



EFFICIENT SPARSE MATRIX-VECTOR MULTIPLICATION ON GPUS USING THE CSR STORAGE FORMAT

JOSEPH L. GREATHOUSE, MAYANK DAGA
AMD RESEARCH
11/20/2014

THIS TALK IN ONE SLIDE



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

Optimized GPU-based SpMV algorithm that increases memory coalescing by using LDS

14.7x faster than other CSR-based algorithms
2.3x faster than using other matrix formats



Background.

SPARSE MATRIX-VECTOR MULTIPLICATION



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

 $\times$

1
2
3
4
5

 $=$

SPARSE MATRIX-VECTOR MULTIPLICATION



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

 $\times$

1
2
3
4
5

 $=$

SPARSE MATRIX-VECTOR MULTIPLICATION



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

 $\times$

1
2
3
4
5

 $=$

SPARSE MATRIX-VECTOR MULTIPLICATION



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

 $\times$

1
2
3
4
5

 $=$

SPARSE MATRIX-VECTOR MULTIPLICATION



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

 $\times$

1
2
3
4
5

 $=$

22

SPARSE MATRIX-VECTOR MULTIPLICATION



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

 $\times$

1
2
3
4
5

 $=$

22
28
18
39
18

SPARSE MATRIX-VECTOR MULTIPLICATION



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

 $\times$

1
2
3
4
5

 $=$

22
28
18
39
18

Traditionally Bandwidth Limited

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)



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)



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
---	---	---	---	---	---	---	---	---

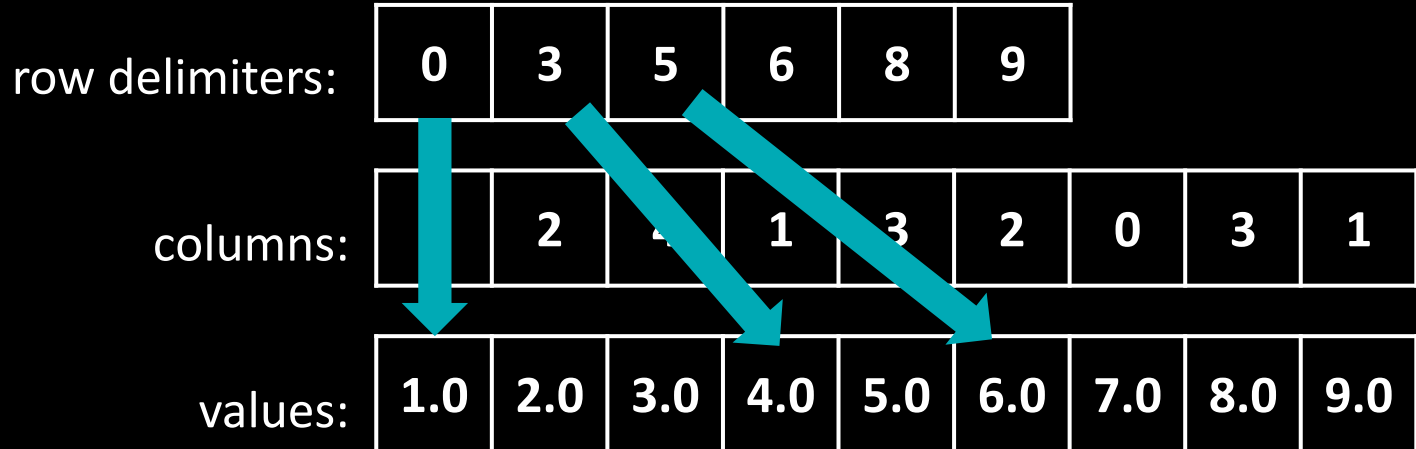
values:

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
---	---	---	---	---	---

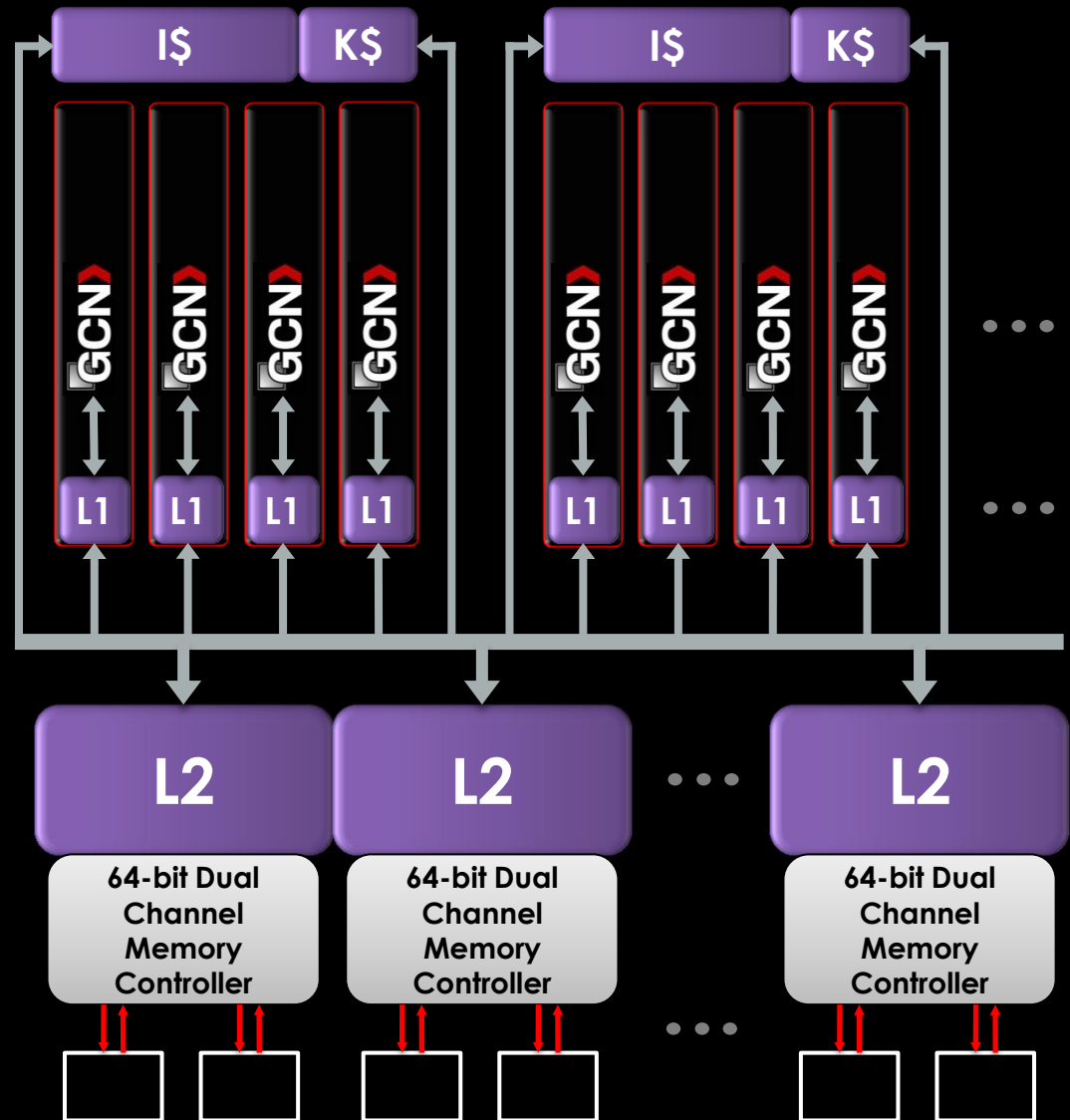
columns:

0	2	4	1	3	2	0	3	1
---	---	---	---	---	---	---	---	---

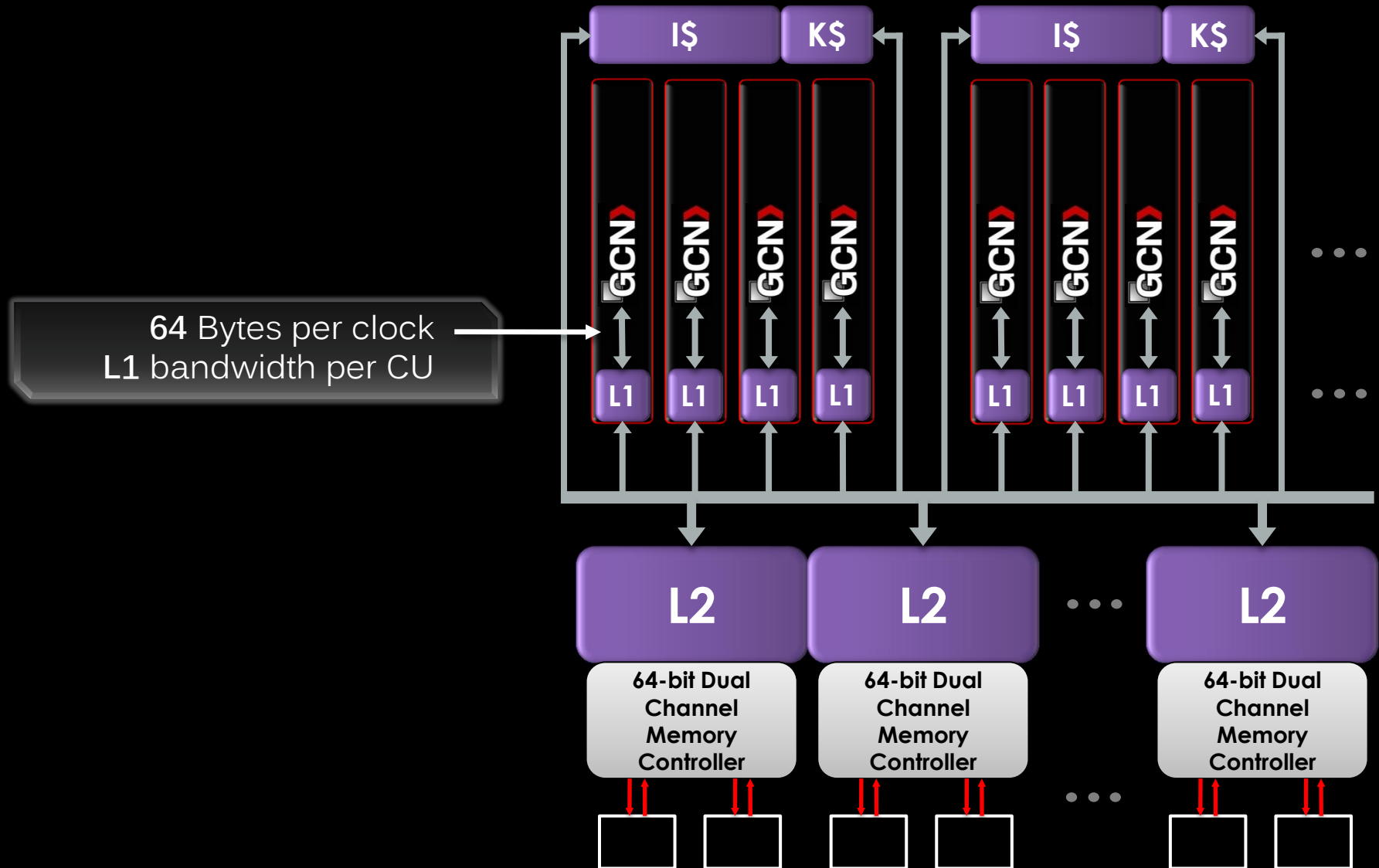
values:

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

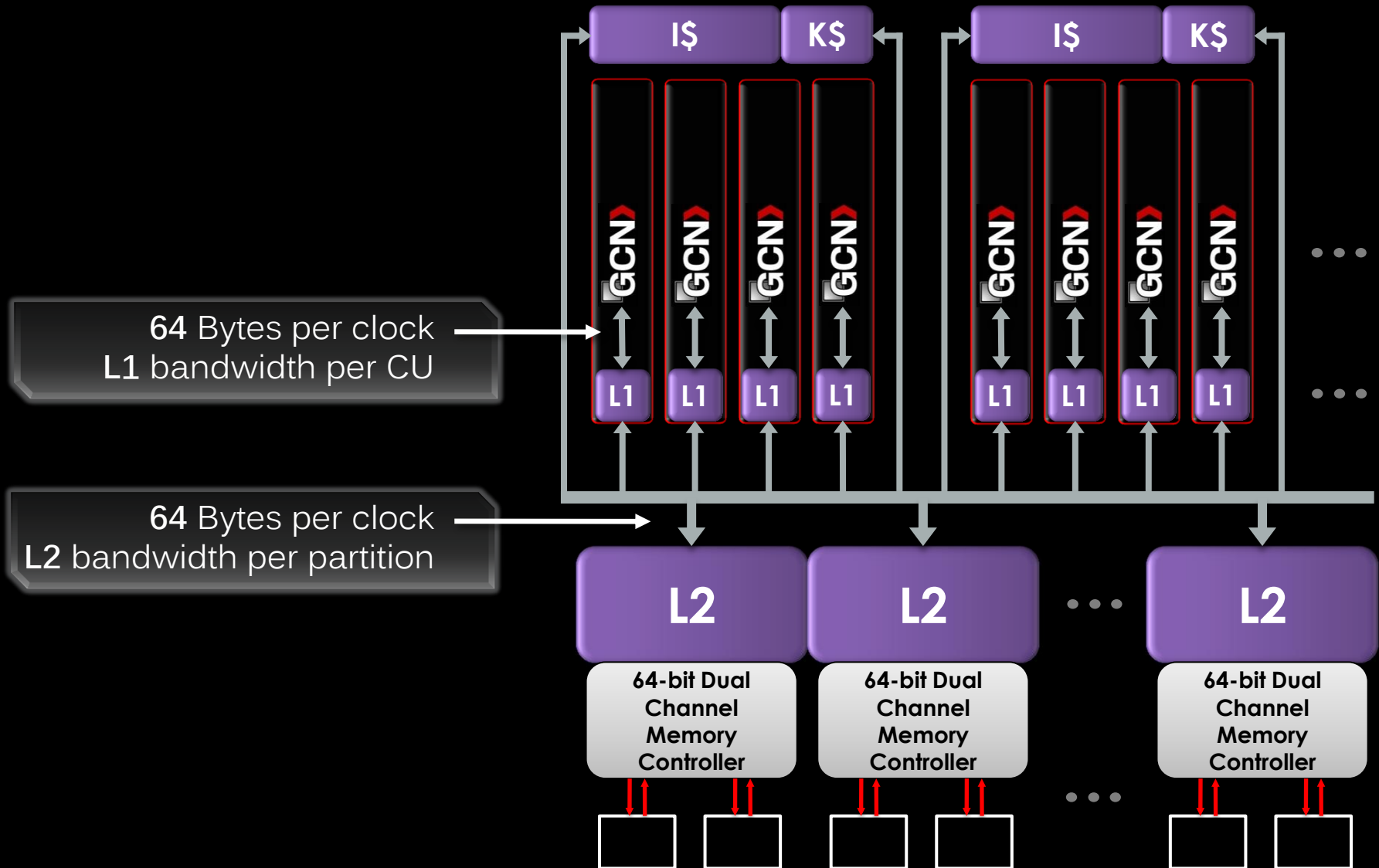
AMD GPU MEMORY SYSTEM



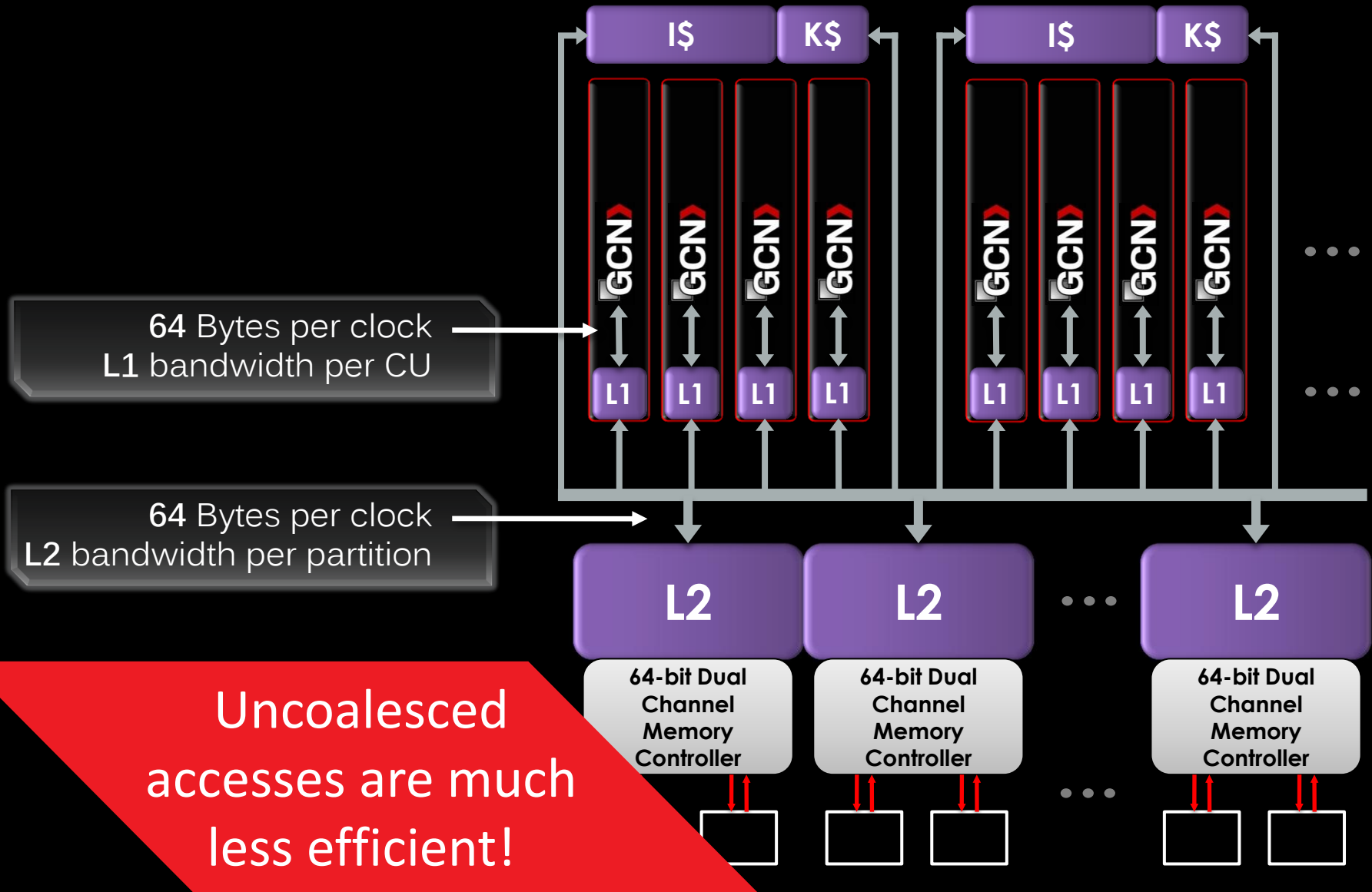
AMD GPU MEMORY SYSTEM



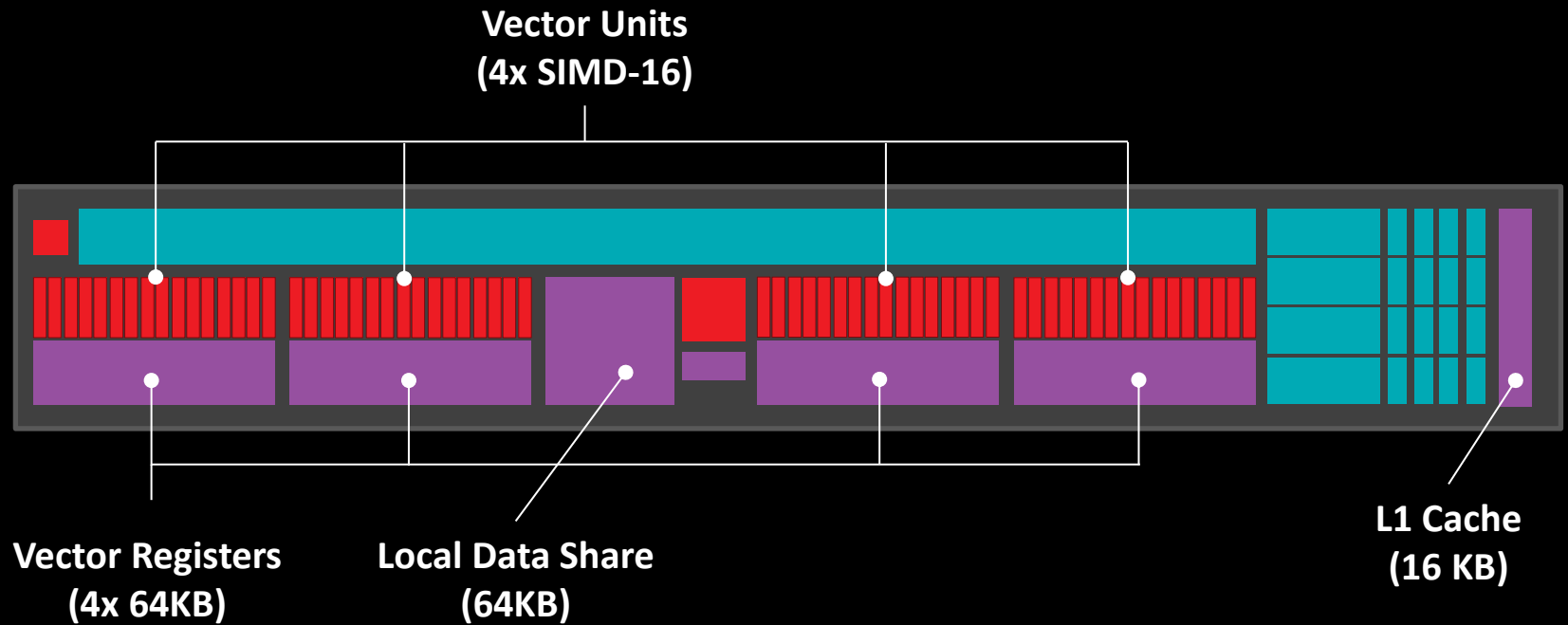
AMD GPU MEMORY SYSTEM



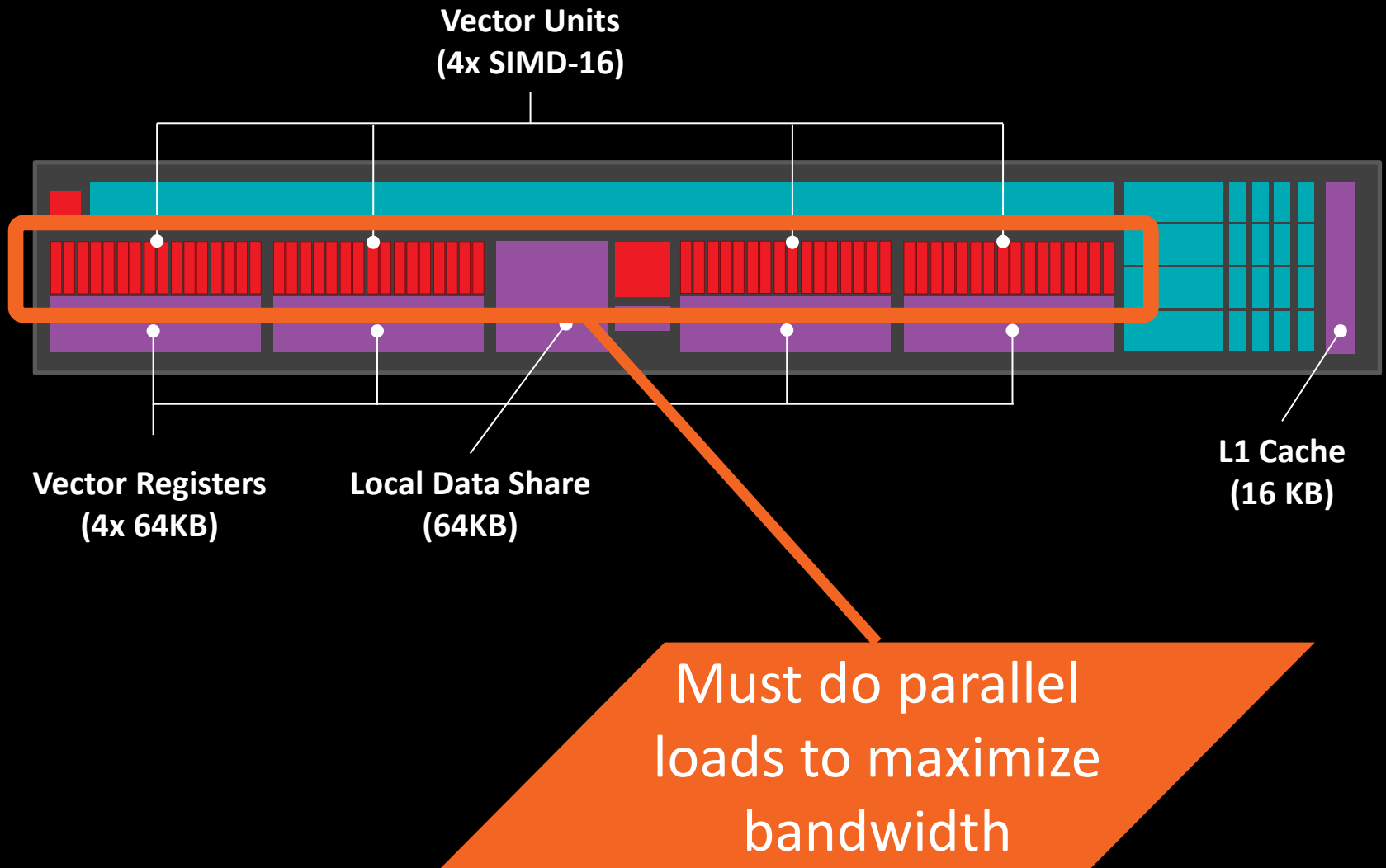
AMD GPU MEMORY SYSTEM



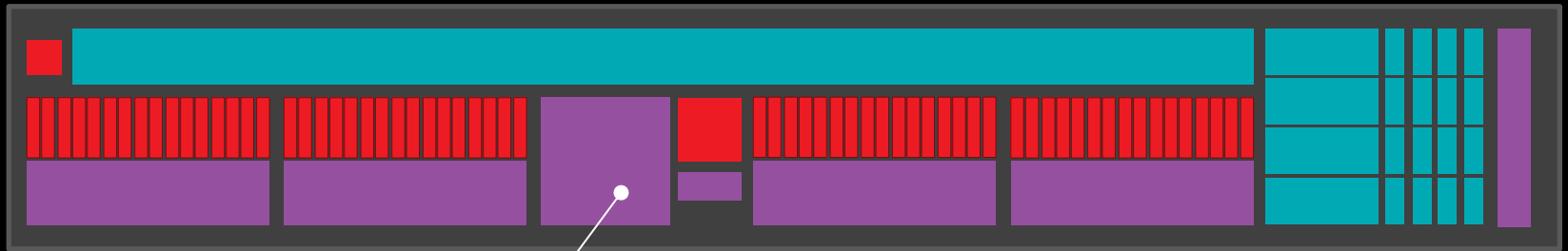
AMD GPU COMPUTE UNIT



AMD GPU COMPUTE UNIT



LOCAL DATA SHARE (LDS) MEMORY



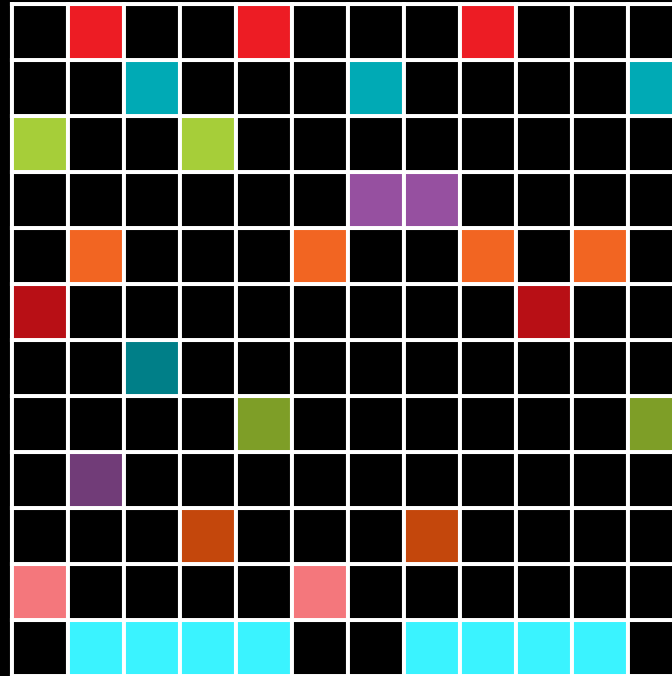
Local Data Share
(64KB)

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

CSR-SCALAR



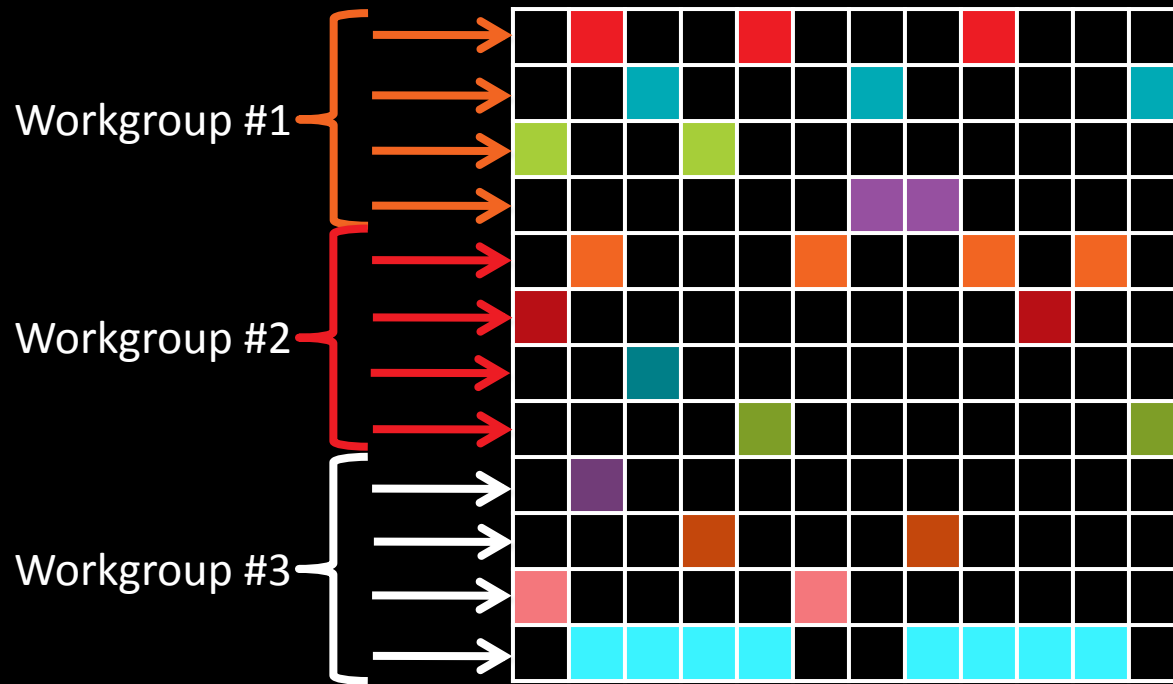
▲ One thread/work-item per matrix row



CSR-SCALAR



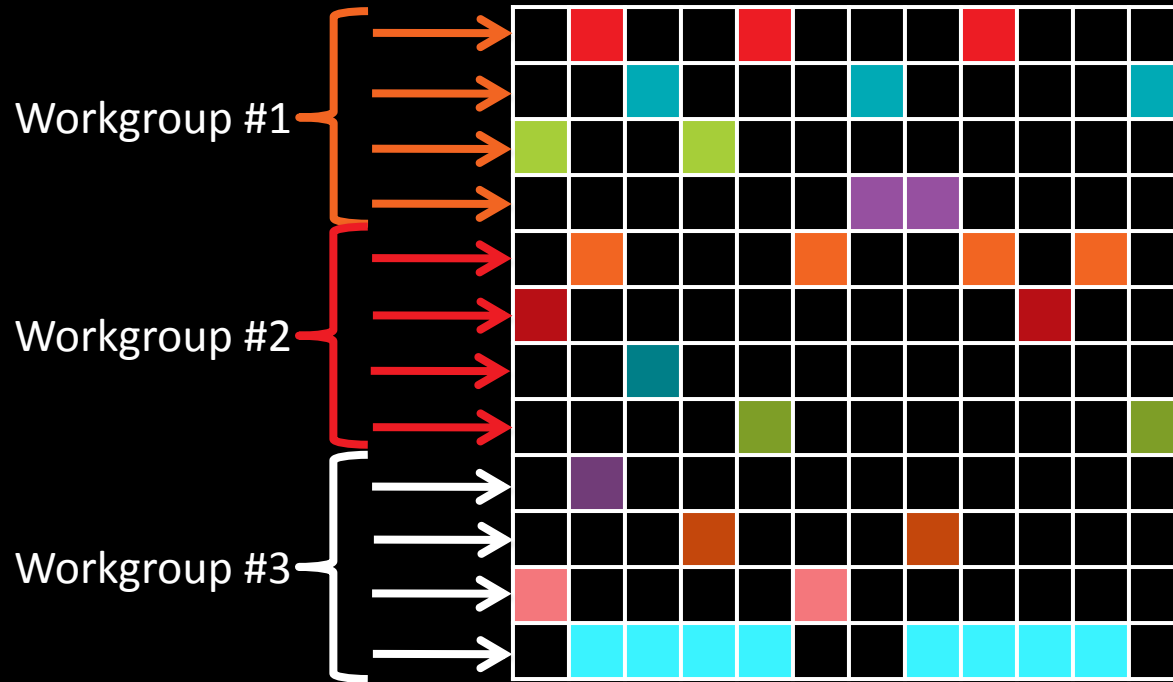
▲ One thread/work-item per matrix row



CSR-SCALAR



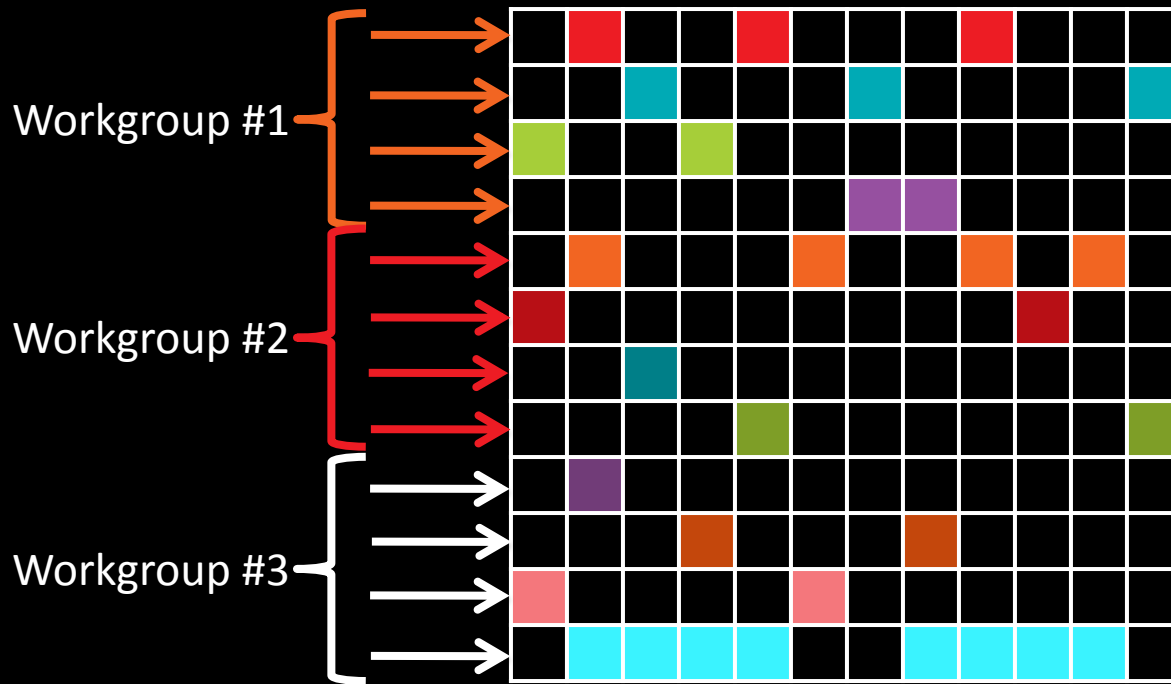
▲ One thread/work-item per matrix row



CSR-SCALAR



▲ One thread/work-item per matrix row



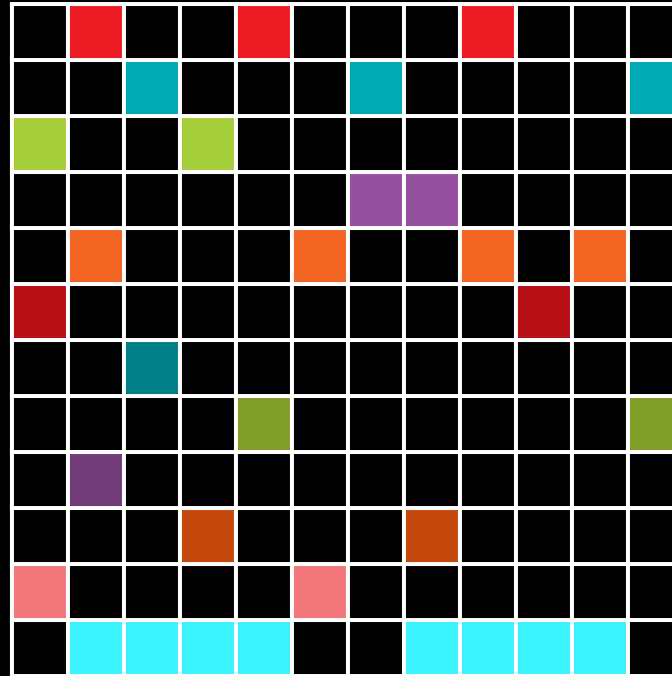
Uncoalesced accesses



CSR-VECTOR



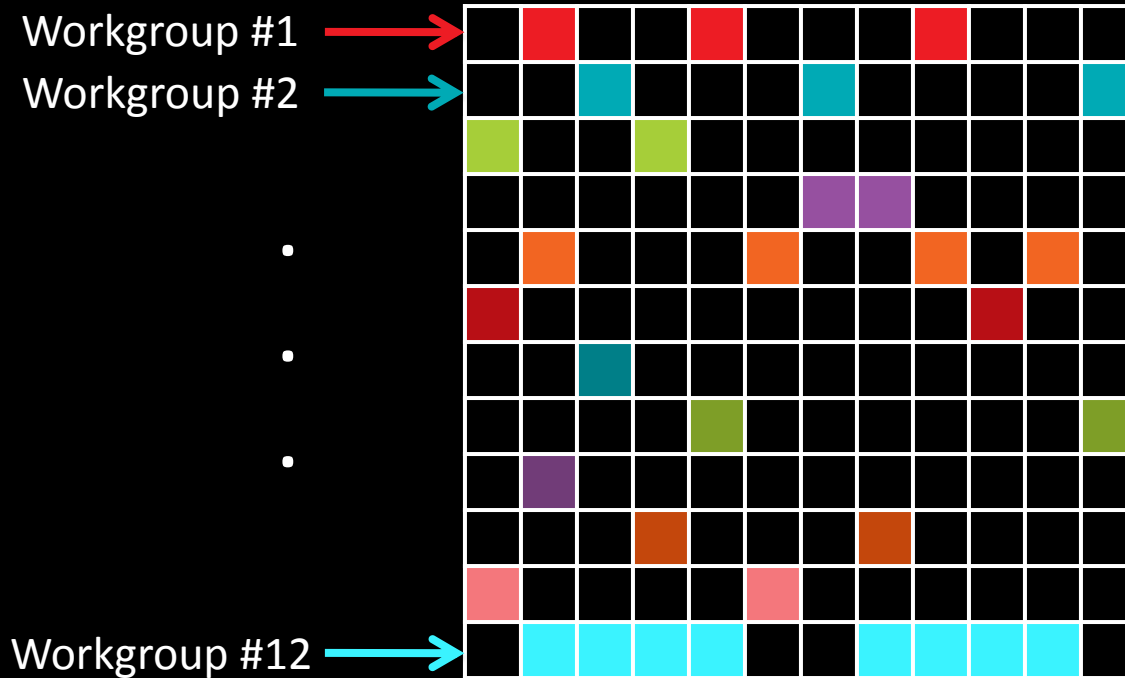
- ▲ One workgroup (multiple threads) per matrix row



CSR-VECTOR



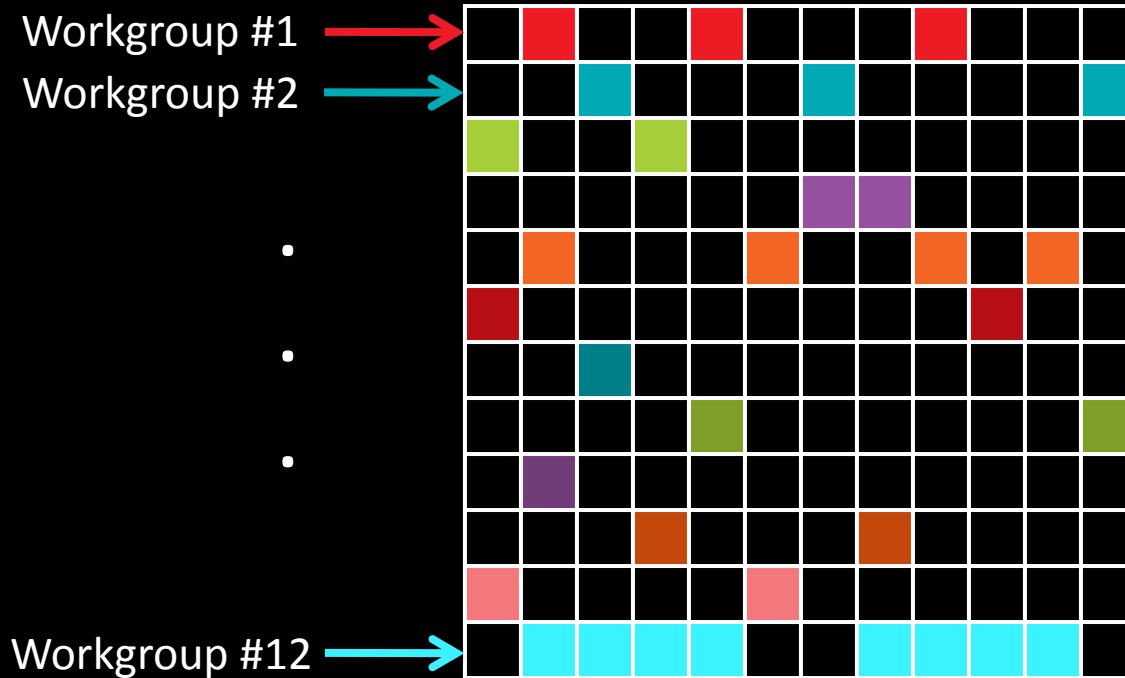
▲ One workgroup (multiple threads) per matrix row



CSR-VECTOR



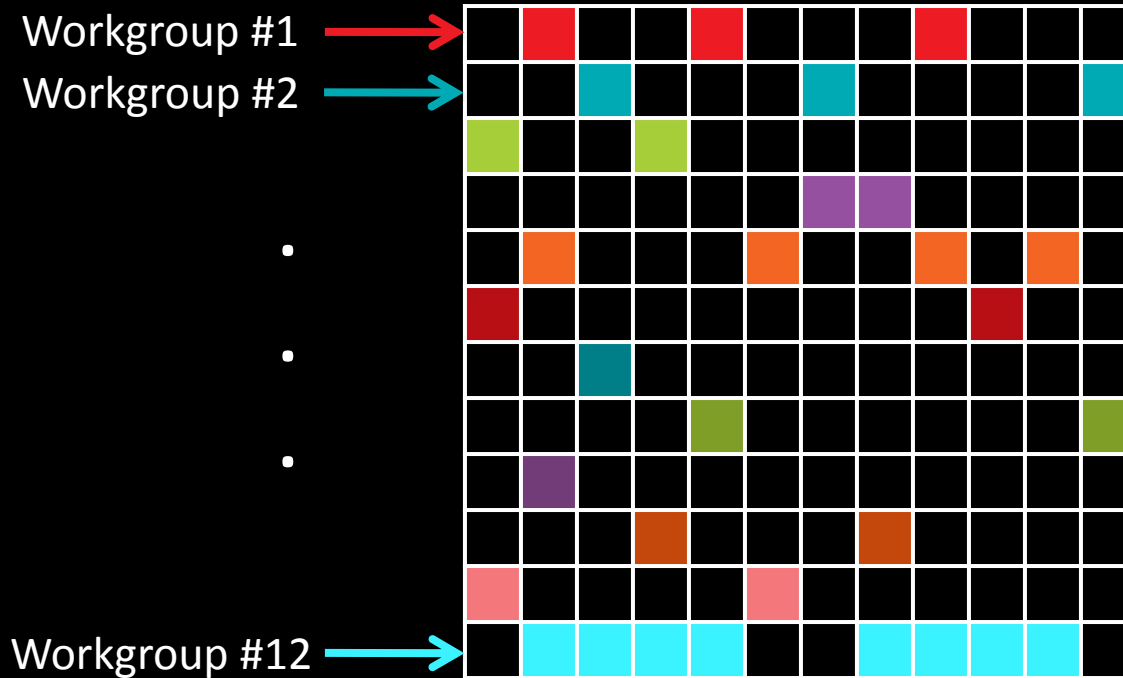
▲ One workgroup (multiple threads) per matrix row



CSR-VECTOR



▲ One workgroup (multiple threads) per matrix row



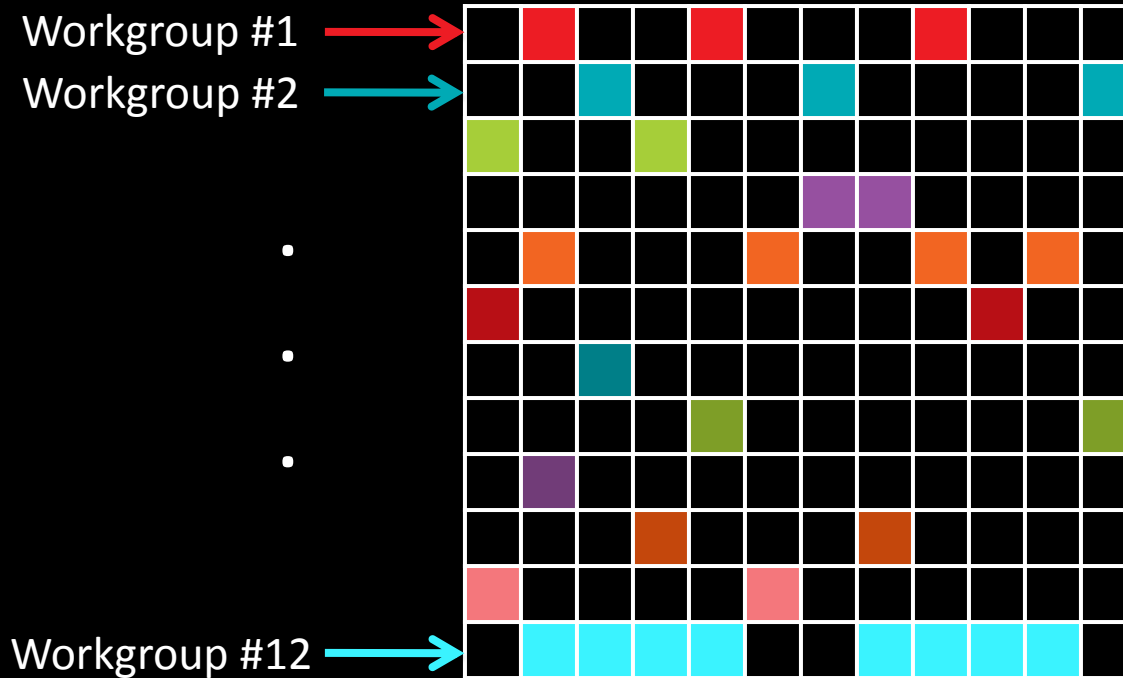
Good: coalesced accesses



CSR-VECTOR



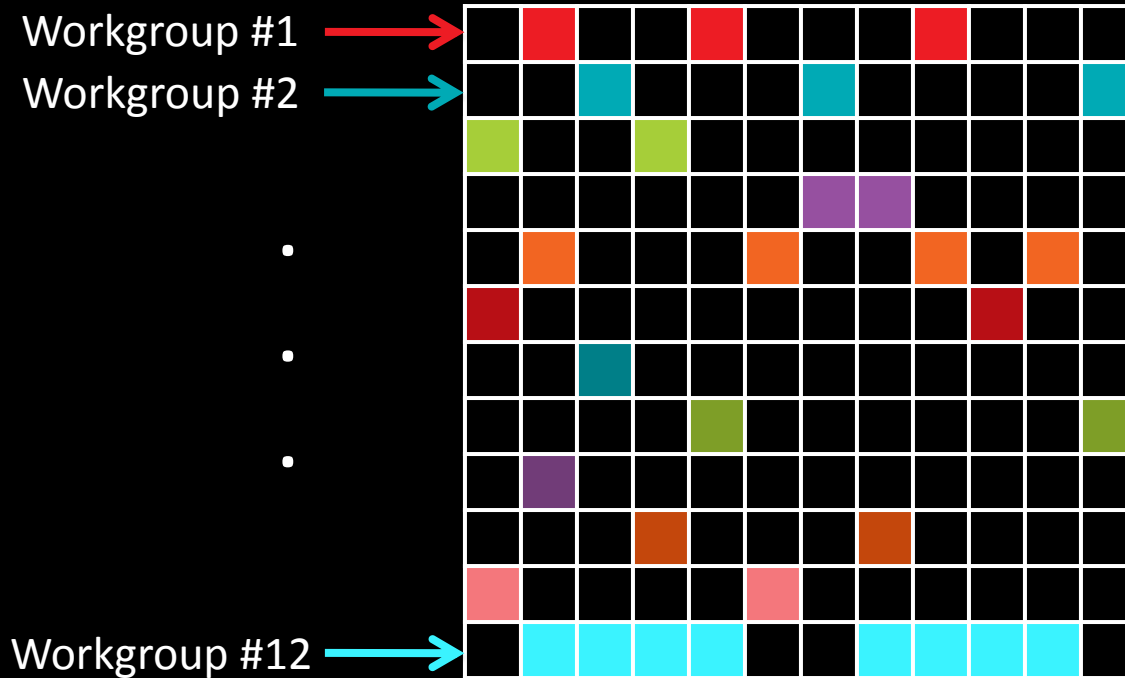
▲ One workgroup (multiple threads) per matrix row



CSR-VECTOR



▲ One workgroup (multiple threads) per matrix row



Bad: poor parallelism

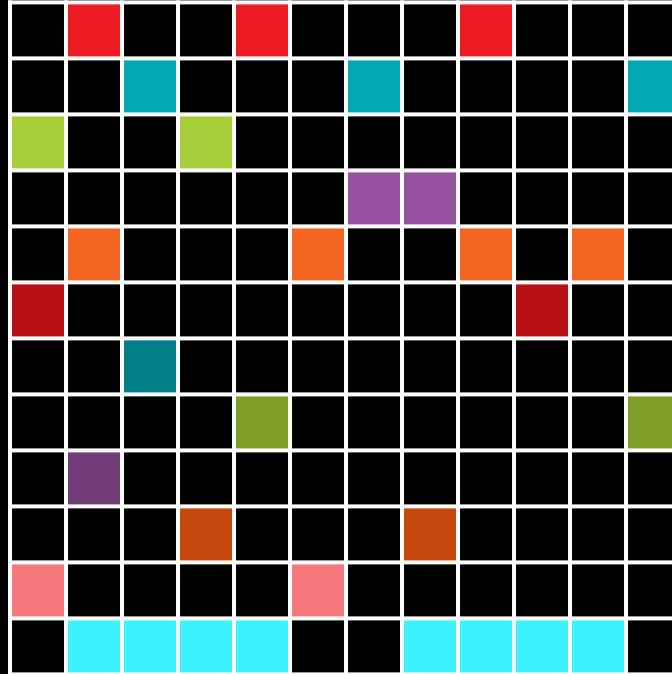


- ▲ Neither CSR-Scalar nor CSR-Vector offer ideal performance.
- ▲ CSR works well on CPUs and is widely used
- ▲ Traditional solution: new matrix storage format + new algorithms
 - ELLPACK/ITPACK, BCSR, DIA, BDIA, SELL, SBELL, ELL+COO Hybrid, many others
 - Auto-tuning frameworks like clSpMV try to find the best format for each matrix
 - Can require more storage space or dynamic transformation overhead
- ▲ Can we find a good GPU algorithm that does not modify the original CSR data?

The background of the slide is black. It features several large, overlapping geometric shapes. A large red triangle points upwards from the bottom left towards the center. A cyan parallelogram is positioned in the upper left, partially overlapping the red shape. A smaller cyan parallelogram is located to the right of the larger cyan one. A thin, dark grey horizontal bar is situated between the red triangle and the cyan shapes.

CSR-Adaptive.

INCREASING BANDWIDTH EFFICIENCY



INCREASING BANDWIDTH EFFICIENCY



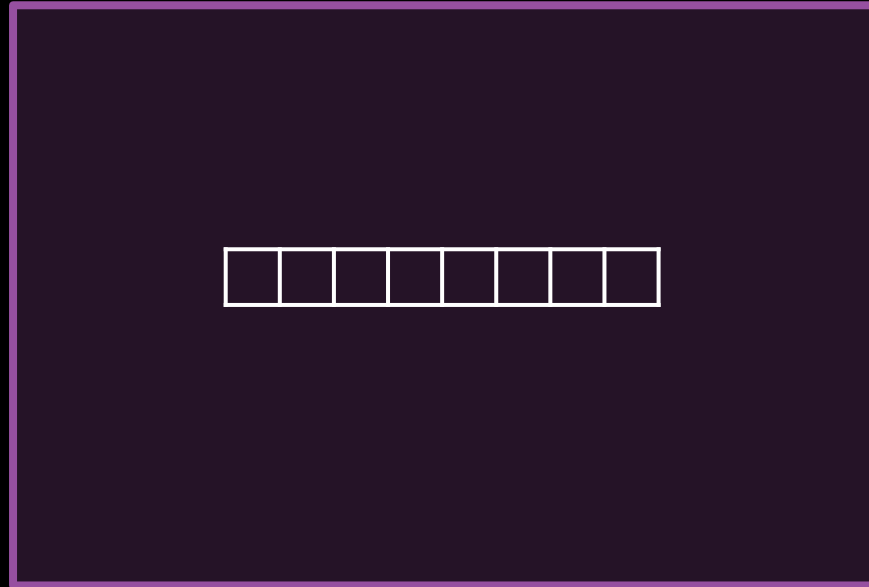
INCREASING BANDWIDTH EFFICIENCY



CSR-VECTOR USING THE LDS



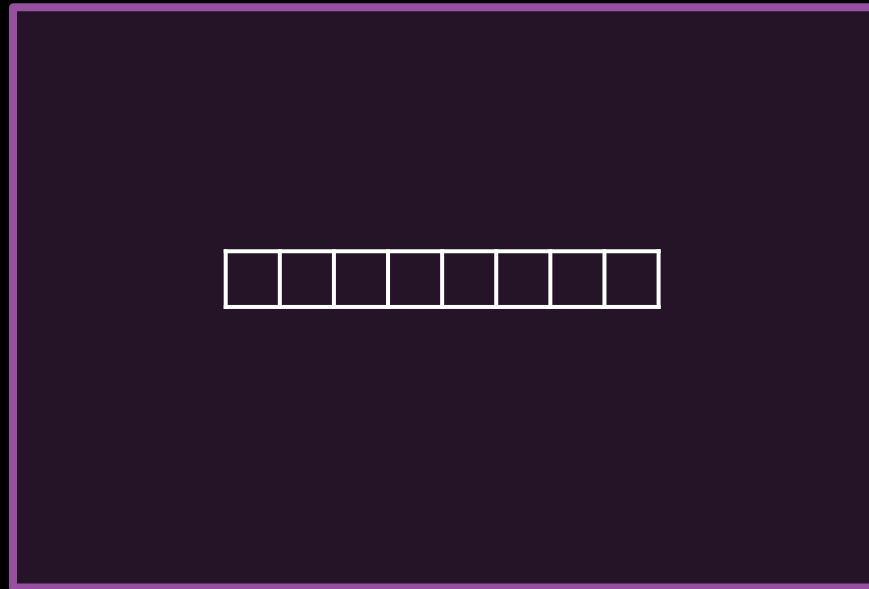
Local Data Share



CSR-VECTOR USING THE LDS



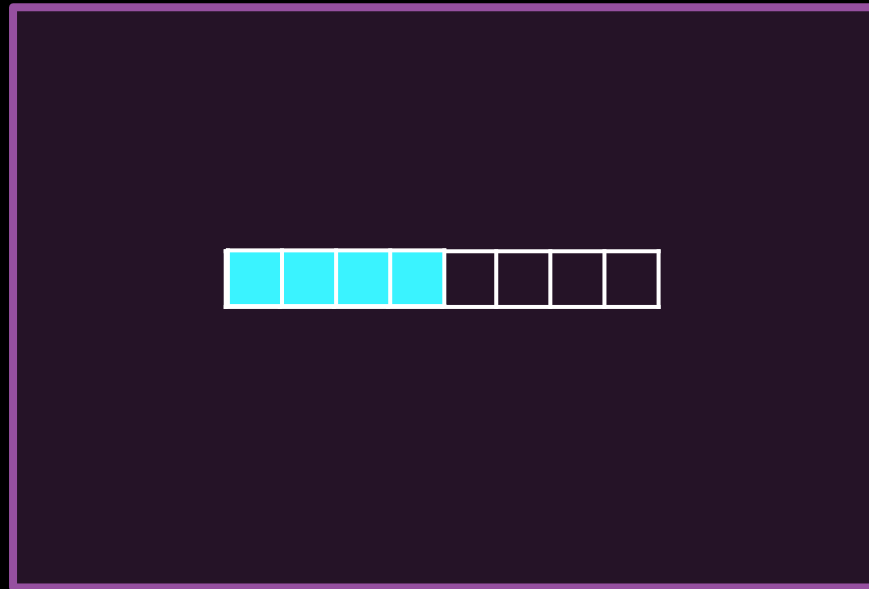
Local Data Share



CSR-VECTOR USING THE LDS



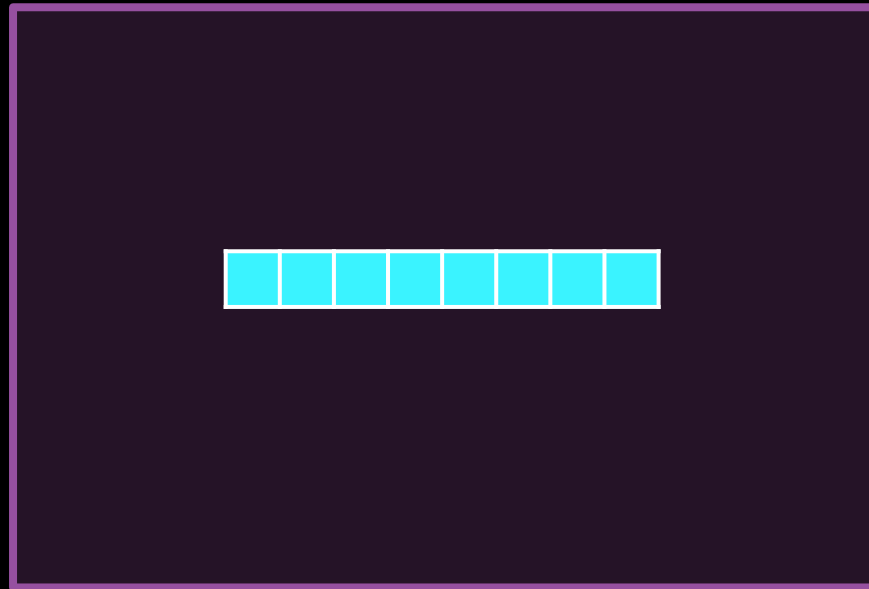
Local Data Share



CSR-VECTOR USING THE LDS



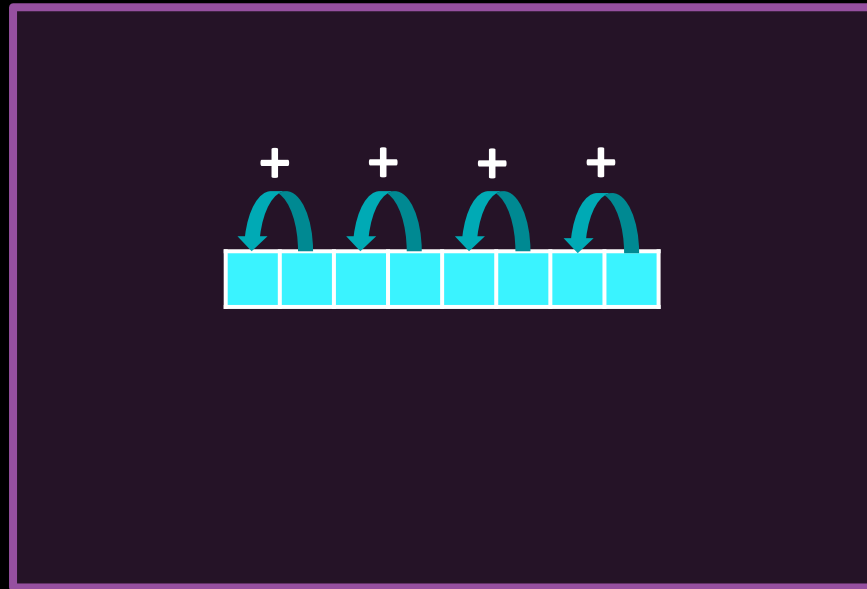
Local Data Share



CSR-VECTOR USING THE LDS



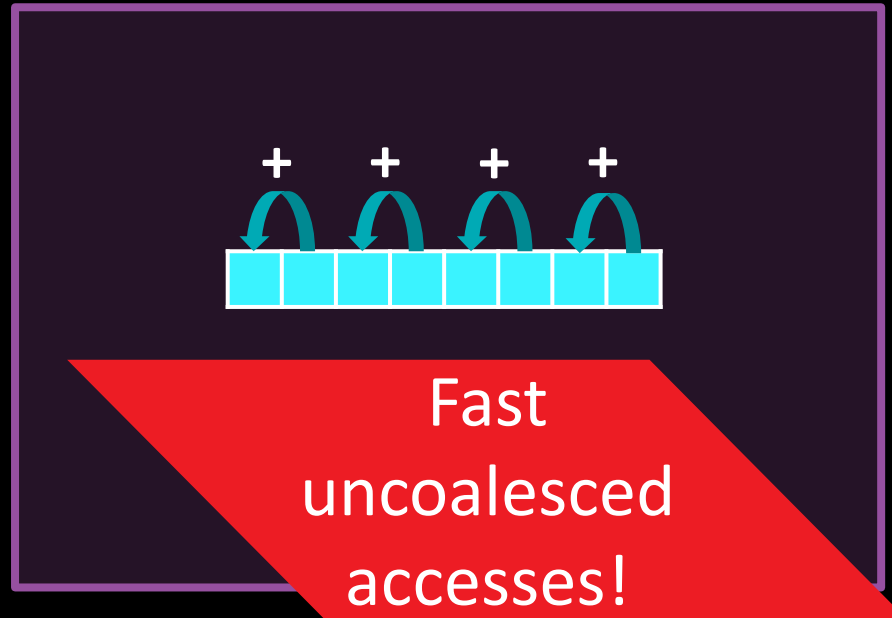
Local Data Share



CSR-VECTOR USING THE LDS



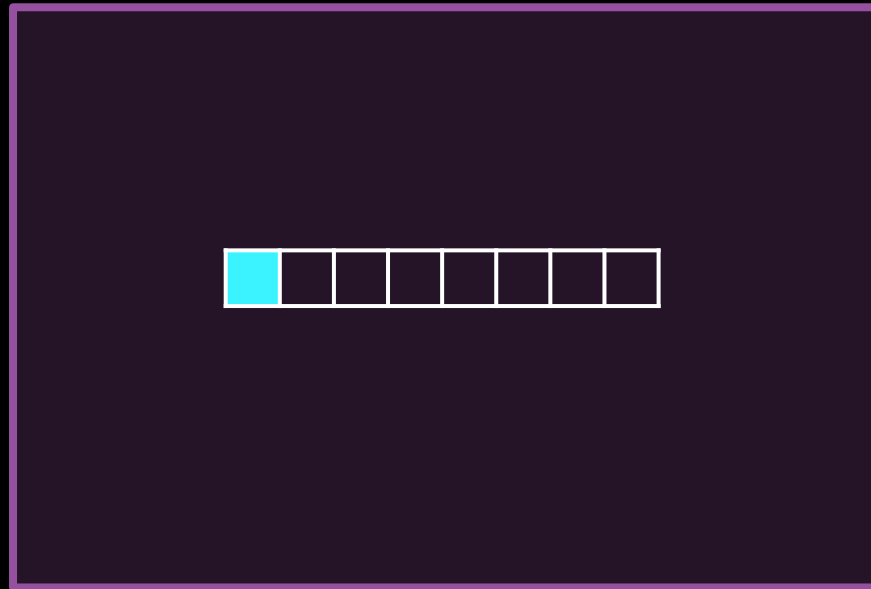
Local Data Share



CSR-VECTOR USING THE LDS



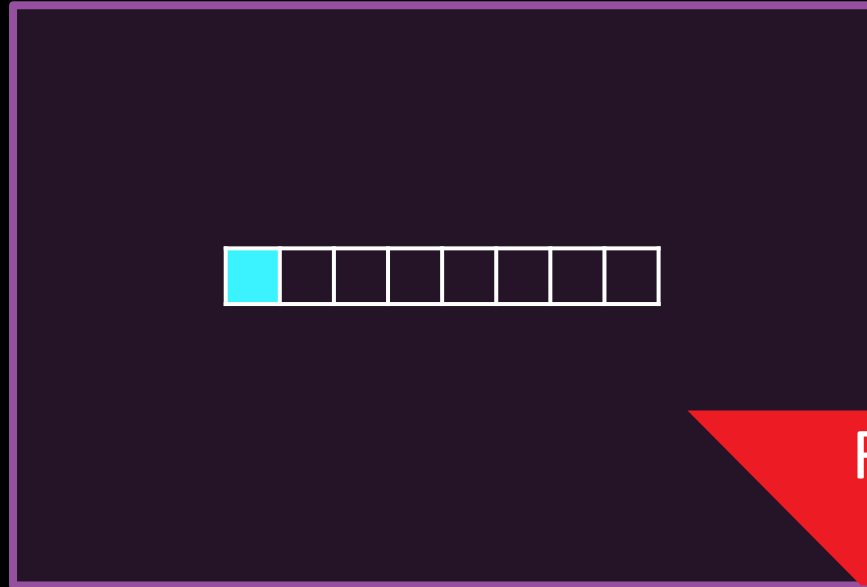
Local Data Share



CSR-VECTOR USING THE LDS



Local Data Share



Fast accesses
everywhere



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



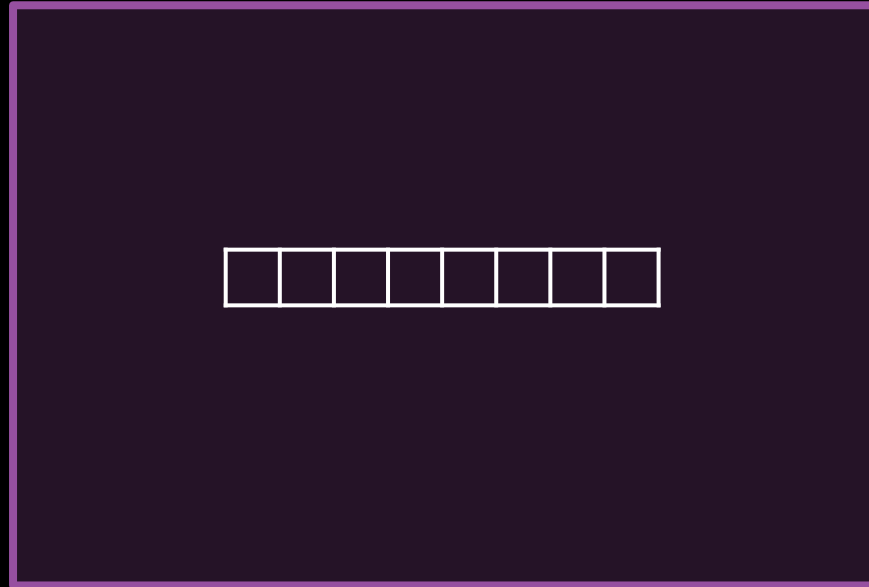
▲ How can we get good bandwidth for other “short” rows?



CSR-STREAM: STREAM MATRIX INTO THE LDS



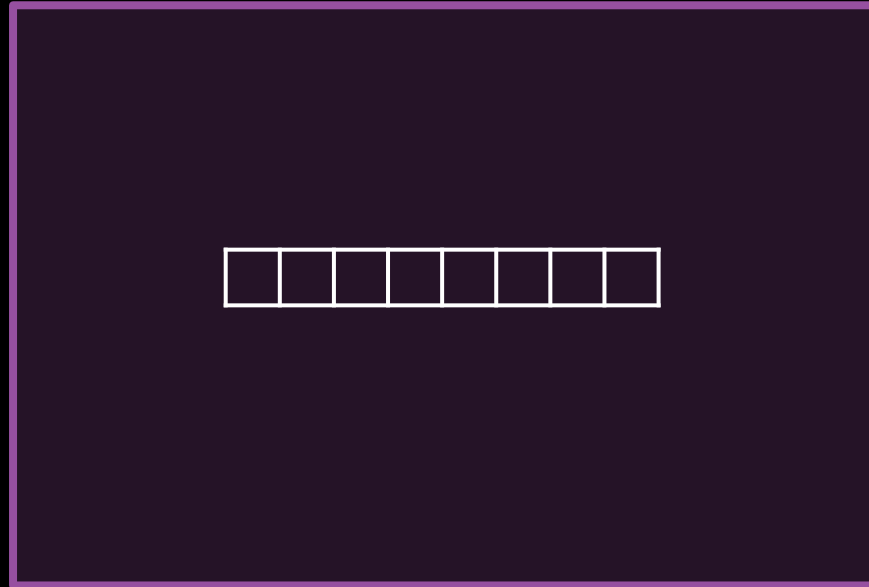
Local Data Share



CSR-STREAM: STREAM MATRIX INTO THE LDS



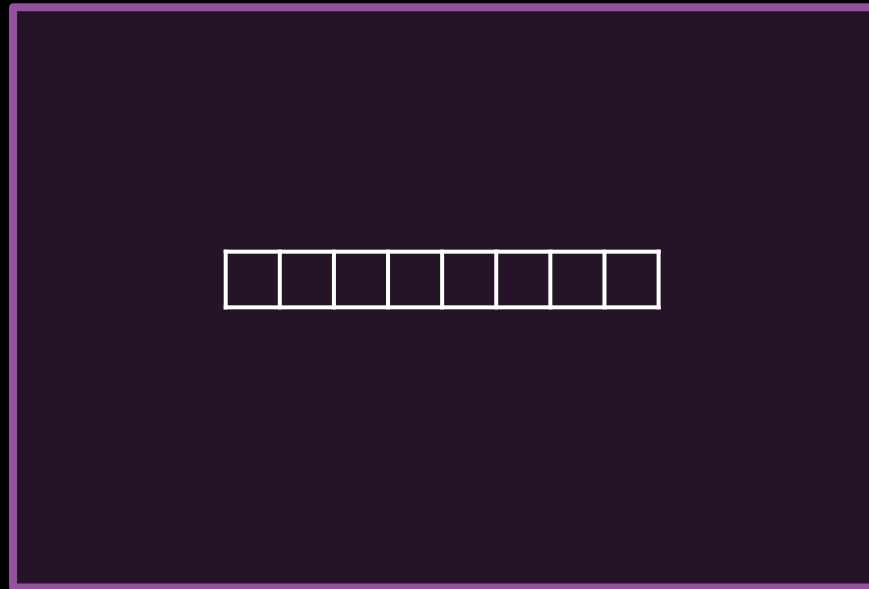
Local Data Share



CSR-STREAM: STREAM MATRIX INTO THE LDS



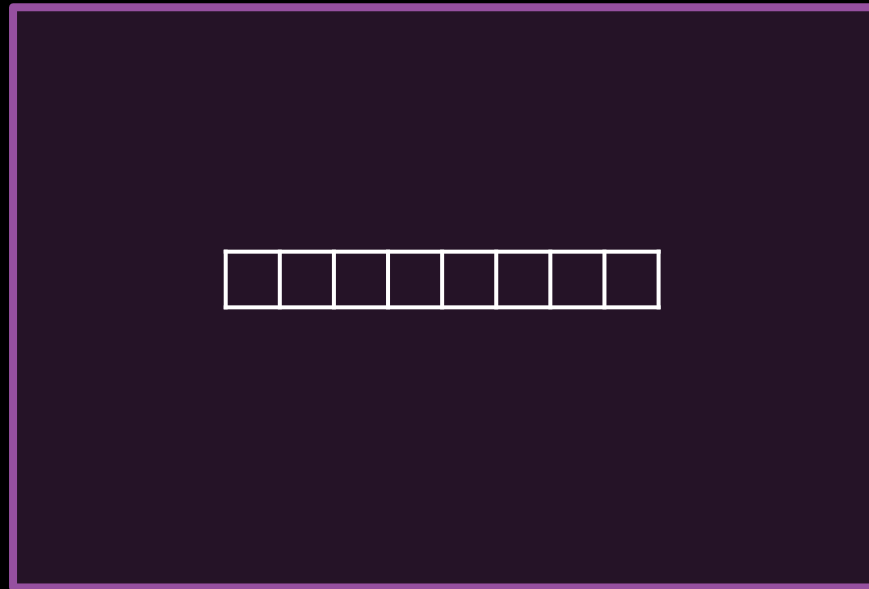
Local Data Share



CSR-STREAM: STREAM MATRIX INTO THE LDS



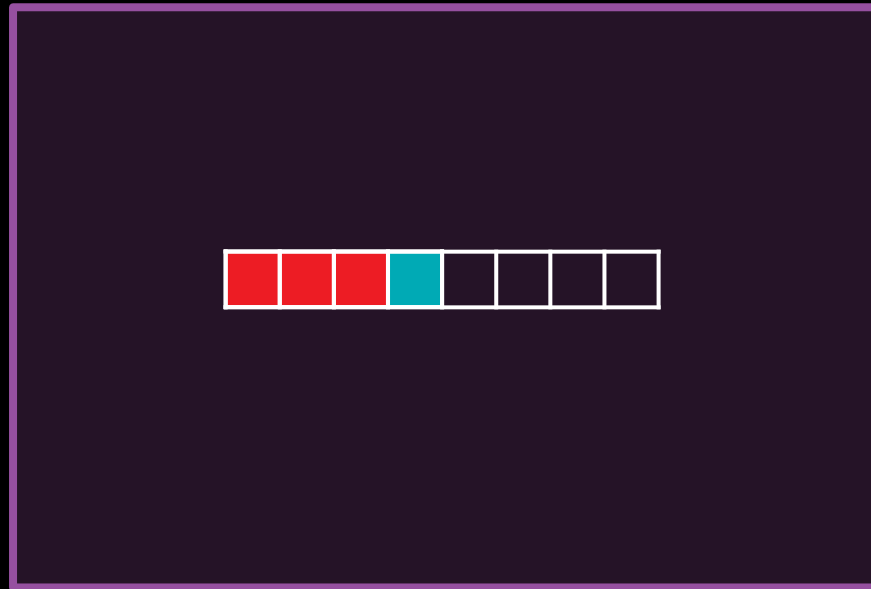
Local Data Share



CSR-STREAM: STREAM MATRIX INTO THE LDS



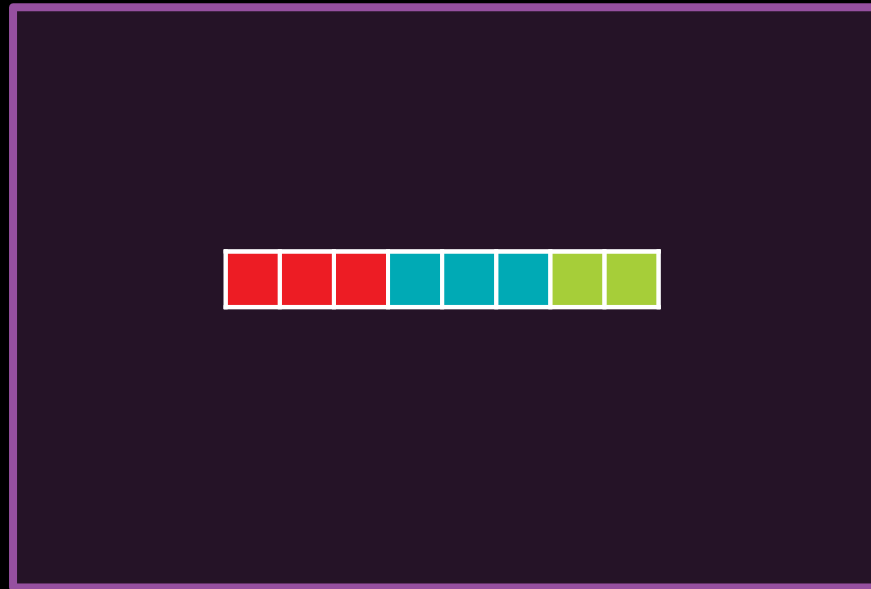
Local Data Share



CSR-STREAM: STREAM MATRIX INTO THE LDS



Local Data Share



CSR-STREAM: STREAM MATRIX INTO THE LDS



Local Data Share



CSR-STREAM: STREAM MATRIX INTO THE LDS



Local Data Share



CSR-STREAM: STREAM MATRIX INTO THE LDS



Local Data Share



Fast accesses
everywhere



FINDING BLOCKS FOR CSR-STREAM

DONE ONCE ON THE CPU



row delimiters:

0	3	6	8	10	14	16	17	19	20	22	24	32
---	---	---	---	----	----	----	----	----	----	----	----	----

columns:



values:



FINDING BLOCKS FOR CSR-STREAM

DONE ONCE ON THE CPU



row delimiters:

0	3	6	8	10	14	16	17	19	20	22	24	32
---	---	---	---	----	----	----	----	----	----	----	----	----

columns:



values:



FINDING BLOCKS FOR CSR-STREAM

DONE ONCE ON THE CPU



row delimiters:



columns:



values:



FINDING BLOCKS FOR CSR-STREAM

DONE ONCE ON THE CPU



Assuming 8 LDS entries per workgroup

row blocks:

0

row delimiters:



columns:



values:



FINDING BLOCKS FOR CSR-STREAM

DONE ONCE ON THE CPU



Assuming 8 LDS entries per workgroup

row blocks:

0

row delimiters:



columns:



values:



FINDING BLOCKS FOR CSR-STREAM

DONE ONCE ON THE CPU



Assuming 8 LDS entries per workgroup

row blocks:

0

row delimiters:



columns:



values:



FINDING BLOCKS FOR CSR-STREAM

DONE ONCE ON THE CPU



Assuming 8 LDS entries per workgroup

row blocks:

0

row delimiters:



columns:



values:



FINDING BLOCKS FOR CSR-STREAM

DONE ONCE ON THE CPU



Assuming 8 LDS entries per workgroup

row blocks:

0	3
---	---

row delimiters:



columns:



values:



Block 1

FINDING BLOCKS FOR CSR-STREAM

DONE ONCE ON THE CPU



Assuming 8 LDS entries per workgroup

row blocks:

0	3	6
---	---	---

row delimiters:



columns:



values:



Block 1

Block 2

FINDING BLOCKS FOR CSR-STREAM

DONE ONCE ON THE CPU



Assuming 8 LDS entries per workgroup

row blocks:

0	3	6	11	12
---	---	---	----	----

row delimiters:



columns:



values:



Block 1

Block 2

Block 3

Block 4

FINDING BLOCKS FOR CSR-STREAM

DONE ONCE ON THE CPU



Assuming 8 LDS entries per workgroup

row blocks:

0	3	6	11	12
---	---	---	----	----

CSR Structure
Unchanged

row delimiters:



columns:



values:



Block 1

Block 2

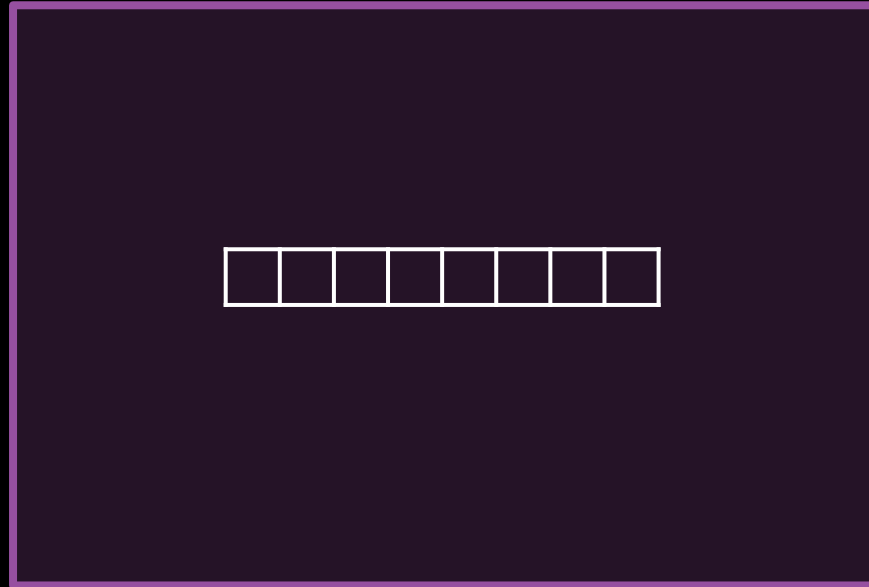
Block 3

Block 4

WHAT ABOUT VERY LONG ROWS?



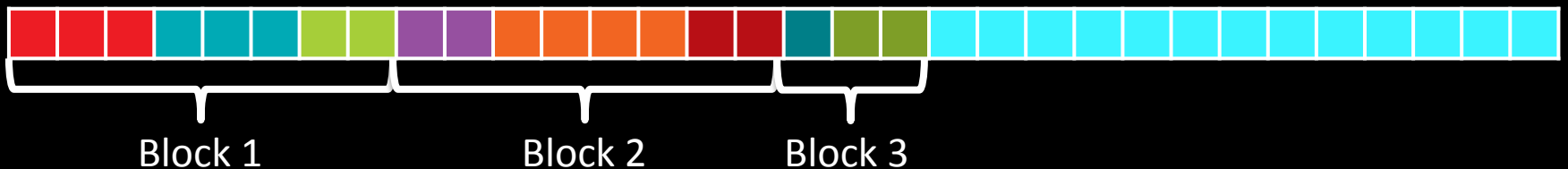
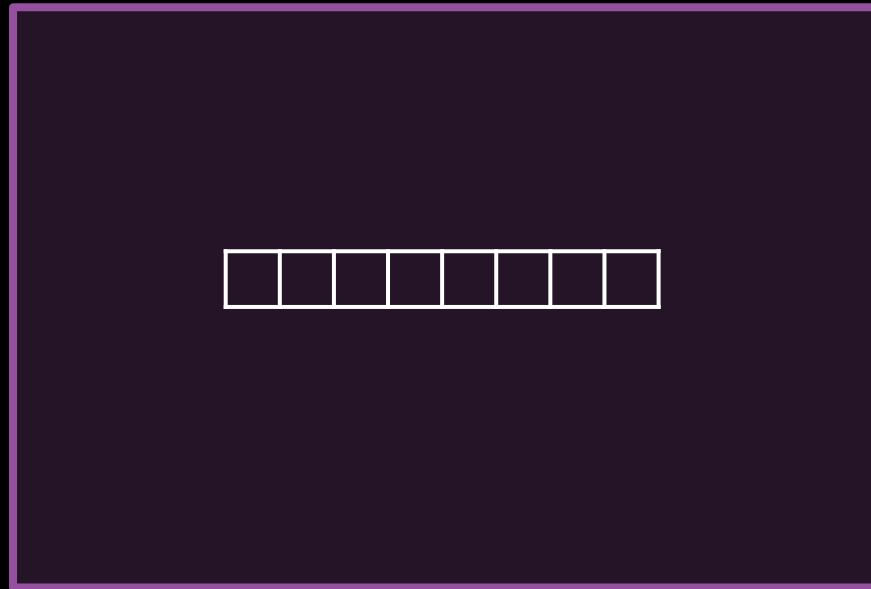
Local Data Share



WHAT ABOUT VERY LONG ROWS?



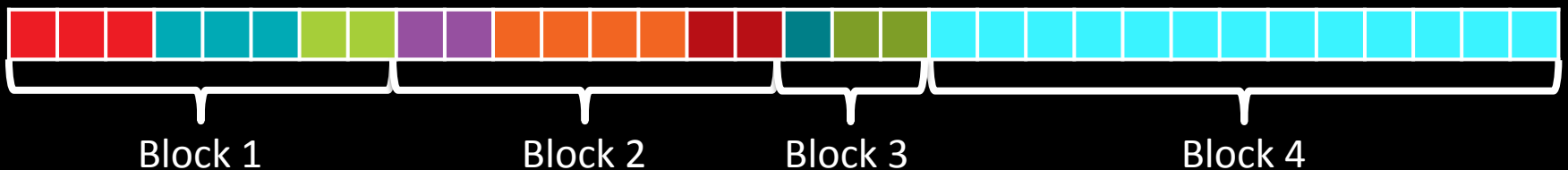
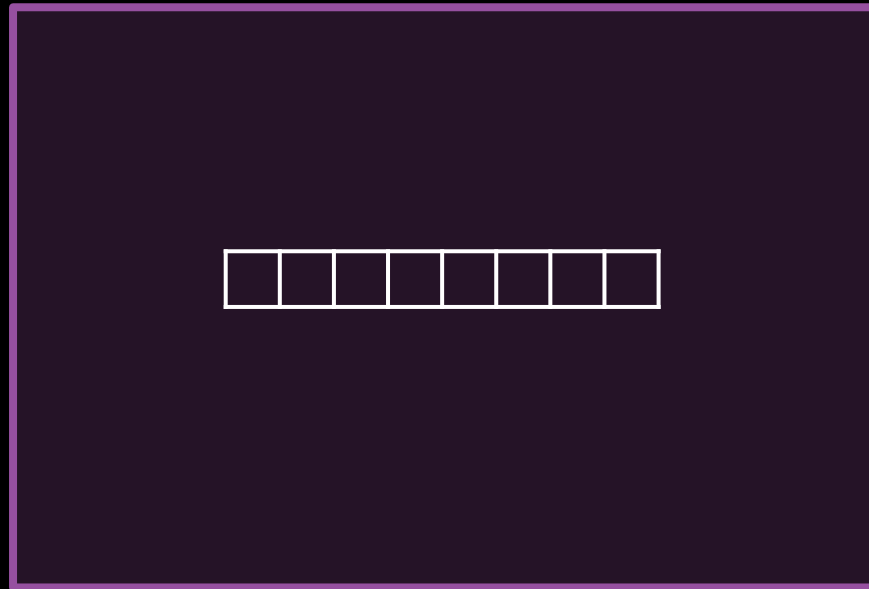
Local Data Share



WHAT ABOUT VERY LONG ROWS?



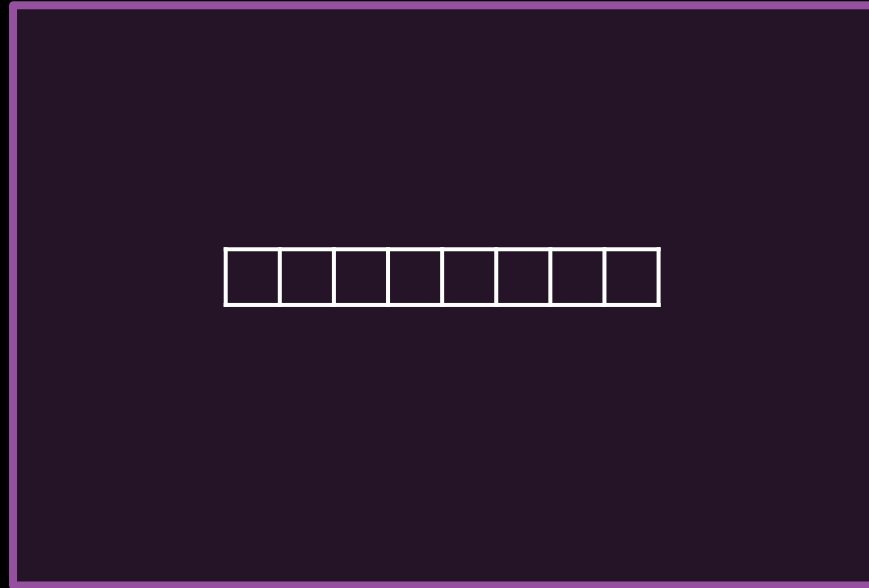
Local Data Share



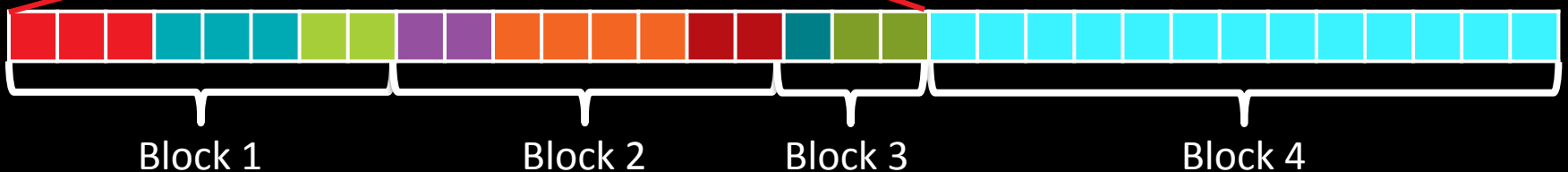
WHAT ABOUT VERY LONG ROWS?



Local Data Share



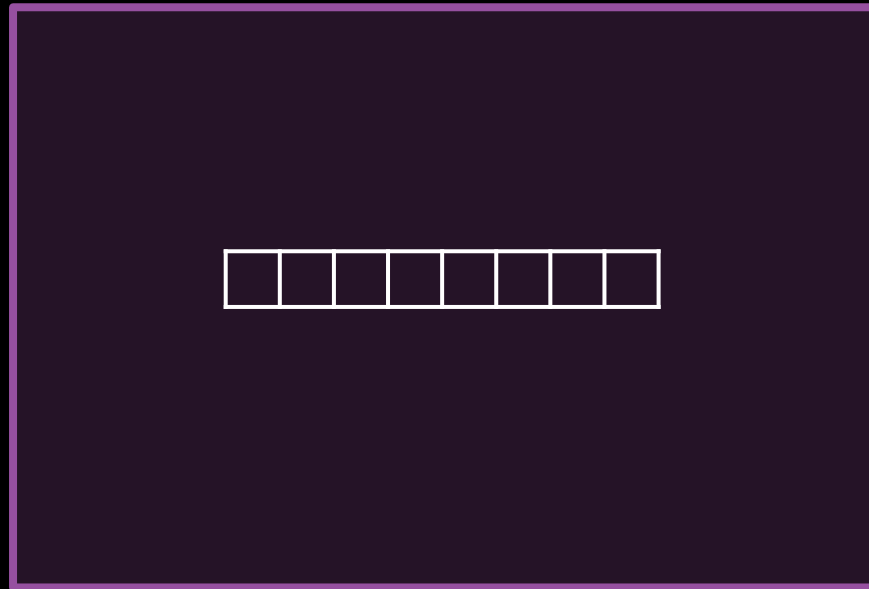
CSR-Stream



WHAT ABOUT VERY LONG ROWS?

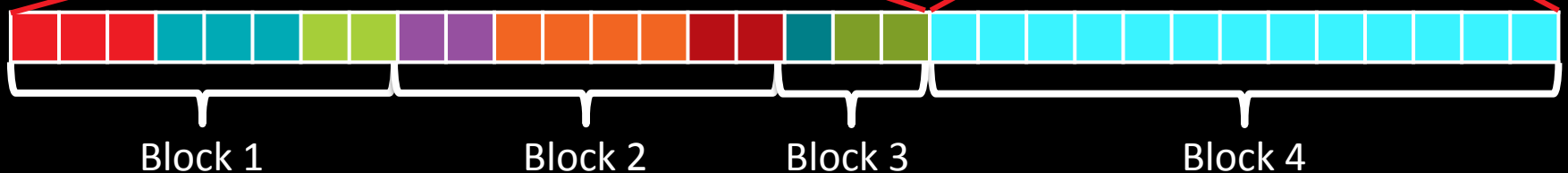


Local Data Share

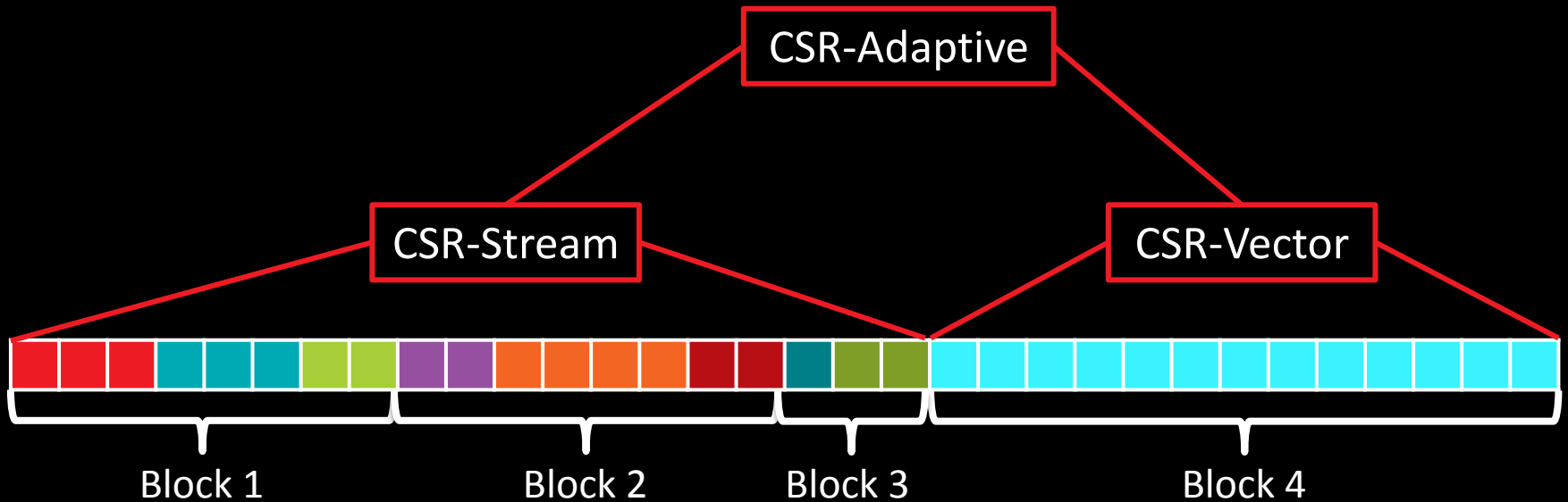


CSR-Stream

CSR-Vector



WHAT ABOUT VERY LONG ROWS?



CSR-ADAPTIVE



(Once per matrix) On CPU:

Create Row
Blocks

CSR-ADAPTIVE



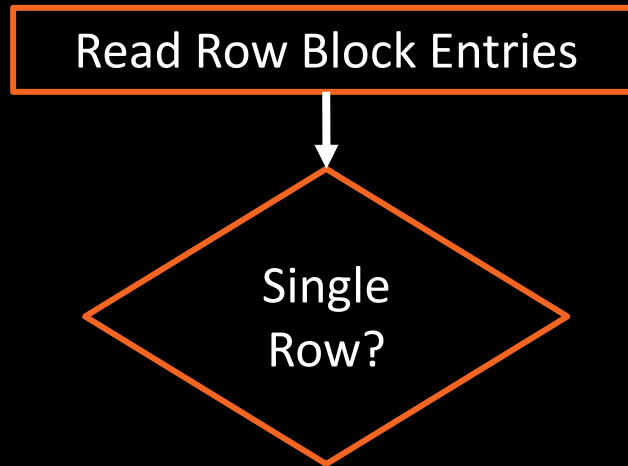
(Once per matrix) On CPU: Create Row Blocks

On GPU (each workgroup assigned one row block):

Read Row Block Entries

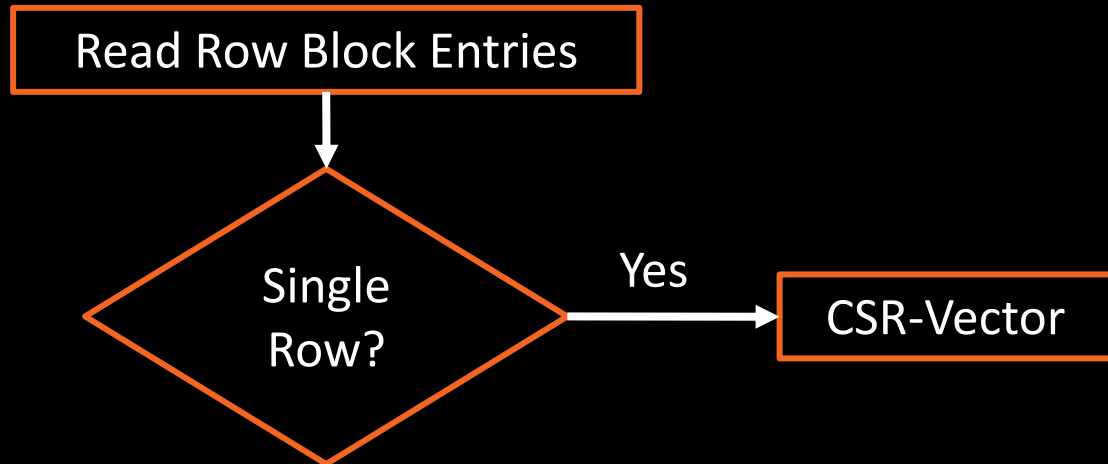
(Once per matrix) On CPU: Create Row
Blocks

On GPU (each workgroup assigned one row block):



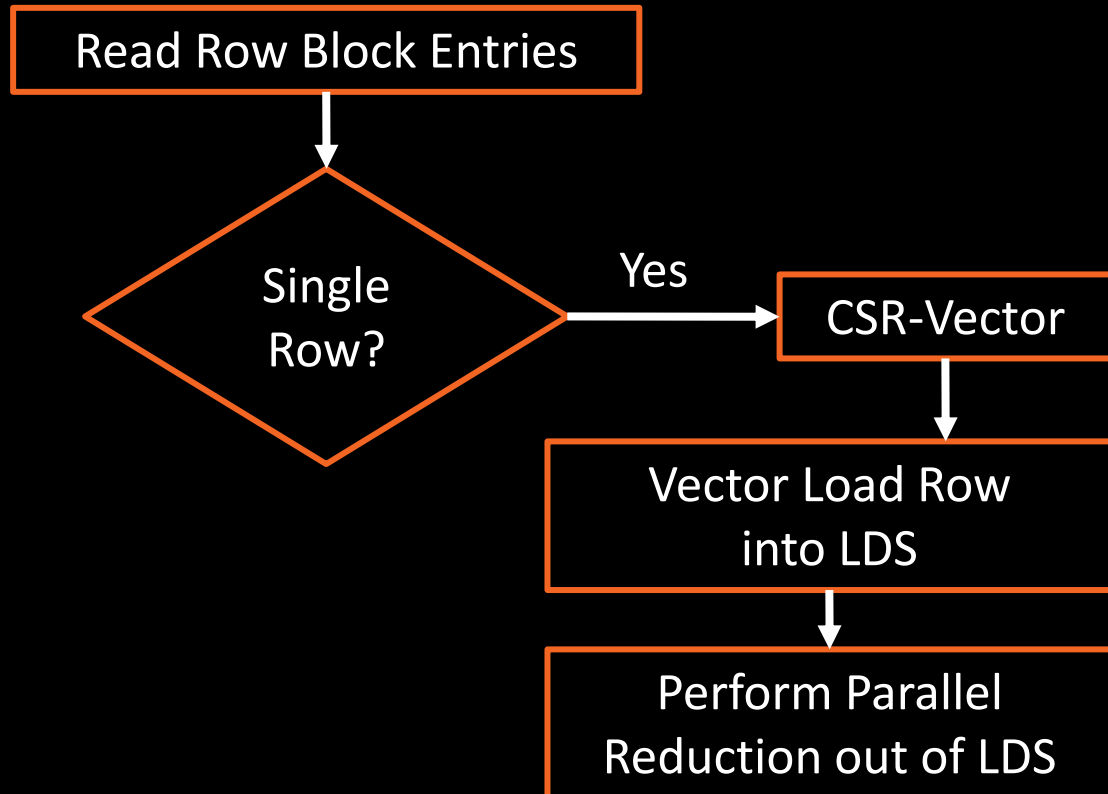
(Once per matrix) On CPU: Create Row Blocks

On GPU (each workgroup assigned one row block):



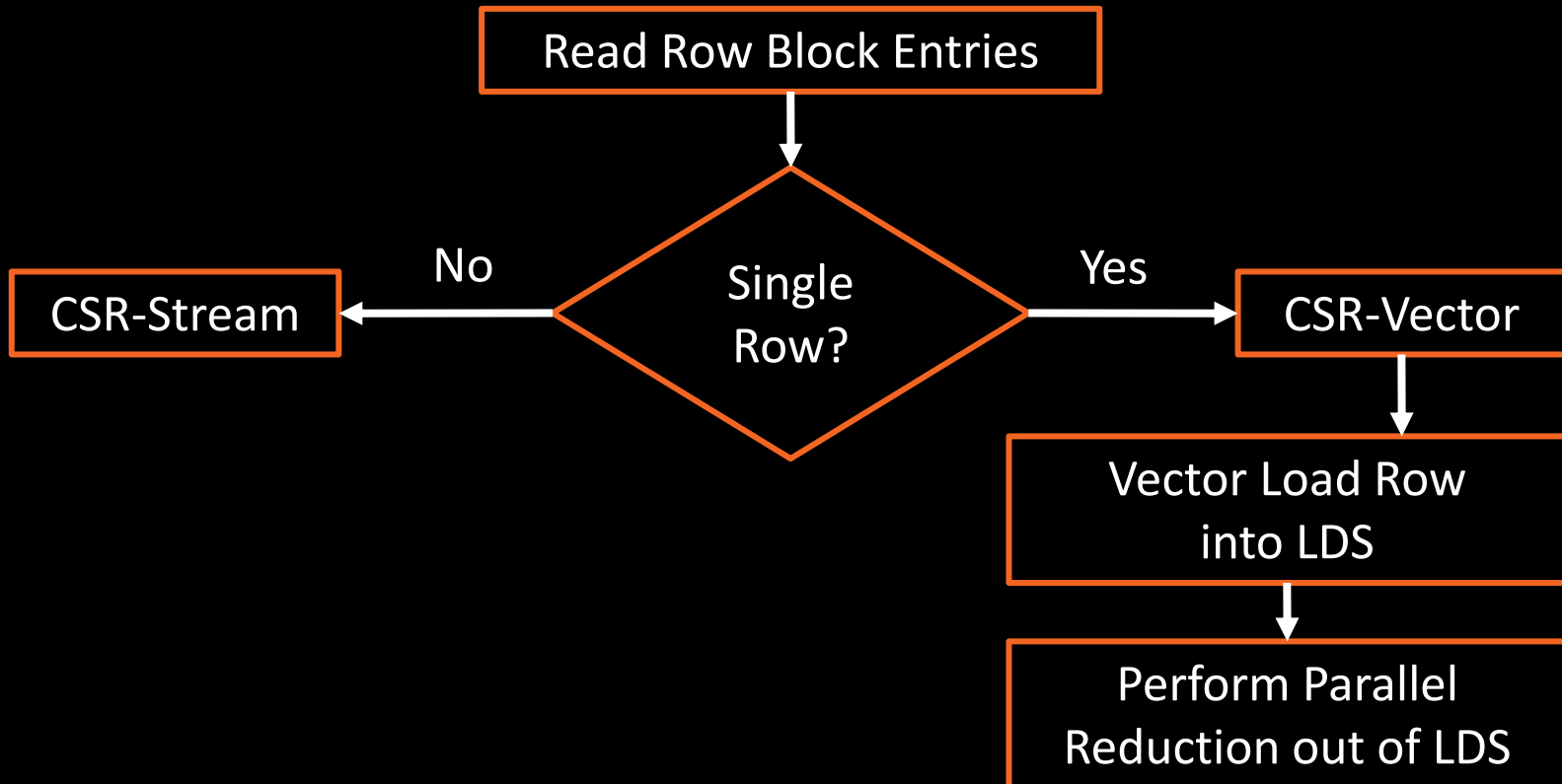
(Once per matrix) On CPU: Create Row Blocks

On GPU (each workgroup assigned one row block):



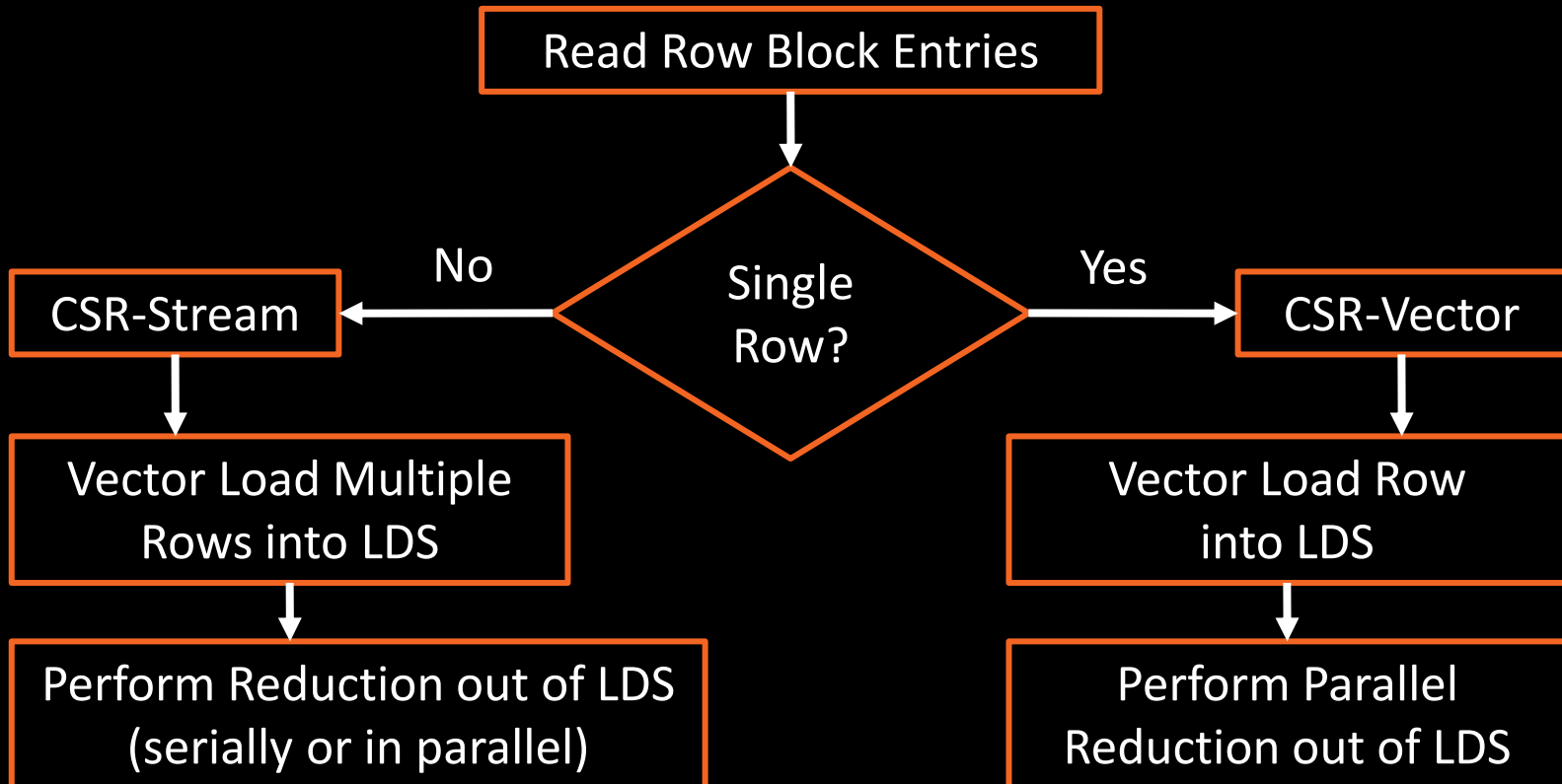
(Once per matrix) On CPU: **Create Row Blocks**

On GPU (each workgroup assigned one row block):



(Once per matrix) On CPU: **Create Row Blocks**

On GPU (each workgroup assigned one row block):





Experiments.

EXPERIMENTAL SETUP



- ▲ CPU: AMD A10-7850K (3.7 GHz)
 - 32 GB dual-channel DDR3-2133

- ▲ GPU: AMD FirePro™ W9100
 - 44 parallel compute units (CUs)
 - 930 MHz core clock rate (5.2 single-precision TFLOPs)
 - 16 GDDR5 channels at 1250 MHz (320 GB/s)

- ▲ CentOS 6.4 (kernel 2.6.32-358.23.2)
 - AMD FirePro Driver version 14.20 beta
 - AMD APP SDK v2.9

SPMV SETUP



- ▲ SHOC Benchmark Suite (Aug. 2013 version)
 - CSR-Vector (details in paper: CSR-Scalar, ELLPACK)

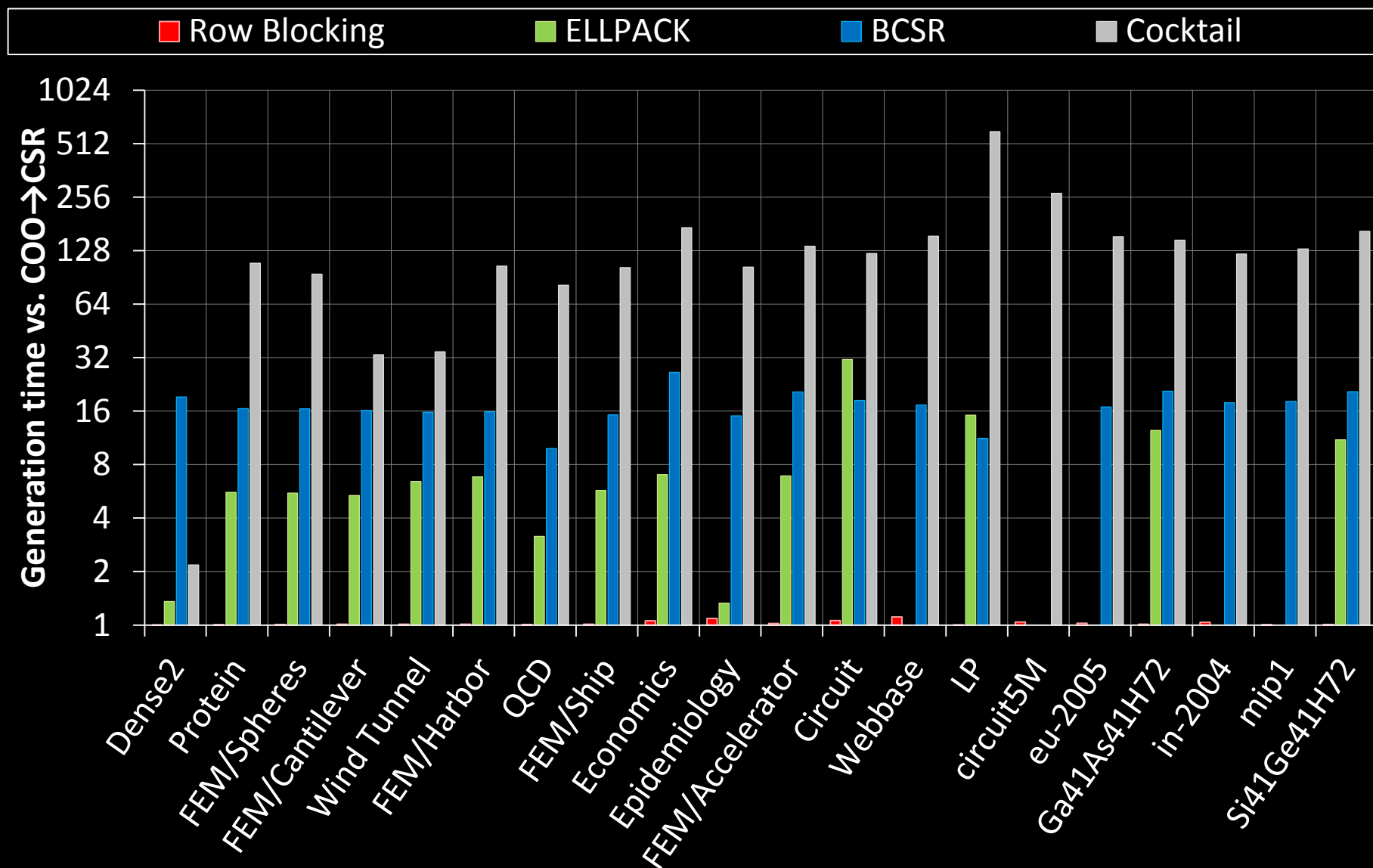
- ▲ ViennaCL v1.5.2
 - CSR-Scalar, COO, ELLPACK, ELLPACK+COO HYB
 - 4-value and 8-value padded versions of CSR-Scalar
 - Best of these for each matrix: “ViennaCL Best”

- ▲ clSpMV v0.1
 - CSR-Scalar, CSR-Vector, BCSR, COO, DIA, BDIA, ELLPACK, SELL, BELL, SBELL
 - Best of these for each matrix: “clSpMV Best”
 - Details in paper: clSpMV “Cocktail” format

- ▲ CSR-Adaptive

- ▲ 20 matrices from Univ. of Florida Sparse Matrix Collection

DATA STRUCTURE GENERATION TIMES



Note: Some matrices could not be held in ELLPACK or BCSR

PERFORMANCE IN SINGLE-PRECISION GFLOPS



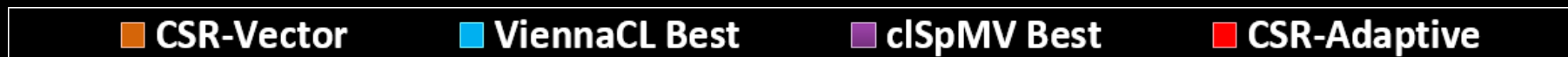
■ CSR-Vector

■ ViennaCL Best

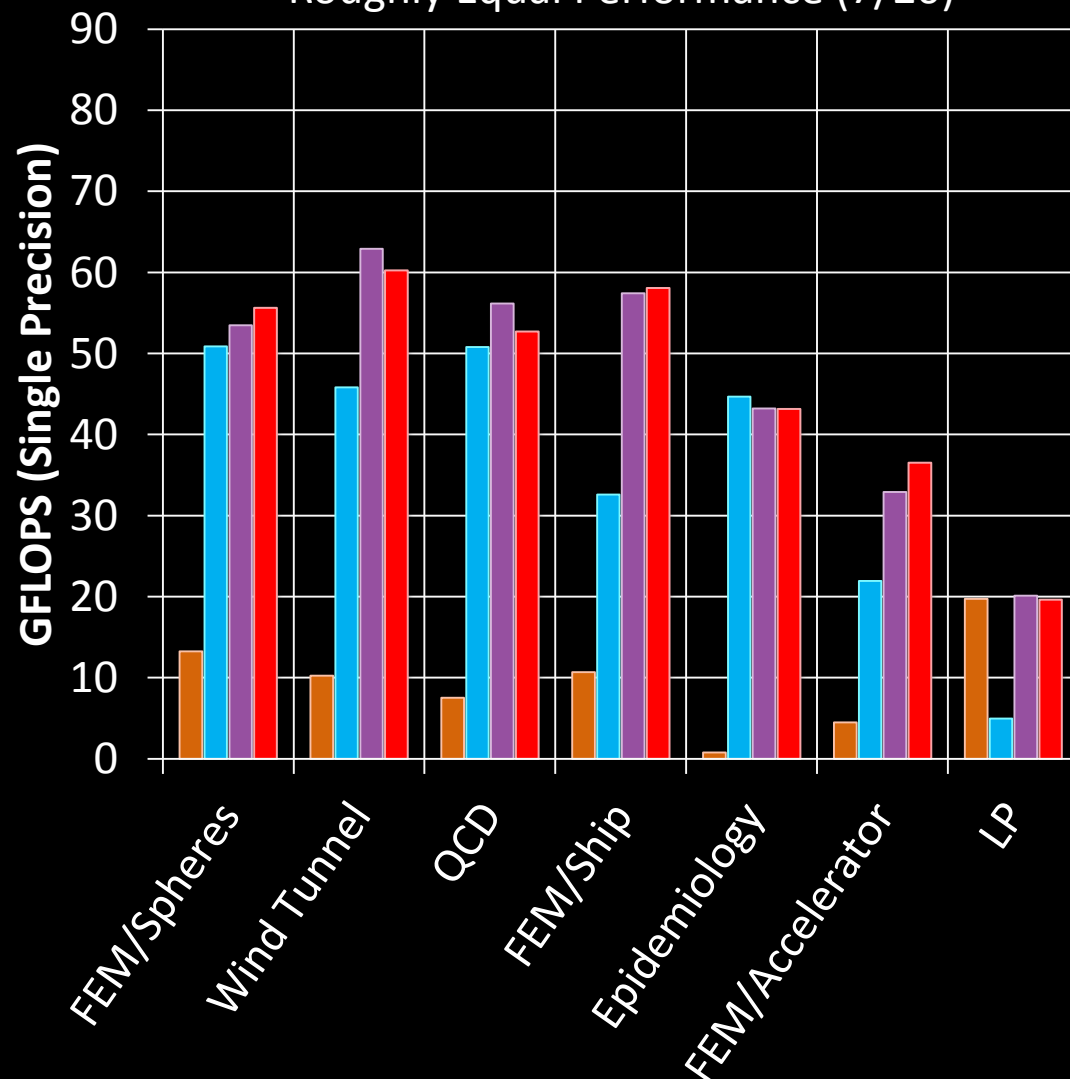
■ clSpMV Best

■ CSR-Adaptive

PERFORMANCE IN SINGLE-PRECISION GFLOPS



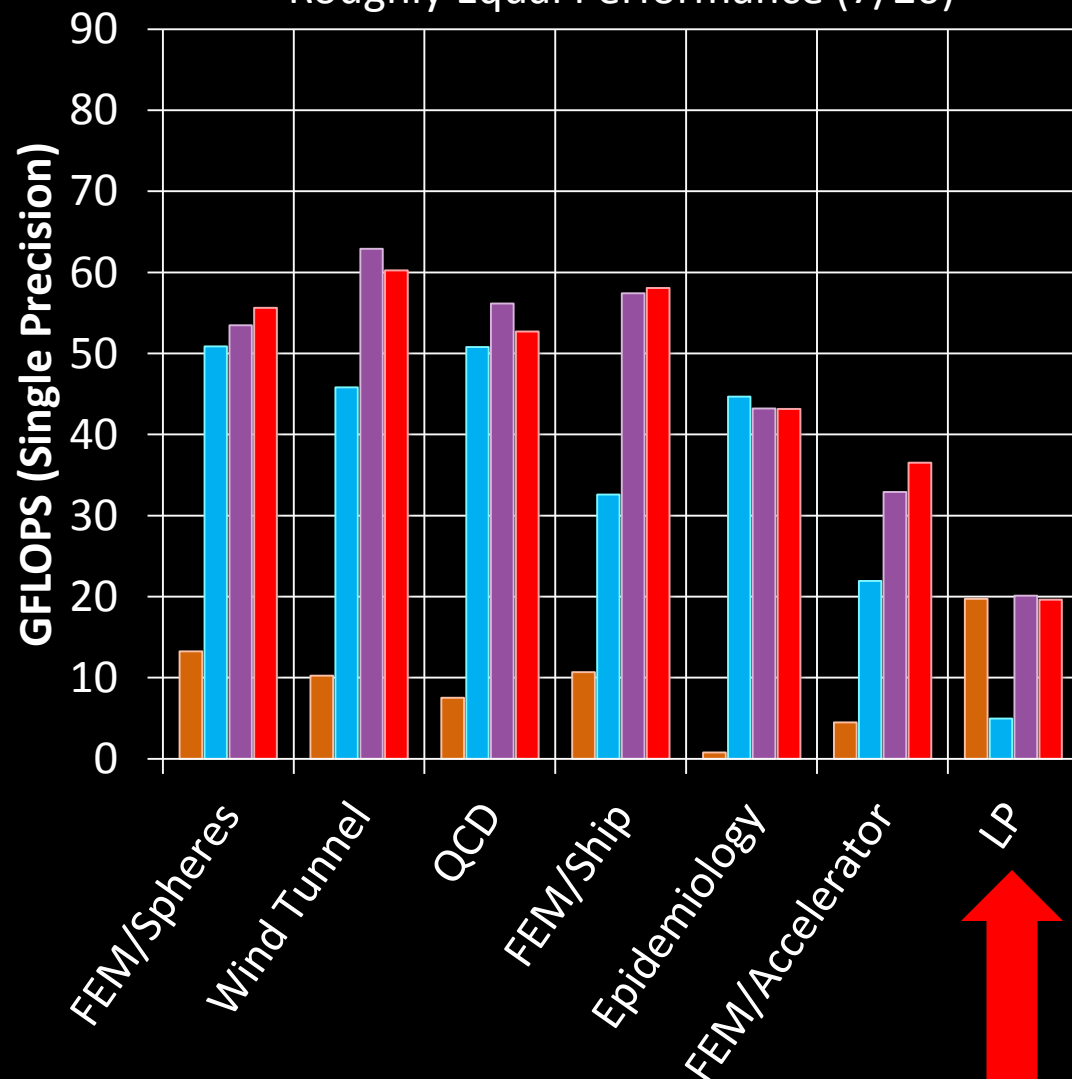
Roughly Equal Performance (7/20)



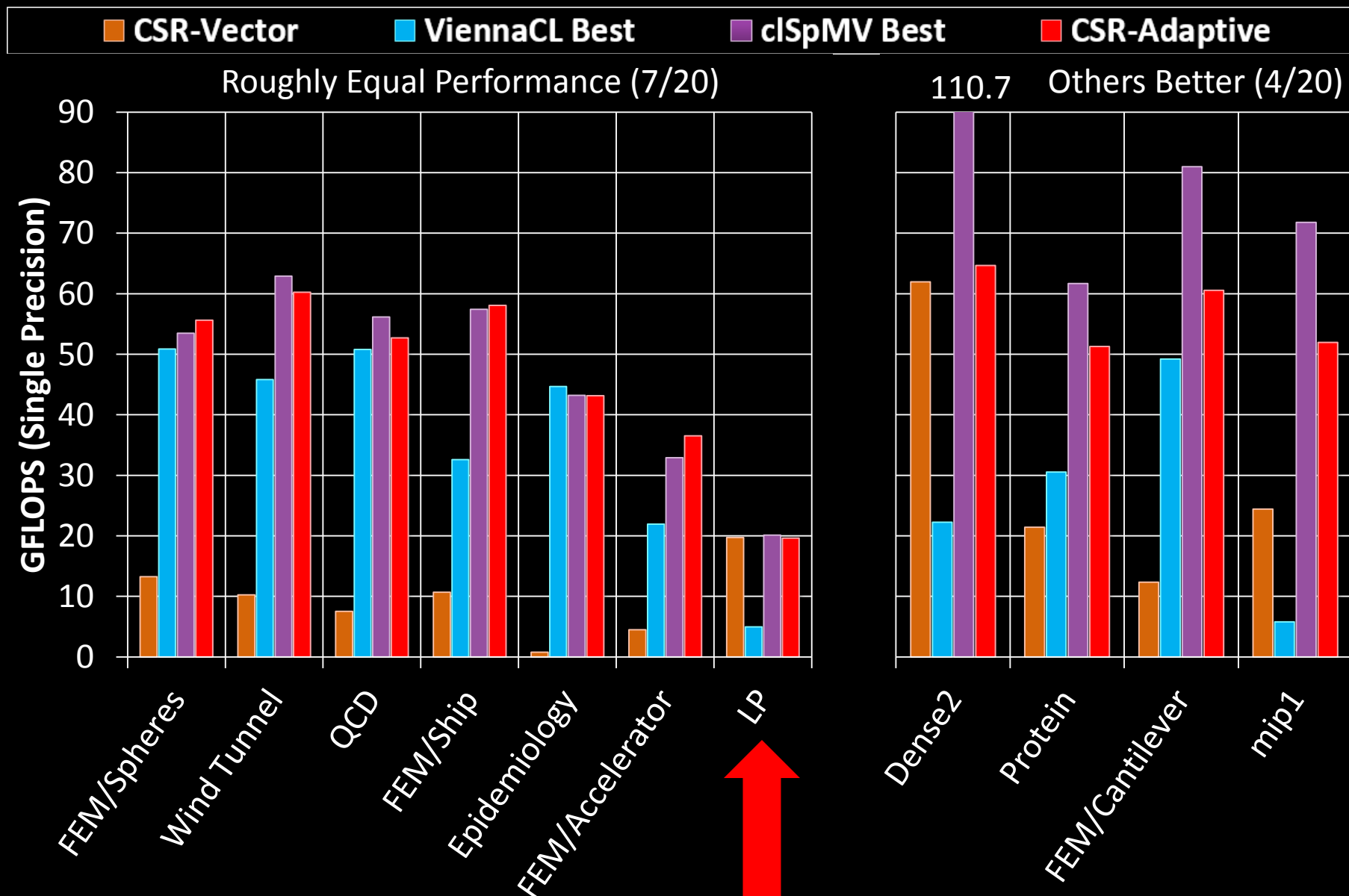
PERFORMANCE IN SINGLE-PRECISION GFLOPS



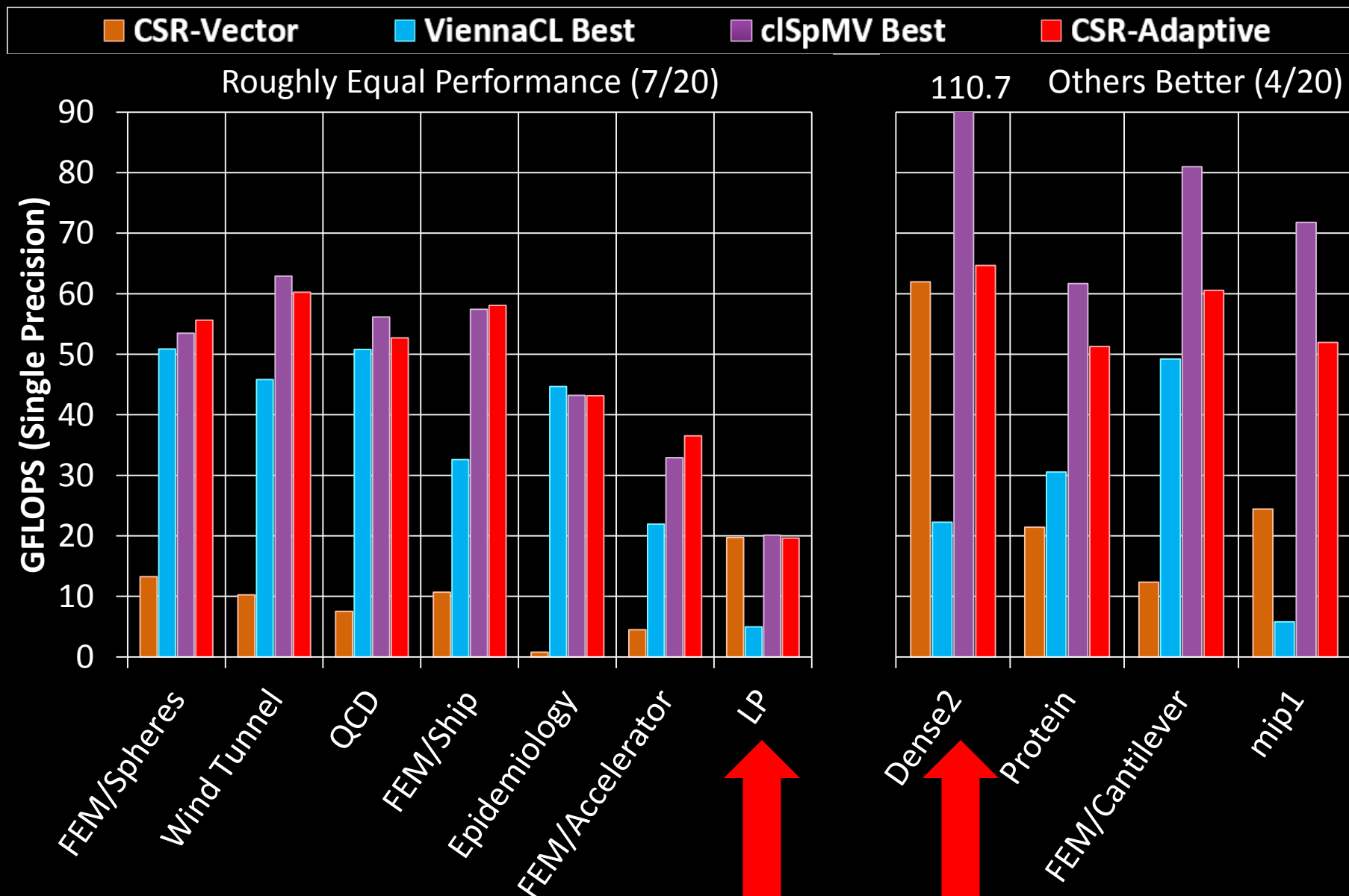
Roughly Equal Performance (7/20)



PERFORMANCE IN SINGLE-PRECISION GFLOPS



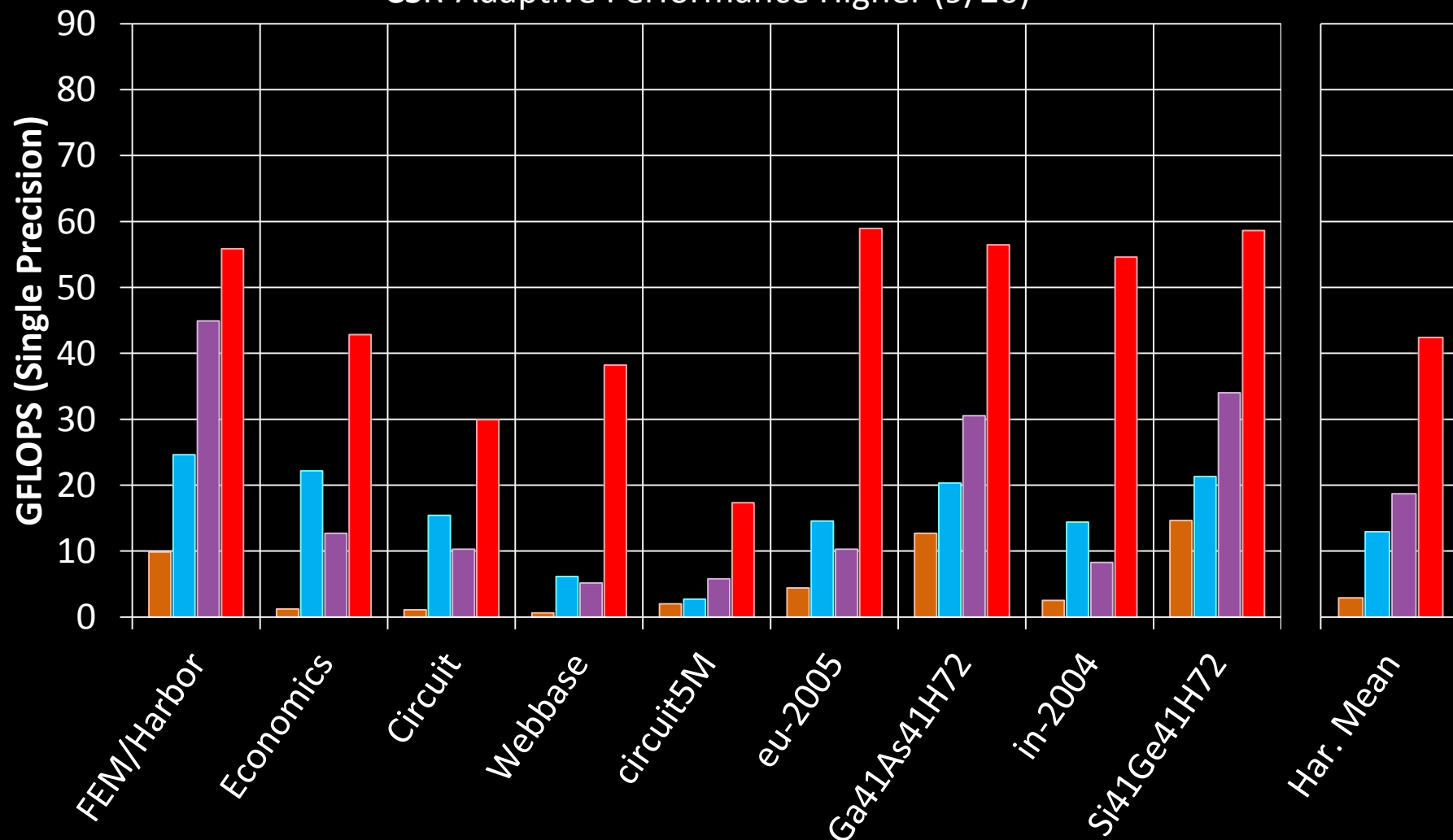
PERFORMANCE IN SINGLE-PRECISION GFLOPS



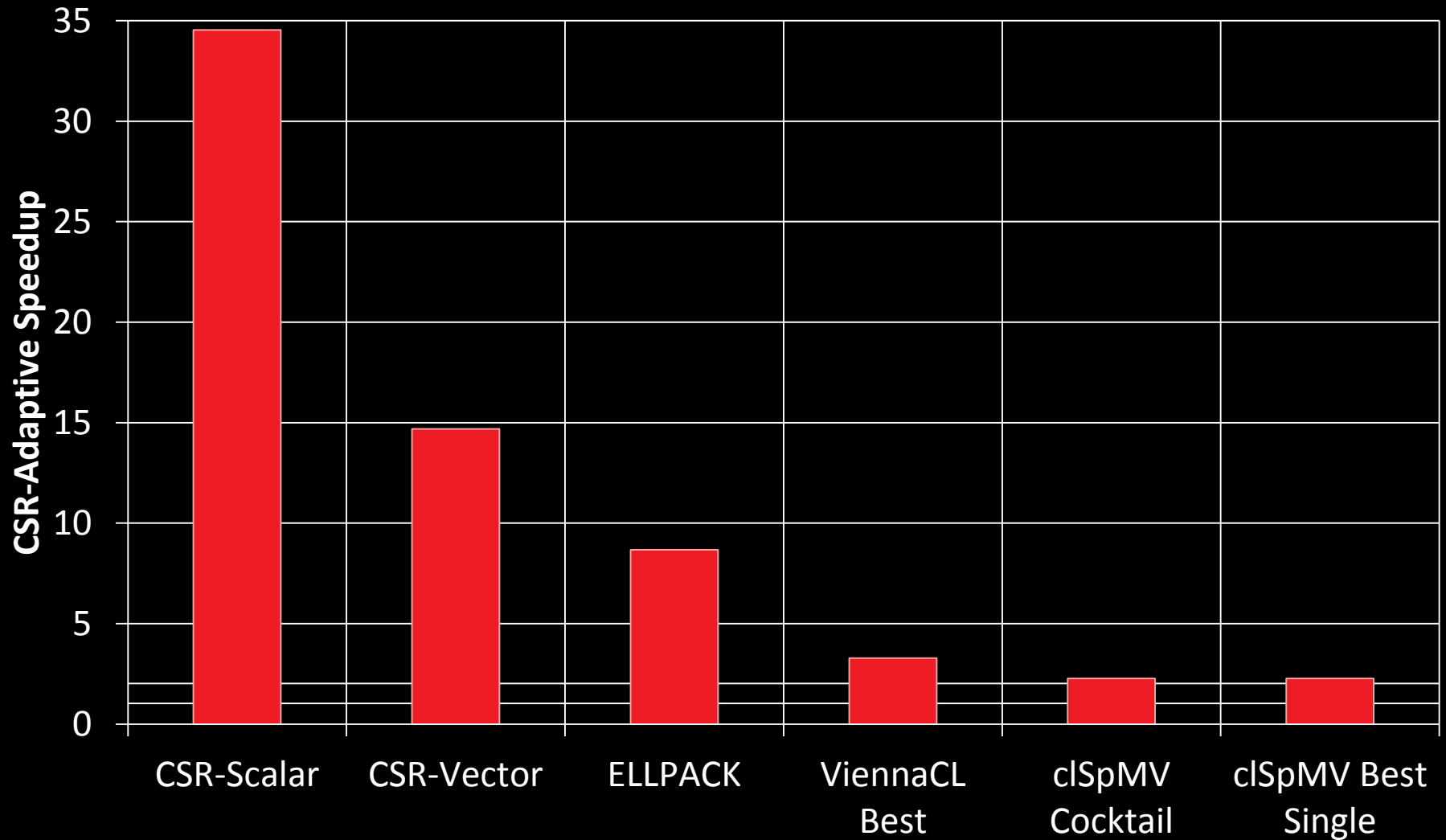
PERFORMANCE IN SINGLE-PRECISION GFLOPS



CSR-Adaptive Performance Higher (9/20)



PERFORMANCE BENEFIT OF CSR-ADAPTIVE



CONCLUSION



- ▲ CSR-Adaptive up to 14.7x faster than previous GPU-based SpMV algorithms
- ▲ Requires little work beyond generating the traditional CSR data structure
 - Less than 0.1% extra storage compared to CSR
- ▲ Algorithm is not complex: easy to integrate into libraries and existing SW

CONCLUSION



- ▲ CSR-Adaptive up to 14.7x faster than previous GPU-based SpMV algorithms
- ▲ Requires little work beyond generating the traditional CSR data structure
 - Less than 0.1% extra storage compared to CSR
- ▲ Algorithm is not complex: easy to integrate into libraries and existing SW

AMD Sample Code
Available Soon – Ask Us!

CONCLUSION



- ▲ CSR-Adaptive up to 14.7x faster than previous GPU-based SpMV algorithms
- ▲ Requires little work beyond generating the traditional CSR data structure
 - Less than 0.1% extra storage compared to CSR
- ▲ Algorithm is not complex: easy to integrate into libraries and existing SW

AMD Sample Code
Available Soon – Ask Us!

Independently
Implemented Version in
ViennaCL 1.6.1

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

© 2014 Advanced Micro Devices, Inc. All rights reserved. AMD, the AMD Arrow logo, FirePro and combinations thereof are trademarks of Advanced Micro Devices, Inc. in the United States and/or other jurisdictions. Other names are for informational purposes only and may be trademarks of their respective owners.



Backup Slides

AMD GPU MICROARCHITECTURE



AMD GPU MICROARCHITECTURE



Many highly-threaded vector processors

Up to 112,640 work-items in flight

AMD GPU MICROARCHITECTURE

