Berkeley Winter School

Advanced Algorithmic Techniques for GPUs

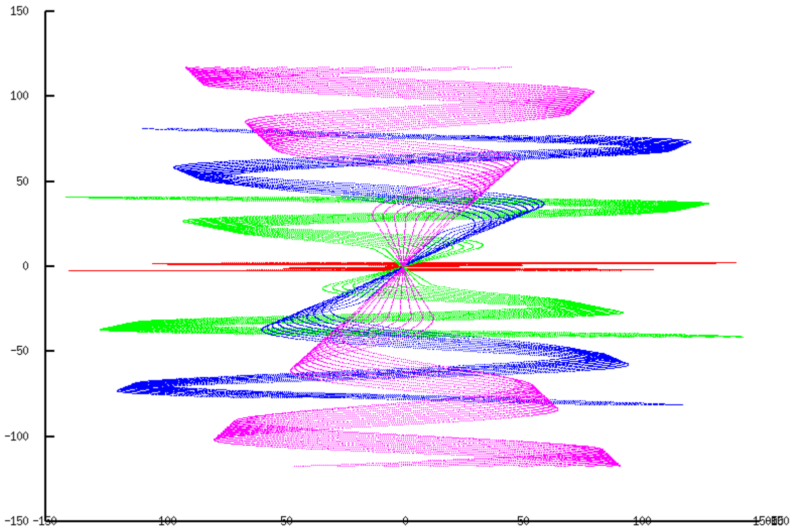
Lecture 7: Input Compaction

Objective

- To learn the key techniques for compacting input data for reduced consumption of memory bandwidth
  - Via better utilization of on-chip memory
  - As well as fewer bytes transferred to on-chip memory
- To understand the tradeoffs between input compaction and input binning/regularization

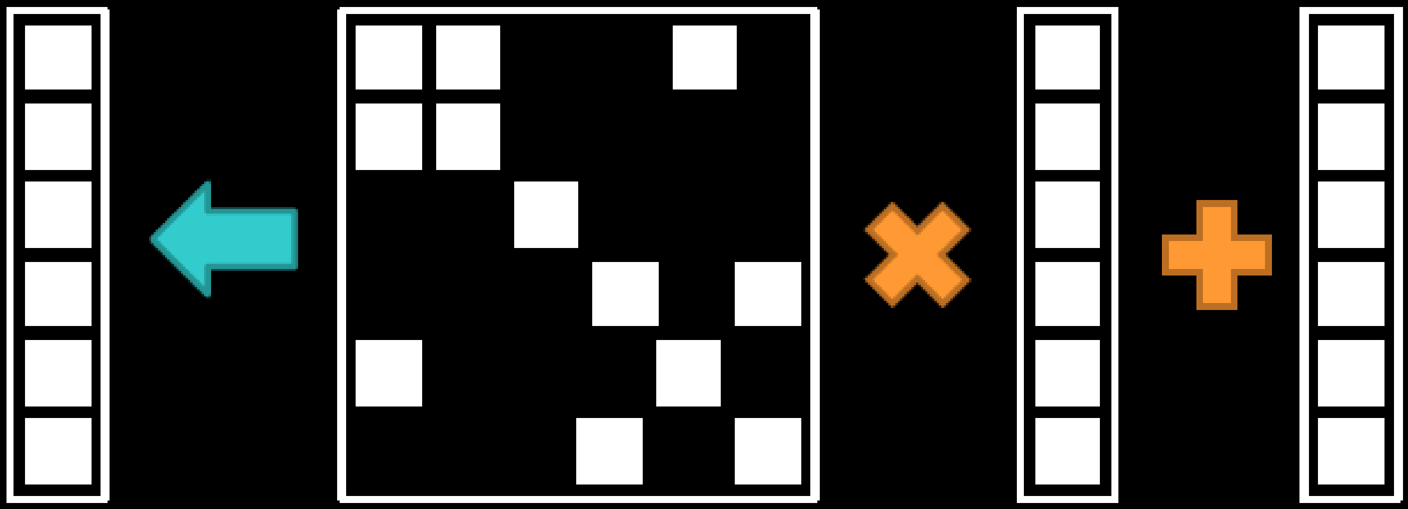
Sparse Data
Motivation for Compaction

- Many real-world inputs are sparse/non-uniform
- Signal samples, mesh models, transportation networks, communication networks, etc.



Sparse matrix-vector multiplication

- Compute $y \leftarrow Ax + y$
  - where A is sparse and x, y are dense



- Unlike dense methods, SpMV is generally
  - unstructured / irregular
  - entirely bound by memory bandwidth

Parallelizing CSR SpMV

Compressed Sparse Row

- **Straightforward approach**
  - **one thread per matrix row**

Thread 0	3	0	1	0
Thread 1	0	0	0	0
Thread 2	0	2	4	1
Thread 3	1	0	0	1

CSR SpMV Kernel (CUDA)

```
int row = blockDim.x * blockIdx.x + threadIdx.x;
if ( row < num_rows ){
    float dot = 0;
    int row_start = ptr[row];
    int row_end   = ptr[row + 1];
    for (int jj = row_start; jj < row_end; jj++)
        dot += data[jj] * x[indices[jj]];
    y[row] += dot;
}
```

			Row 0	Row 2	Row 3	
Nonzero values	data[7]	= {	3, 1,	2, 4, 1,	1, 1	};
Column indices	indices[7]	= {	0, 2,	1, 2, 3,	0, 3	};

Row pointers `ptr[5] = { 0, 2, 2, 5, 7 };`

Problems with simple CSR kernel

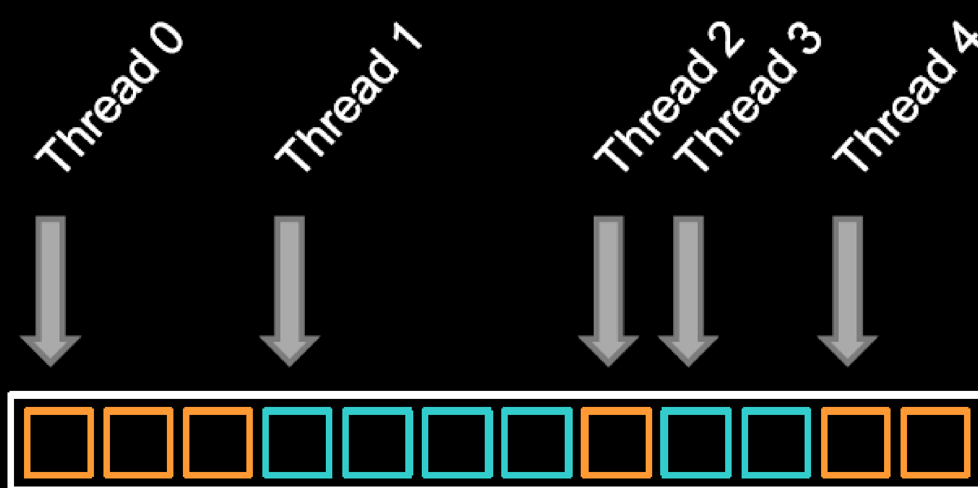
- **Execution divergence**
  - varying row lengths
- **Memory divergence**
  - minimal coalescing

Thread 0	3	0	1	0
Thread 1	0	0	0	0
Thread 2	0	2	4	1
Thread 3	1	0	0	1

```
                                     #0 #1 #0 #1 #0 #2 #1   Iteration
Nonzero values   data[7]     = {  3, 1, 2, 4, 1, 1, 1 };
Column indices   indices[7]  = {  0, 2, 1, 2, 3, 0, 3 };
Row pointers     ptr[5]      = {  0, 2, 2, 5, 7 };
```

Problems with simple CSR kernel

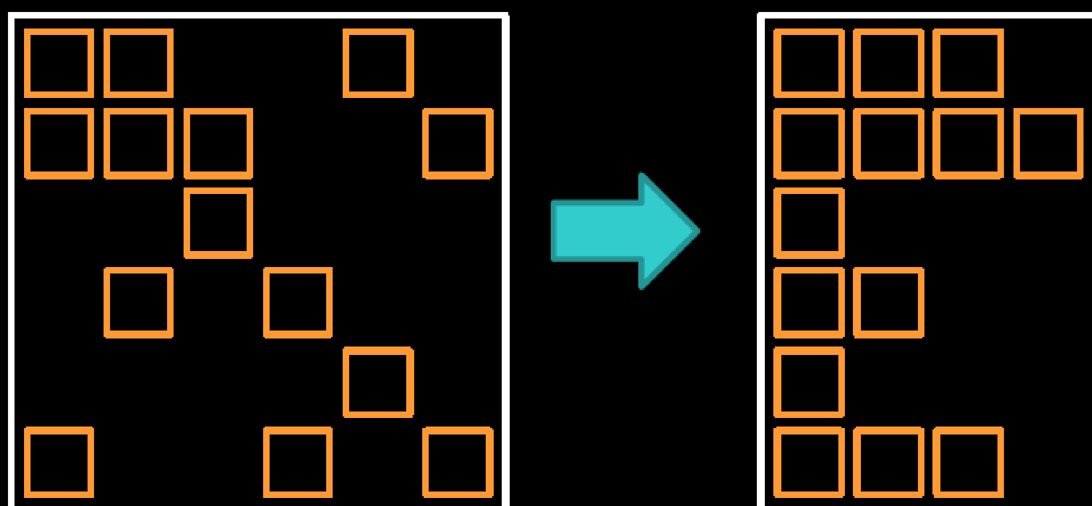
- Memory divergence
  - minimal coalescing



Regularizing SpMV with ELL format

- Storage for K nonzeros per row
  - pad rows with fewer than K nonzeros
  - inefficient when row length varies

ELLPACK



Regularizing SpMV with ELL format

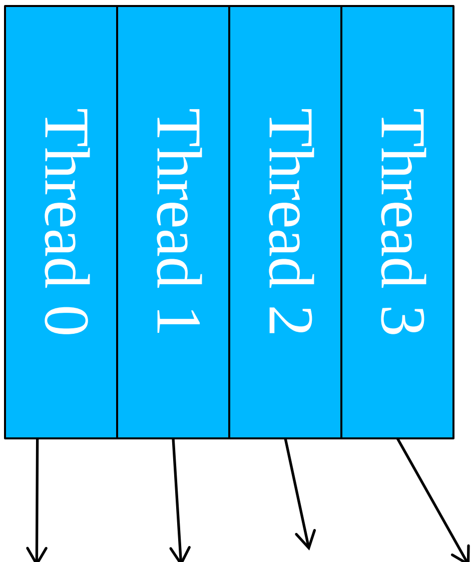
- Quantize each row to a fix length K

	Values			Columns		
Thread 0	3	1	*	0	2	*
Thread 1	*	*	*	*	*	*
Thread 2	2	4	1	1	2	3
Thread 3	1	1	*	0	3	*

- Layout in column-major order
  - yields full coalescing

Memory Coalescing with ELL

	Values			Columns		
Thread 0	3	1	*	0	2	*
Thread 1	*	*	*	*	*	*
Thread 2	2	4	1	1	2	3
Thread 3	1	1	*	0	3	*



data	3	*	2	1	1	*	4	1	*	*	1
index	0	*	1	1	2	*	2	3	*	*	3

Exposing maximal parallelism

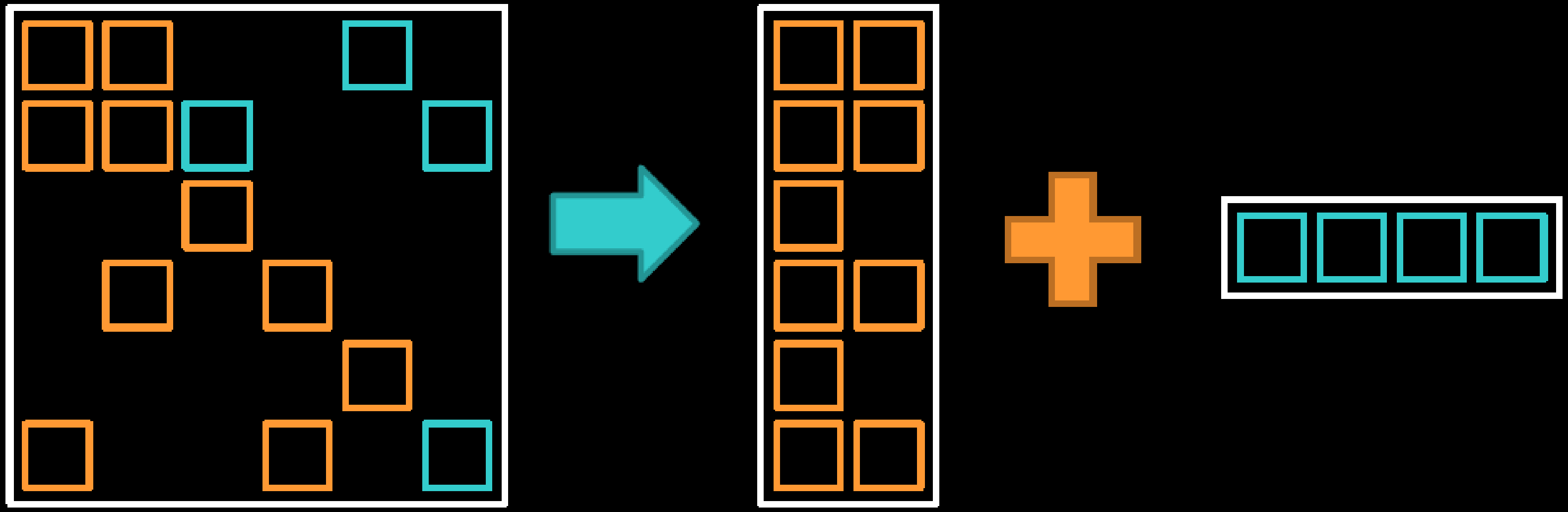
- Use COO (Coordinate) format
  - list row, column, and value for every non-zero entry

Nonzero values	`data[7] = { 3, 1, 2, 4, 1, 1, 1 };`
Column indices	`cols[7] = { 0, 2, 1, 2, 3, 0, 3 };`
Row indices	`rows[7] = { 0, 0, 1, 1, 1, 2, 2 };`

- Assign one thread to each non-zero entry
  - each thread computes an `A[i,j]*x[j]` product
  - sum products with segmented reduction algorithm
  - largely insensitive to row length distribution

Hybrid Format

- ELL handles *typical* entries
- COO handles *exceptional* entries
  - Implemented with segmented reduction



Any More Ideas?

- JDS format
  - Sort rows according to their number of non-zero elements

- Can use Hybrid with JDS and and launch multiple kernels

Sparse formats for different matrices

(DIA) Diagonal

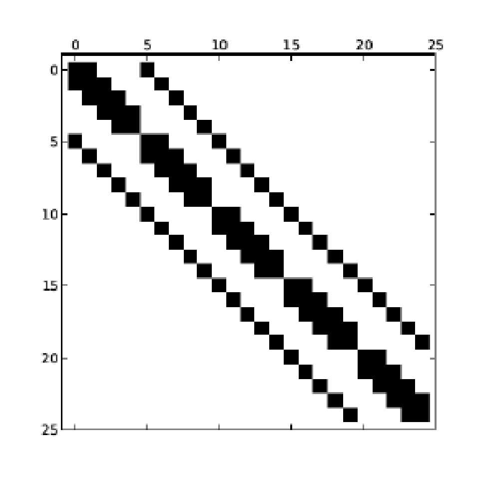
(ELL) ELLPACK

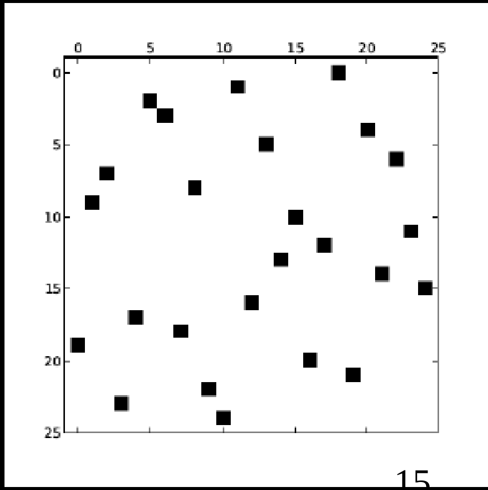
(CSR) Compressed Row

(HYB) Hybrid

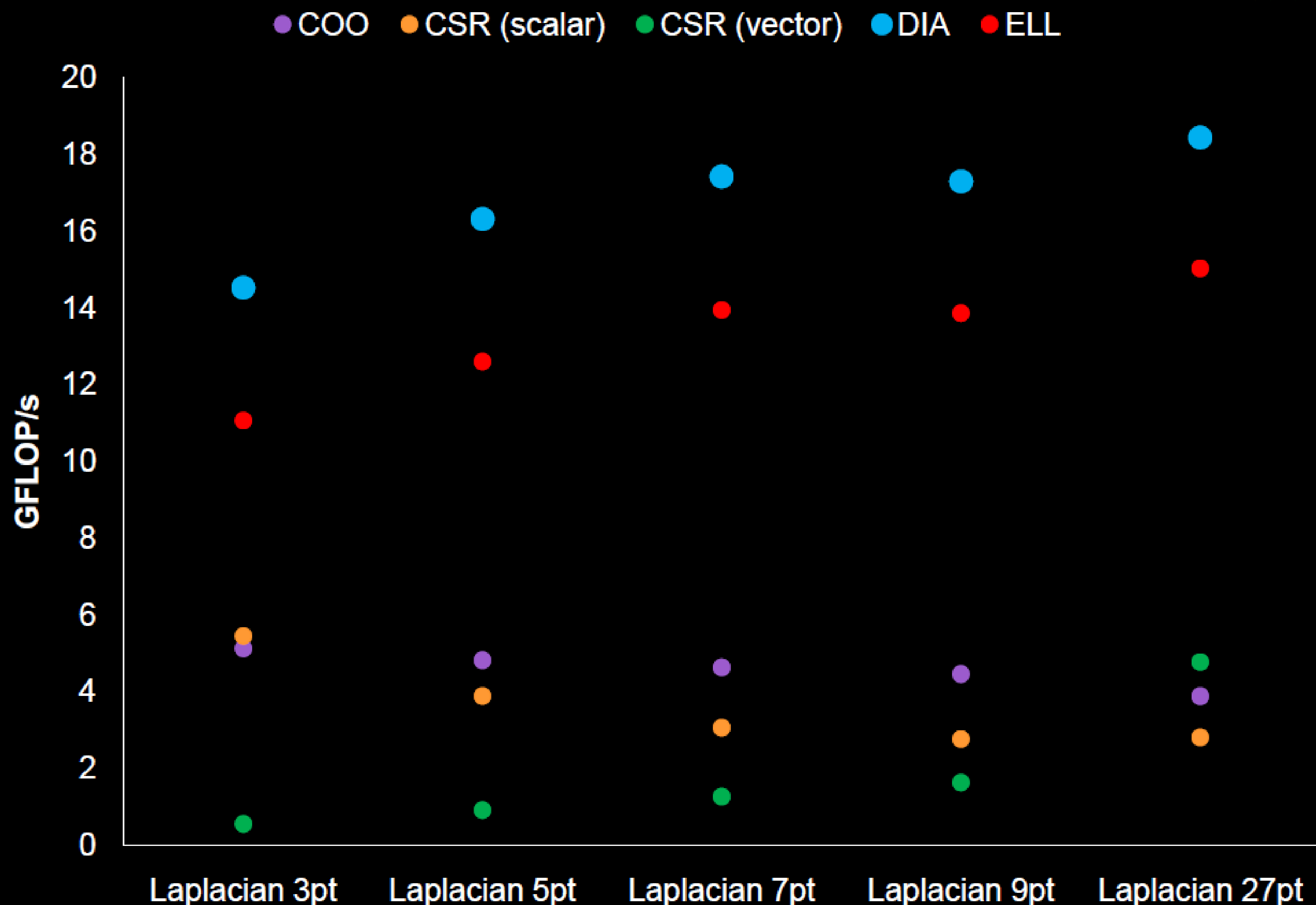
(COO) Coordinate

Structured ⟷ Unstructured

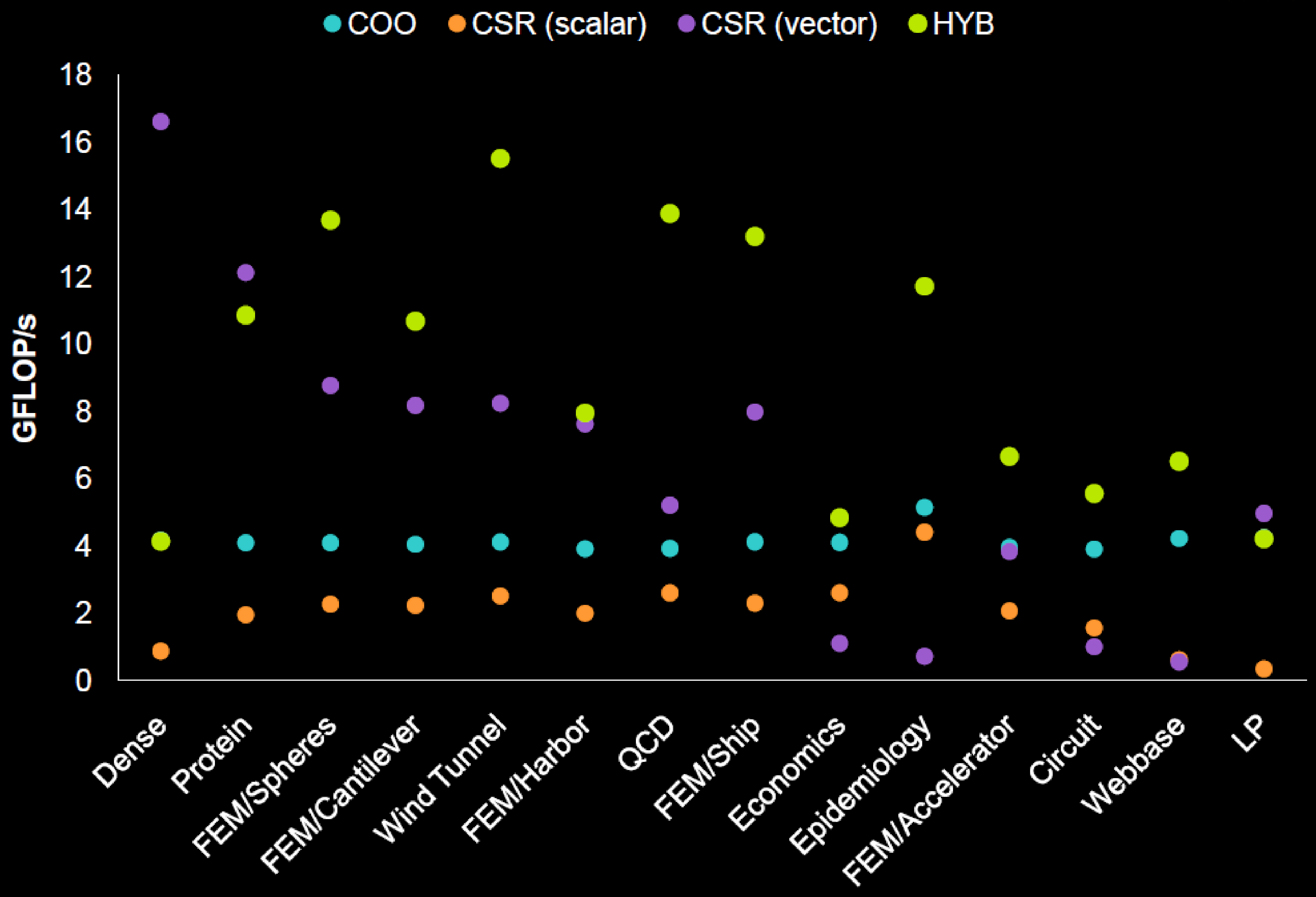




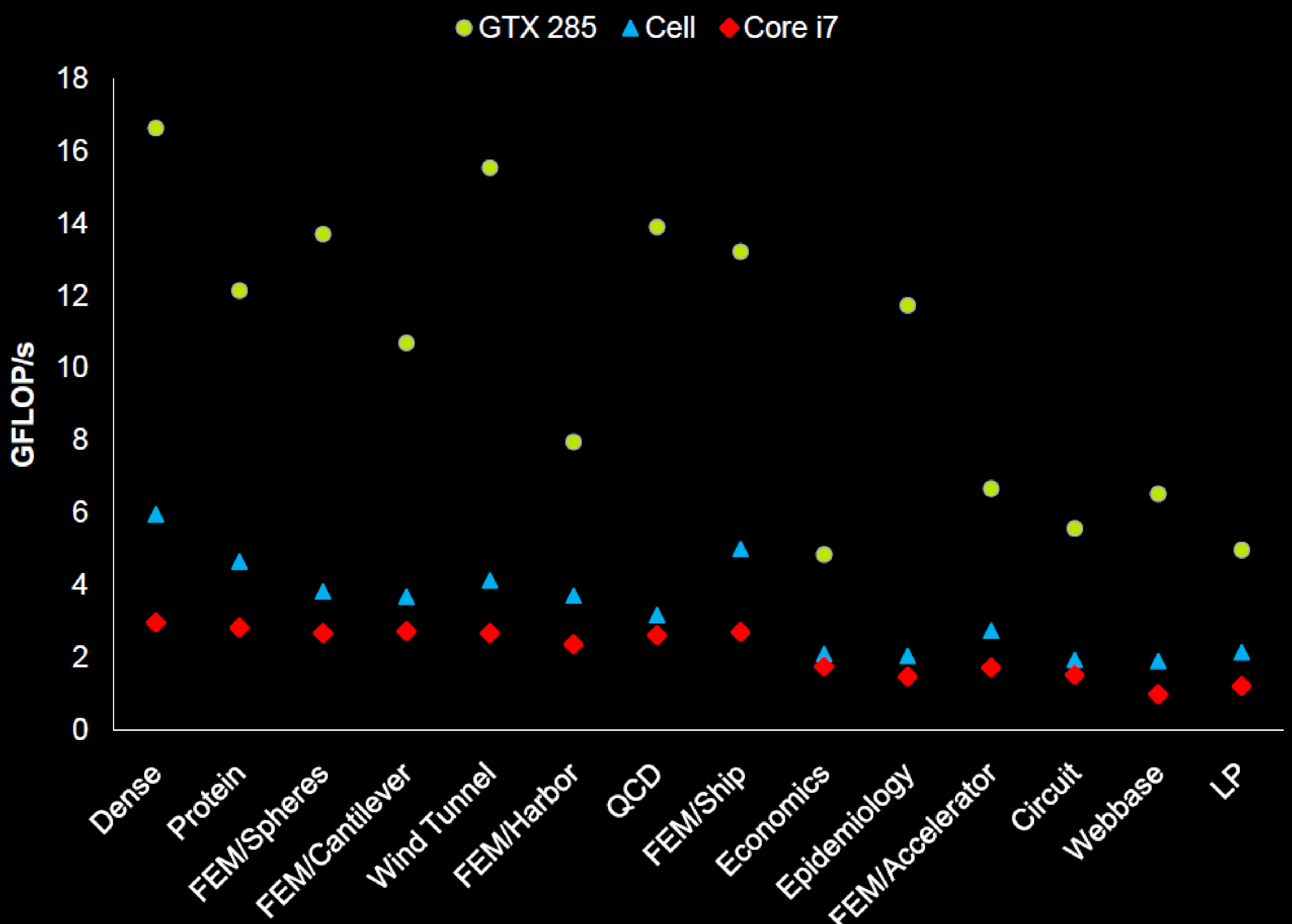
Structured Matrices



Unstructured Matrices

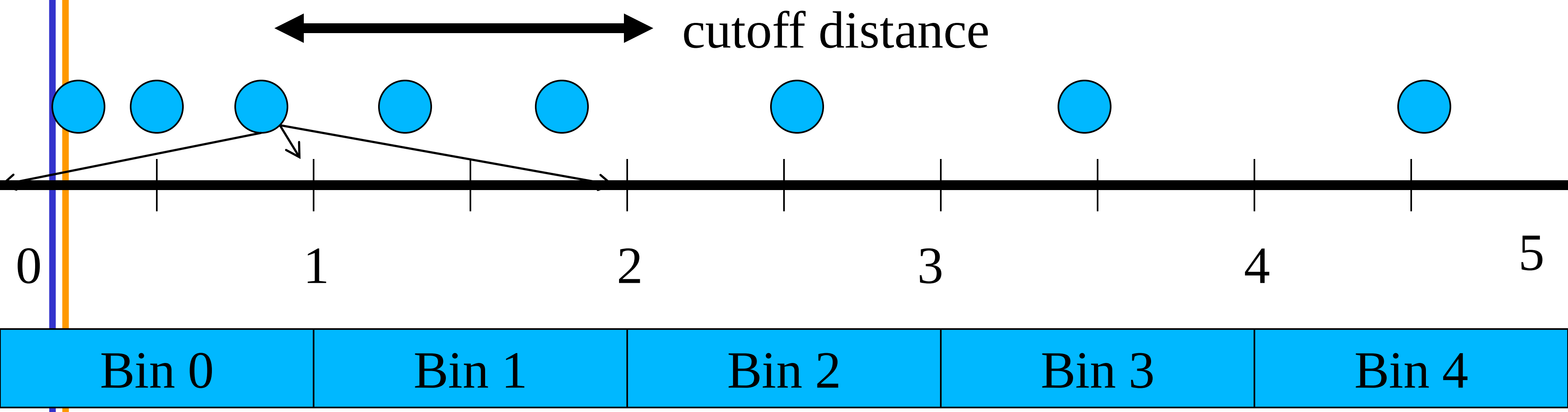


Performance Comparison

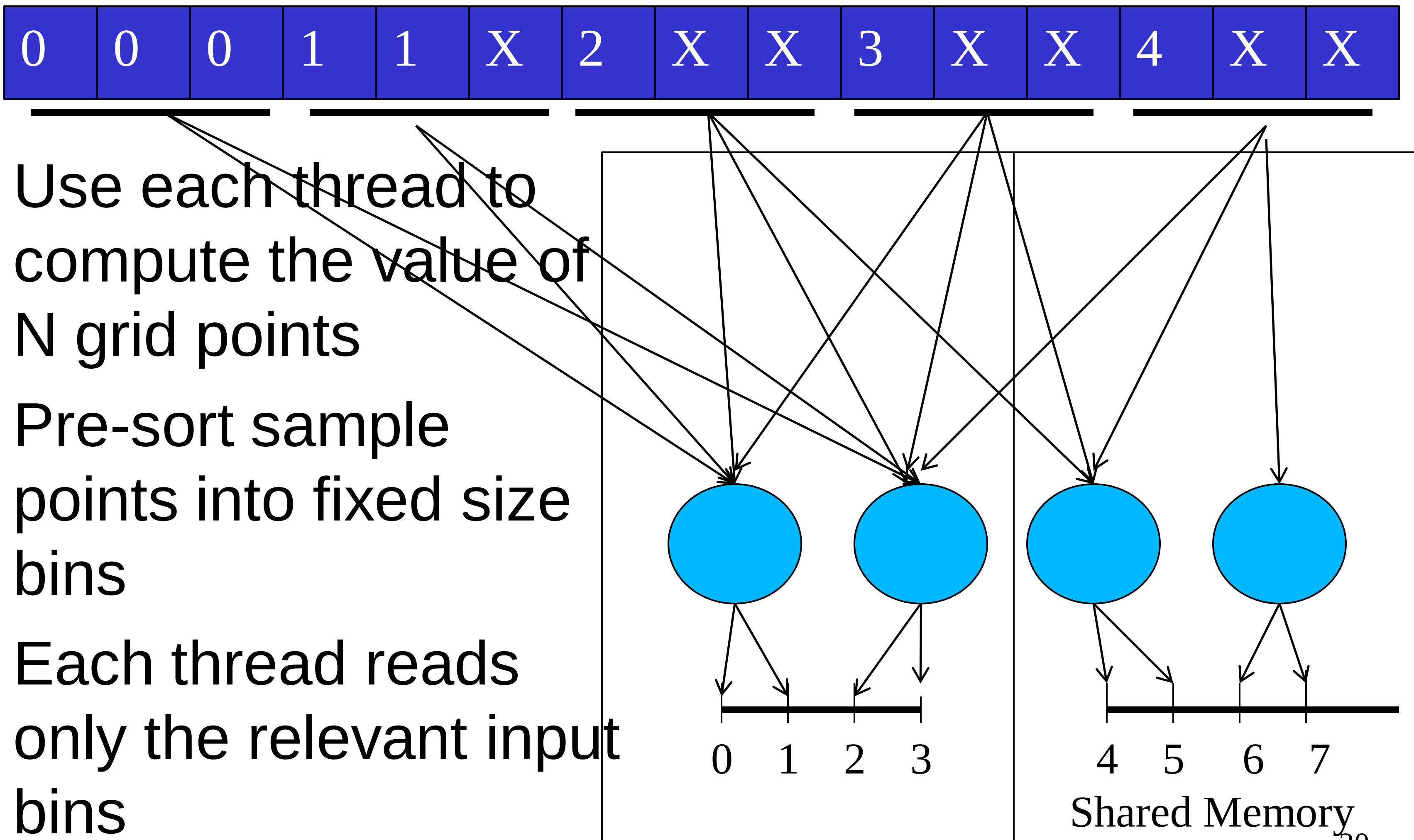


Binning of Sample Points

- For simplicity, we will use 1D gridding examples
- Each sample point has
  - S.x (will be represented with Bin#)
  - S.value (will be omitted unless necessary)

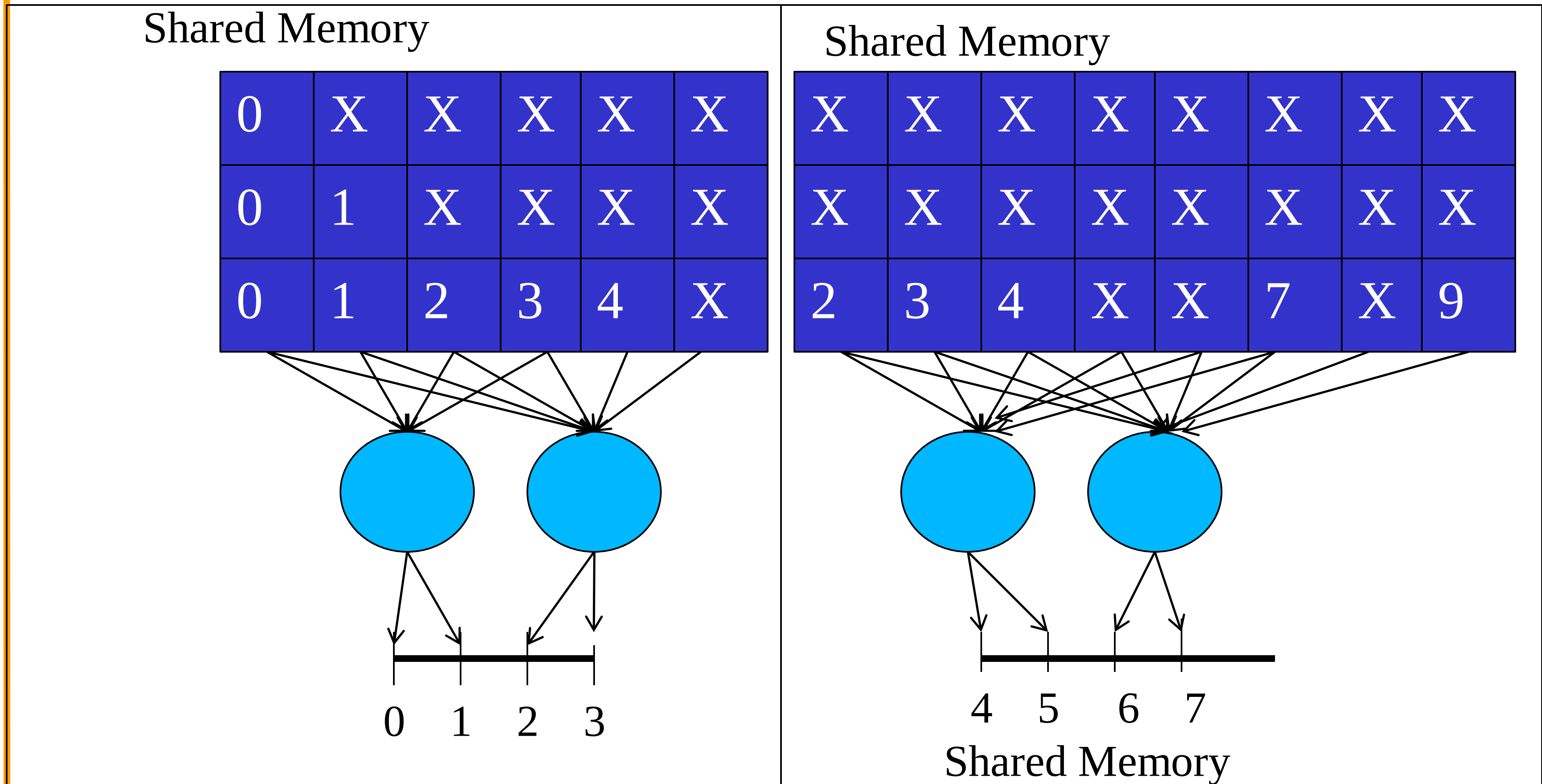


A Binned Gather Parallelization



- Use each thread to compute the value of N grid points
- Pre-sort sample points into fixed size bins
- Each thread reads only the relevant input bins

A Tiled Gather Implementation



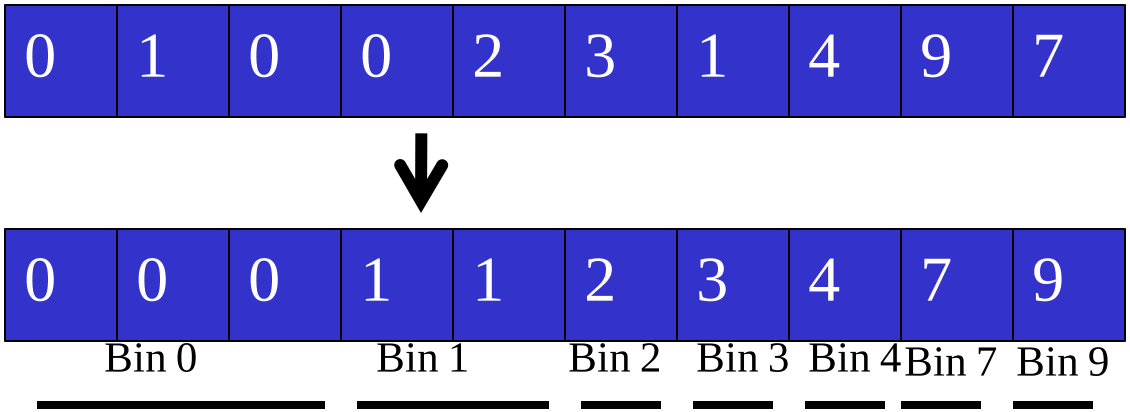
More on Tiled Gather

- Threads cooperate to load all the relevant bins from Global Memory to Shared Memory
- Each thread accesses relevant bins from Shared Memory
- Uniform binning for Non-uniform distribution
  - Large memory overhead for dummy cells
  - Reduced benefit of tiling
  - Many threads spend much time on dummy sample points

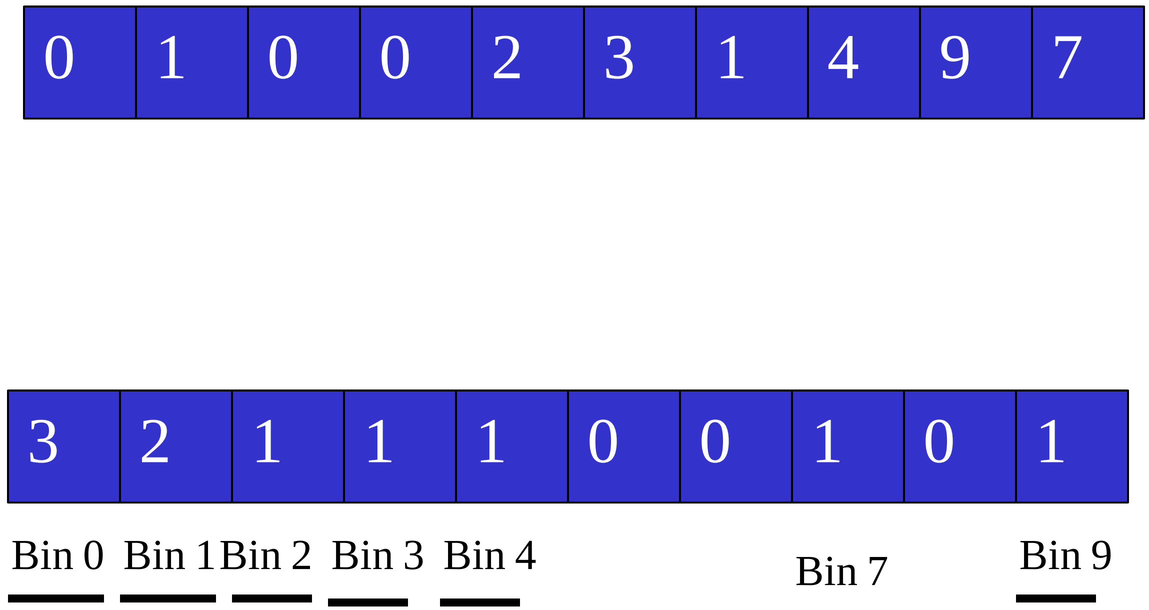
Compact Binning for Gather Parallelization

- Avoid pre-allocated fixed capacity bins (multi-dimensional array)

- Sort samples into bins of varying sizes in input array instead
  - Bins 5, 6, 8 are implicit, zero-sample



GPU Binning - Use Scatter to Generate Bin Capacities

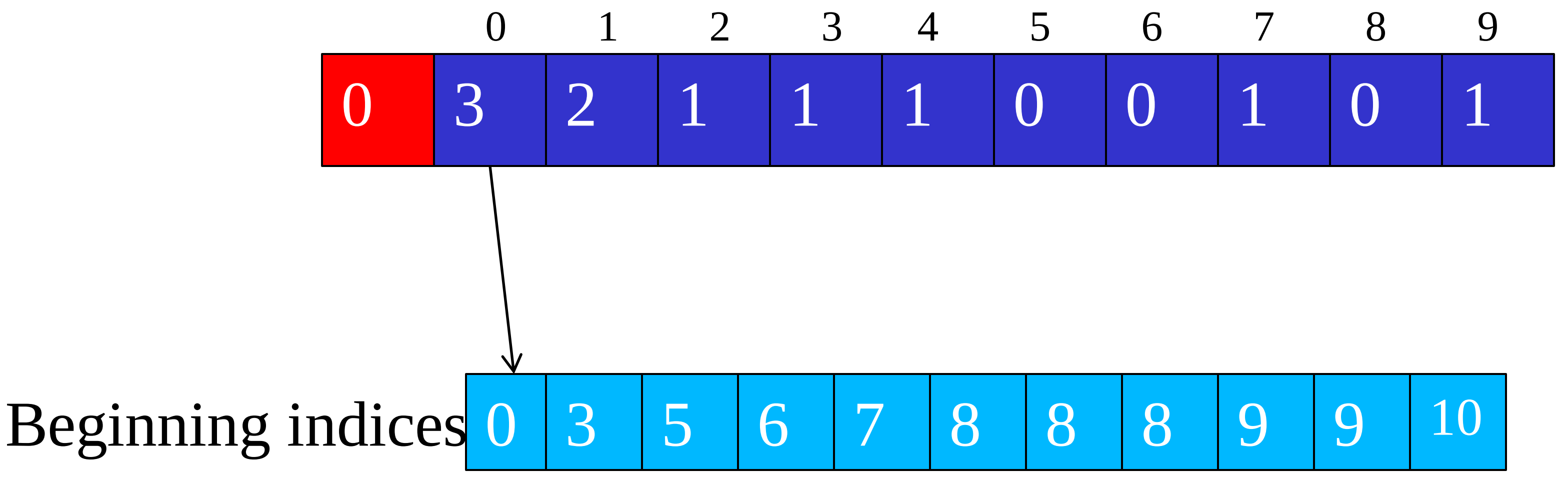


Capacity of Each bin

Need to use atomic operations for counting the capacity

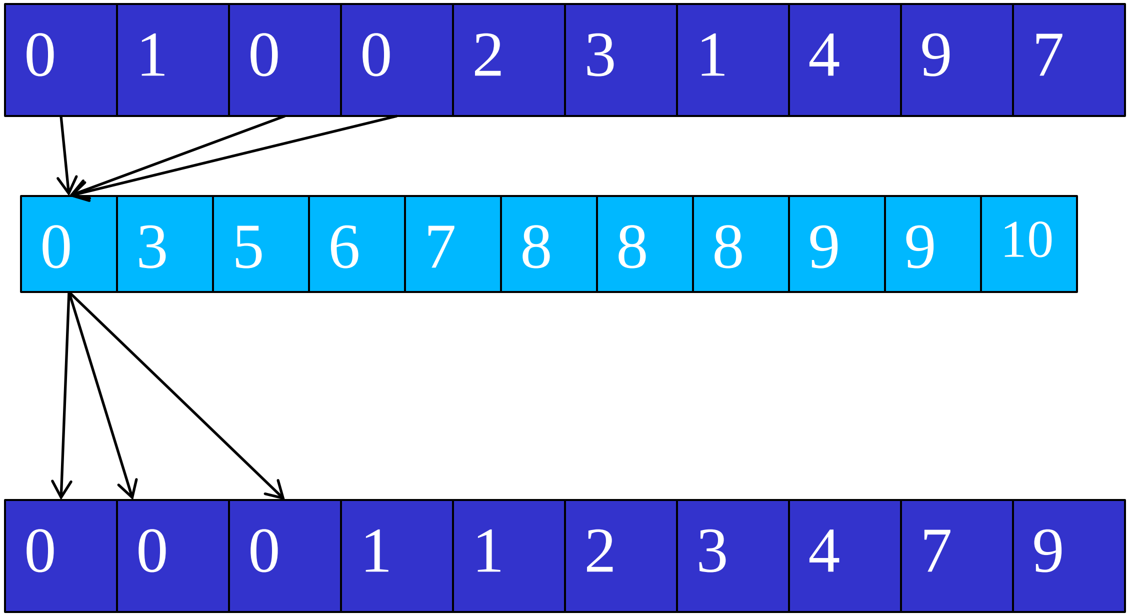
Determine Start and End of Bins

- Use parallel scan operations on the bin capacity array to generate an array of starting points of all bins (CUDPP)

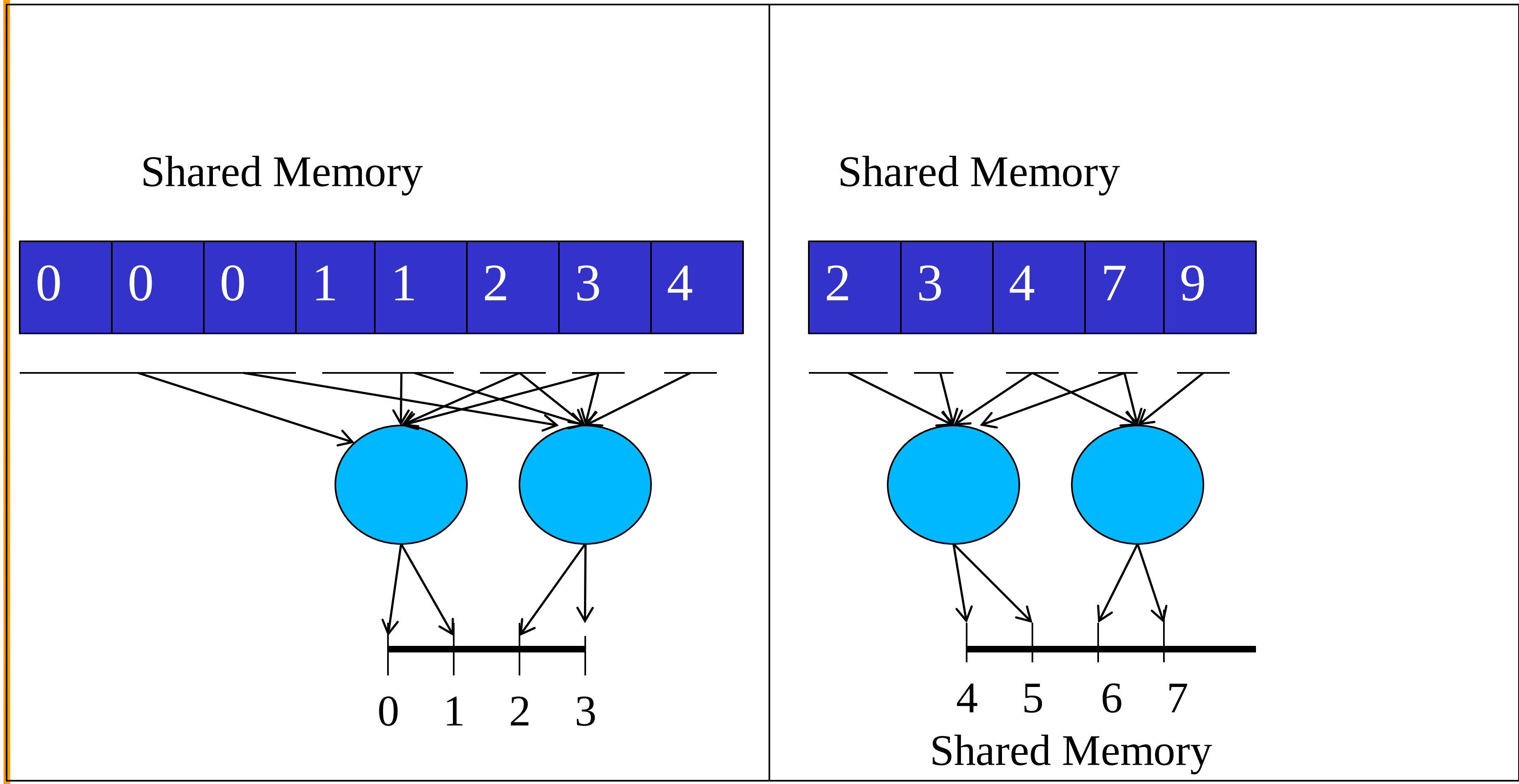


Actual Binning

- All inputs can now be placed into their bins in parallel, using atomic operations

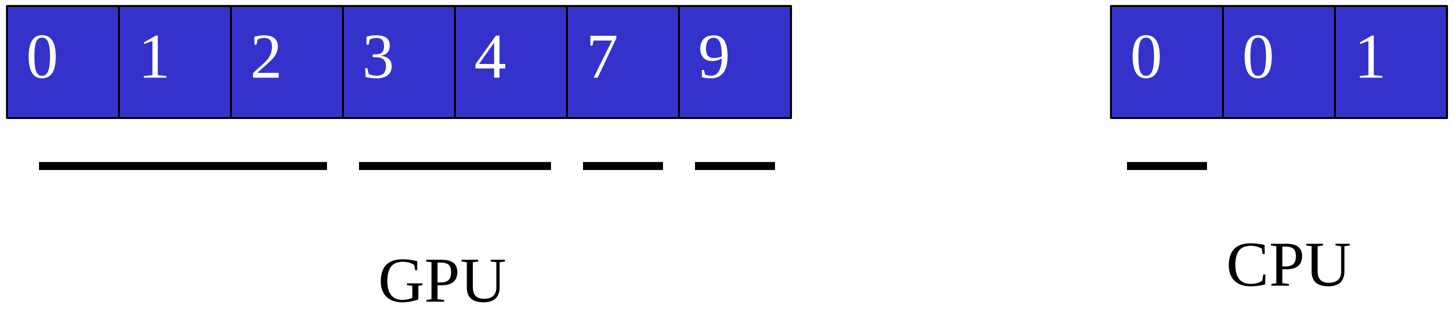


A Tiled Gather Implementation

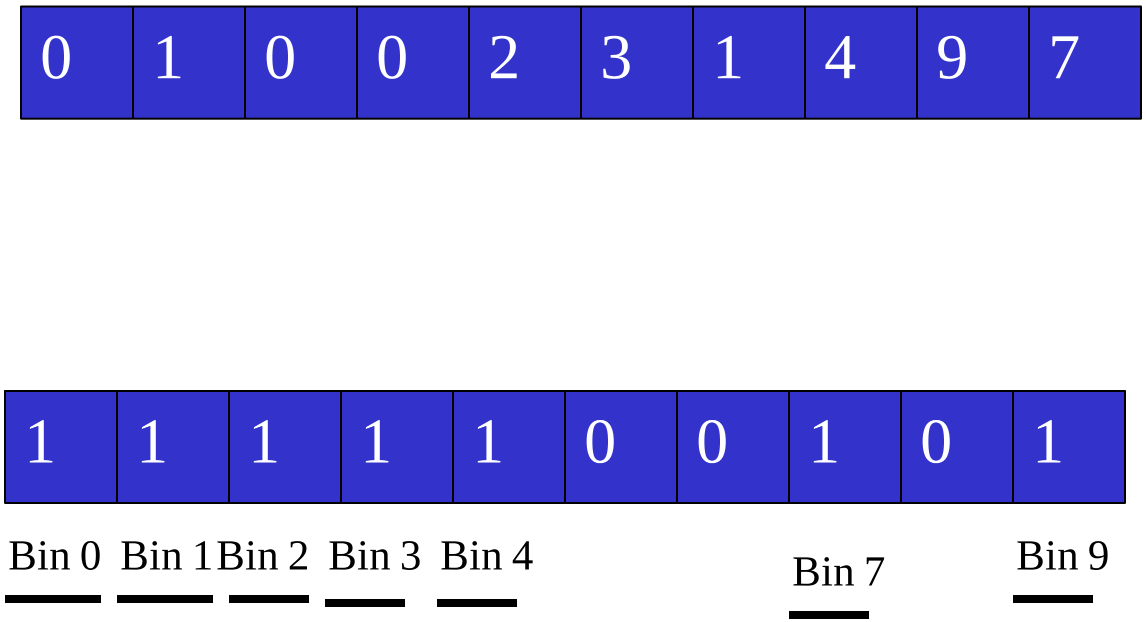


Controlling Load Balance (done during capacity generation)

- Limit the size of each bin
  - When counter exceeds limit for a bin, the input samples are placed into a "CPU" overflow bin
  - CPU places excess sample points into a CPU list
  - CPU does gridding on the excess sample points in parallel with GPU
  - Eventually merge



Set a Limit on Bin Capacities



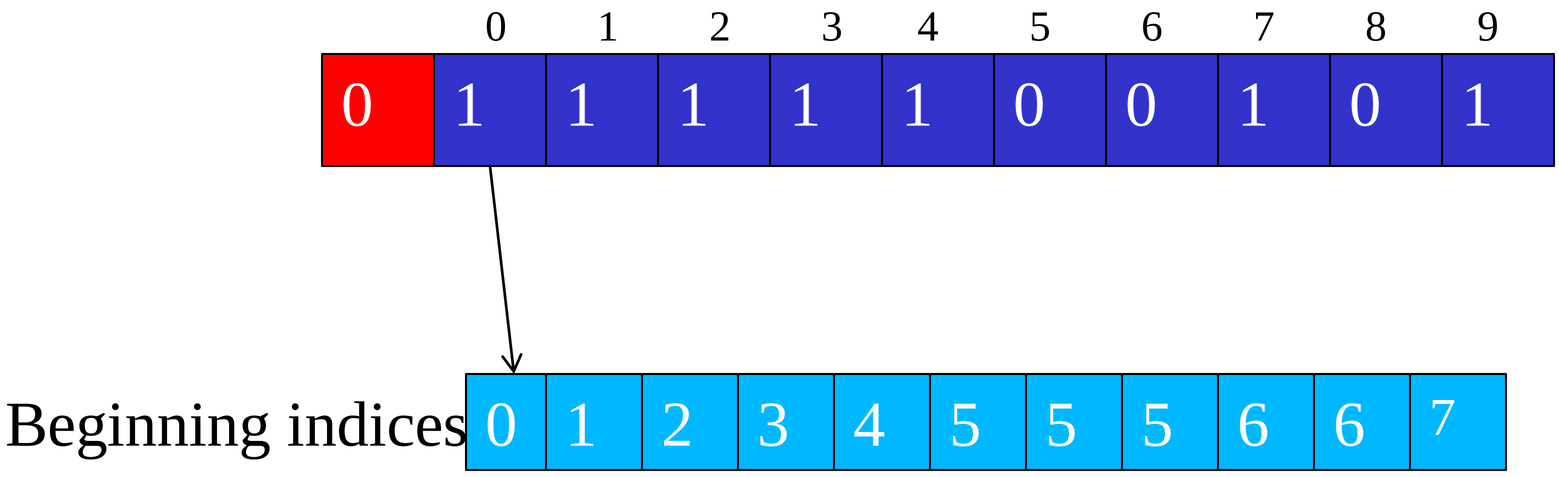
Capacity of each bin limited to 1

When a bin capacity reaches a preset limit, do not further increment the capacity counter

But place the excess input into an overflow bin

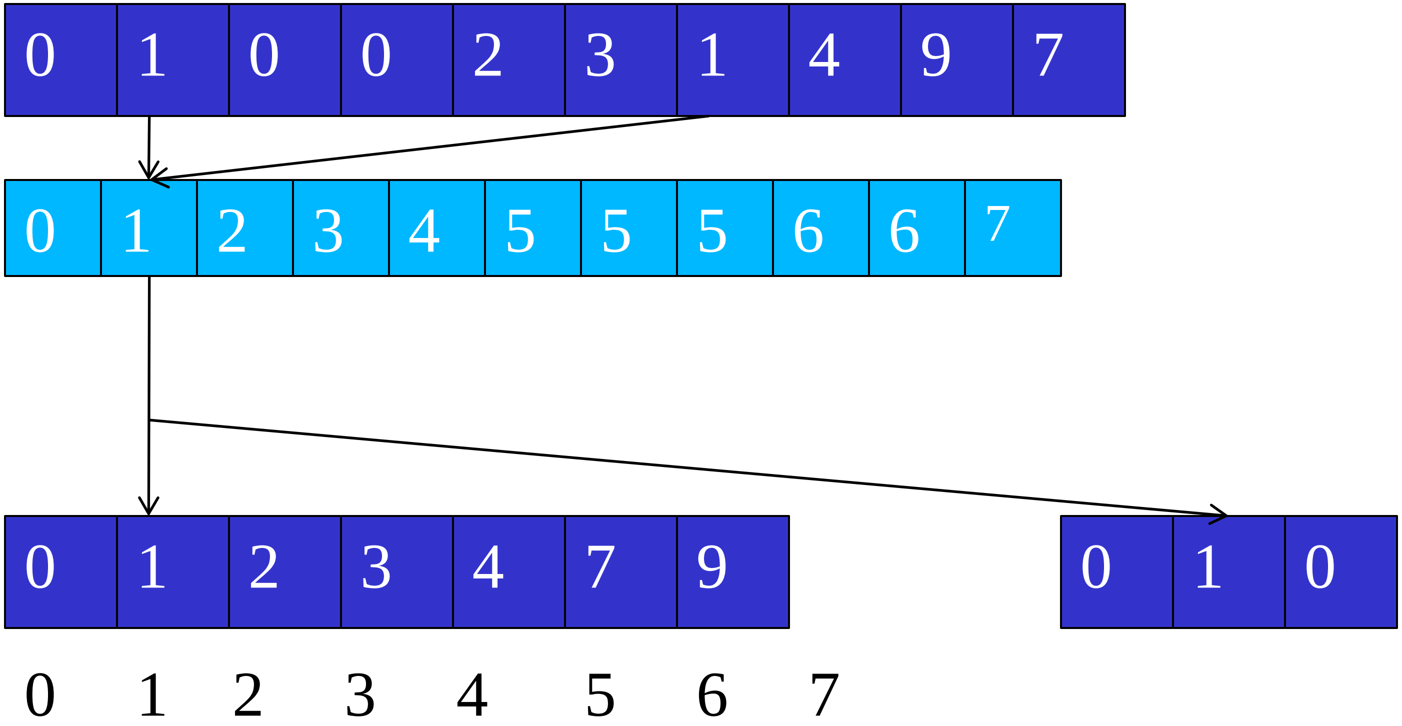
Determine Start and End of Bins

- Use parallel scan operations on the bin capacity array to generate an array of starting points of all bins (CUDPP)



Actual Binning

- All inputs can now be placed into their bins in parallel



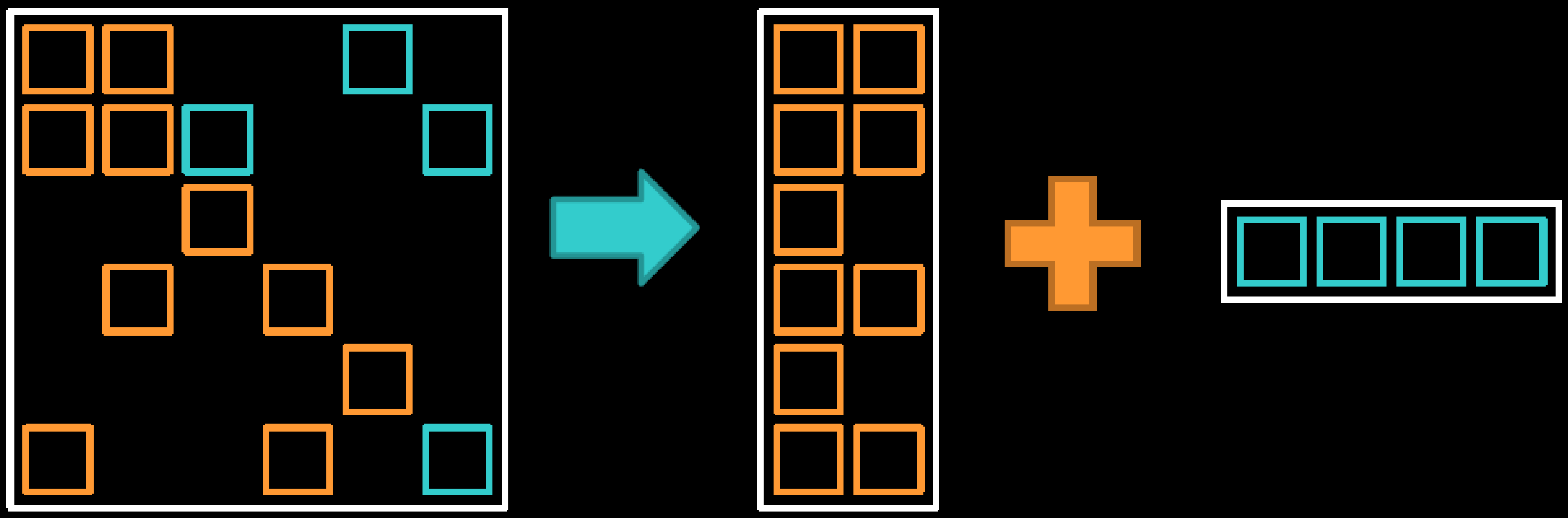
Note the similarity

- Compact bins – CSR
- Overflow bins - COO

- One could use ELL or JDS type of optimization on bins if desired

Hybrid Format

- ELL handles *typical* entries
- COO handles *exceptional* entries
  - Implemented with segmented reduction



ANY FURTHER QUESTION?