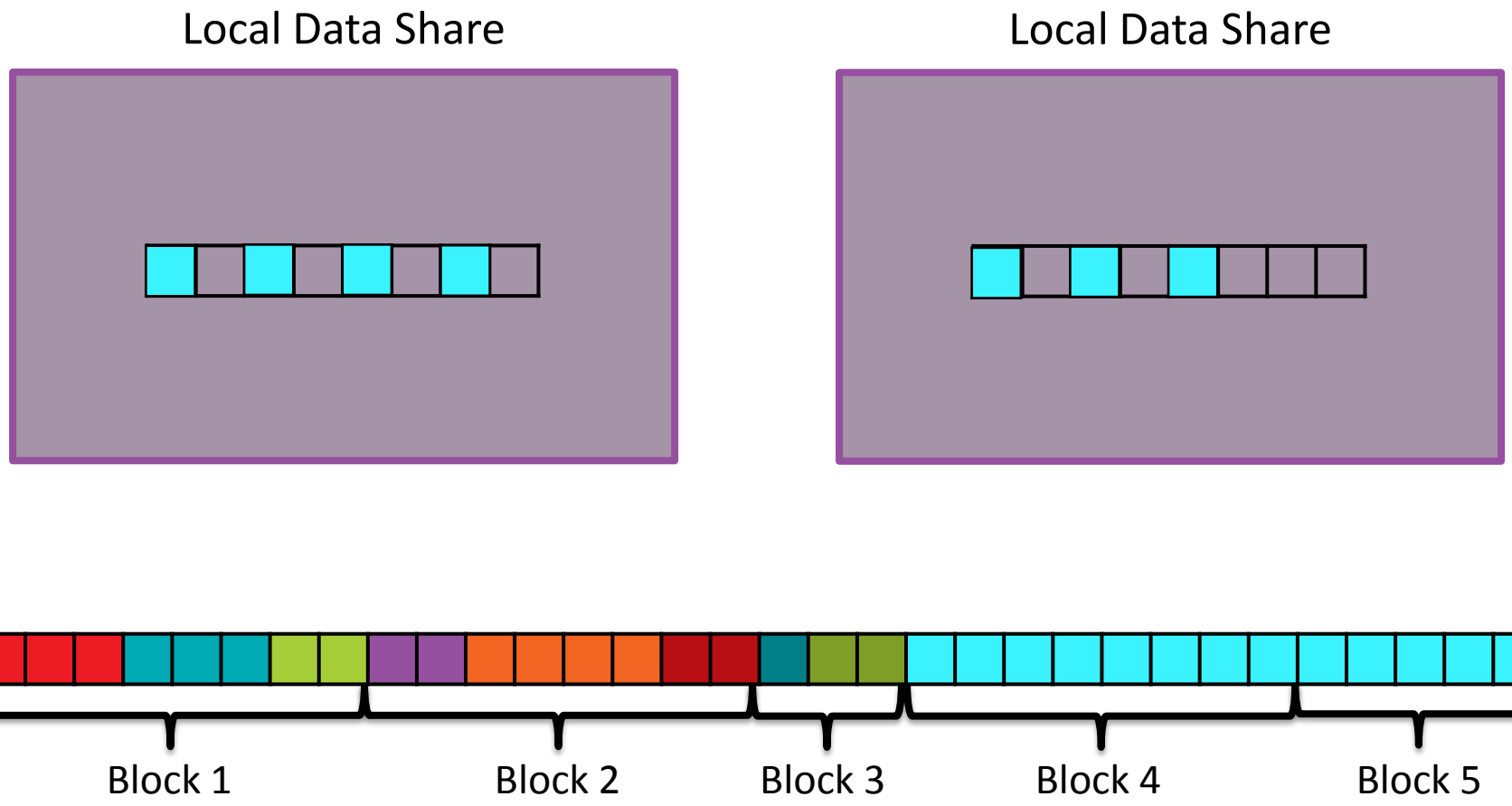
CSR-VECTORL (LONG ROWS)



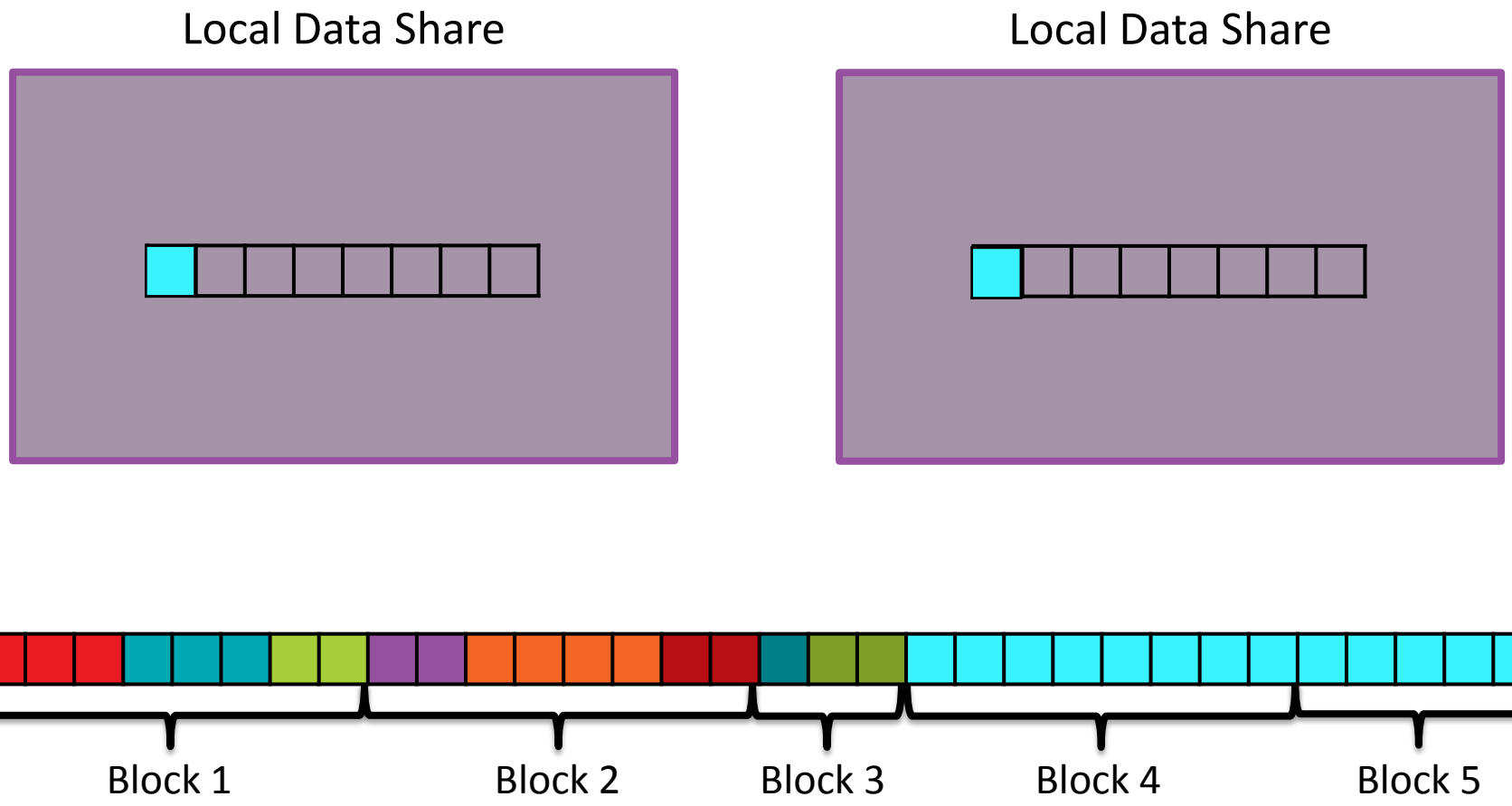
CSR-VECTORL (LONG ROWS)



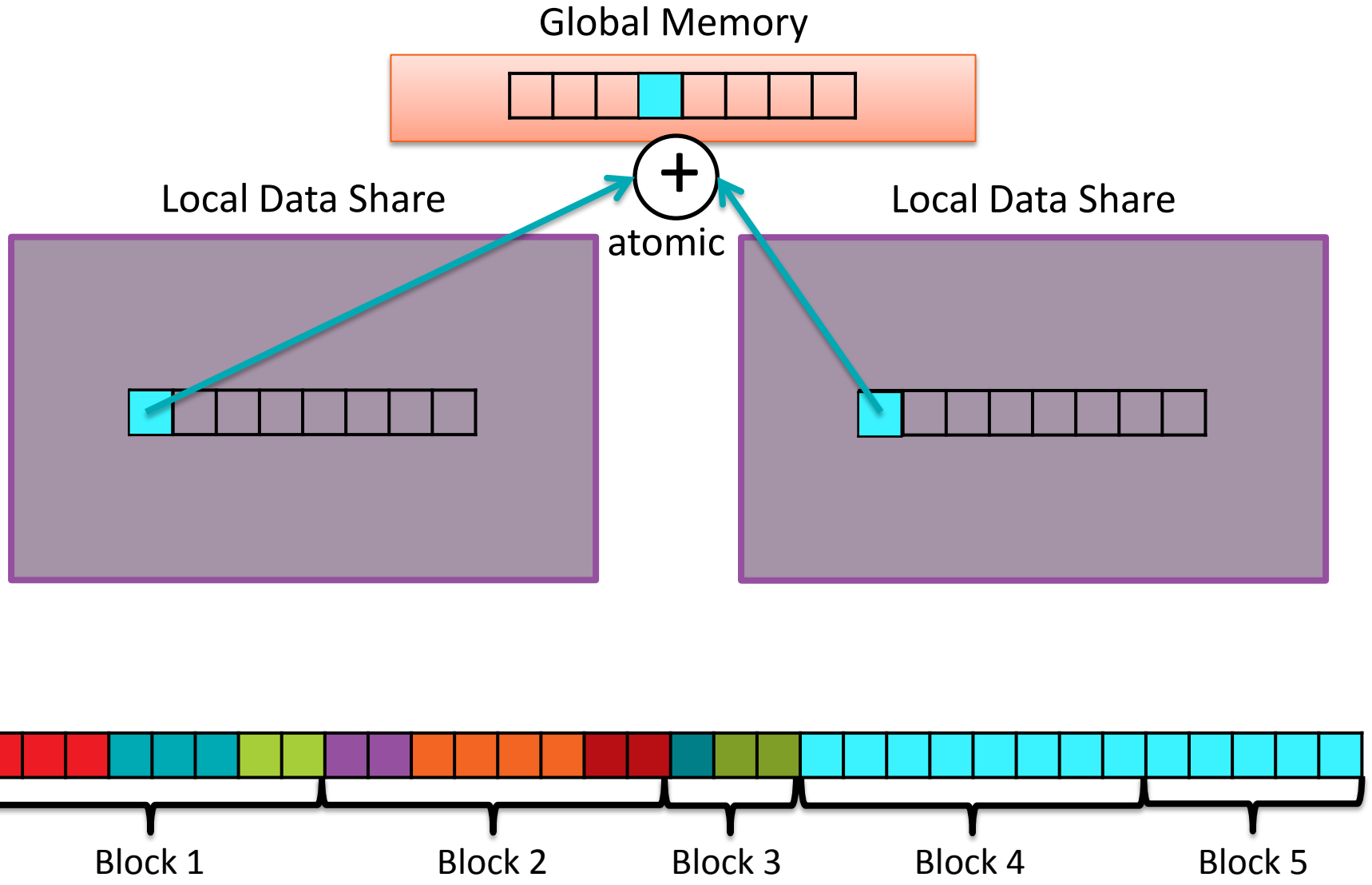
CSR-VECTORL (LONG ROWS)



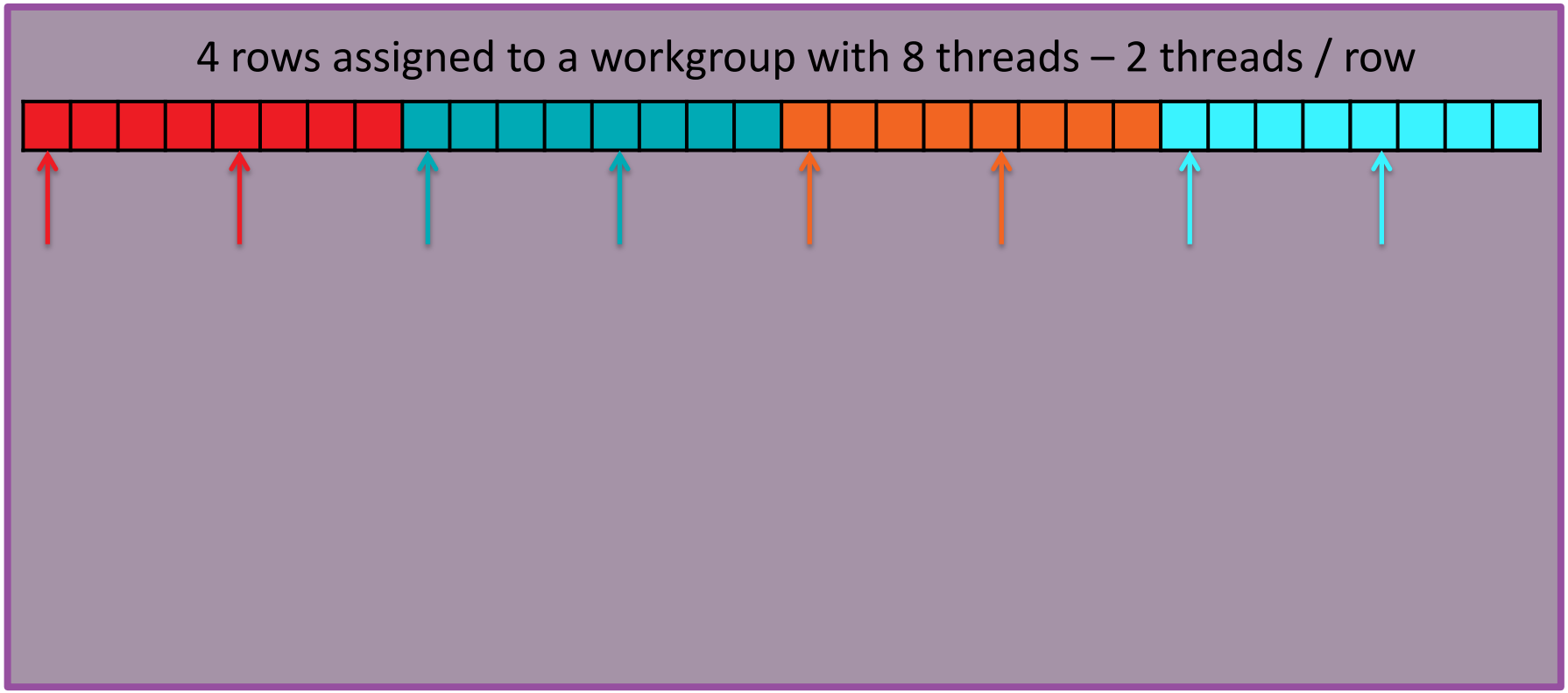
CSR-VECTORL (LONG ROWS)



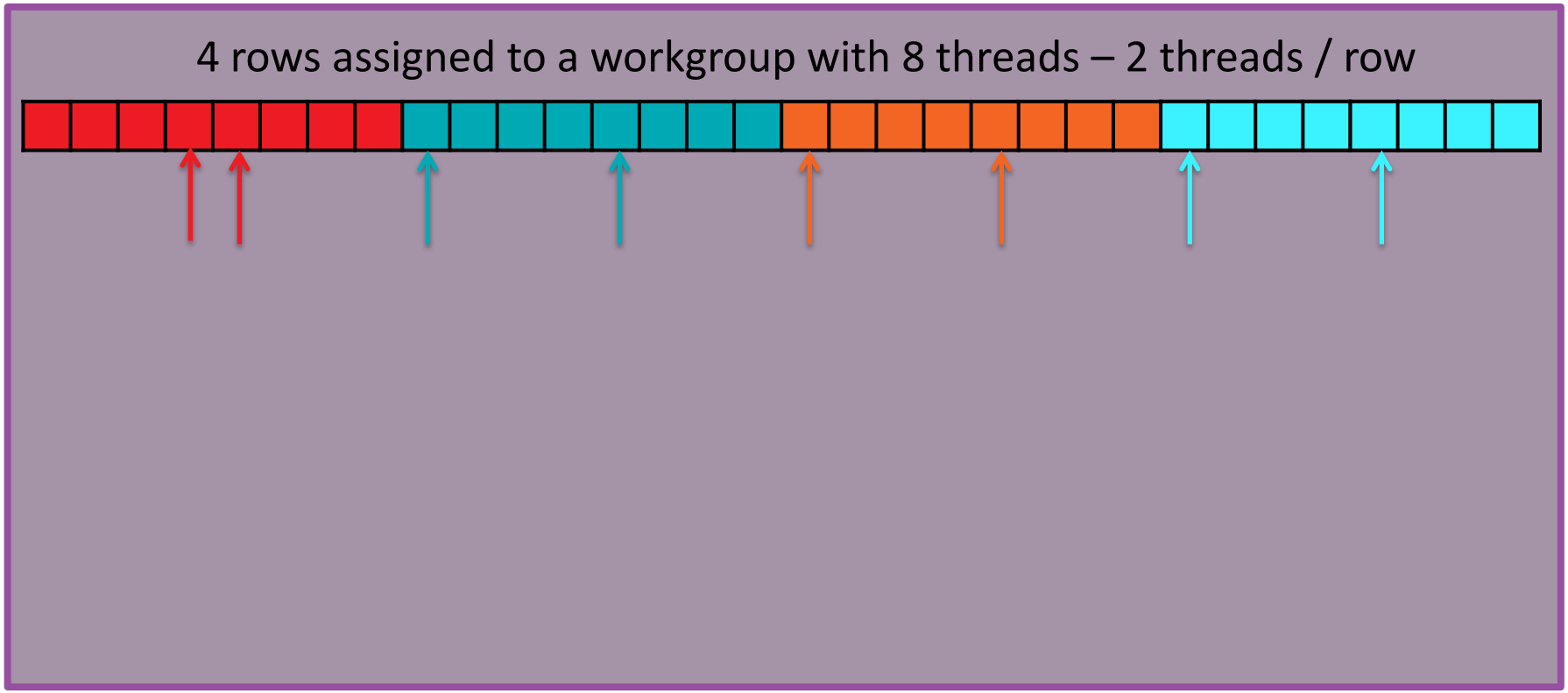
CSR-VECTORL (LONG ROWS)



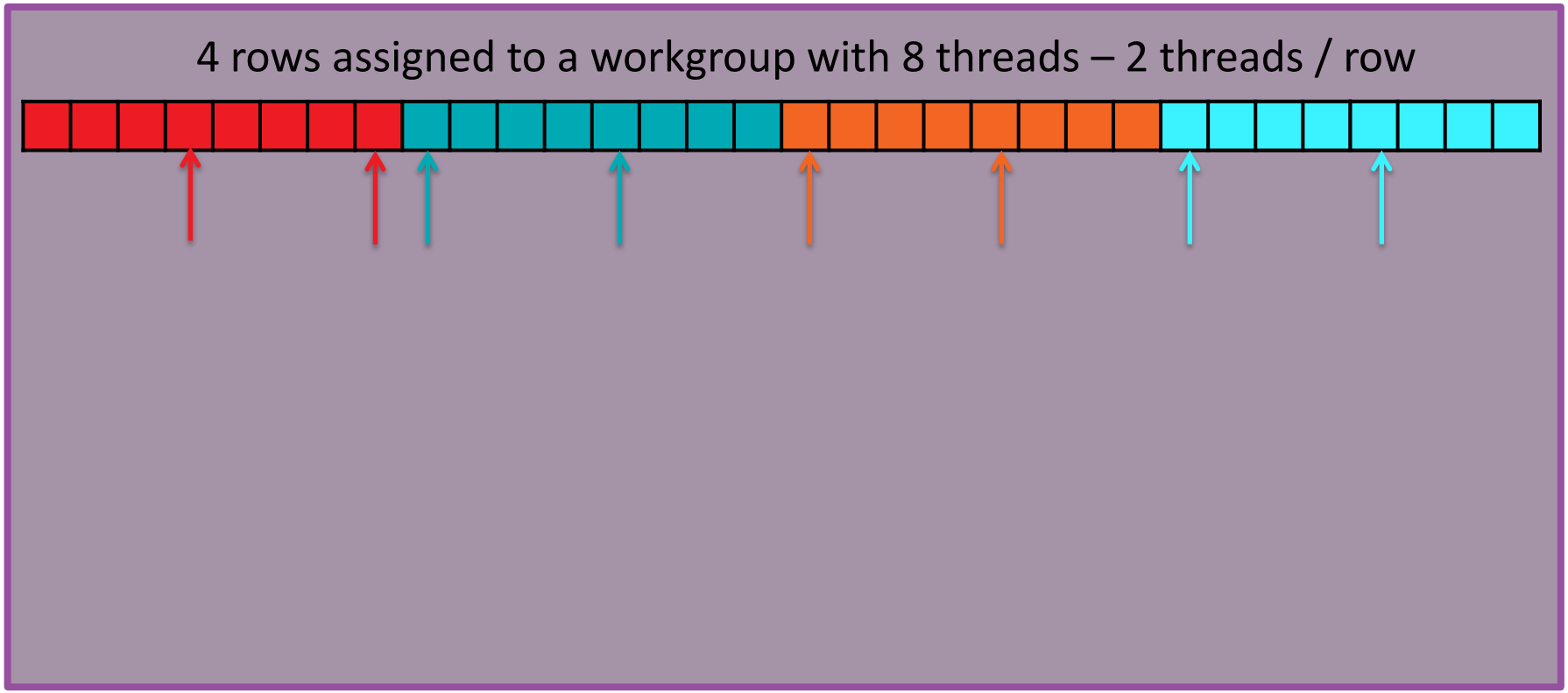
OPTIMIZED CSR-STREAM: LOGARITHMIC REDUCTION



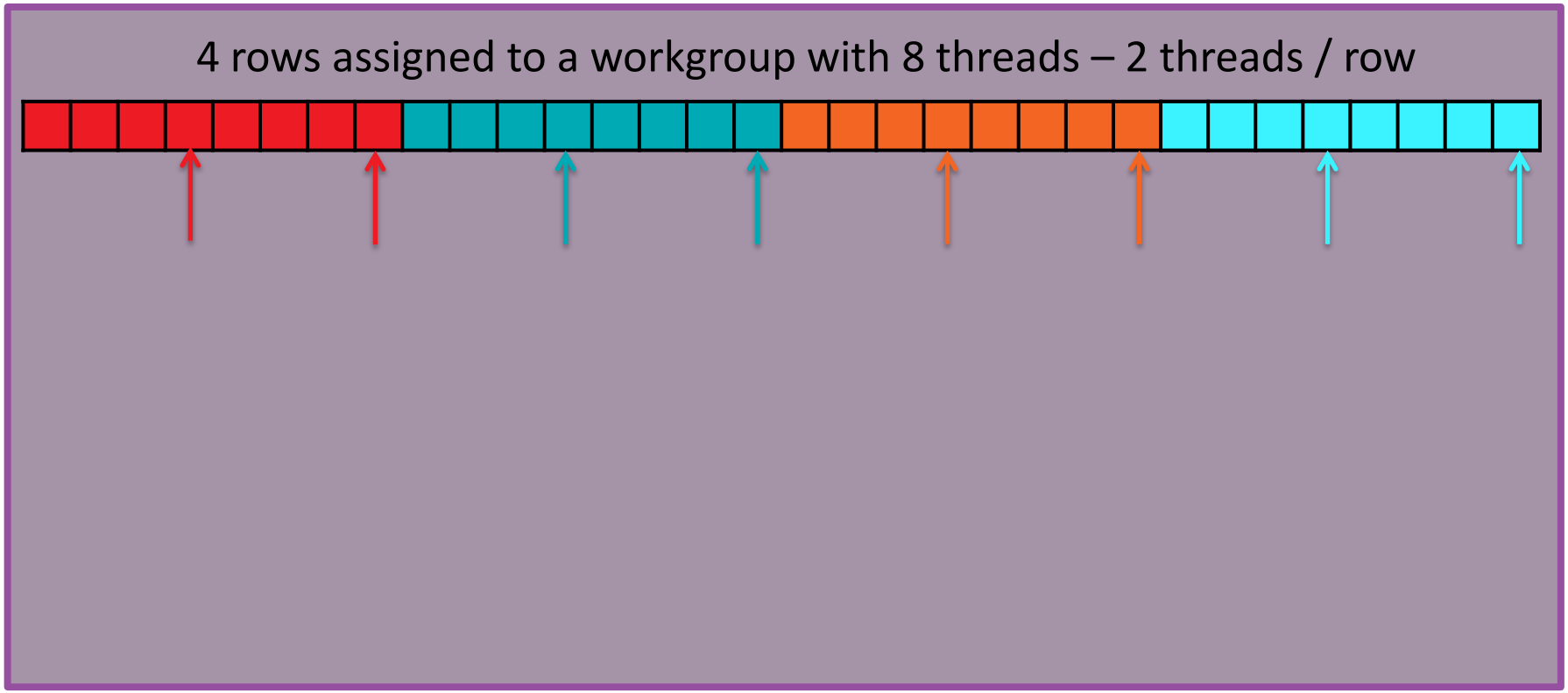
OPTIMIZED CSR-STREAM: LOGARITHMIC REDUCTION



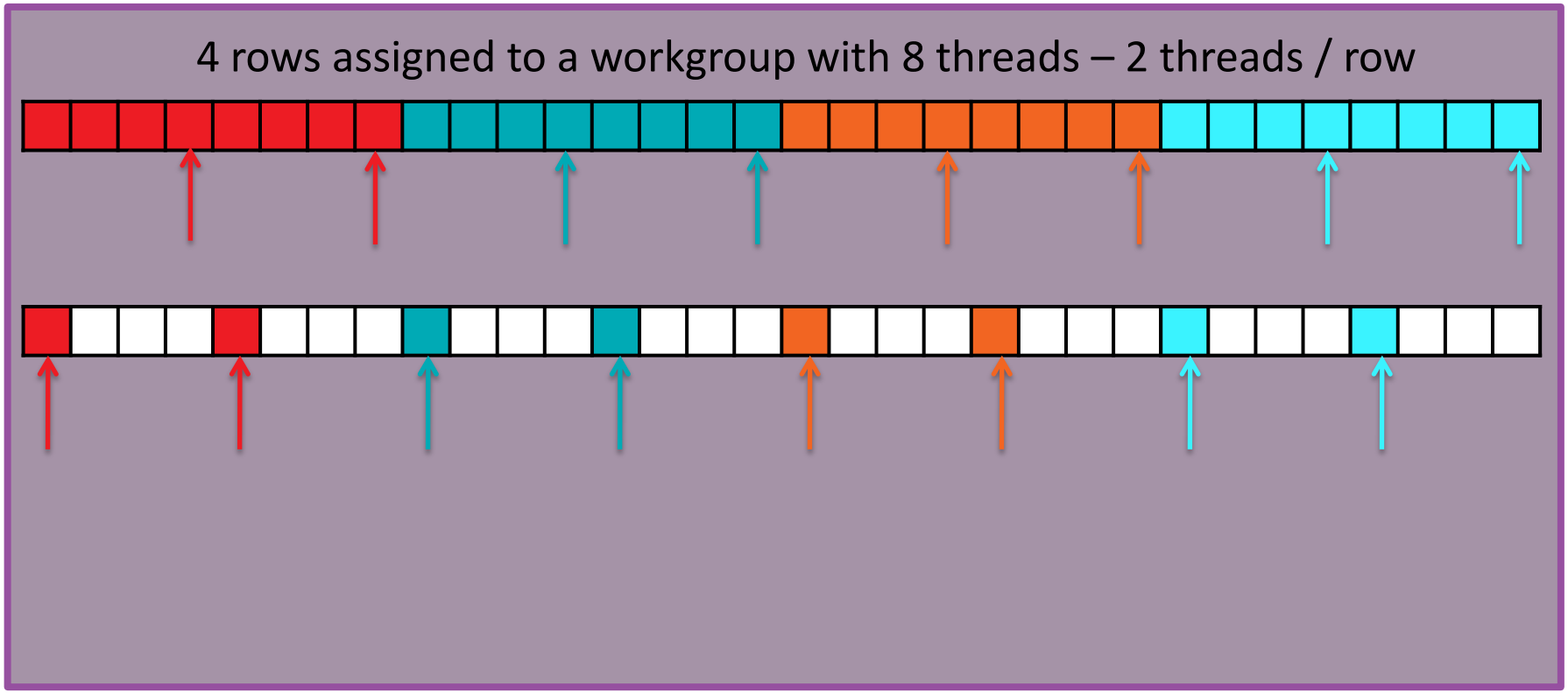
OPTIMIZED CSR-STREAM: LOGARITHMIC REDUCTION



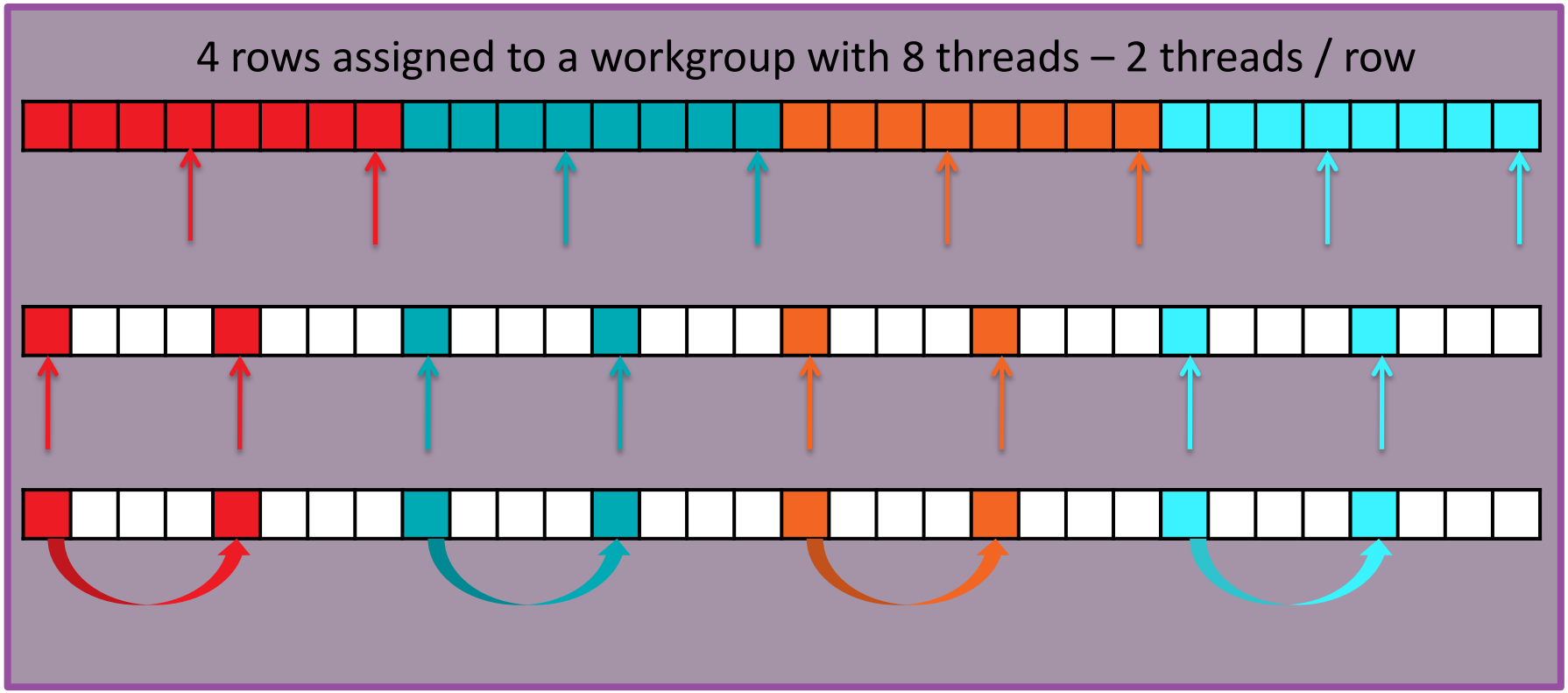
OPTIMIZED CSR-STREAM: LOGARITHMIC REDUCTION



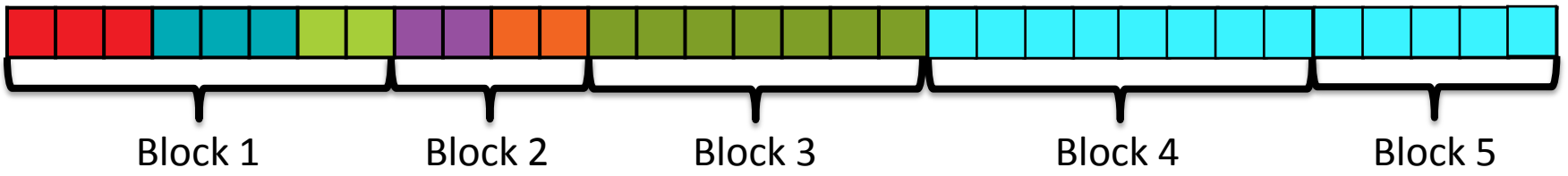
OPTIMIZED CSR-STREAM: LOGARITHMIC REDUCTION



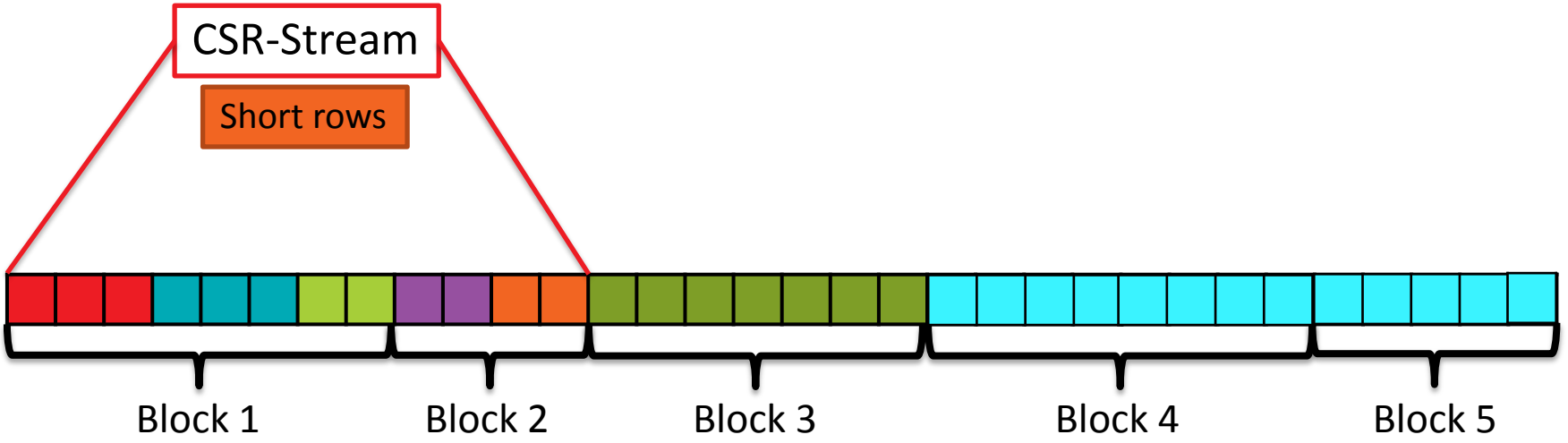
OPTIMIZED CSR-STREAM: LOGARITHMIC REDUCTION



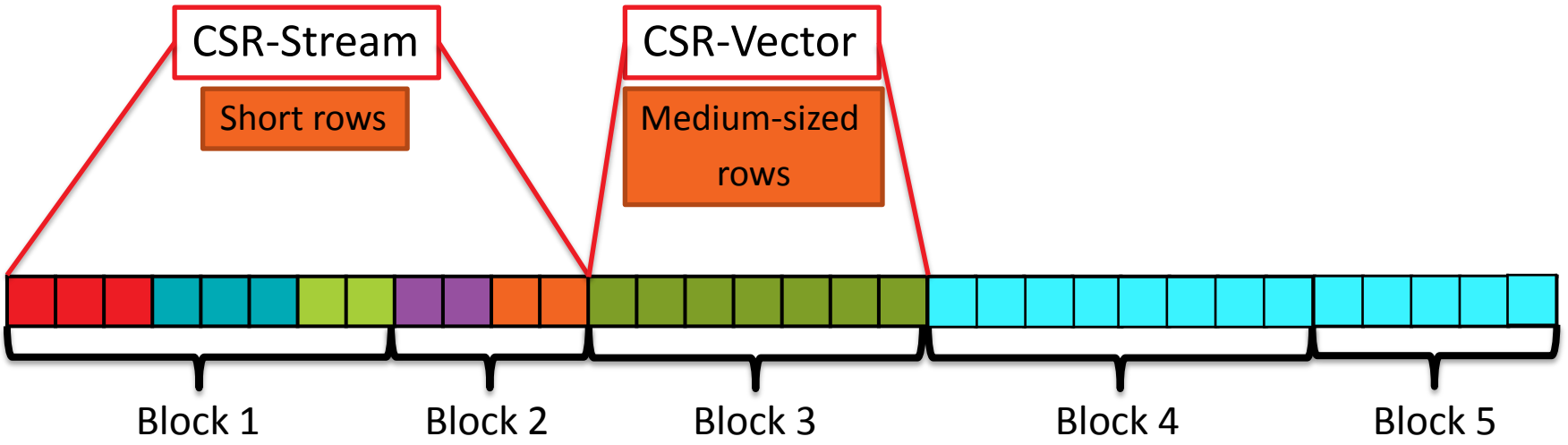
CSR-ADAPTIVE



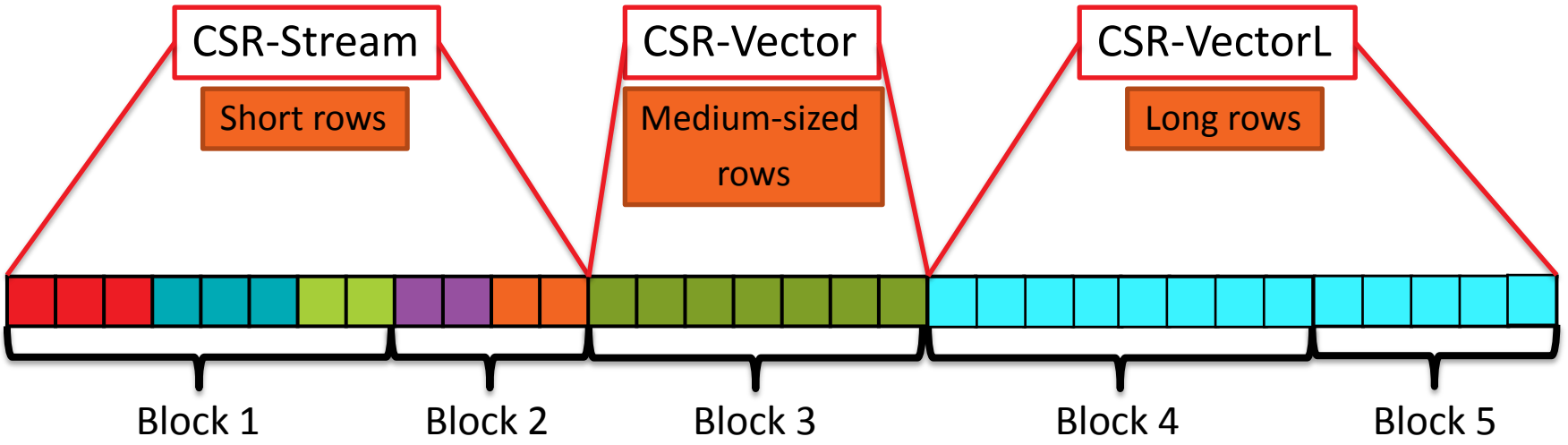
CSR-ADAPTIVE



CSR-ADAPTIVE



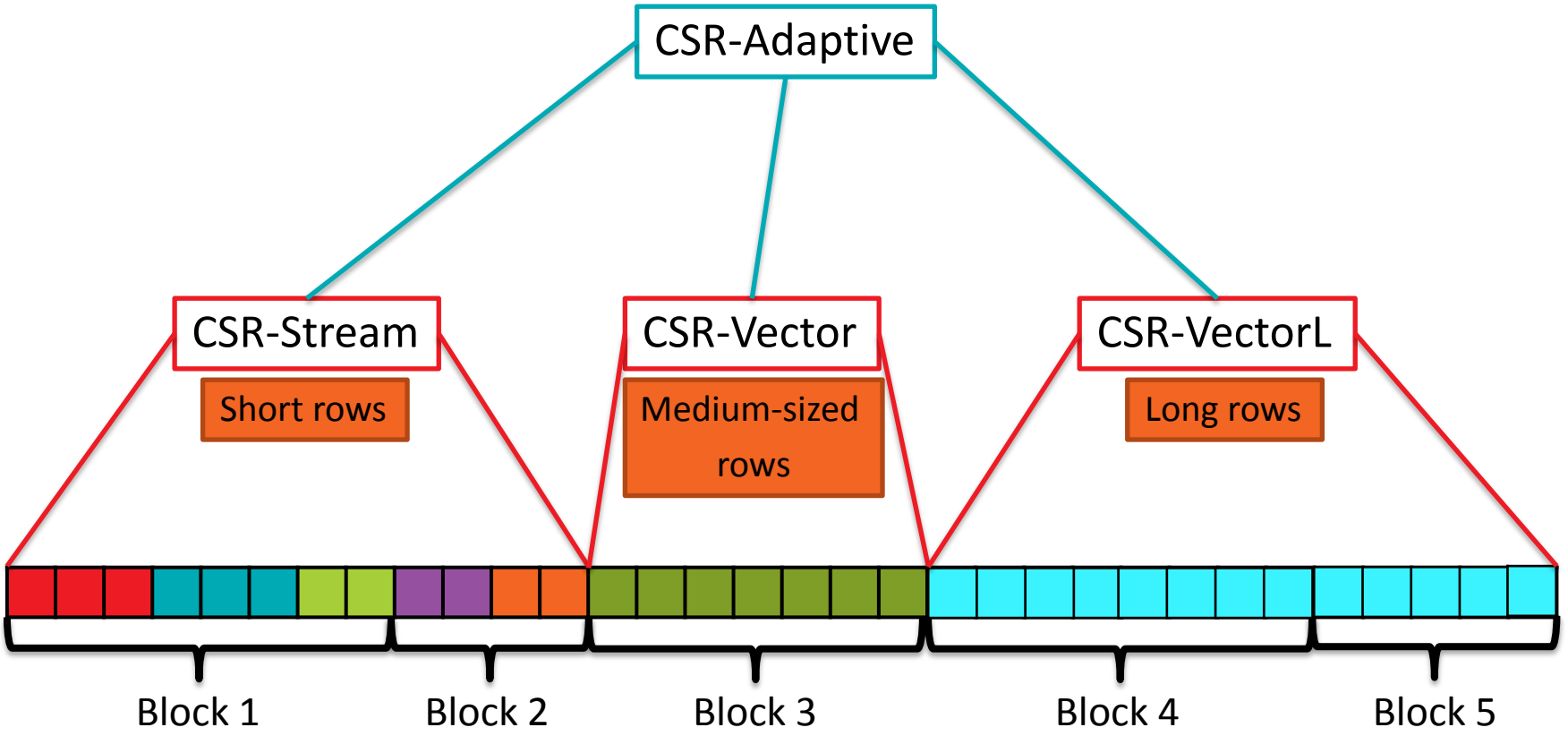
CSR-ADAPTIVE



CSR-ADAPTIVE

AMD

A **complete** SpMV solution



EXPERIMENTAL SETUP

AMD

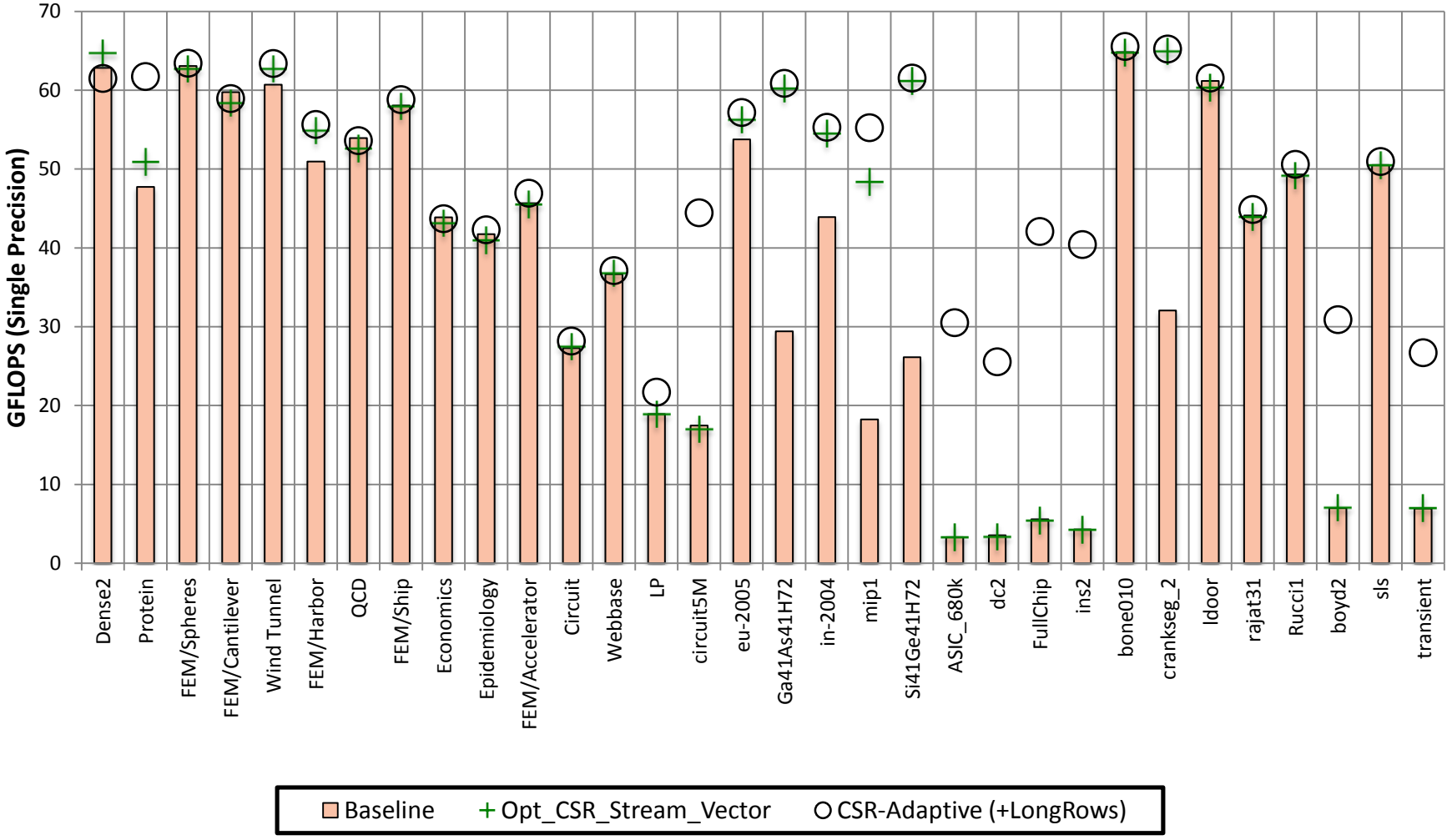
- CPU: AMD A10-7850K
  - 32 GB DDR3-2133 SDRAM
- GPU: AMD FirePro™ W9100
  - 44 parallel 64-wide compute units (CUs) – (2816 ALUs)
  - 5.2 single-precision TFLOPs / 2.6 double-precision TFLOPS
  - 320 GB/s
- 32 different input matrices used in previous works
- AMD APP SDK v2.9
- AMD FirePro driver v14.20 Beta on Ubuntu 14.04.2

CSR-ADAPTIVE OPTIMIZATIONS

AMD FIREPRO W9100 GPU

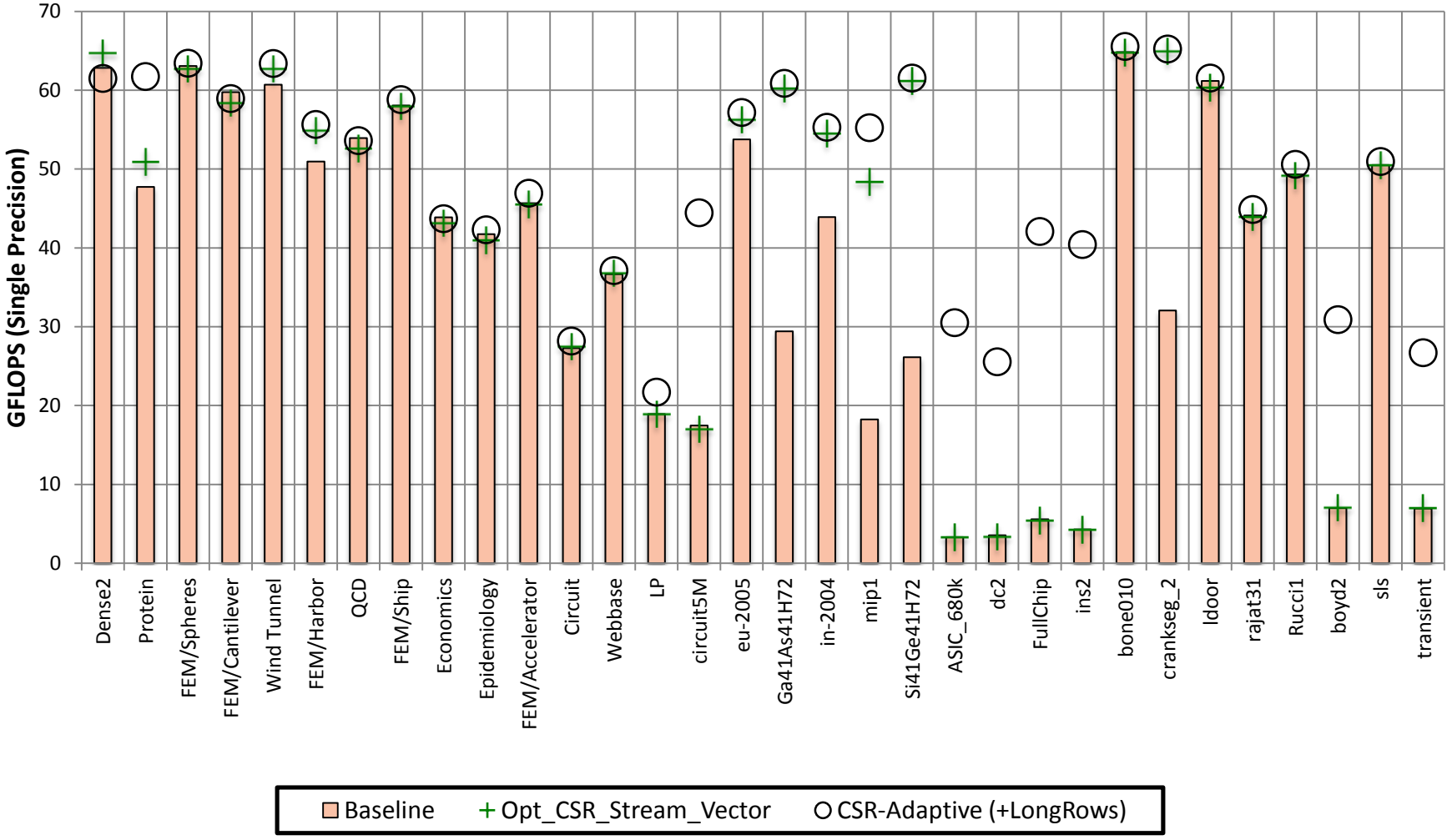
CSR-ADAPTIVE OPTIMIZATIONS

AMD FIREPRO W9100 GPU



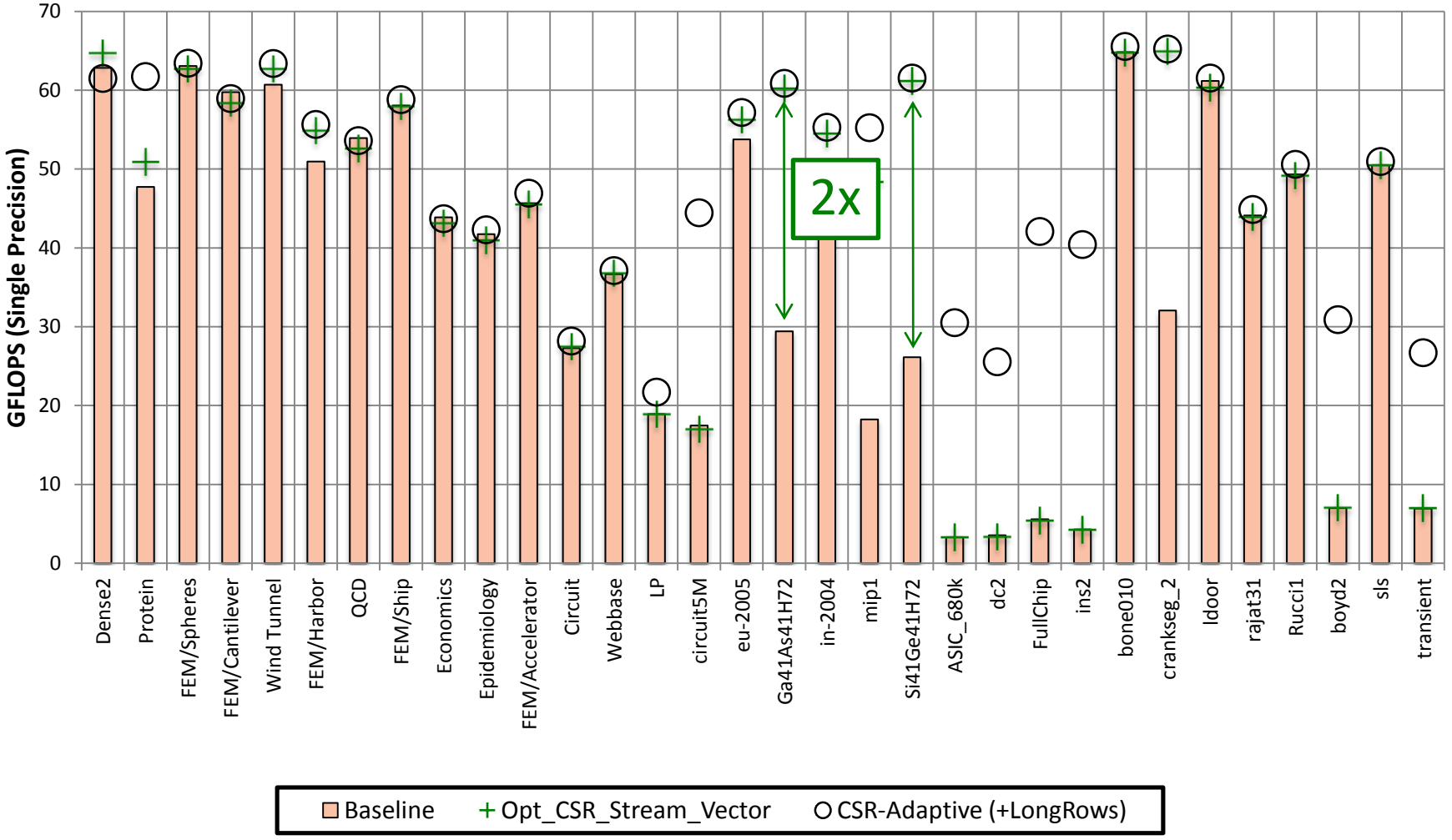
CSR-ADAPTIVE OPTIMIZATIONS

AMD FIREPRO W9100 GPU



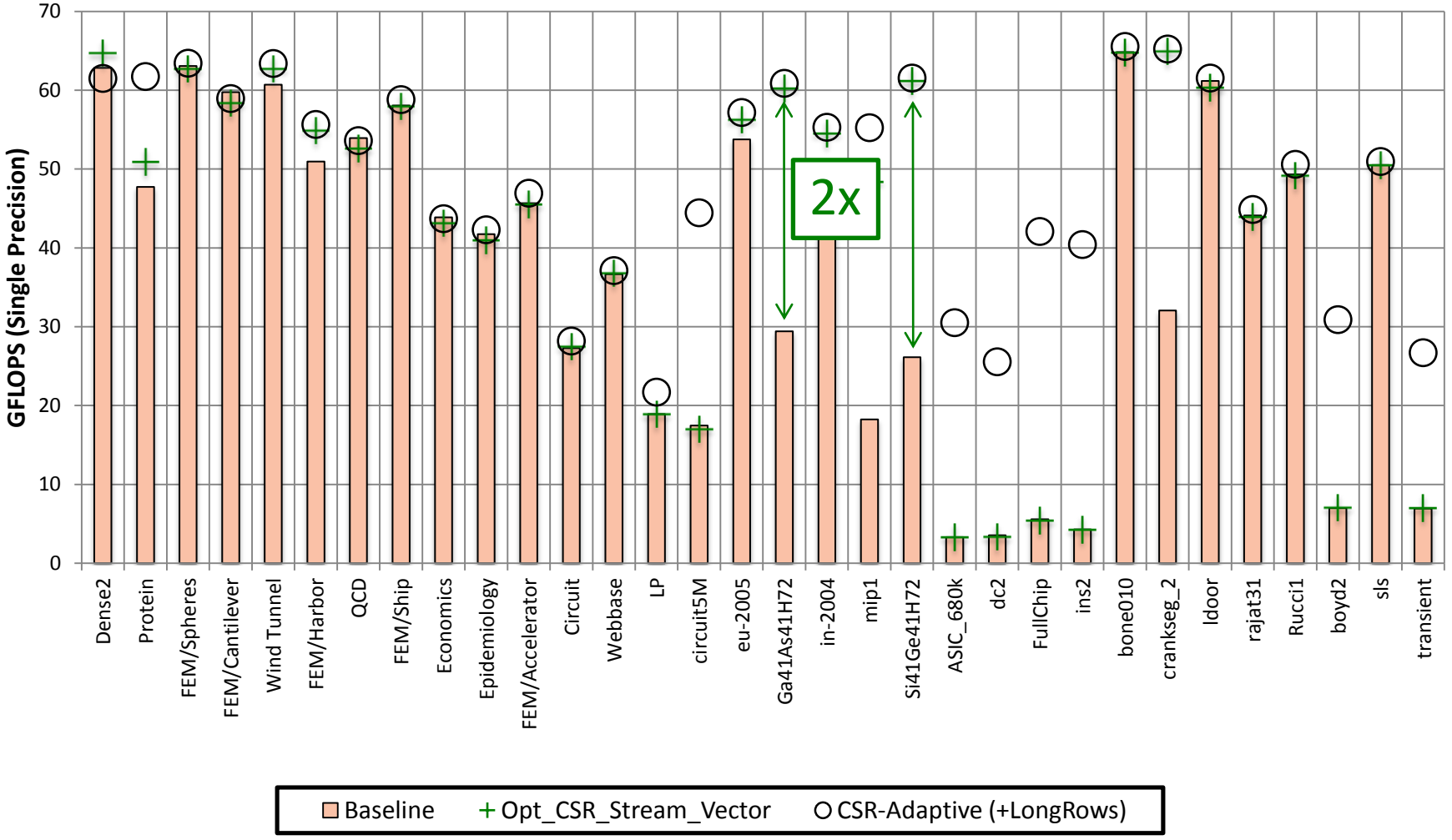
CSR-ADAPTIVE OPTIMIZATIONS

AMD FIREPRO W9100 GPU



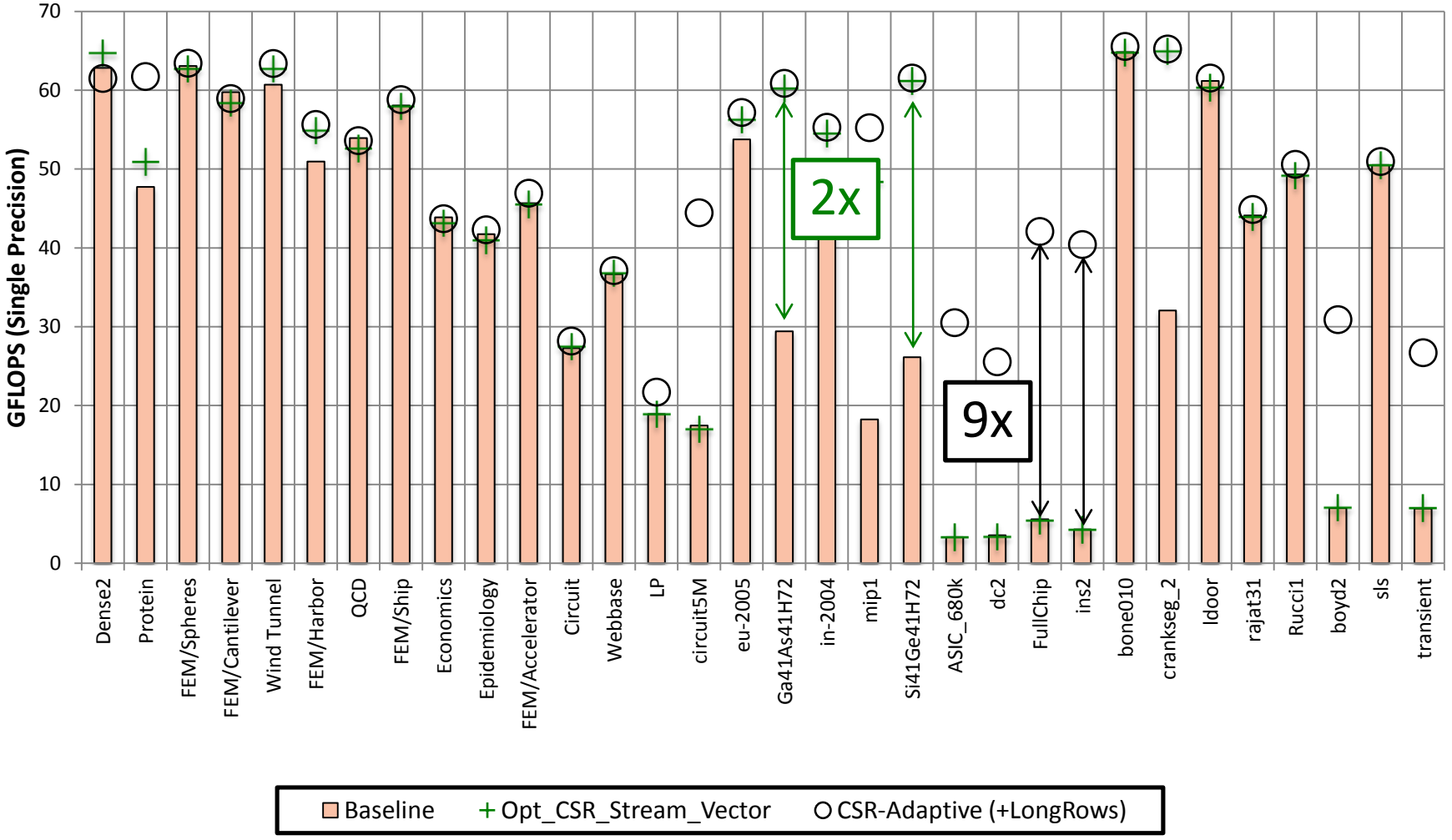
CSR-ADAPTIVE OPTIMIZATIONS

AMD FIREPRO W9100 GPU



CSR-ADAPTIVE OPTIMIZATIONS

AMD FIREPRO W9100 GPU

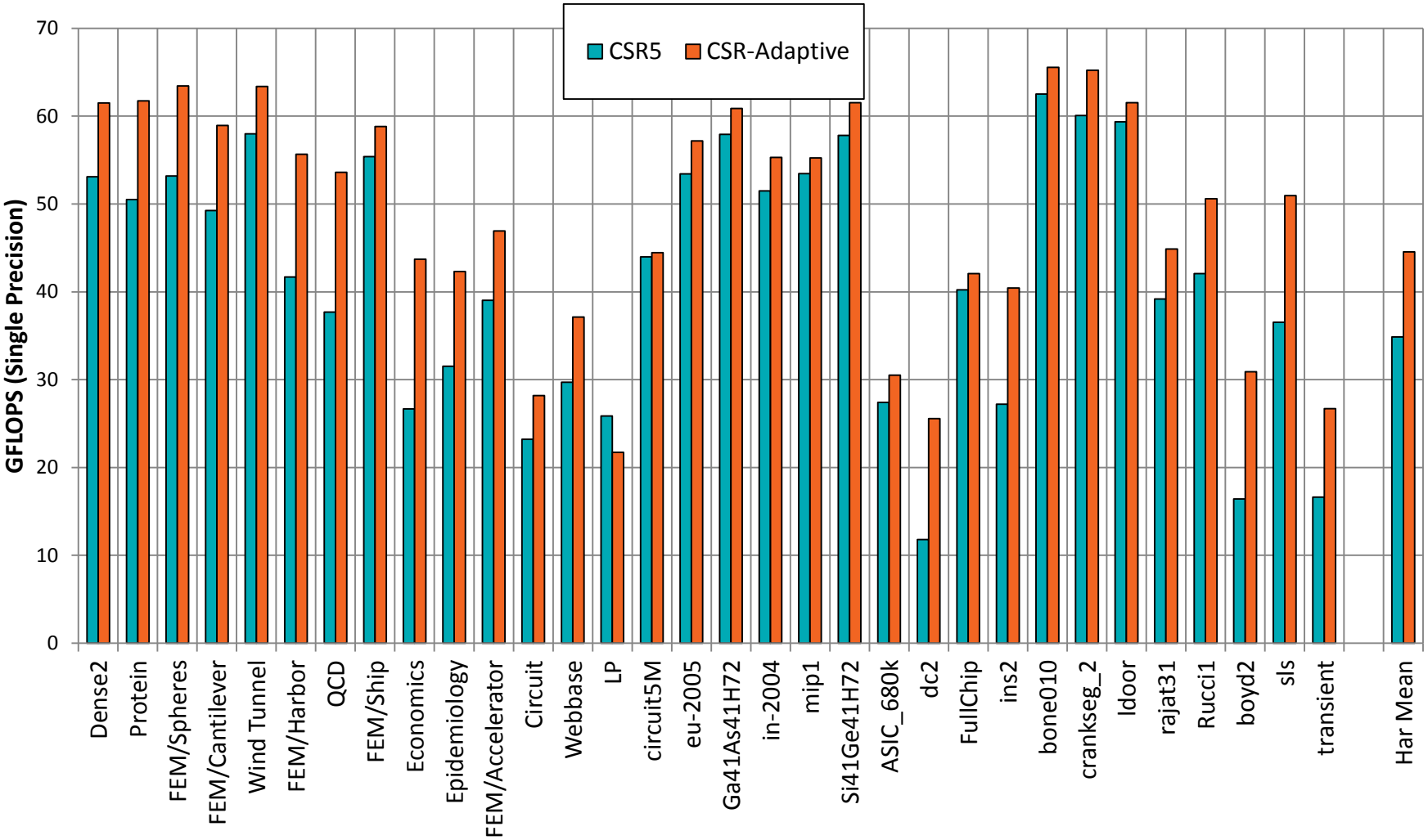


CSR-ADAPTIVE VS CSR5

AMD FIREPRO W9100 GPU (SINGLE PRECISION)

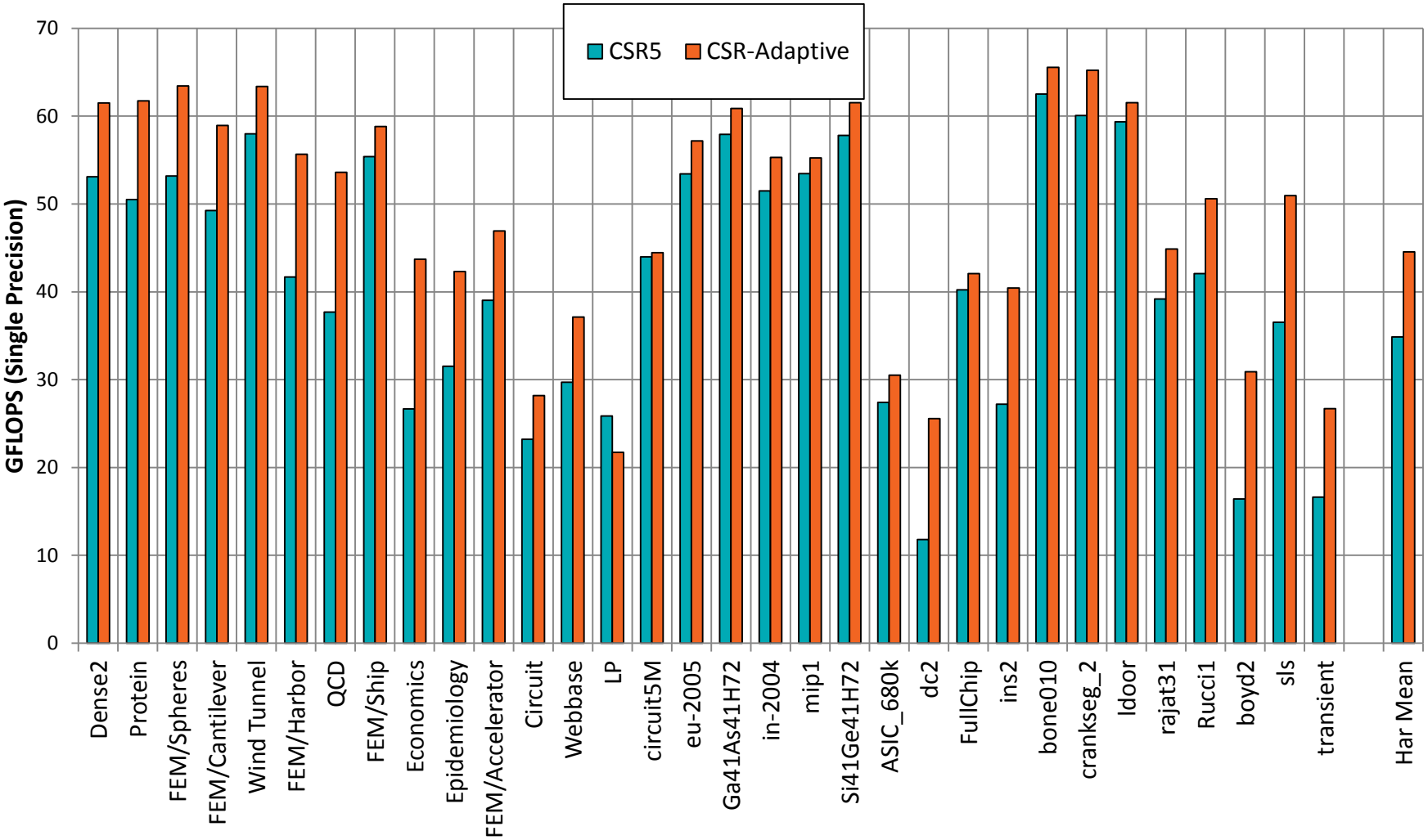
CSR-ADAPTIVE VS CSR5

AMD FIREPRO W9100 GPU (SINGLE PRECISION)



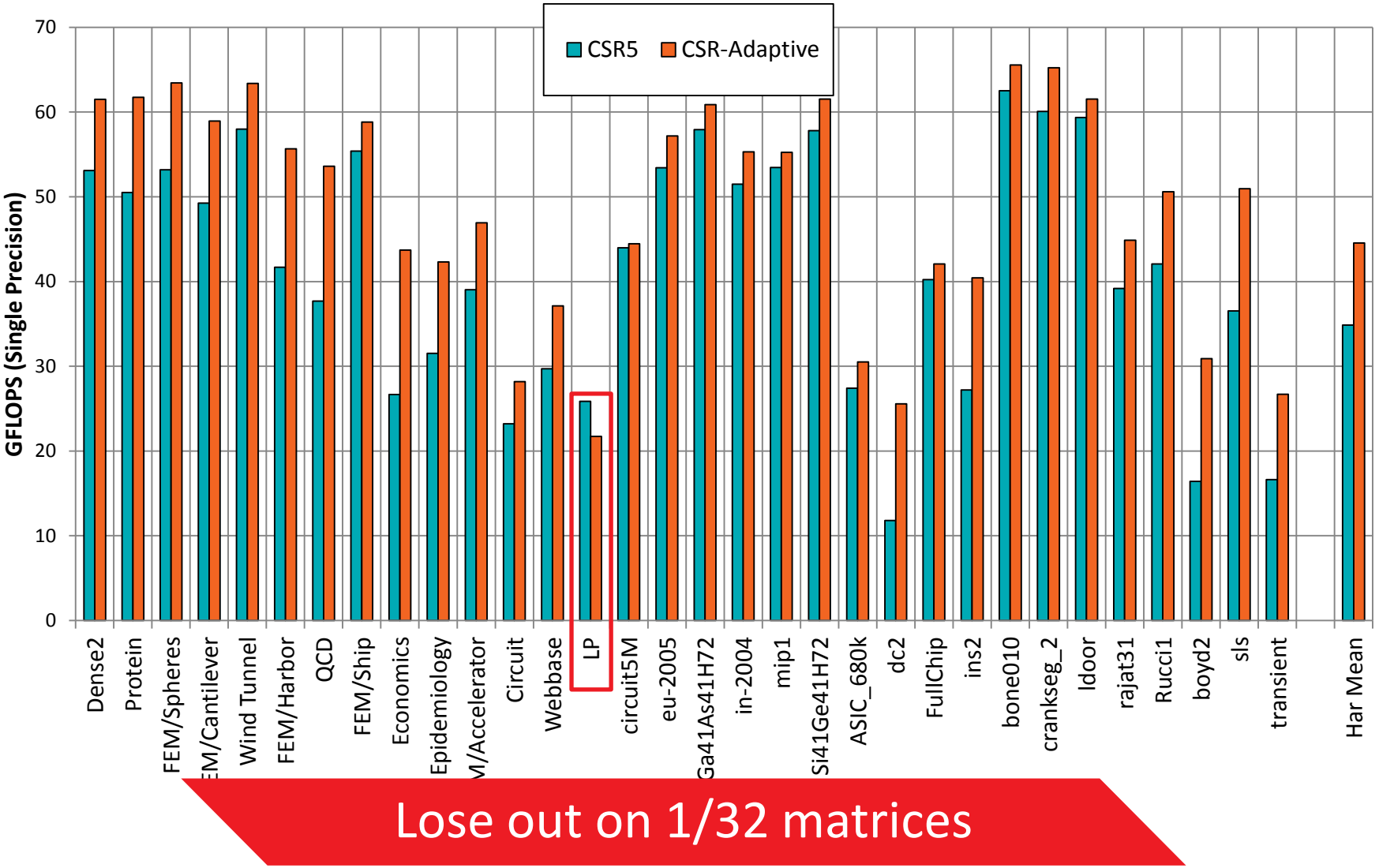
CSR-ADAPTIVE VS CSR5

AMD FIREPRO W9100 GPU (SINGLE PRECISION)



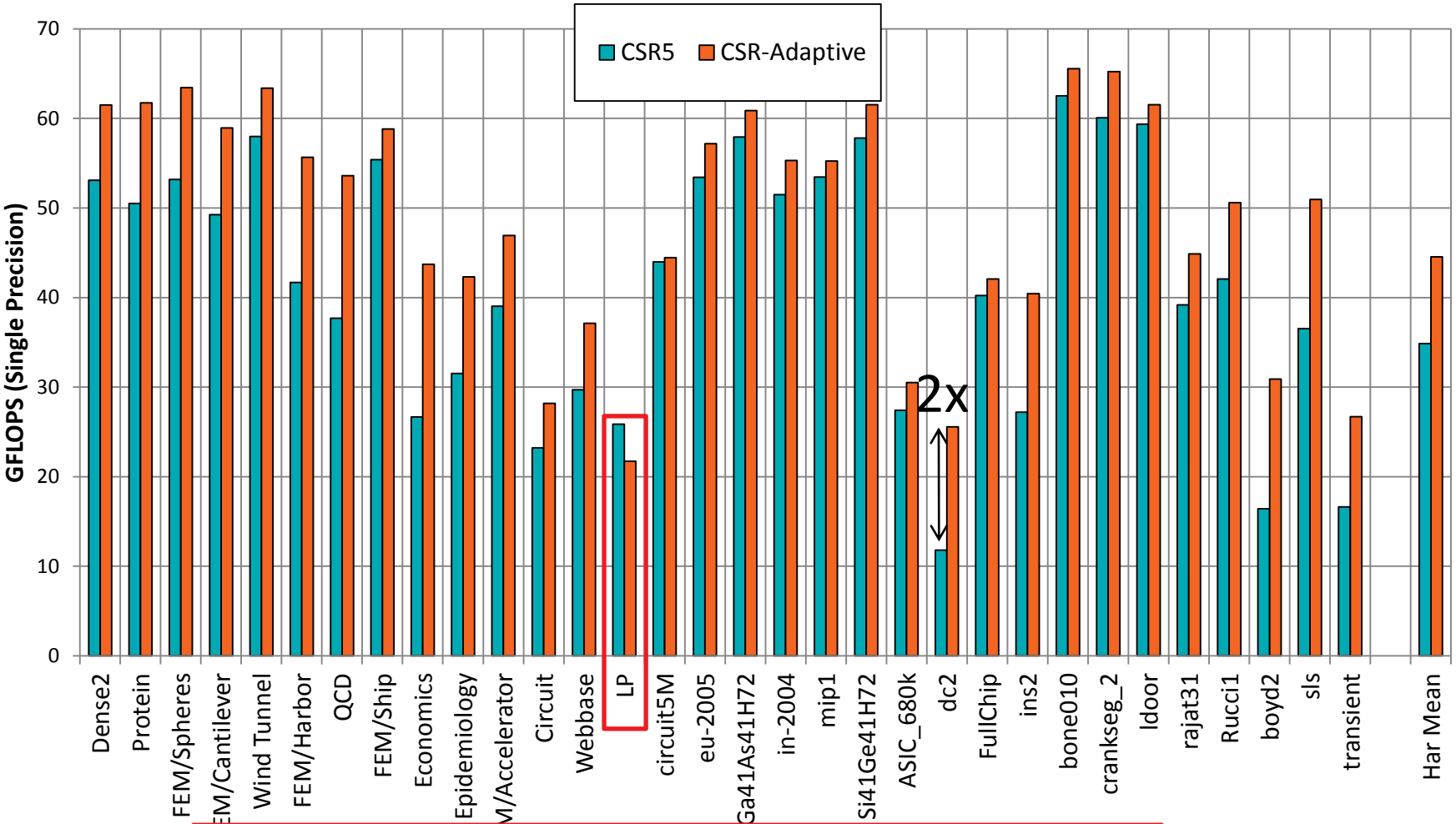
CSR-ADAPTIVE VS CSR5

AMD FIREPRO W9100 GPU (SINGLE PRECISION)



CSR-ADAPTIVE VS CSR5

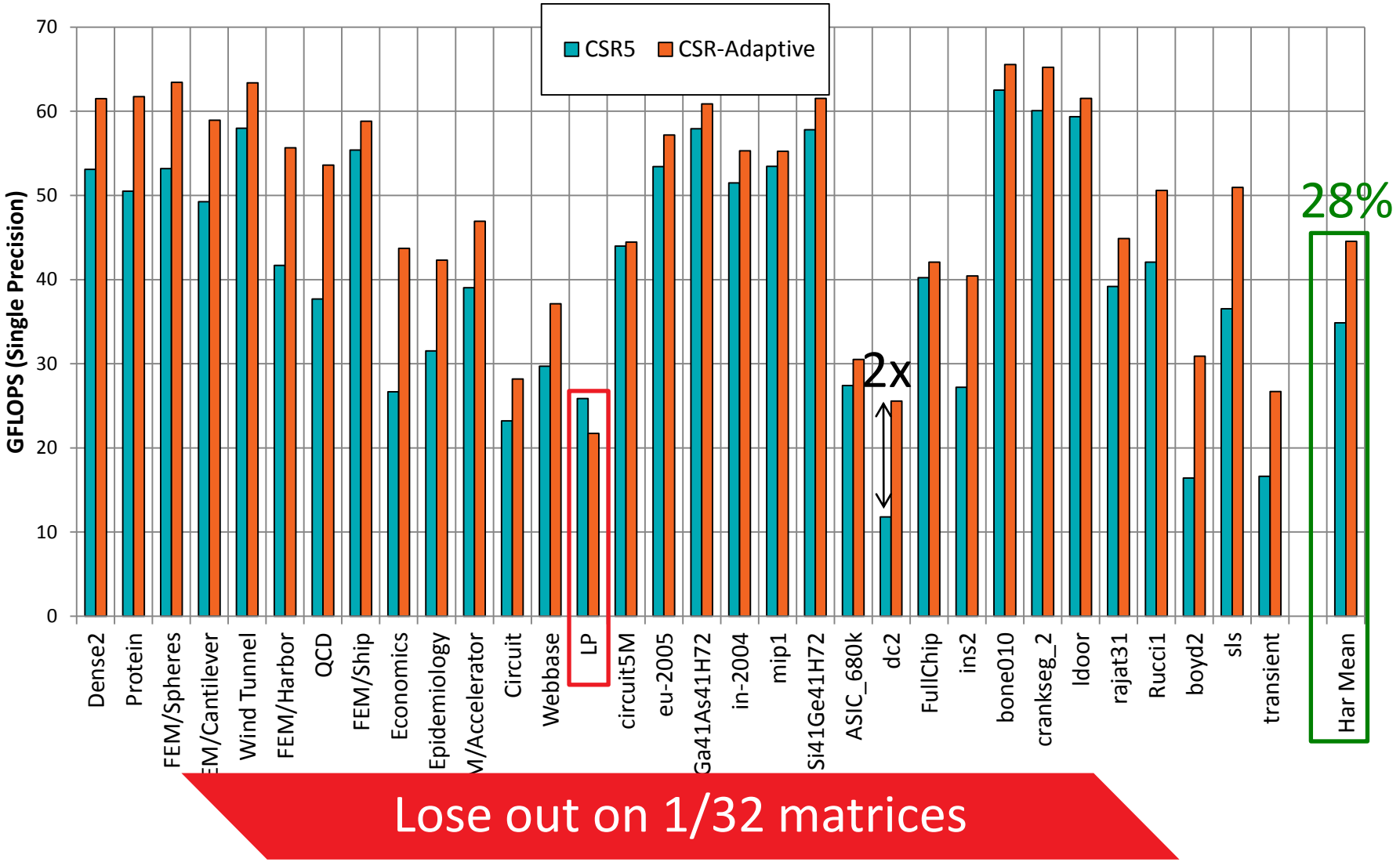
AMD FIREPRO W9100 GPU (SINGLE PRECISION)



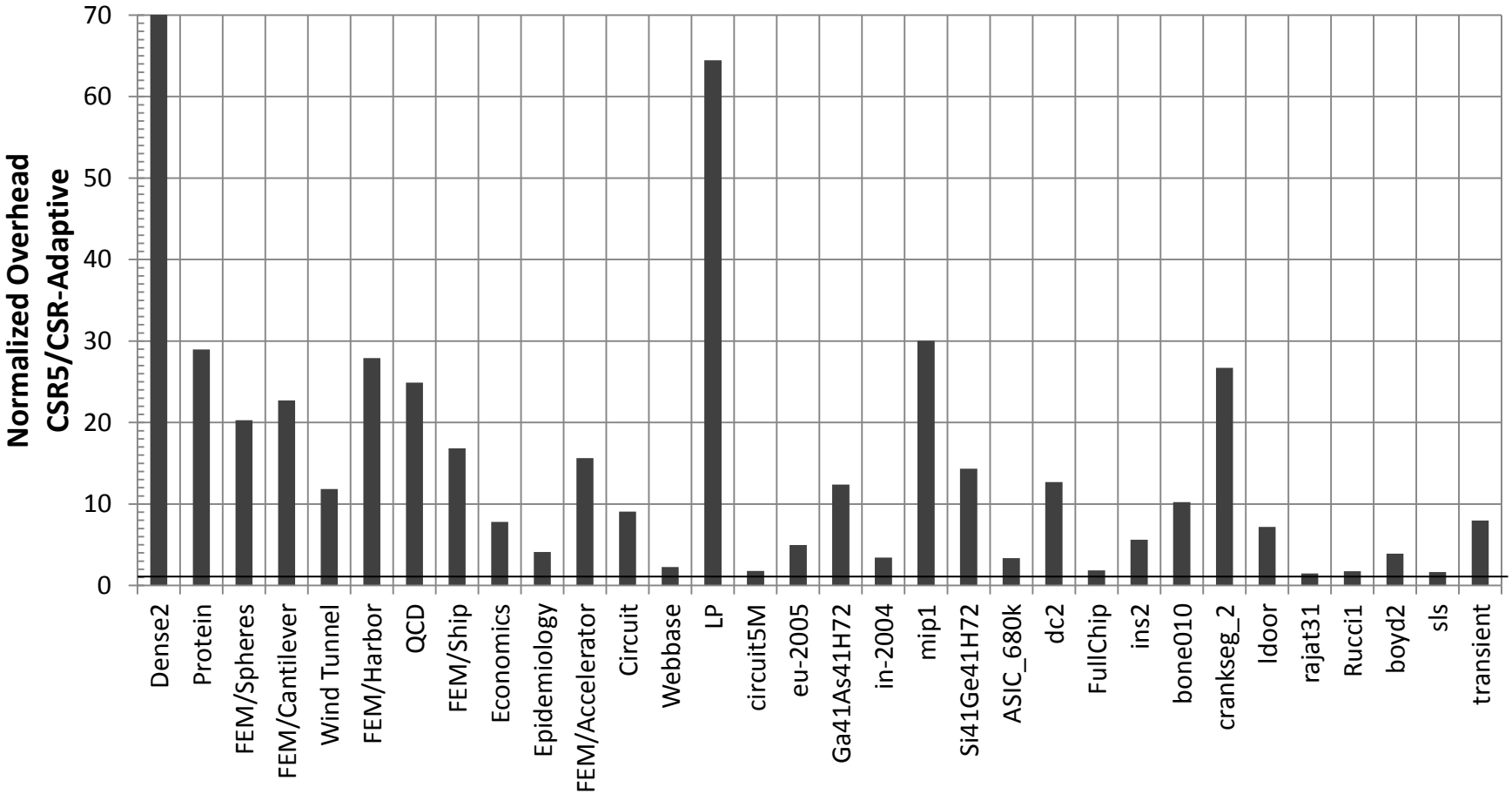
Lose out on 1/32 matrices

CSR-ADAPTIVE VS CSR5

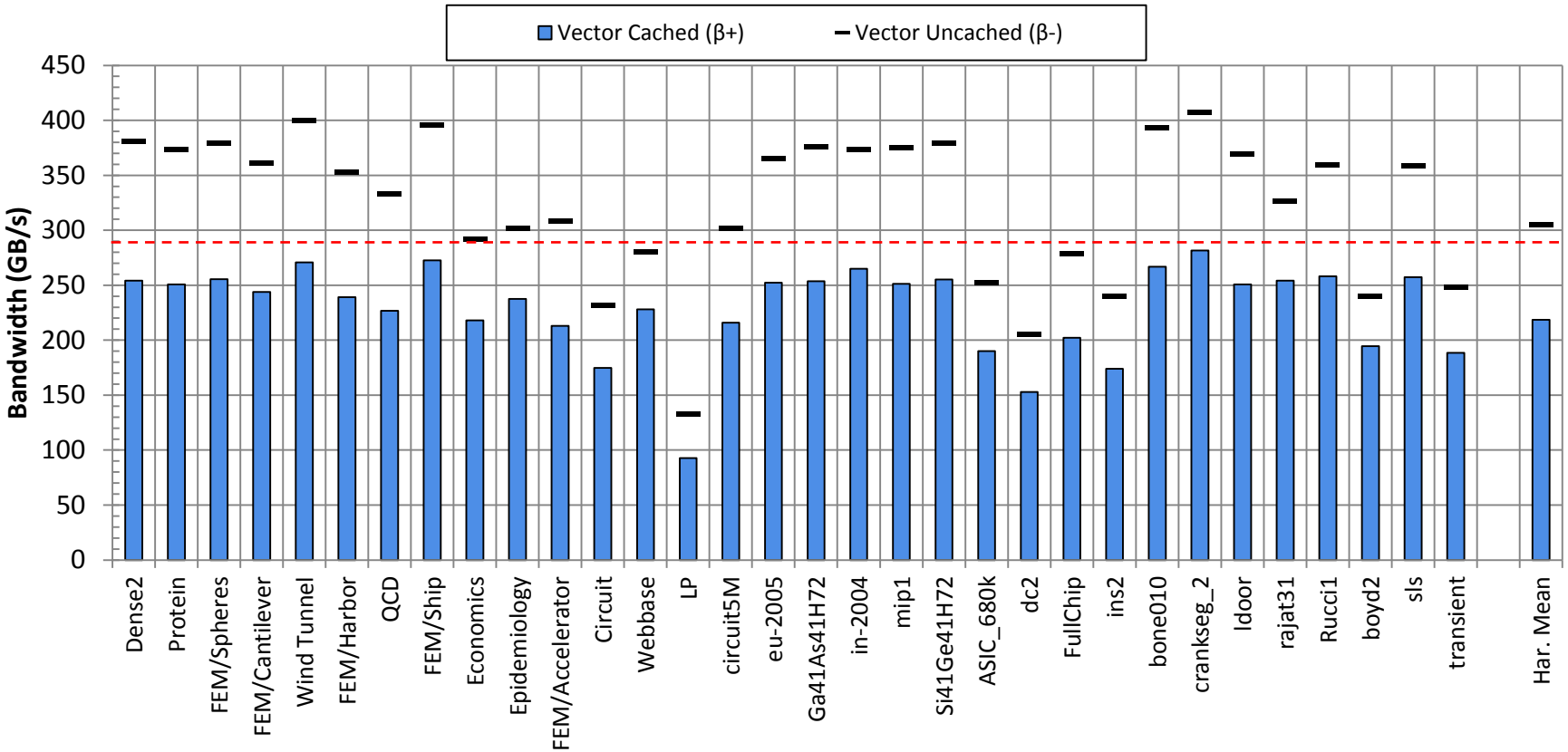
AMD FIREPRO W9100 GPU (SINGLE PRECISION)



OVERHEAD: CSR-ADAPTIVE VS CSR5



BANDWIDTH: CSR-ADAPTIVE



CONCLUSION

AMD

Optimized GPU-based SpMV algorithm that beats the competition by 2x
Operates within 15% of GPU's achievable b/w

Minimal overhead for data structure generation

Shipping now as part of AMD's clSPARSE library
https://github.com/clMathLibraries/clSPARSE

DISCLAIMER & ATTRIBUTION

The information presented in this document is for informational purposes only and may contain technical inaccuracies, omissions and typographical errors.

The information contained herein is subject to change and may be rendered inaccurate for many reasons, including but not limited to product and roadmap changes, component and motherboard version changes, new model and/or product releases, product differences between differing manufacturers, software changes, BIOS flashes, firmware upgrades, or the like. AMD assumes no obligation to update or otherwise correct or revise this information. However, AMD reserves the right to revise this information and to make changes from time to time to the content hereof without obligation of AMD to notify any person of such revisions or changes.

AMD MAKES NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE CONTENTS HEREOF AND ASSUMES NO RESPONSIBILITY FOR ANY INACCURACIES, ERRORS OR OMISSIONS THAT MAY APPEAR IN THIS INFORMATION.

AMD SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT WILL AMD BE LIABLE TO ANY PERSON FOR ANY DIRECT, INDIRECT, SPECIAL OR OTHER CONSEQUENTIAL DAMAGES ARISING FROM THE USE OF ANY INFORMATION CONTAINED HEREIN, EVEN IF AMD IS EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

ATTRIBUTION

Backup Slides

FINDING BLOCKS FOR CSR-ADAPTIVE

DONE ONCE ON THE CPU

AMD

row delimiters:

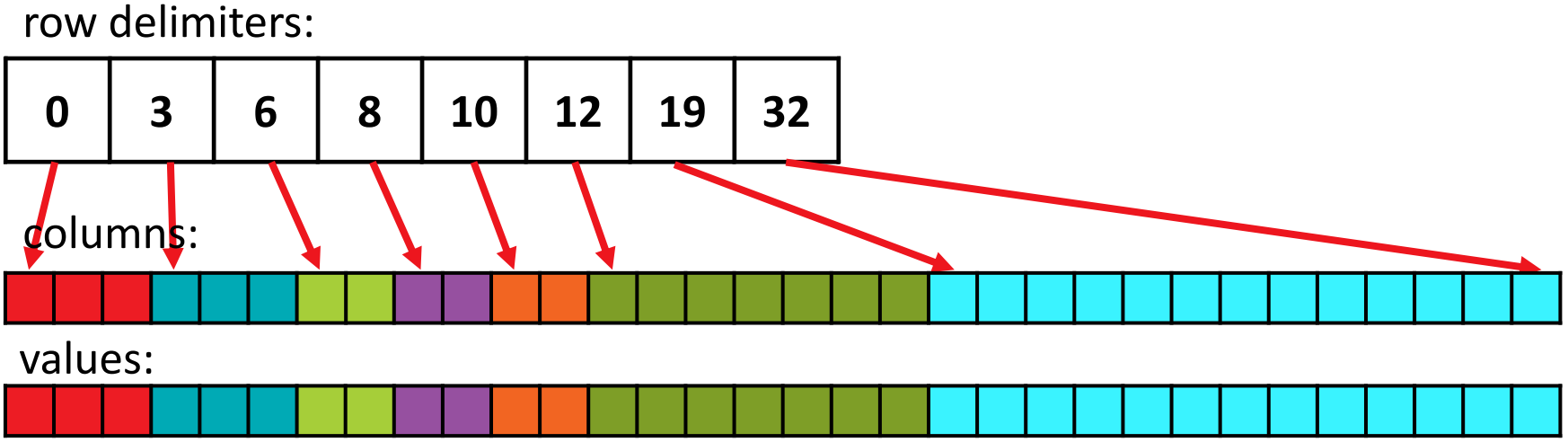
0	3	6	8	10	12	19	32

columns:

values:

FINDING BLOCKS FOR CSR-ADAPTIVE

DONE ONCE ON THE CPU



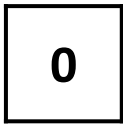
FINDING BLOCKS FOR CSR-ADAPTIVE

DONE ONCE ON THE CPU

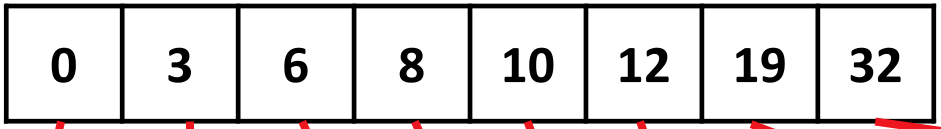
AMD

Assuming 8 LDS entries per workgroup

row blocks:



row delimiters:



columns:

values:

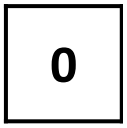
Block 1

FINDING BLOCKS FOR CSR-ADAPTIVE

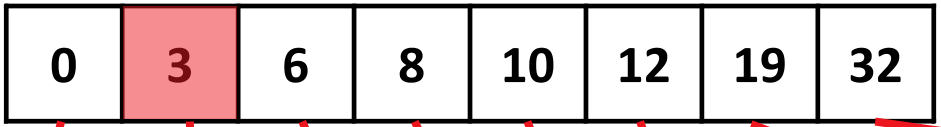
DONE ONCE ON THE CPU

Assuming 8 LDS entries per workgroup

row blocks:



row delimiters:



columns:

values:

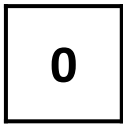
Block 1

FINDING BLOCKS FOR CSR-ADAPTIVE

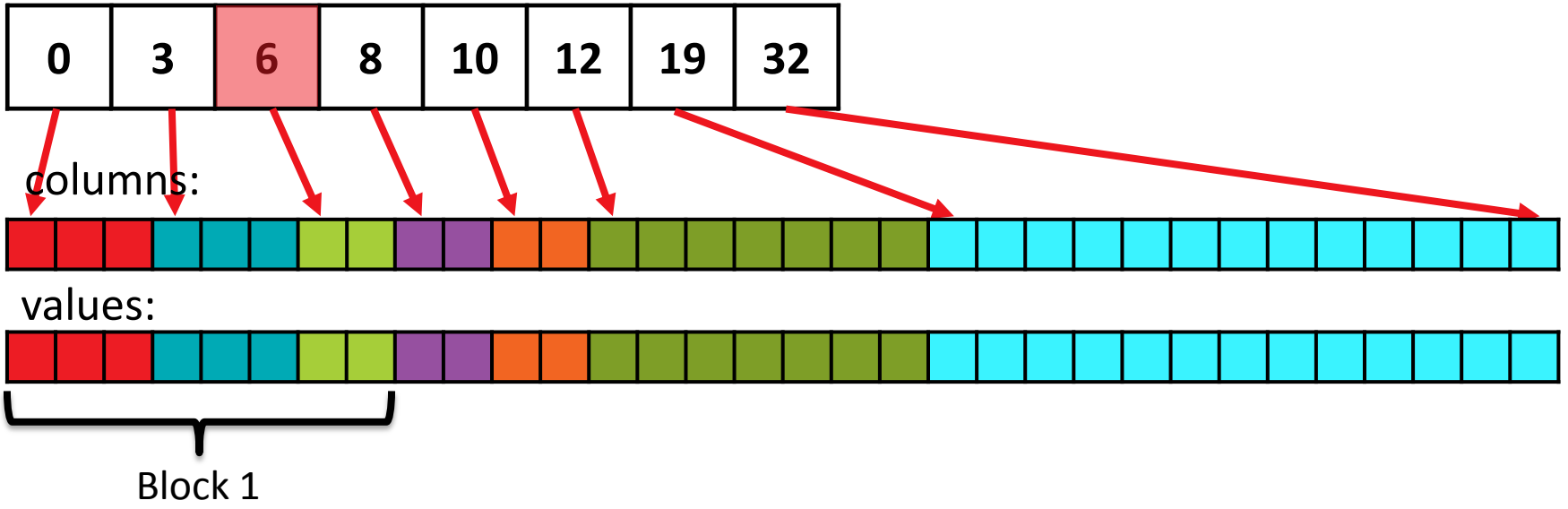
DONE ONCE ON THE CPU

Assuming 8 LDS entries per workgroup

row blocks:



row delimiters:

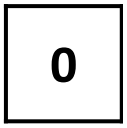


FINDING BLOCKS FOR CSR-ADAPTIVE

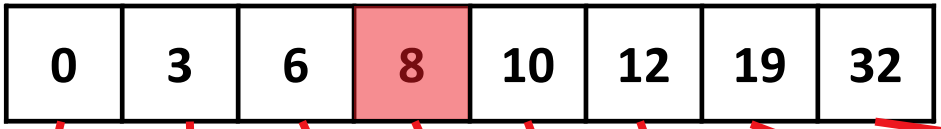
DONE ONCE ON THE CPU

Assuming 8 LDS entries per workgroup

row blocks:



row delimiters:



columns:

values:

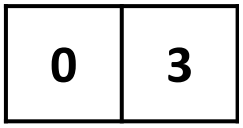
Block 1

FINDING BLOCKS FOR CSR-ADAPTIVE

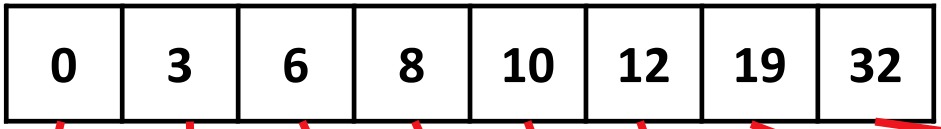
DONE ONCE ON THE CPU

Assuming 8 LDS entries per workgroup

row blocks:



row delimiters:



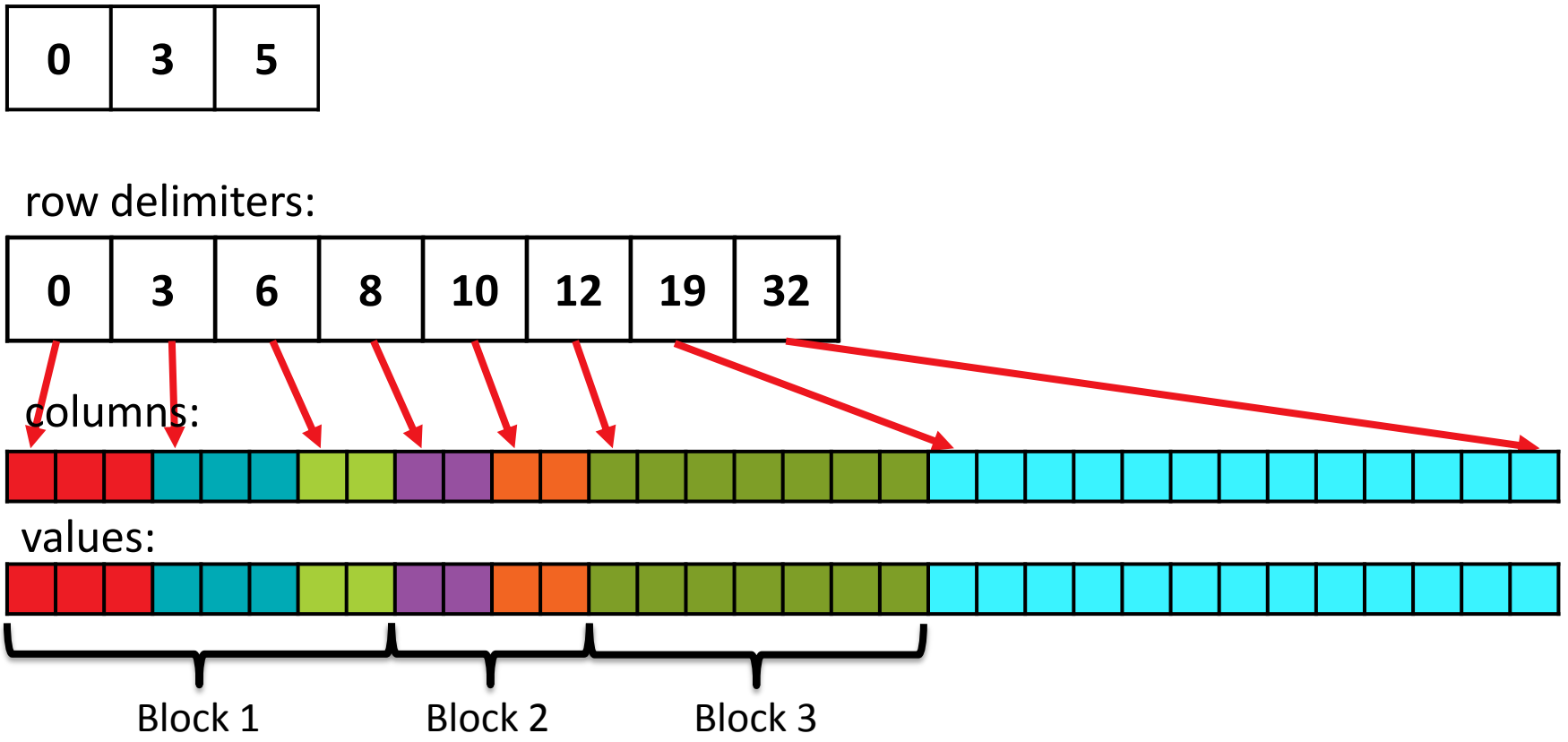
columns:

values:

FINDING BLOCKS FOR CSR-ADAPTIVE

DONE ONCE ON THE CPU

Assuming 8 LDS entries per workgroup



FINDING BLOCKS FOR CSR-ADAPTIVE

DONE ONCE ON THE CPU

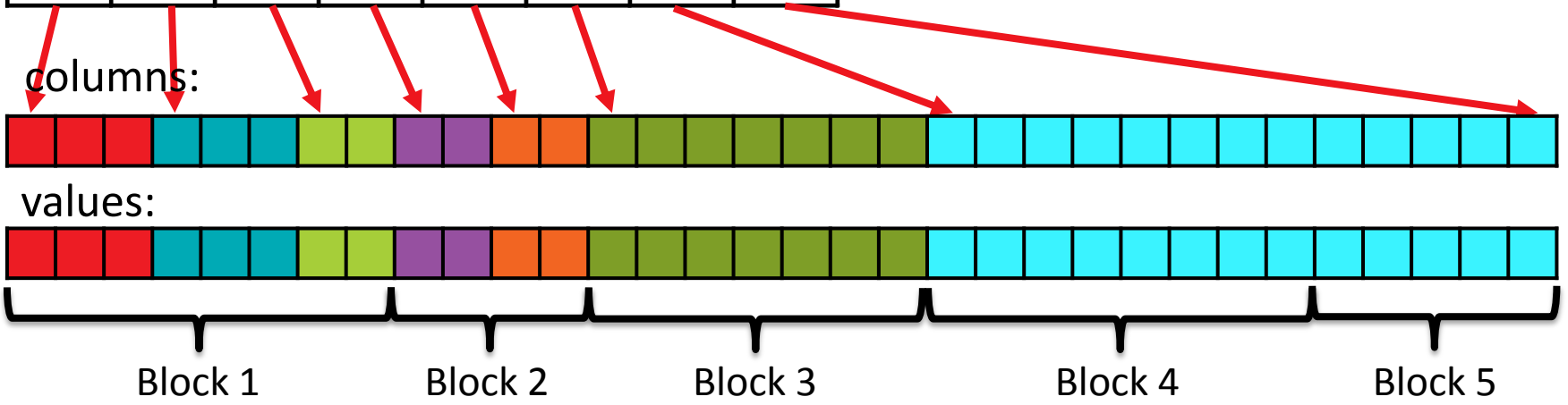
Assuming 8 LDS entries per workgroup

row blocks:

0	3	5	6	6	7

row delimiters:

0	3	6	8	10	12	19	32



FINDING BLOCKS FOR CSR-ADAPTIVE

DONE ONCE ON THE CPU

AMD

Assuming 8 LDS entries per workgroup

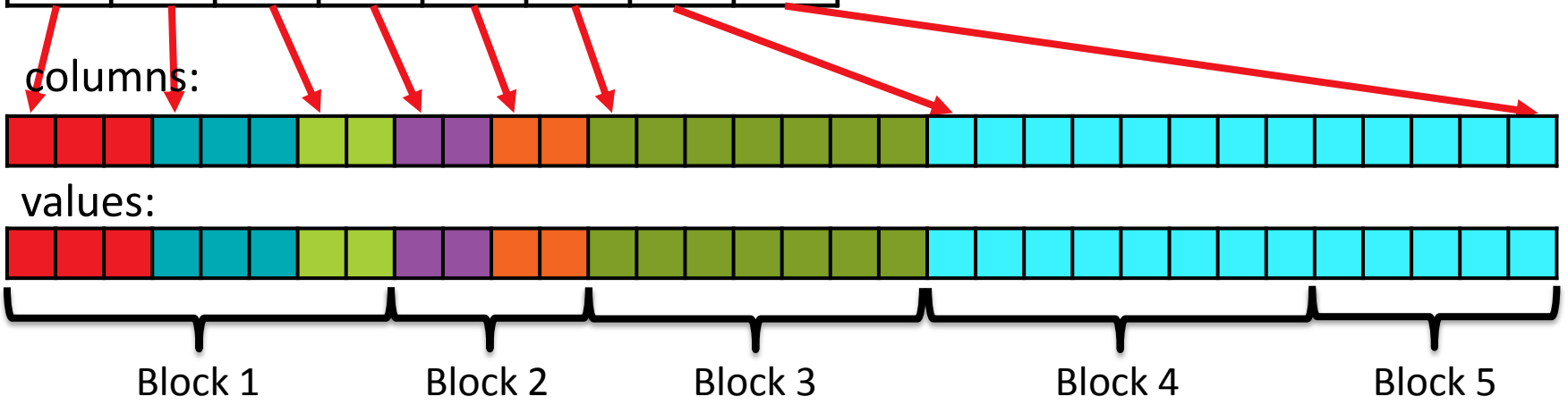
row blocks:

0	3	5	6	6	7

CSR Structure Unchanged

row delimiters:

0	3	6	8	10	12	19	32



A DECENT SPMV FOR THE CPU IS SIMPLE

AMD

```
1. for (i = 0; i < nRows; i++) {
2.   temp = 0;
3.   for (j = rowDs[i]; j < rowDs[i+1]; j++) {
4.     temp += vals[j] * vec[cols[j]];
5.   }
6.   output[i] = temp;
7. }
```

A DECENT SPMV FOR THE CPU IS SIMPLE

AMD

```
#pragma omp parallel for
1. for (i = 0; i < nRows; i++) {
2.   temp = 0;
3.   for (j = rowDs[i]; j < rowDs[i+1]; j++) {
4.     temp += vals[j] * vec[cols[j]];
5.   }
6.   output[i] = temp;
7. }
```

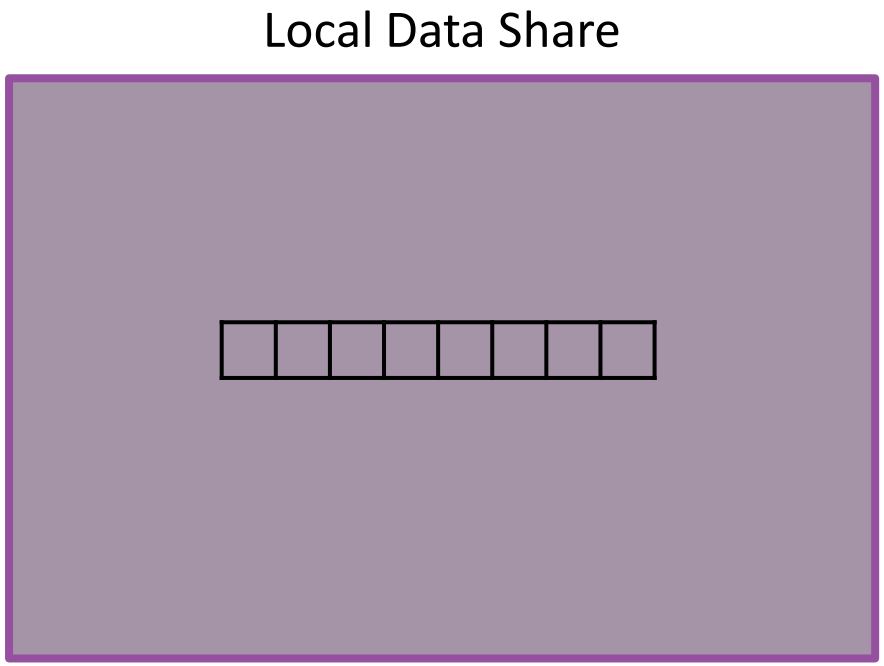
A DECENT SPMV FOR THE CPU IS SIMPLE

AMD

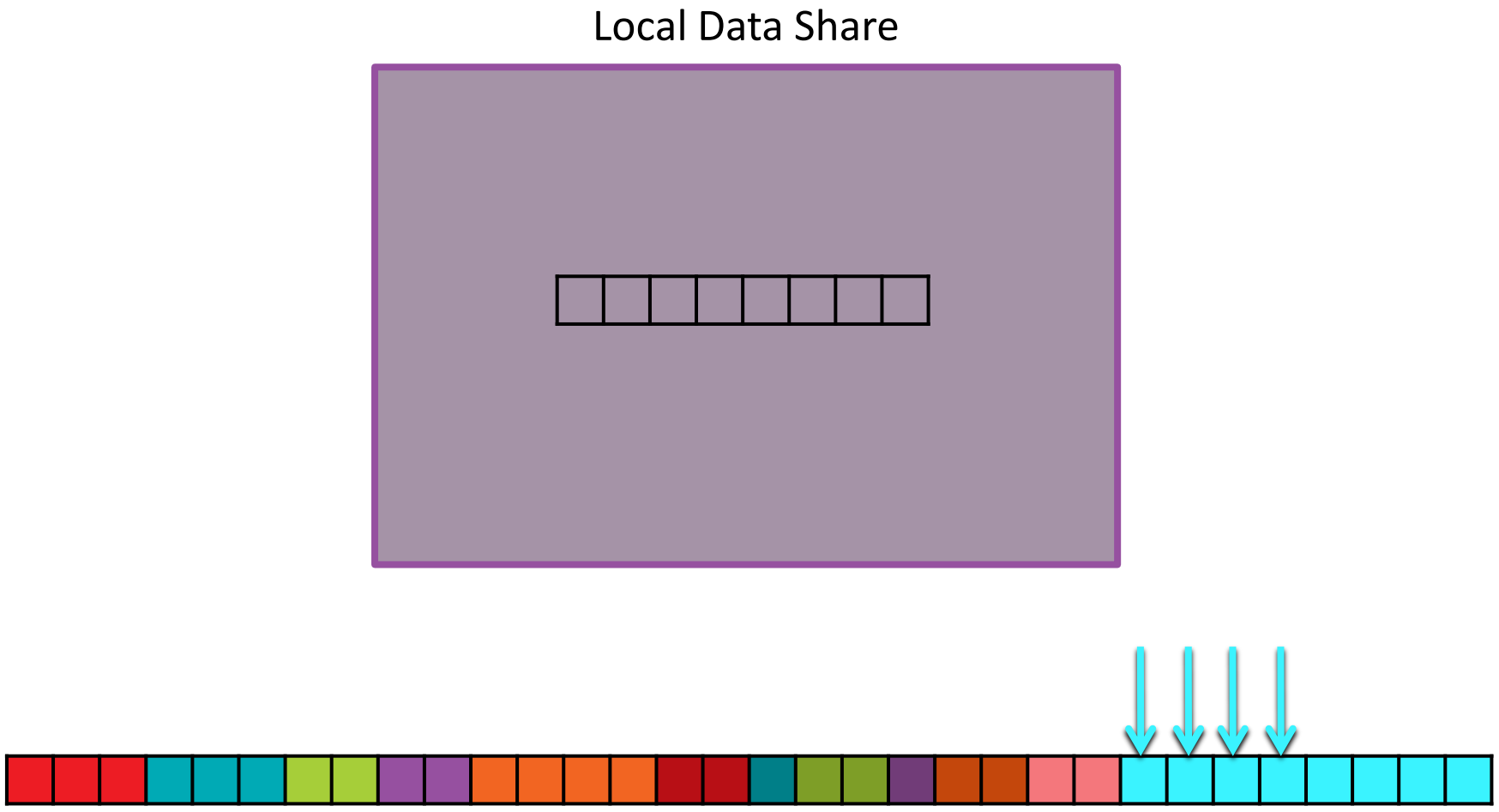
```
#pragma omp parallel for
1. for (i = 0; i < nRows; i++) {
2.   temp = 0;
3.   for (j = rowDs[i]; j < rowDs[i+1]; j++) {
4.     temp += vals[j] * vec[cols[j]];
5.   }
6.   output[i] = temp;
7. }
```

Only **seven** lines of code provides decent SpMV performance on the CPU

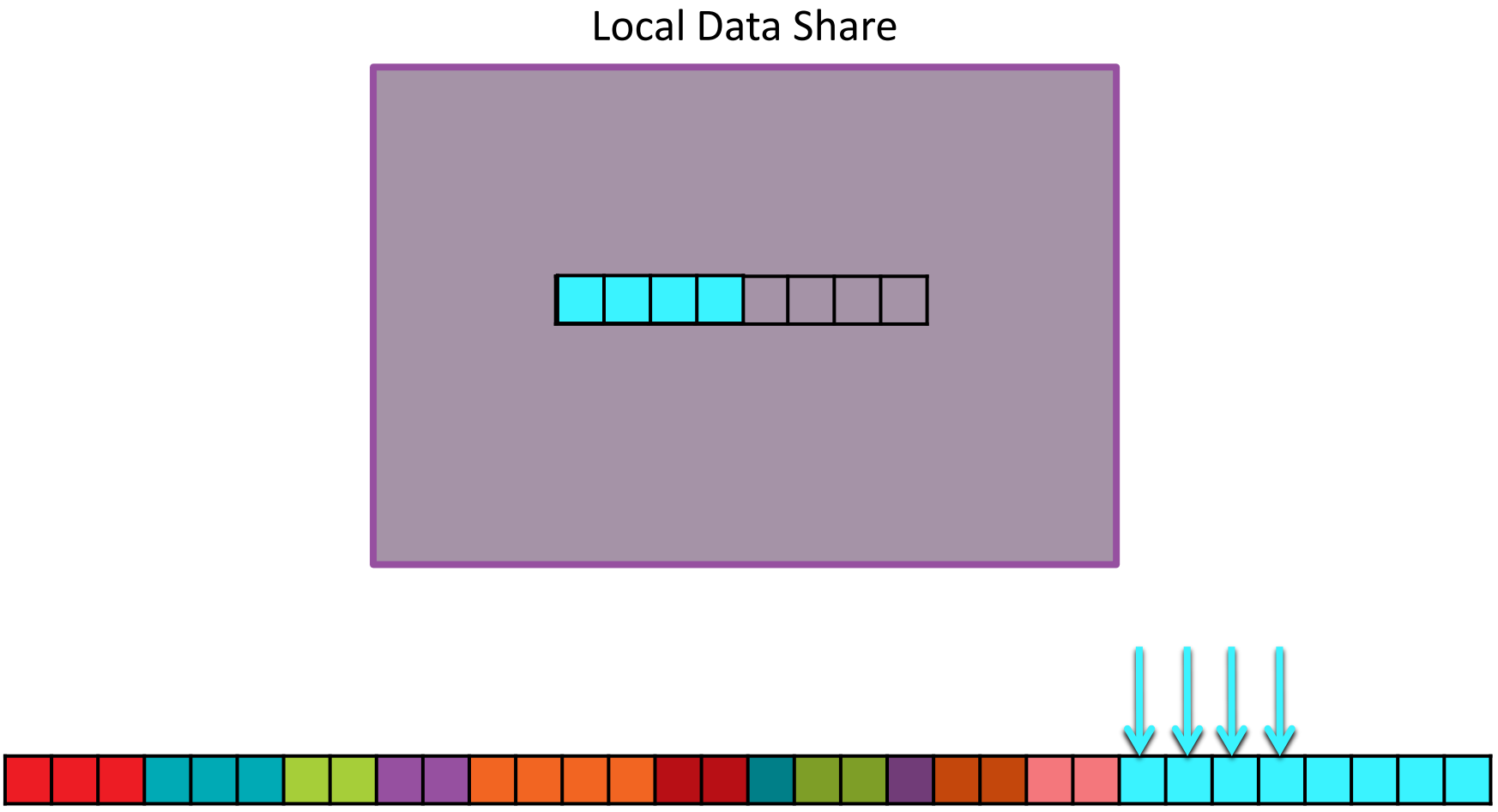
CSR-VECTOR USING THE LDS



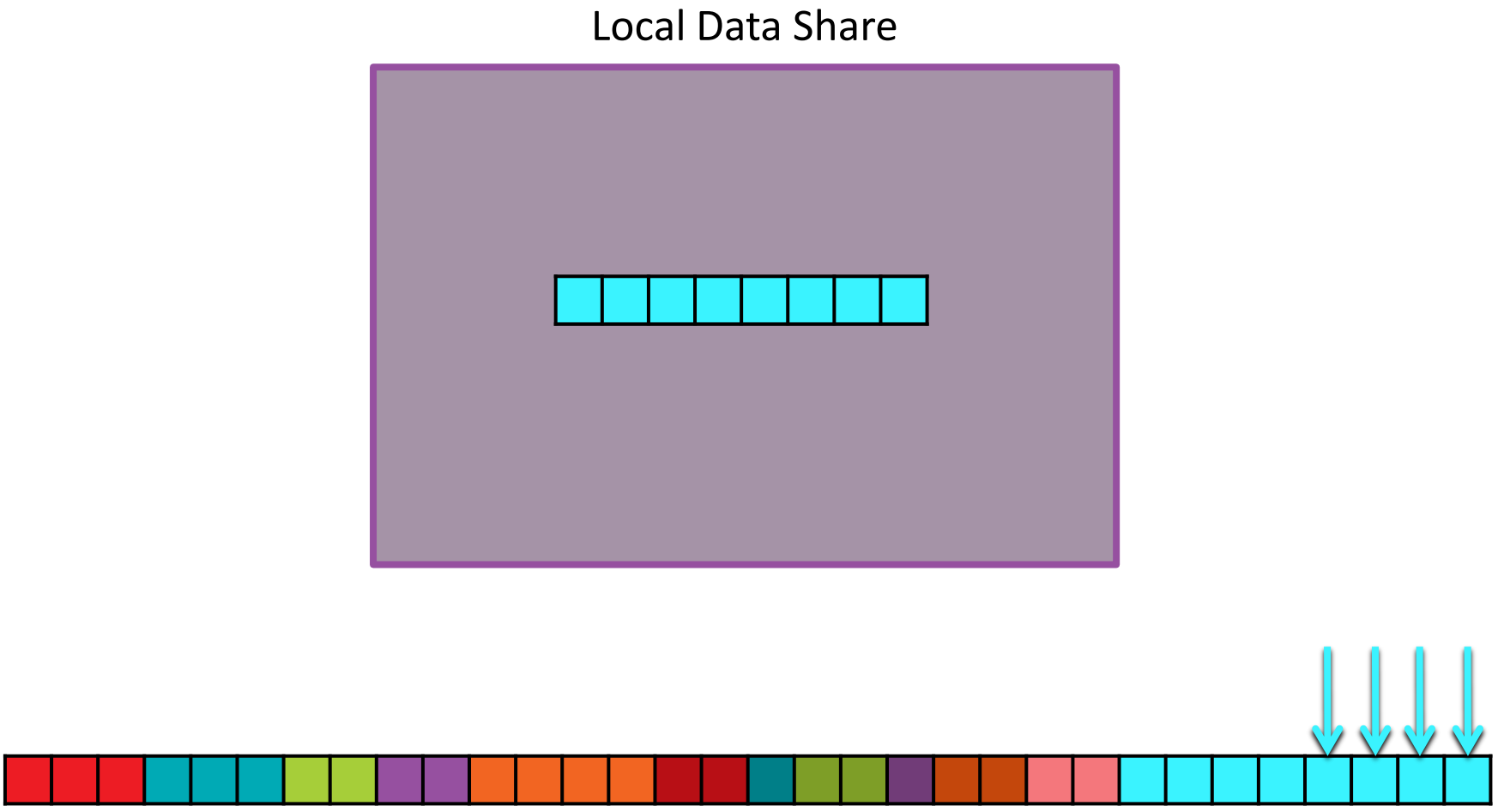
CSR-VECTOR USING THE LDS



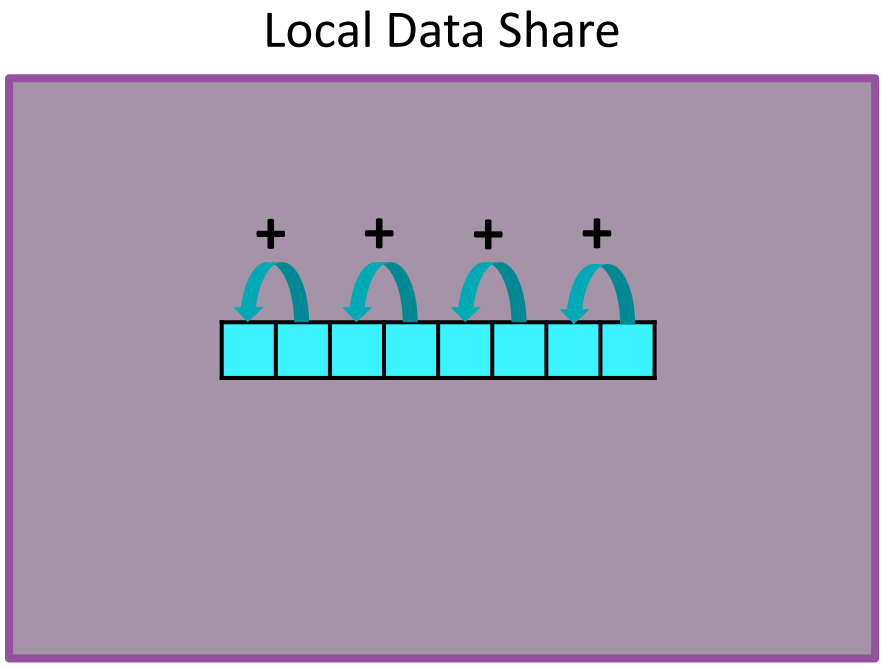
CSR-VECTOR USING THE LDS



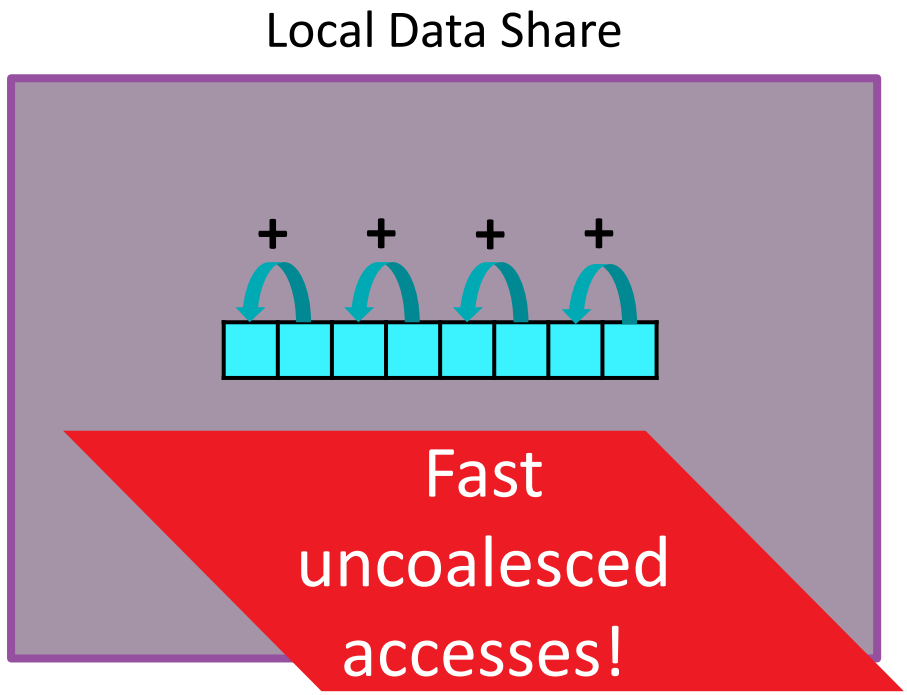
CSR-VECTOR USING THE LDS



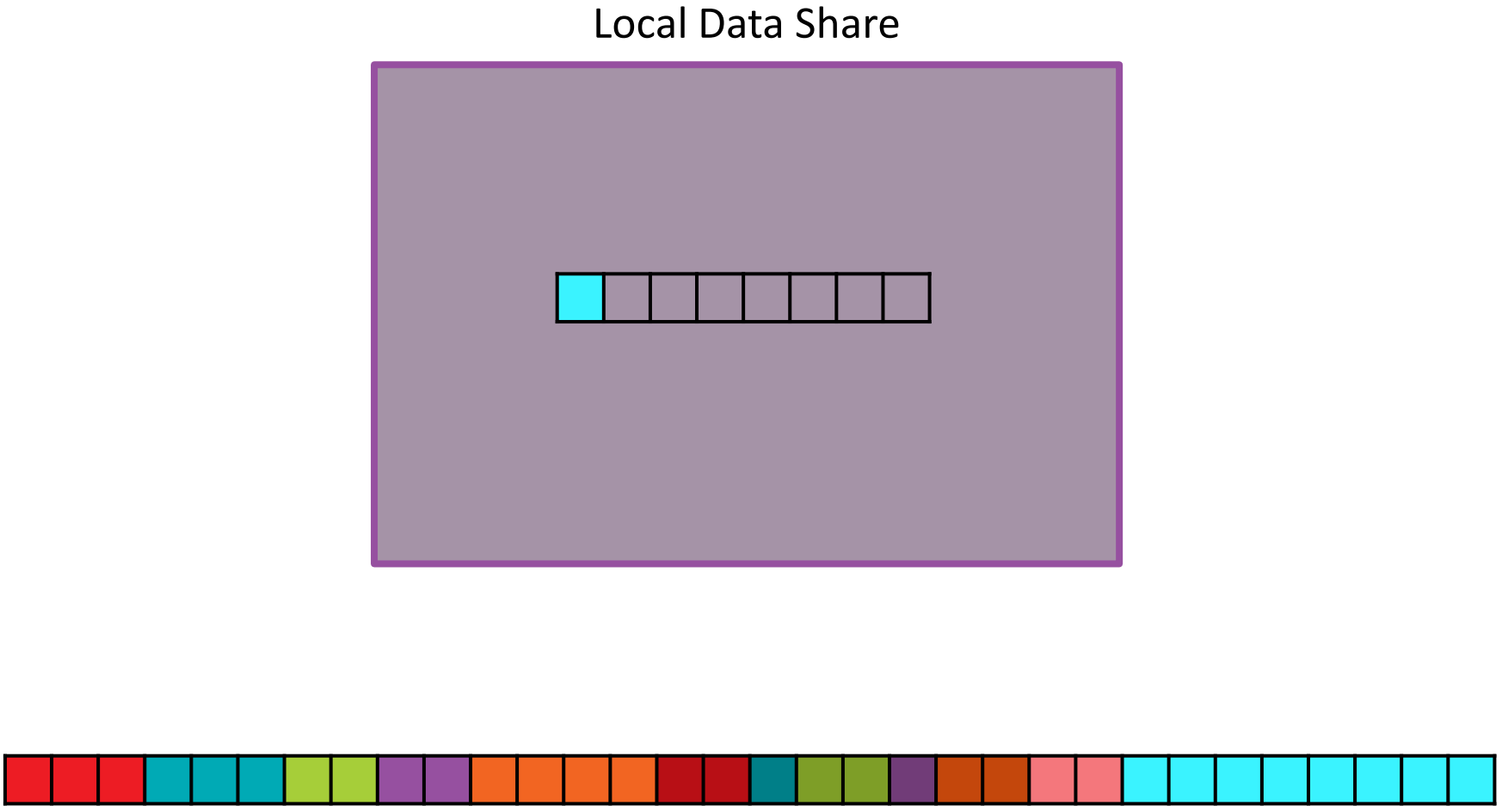
CSR-VECTOR USING THE LDS



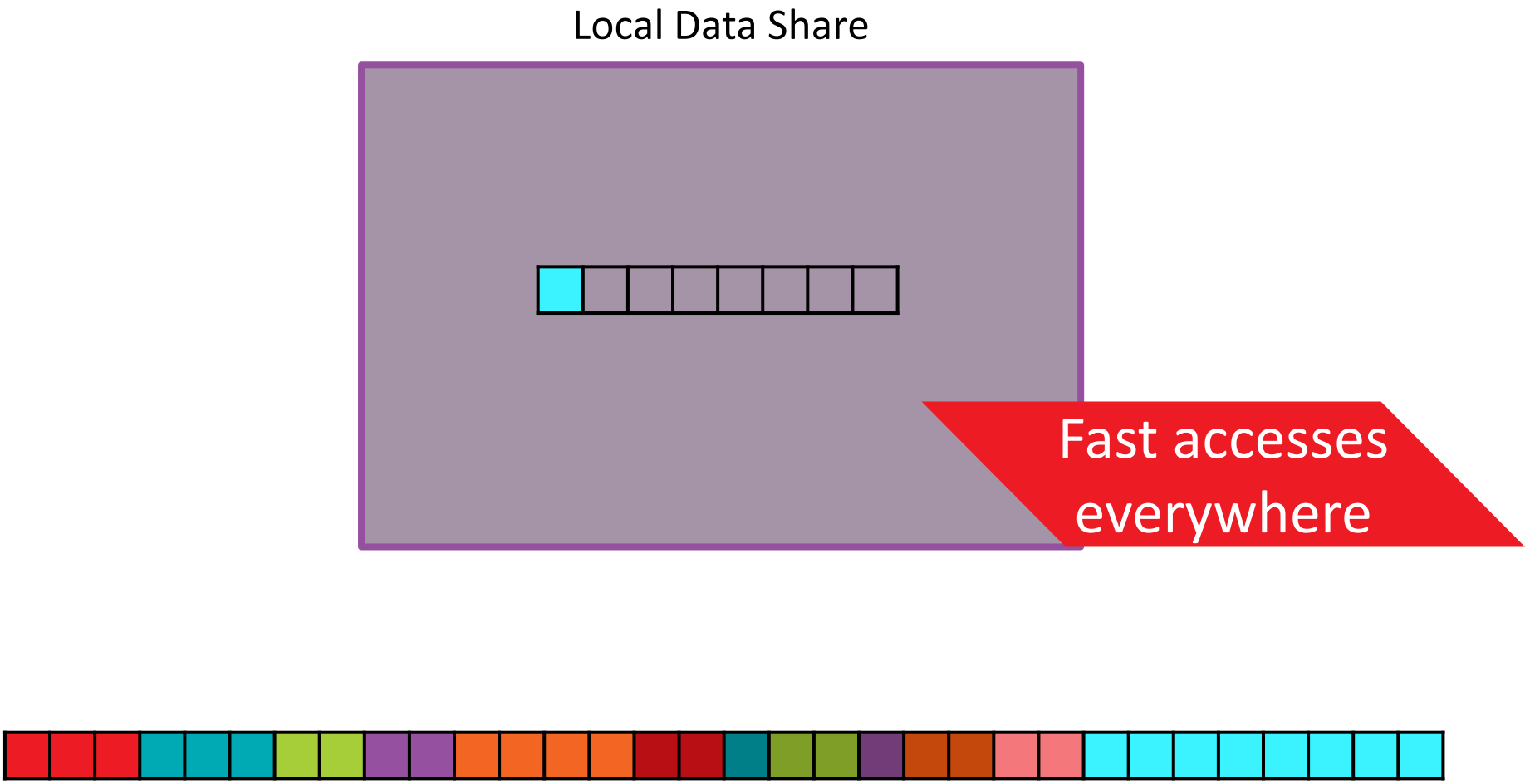
CSR-VECTOR USING THE LDS



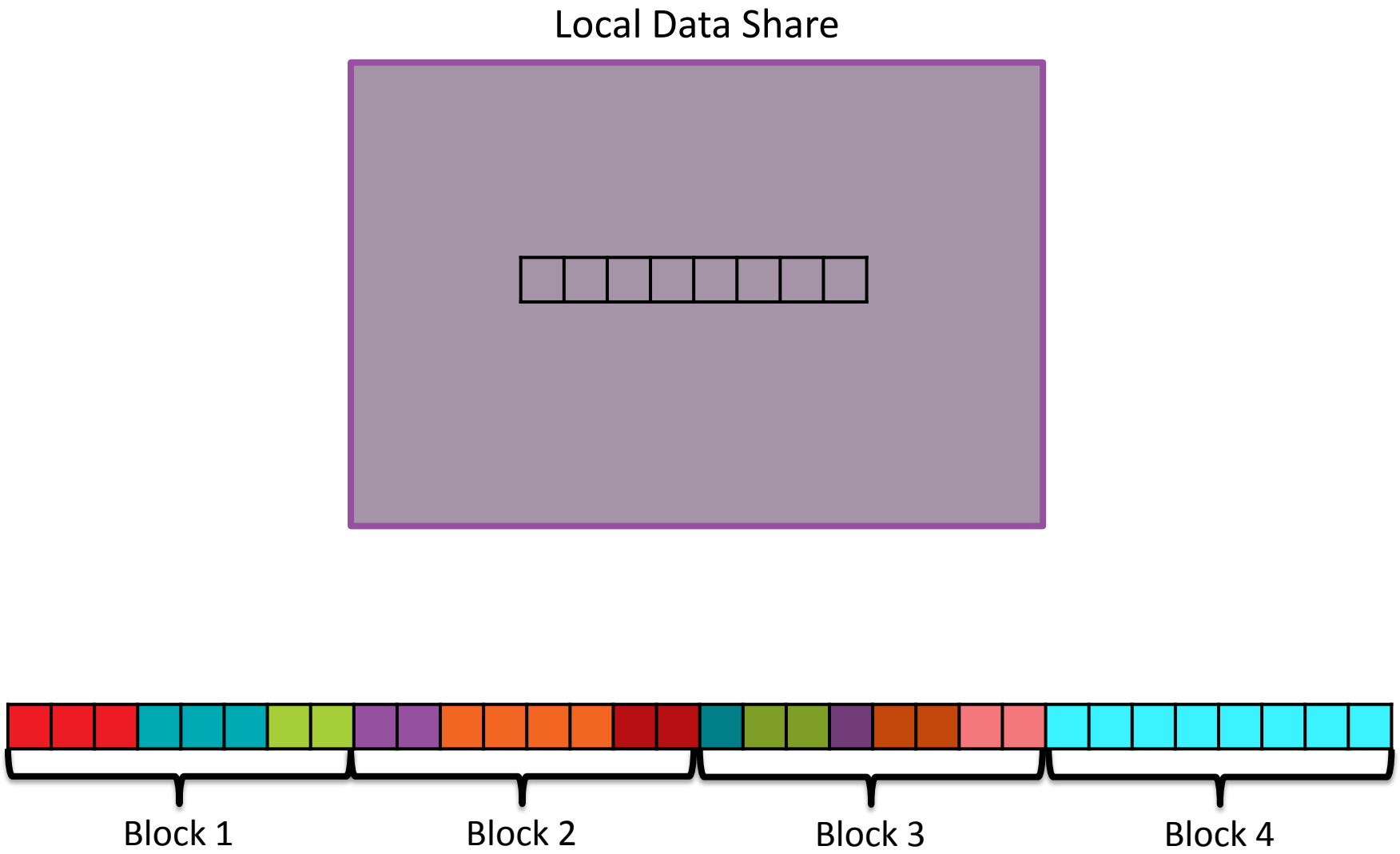
CSR-VECTOR USING THE LDS



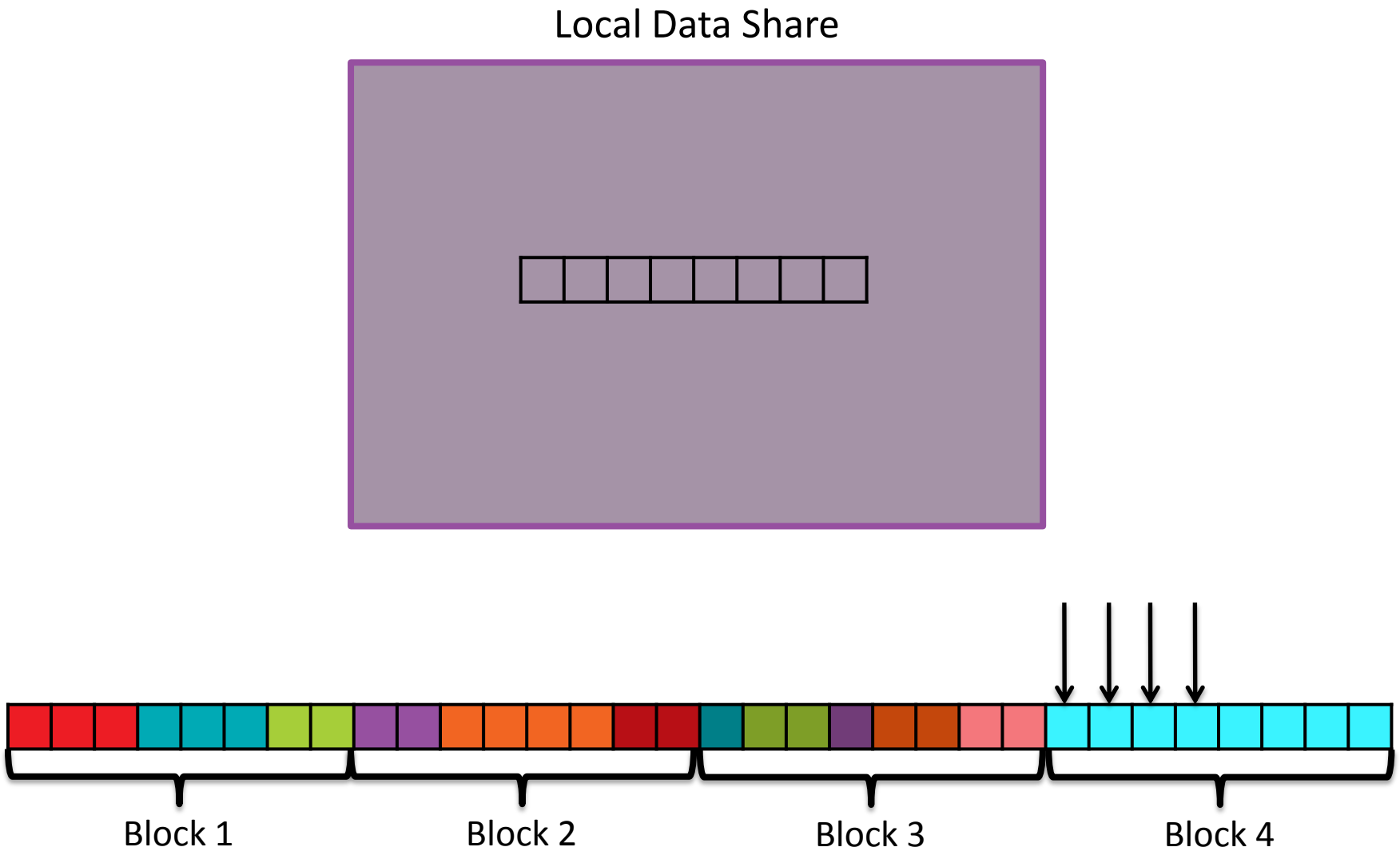
CSR-VECTOR USING THE LDS



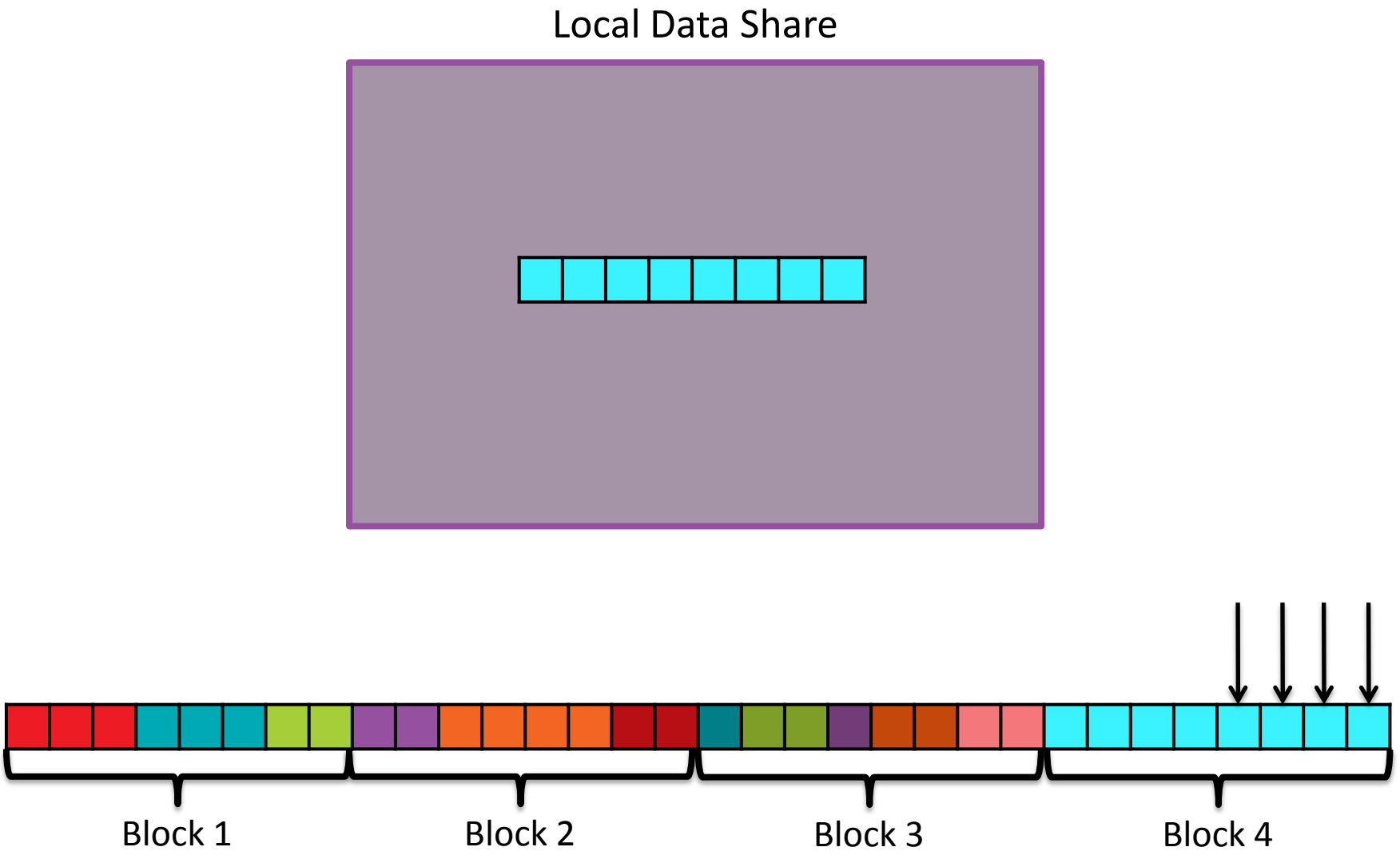
CSR-VECTOR USING THE LDS



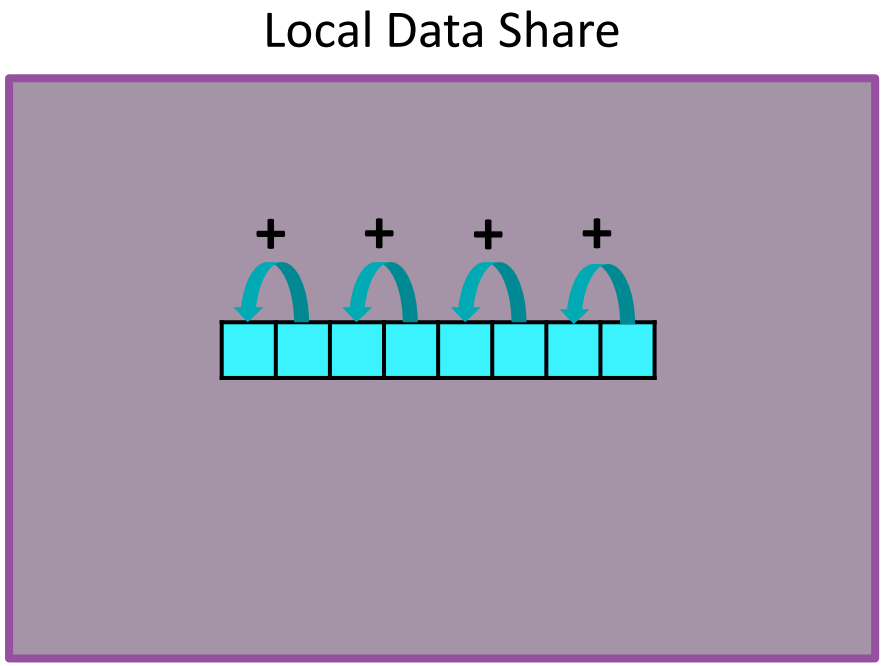
CSR-VECTOR USING THE LDS

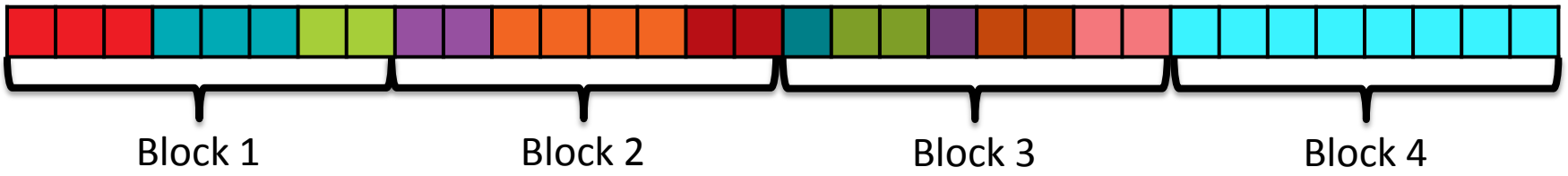


CSR-VECTOR USING THE LDS

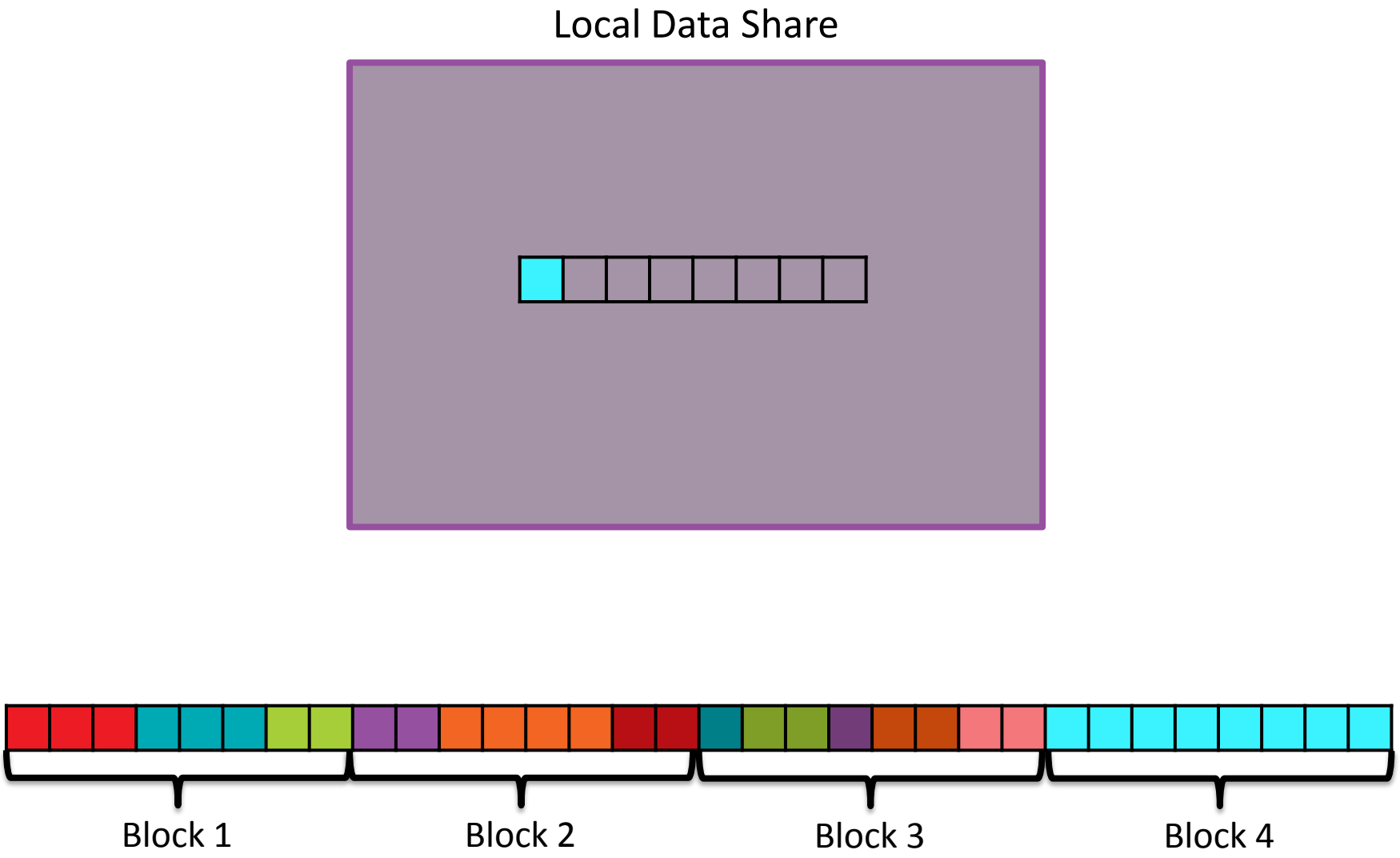


CSR-VECTOR USING THE LDS





CSR-VECTOR USING THE LDS

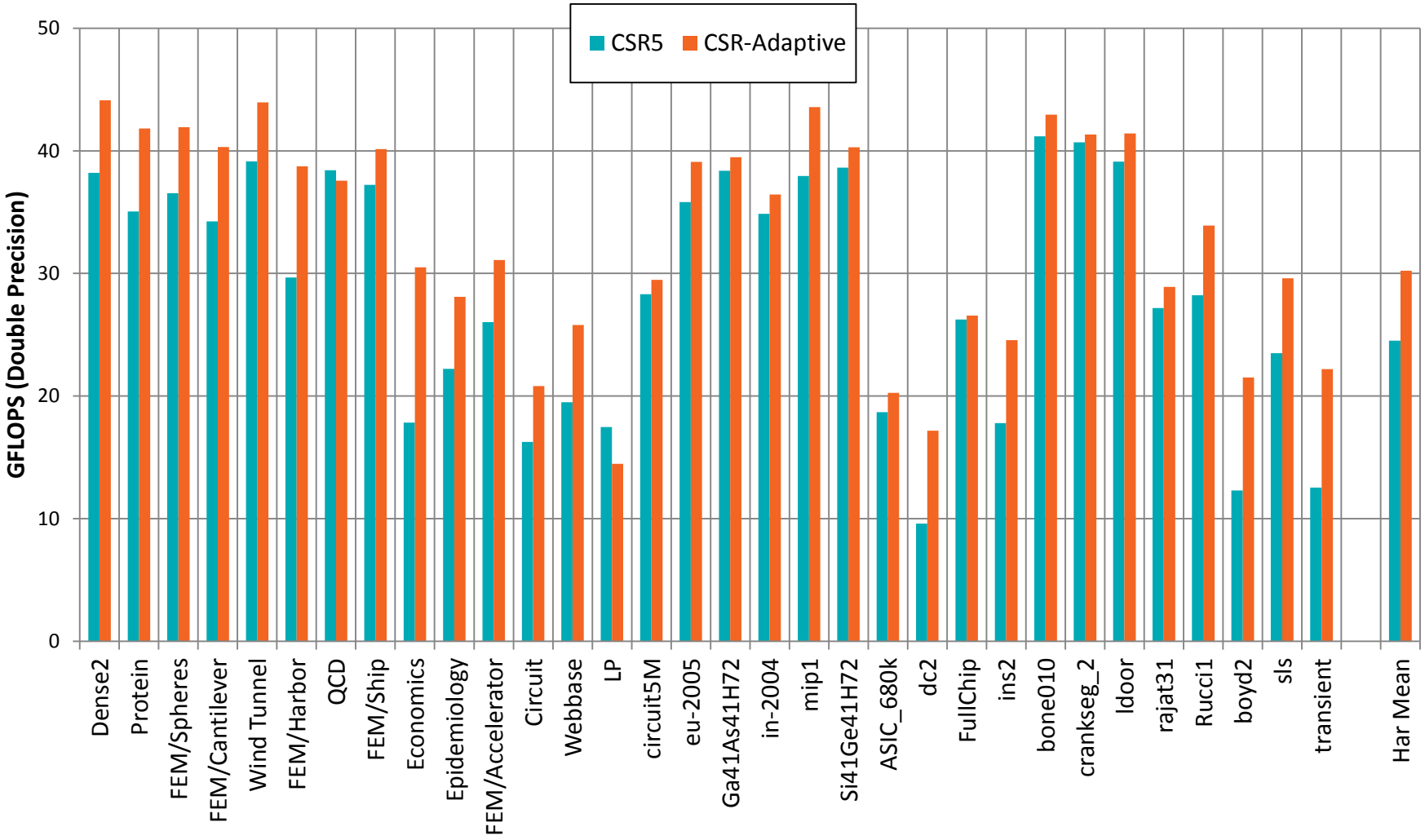


CSR-ADAPTIVE VS CSR5 (BEST IN ACADEMIA)

AMD FIREPRO W9100 GPU (DOUBLE PRECISION)

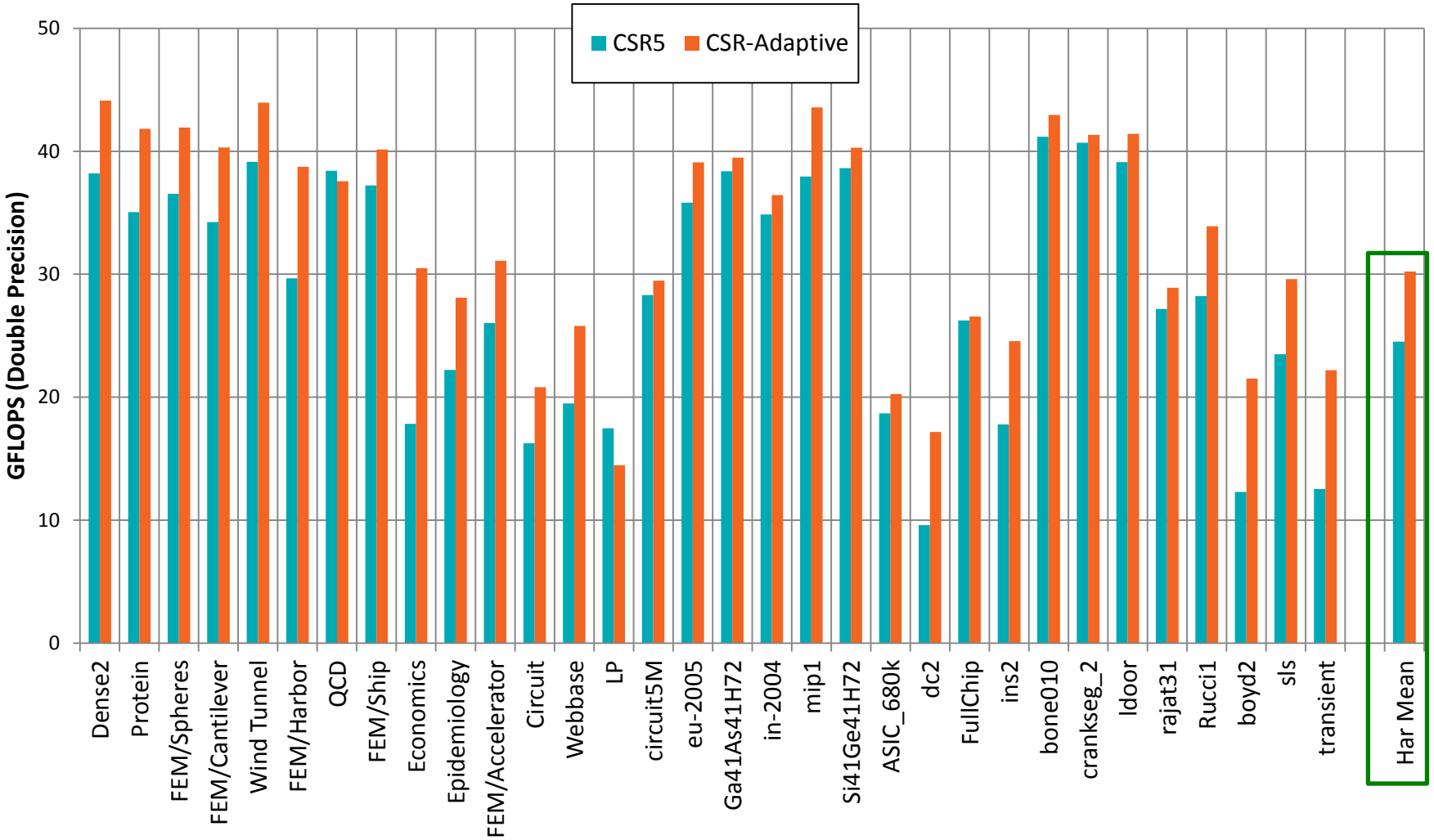
CSR-ADAPTIVE VS CSR5 (BEST IN ACADEMIA)

AMD FIREPRO W9100 GPU (DOUBLE PRECISION)

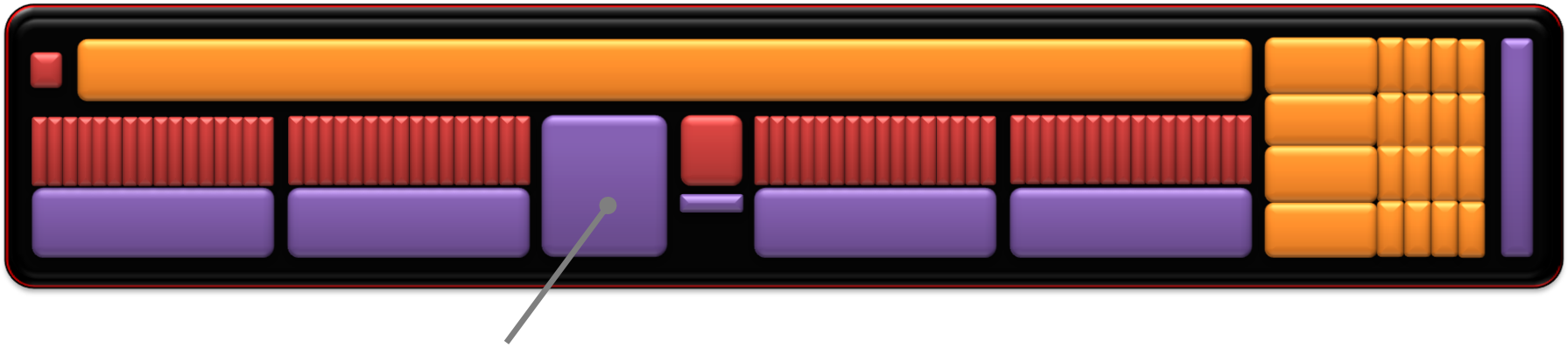


CSR-ADAPTIVE VS CSR5 (BEST IN ACADEMIA)

AMD FIREPRO W9100 GPU (DOUBLE PRECISION)



LOCAL DATA SHARE (LDS) MEMORY



- Scratchpad (software-addressed) on-chip cache
- Statically divided between all workgroups in a CU
- Highly ported to allow scatter/gather (uncoalesced) accesses