An Accelerated Procedure for Hypergraph Coarsening on the GPU

Lin Cheng, Hyunsu Cho, and Peter Yoon

Trinity College

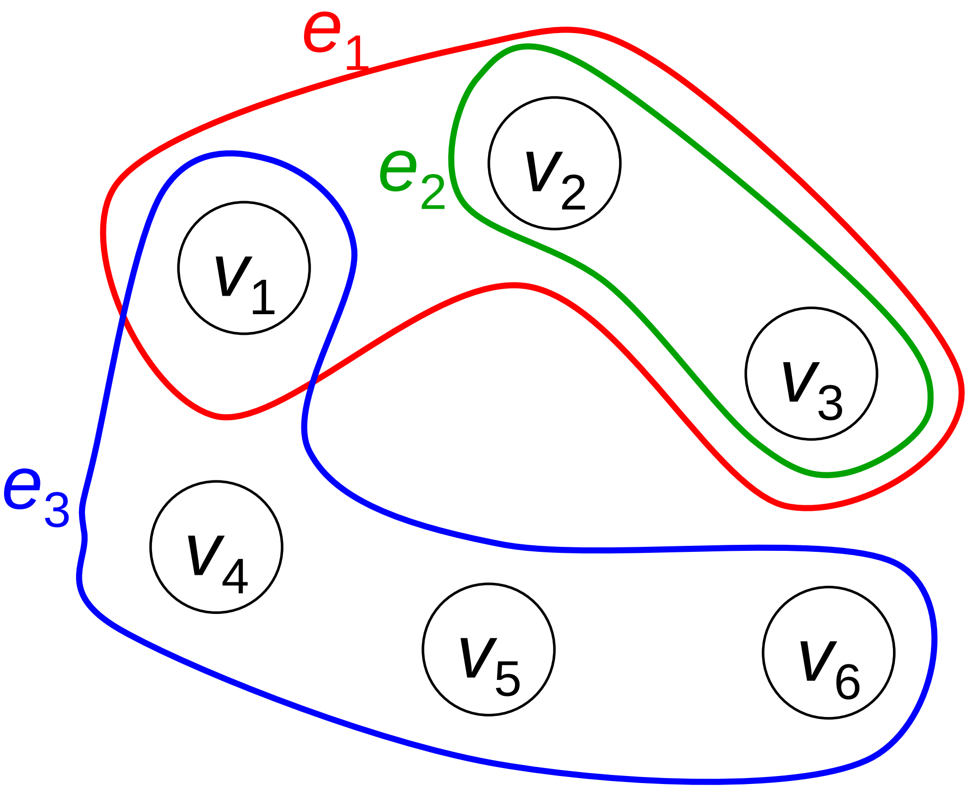
Hartford, CT, USA

Outline

- Hypergraph coarsening
- Implementation challenges
- Runtime task planning
- Results

Hypergraph

- Nodes
- **Hyperedges** (nets)
  - Subsets of nodes



Hypergraph

- Hypergraph partitioning
  - Minimize **edge cut**
  - Balance constraint
- NP-complete

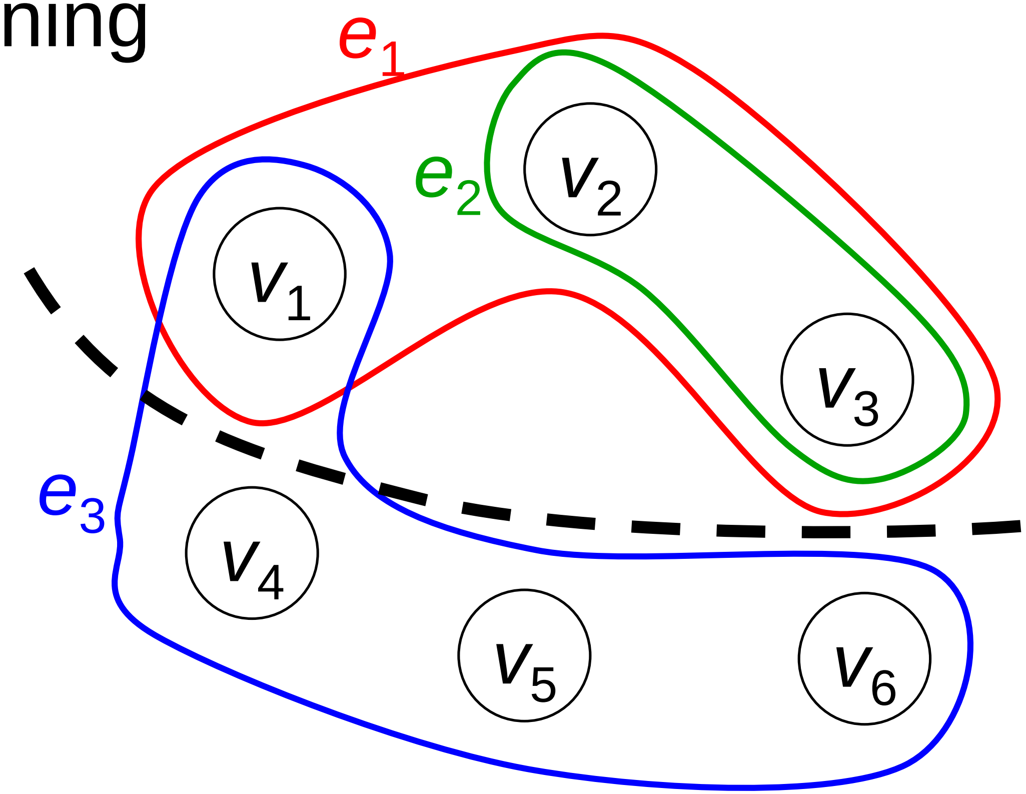
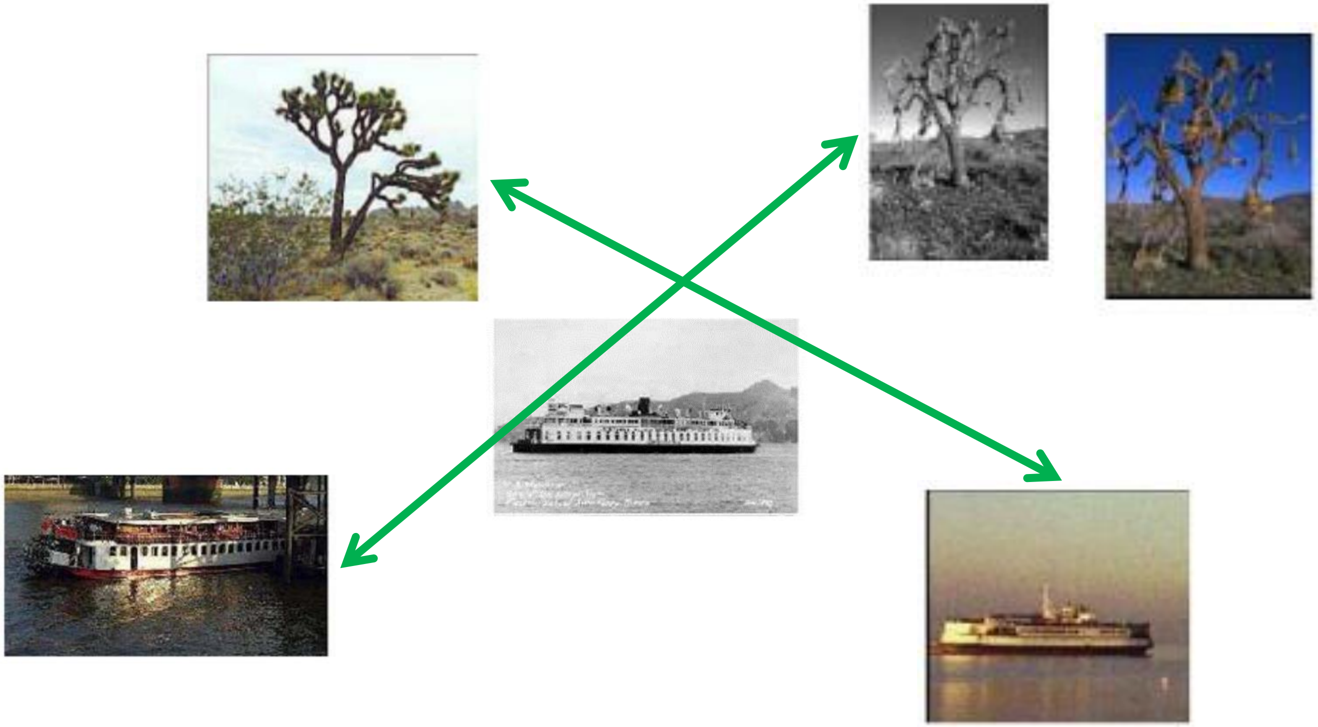


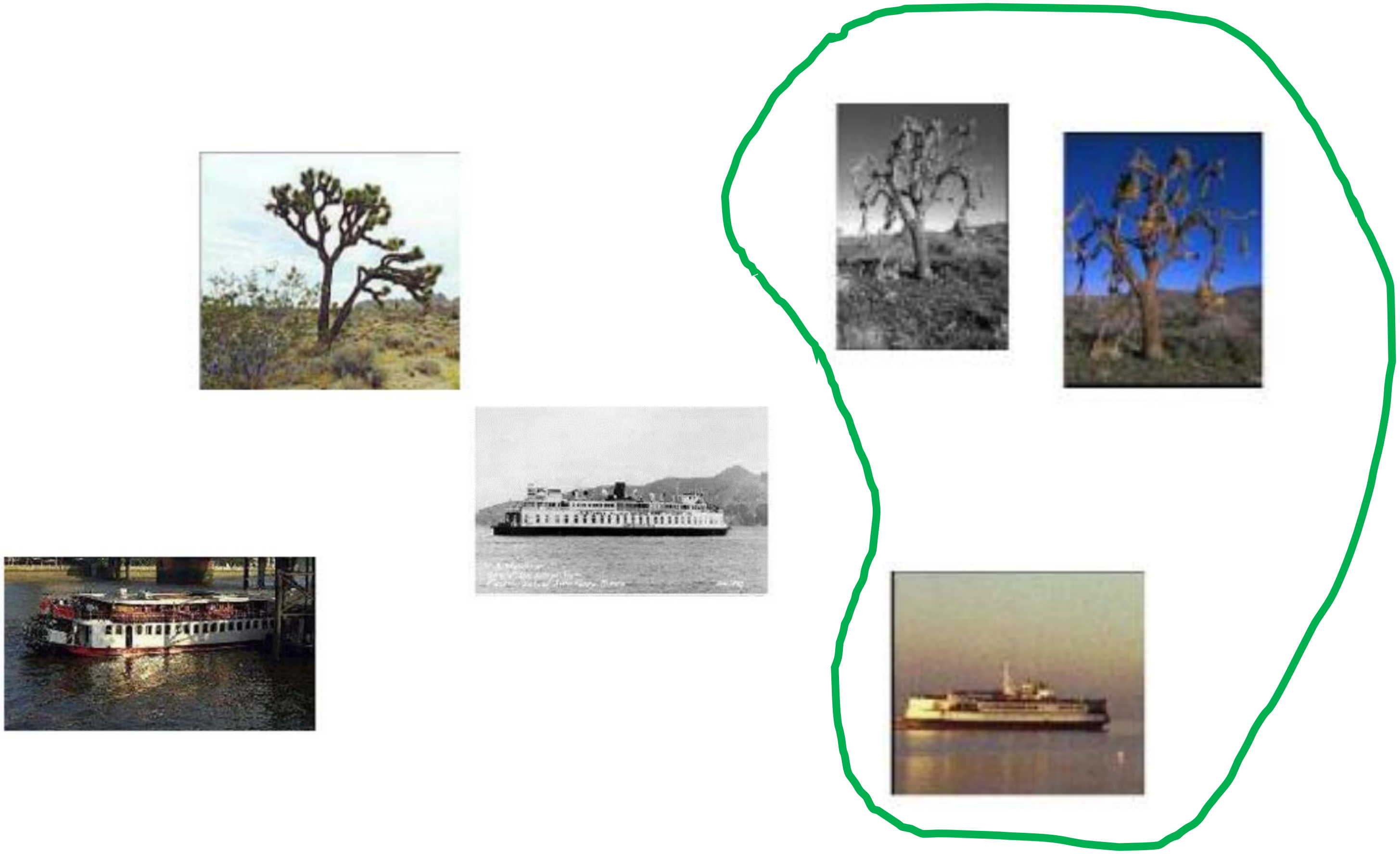
Image classification

- Similar images should go to same category
- Need to **compare** images to one another



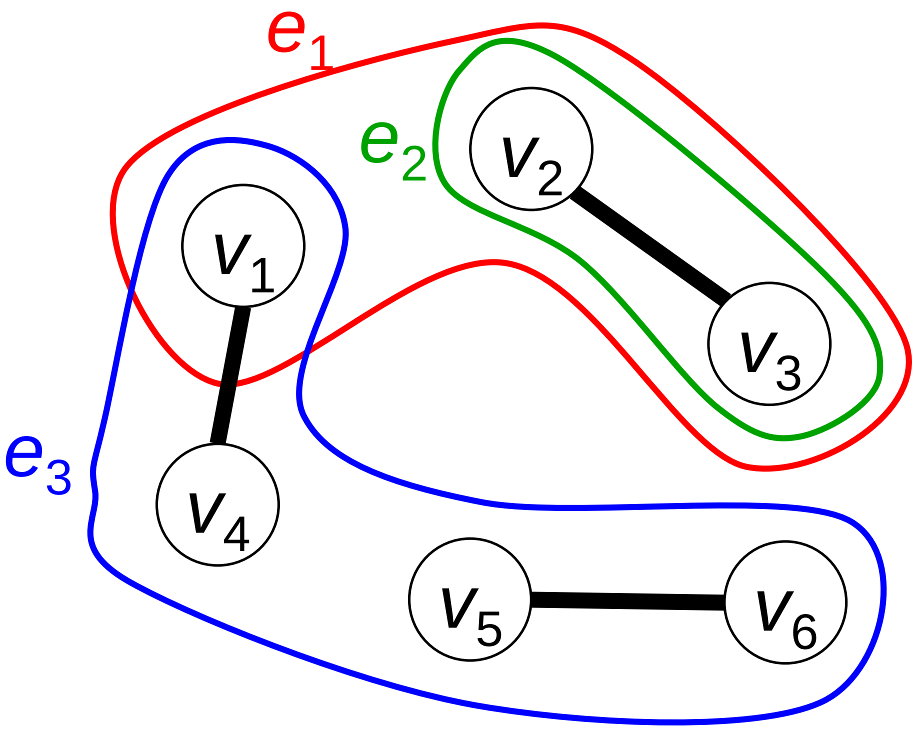
Hypergraph construction

- Compare multiple pictures at a time



Hypergraph coarsening

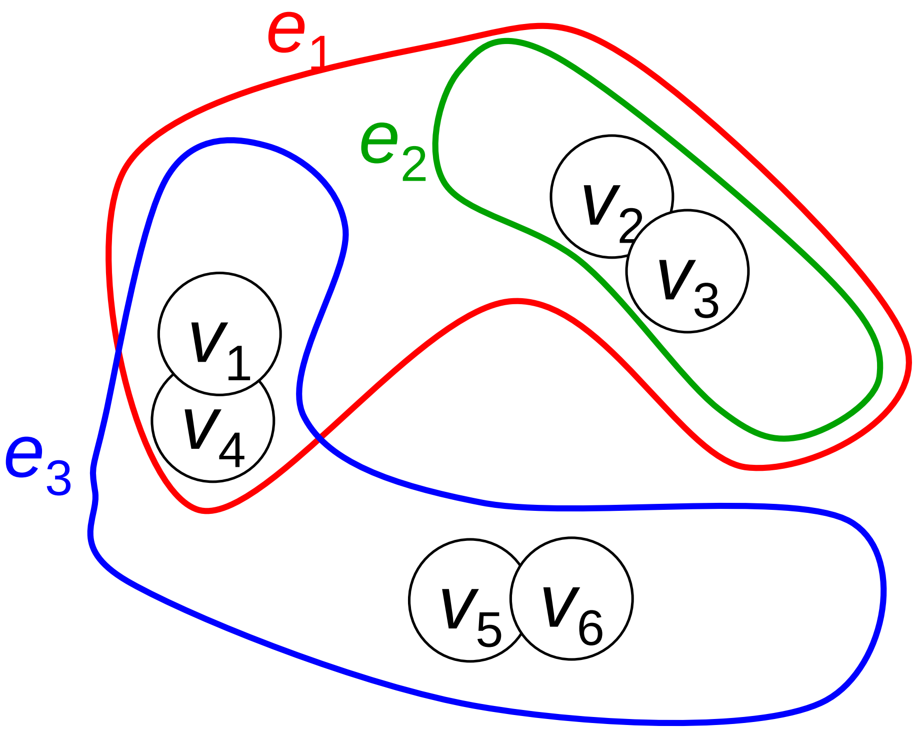
- Heuristic: reduce # nodes by fusing



6 nodes

Hypergraph coarsening

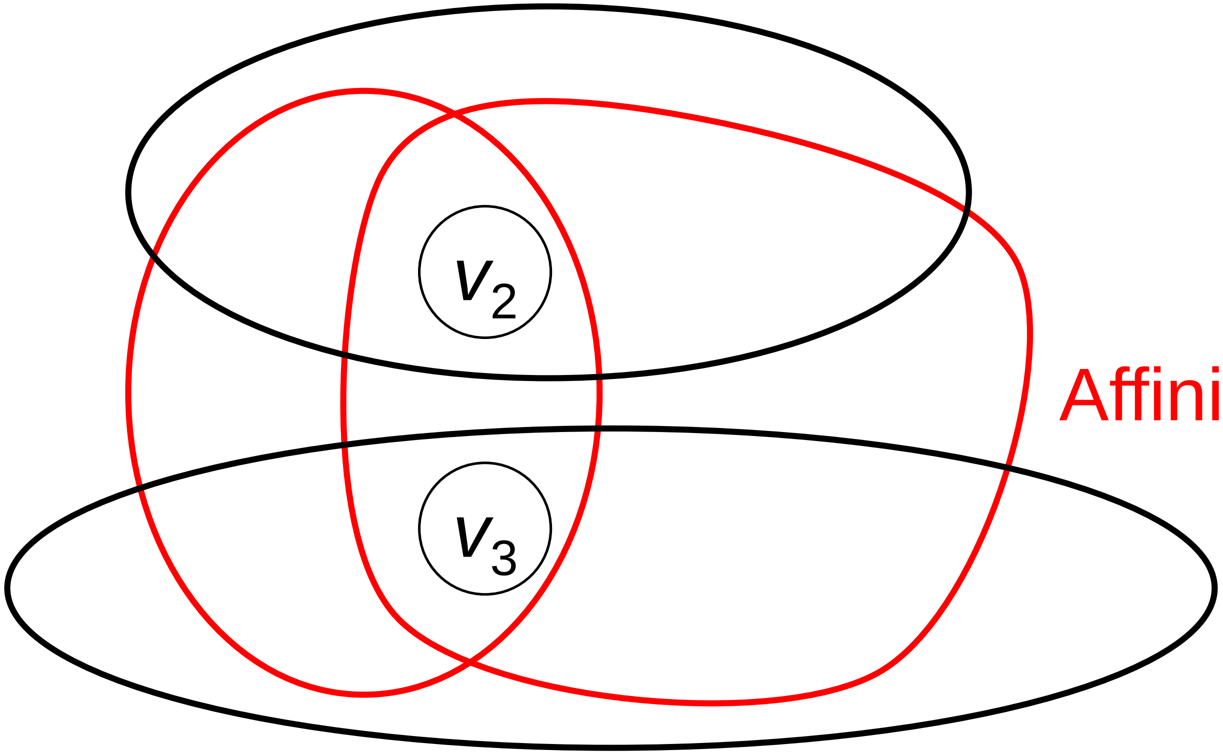
- Heuristic: reduce # nodes by fusing



3 nodes

Mondriaan algorithm

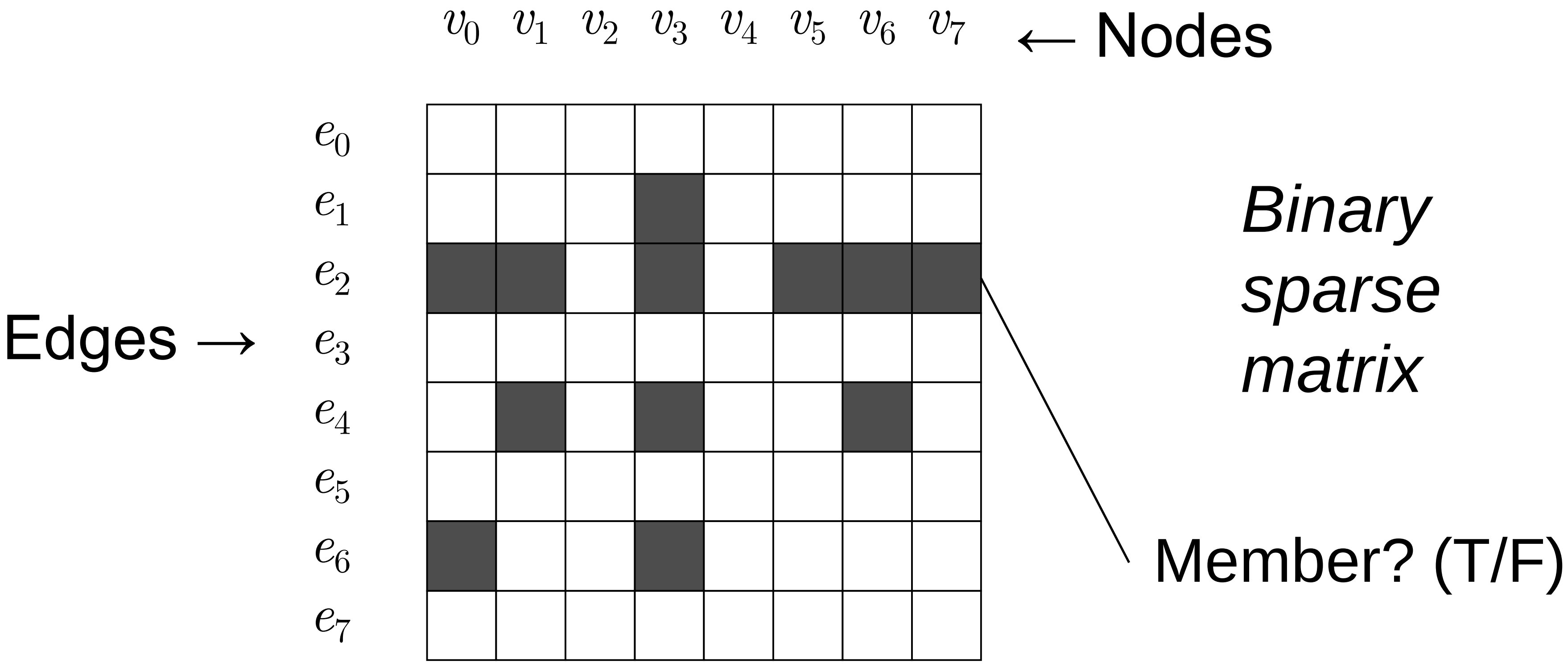
- Given a node, find most “similar” neighbor
- **Similarity = # hyperedges containing both**



Affinity = 2

Mondriaan algorithm

- Given a node, find most “similar” neighbor
- **Similarity = # hyperedges containing both**

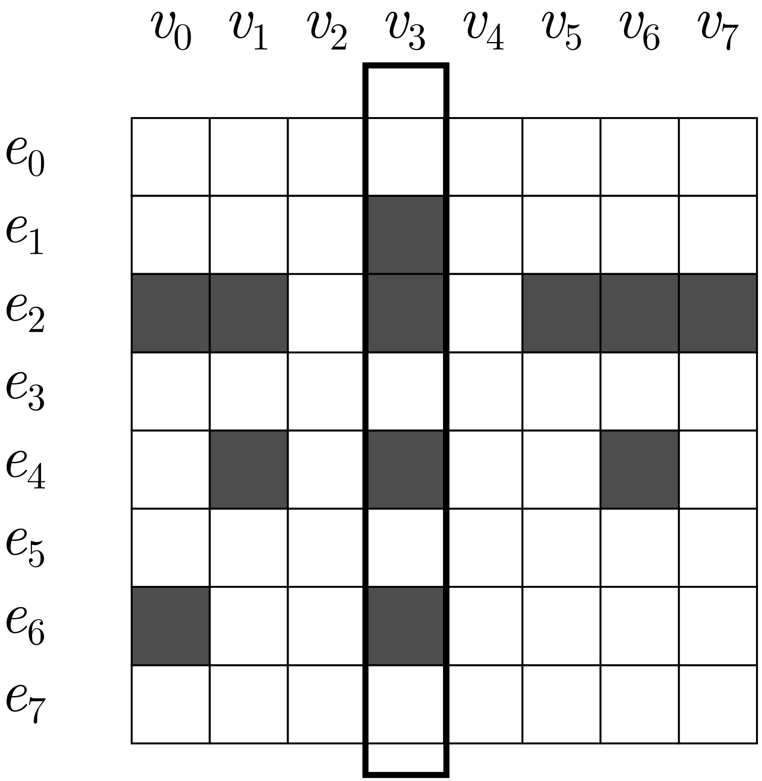


	v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7
e_0								
e_1				■				
e_2	■	■		■		■	■	■
e_3								
e_4		■		■			■	
e_5								
e_6	■			■				
e_7								

Binary sparse matrix

Mondriaan algorithm

- Given a node, find most “similar” neighbor
- **Similarity = # hyperedges containing both**

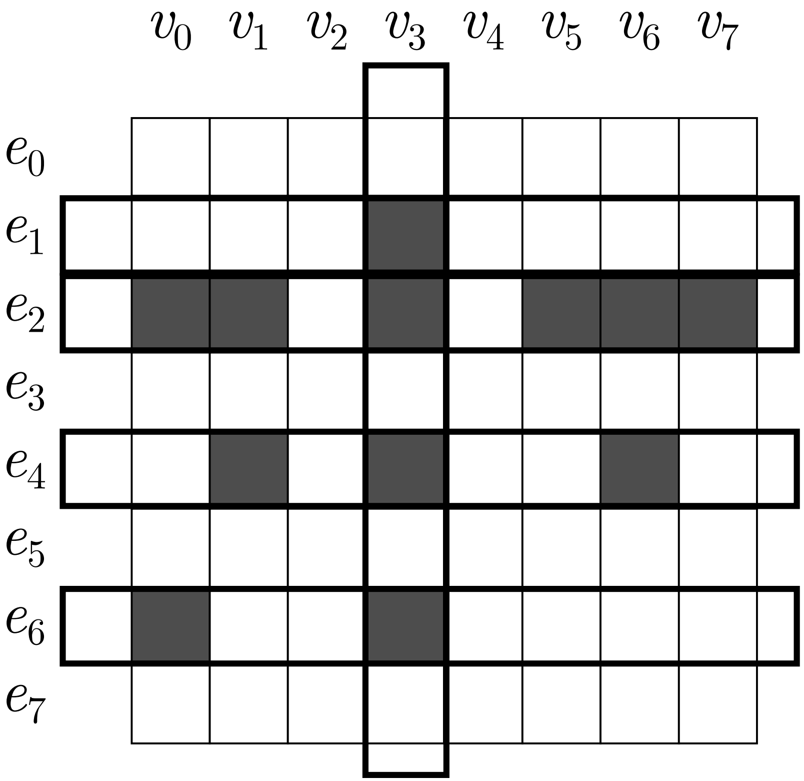


	v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7
e_0								
e_1				■				
e_2	■	■		■		■	■	■
e_3								
e_4		■		■			■	
e_5								
e_6	■			■				
e_7								

1. Find edges containing v_3

Mondriaan algorithm

- Given a node, find most “similar” neighbor
- **Similarity = # hyperedges containing both**

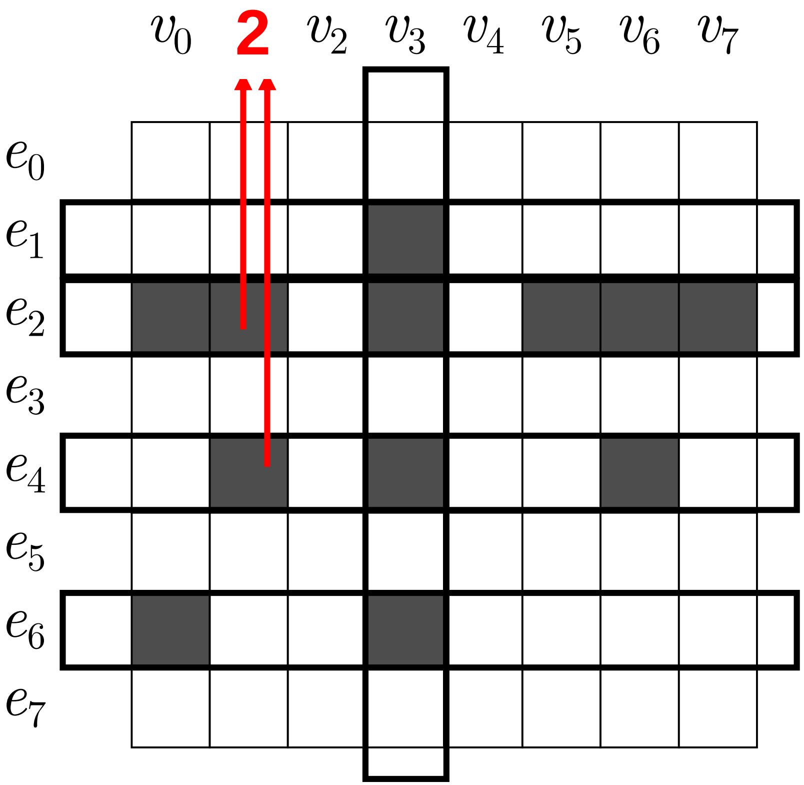


1. Find edges containing v_3

2. Collect nonzeros

Mondriaan algorithm

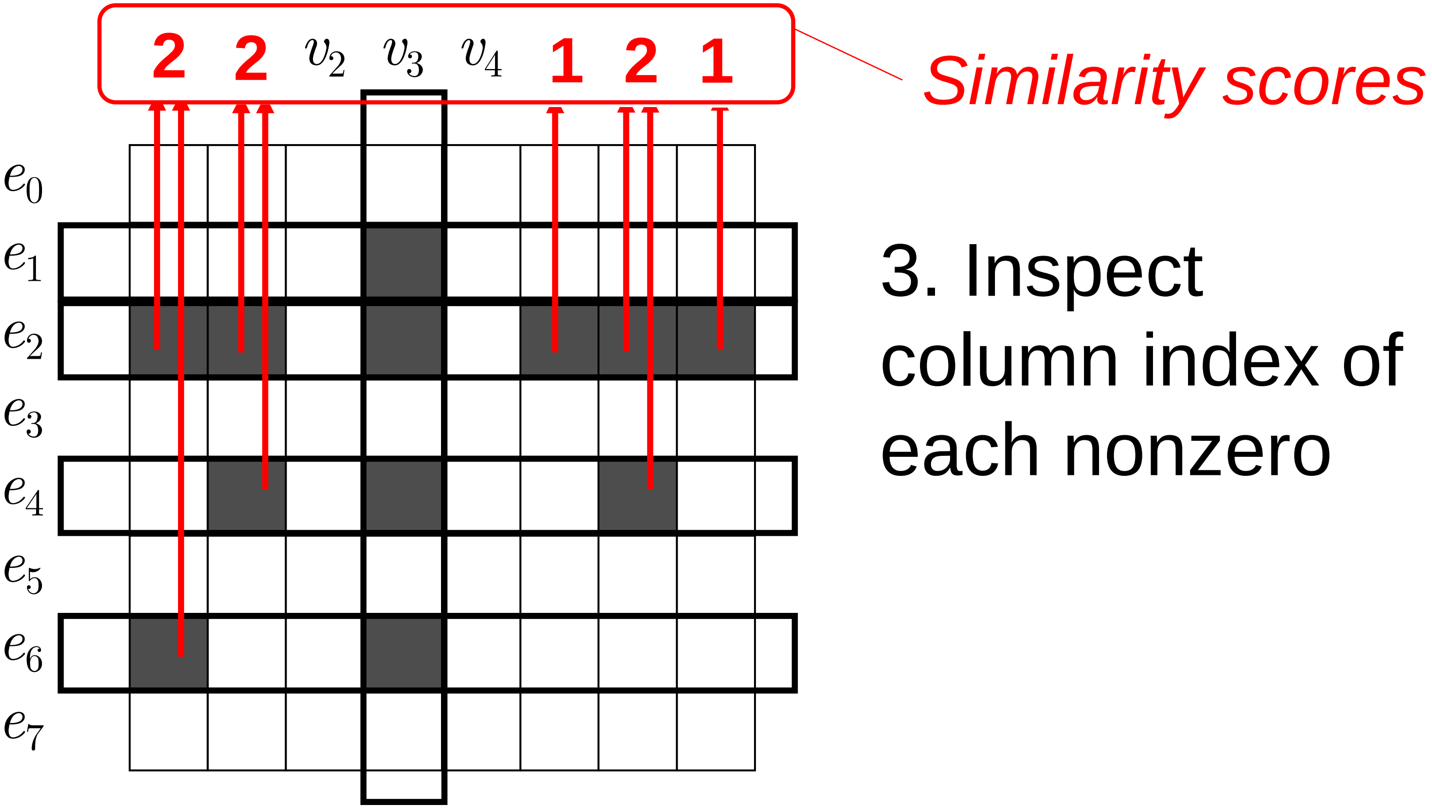
- Given a node, find most “similar” neighbor
- **Similarity = # hyperedges containing both**



3. Inspect column index of each nonzero

Mondriaan algorithm

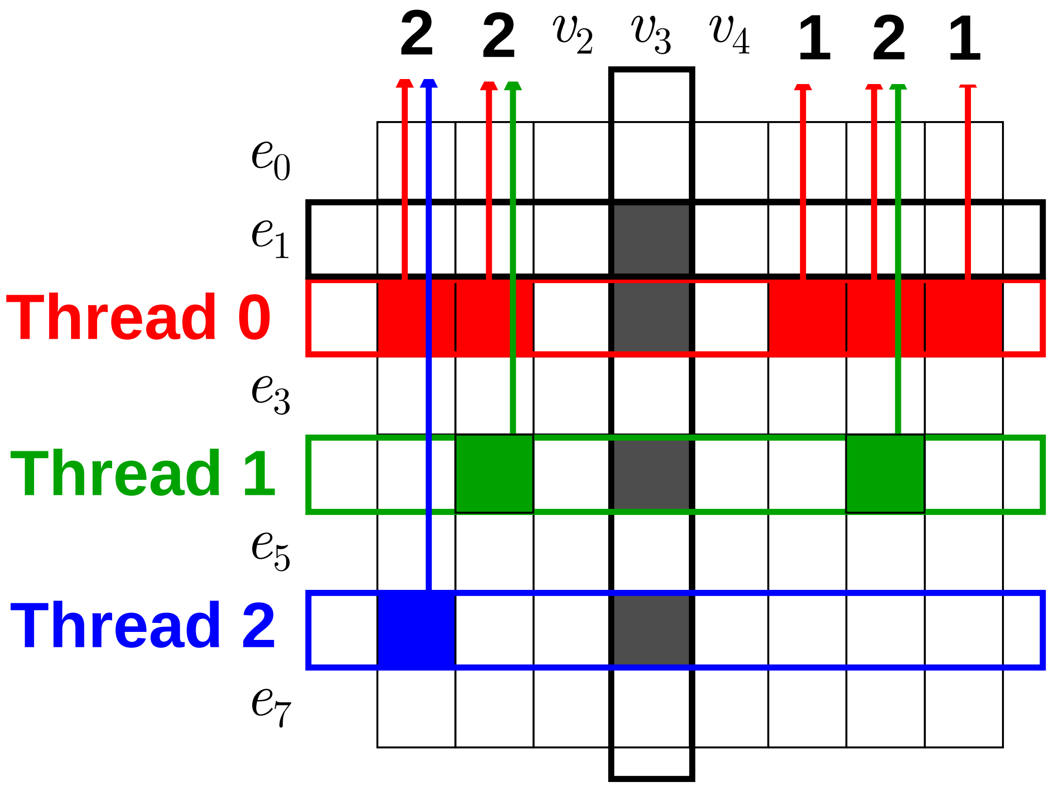
- Given a node, find most “similar” neighbor
- **Similarity = # hyperedges containing both**



3. Inspect column index of each nonzero

Mondriaan algorithm

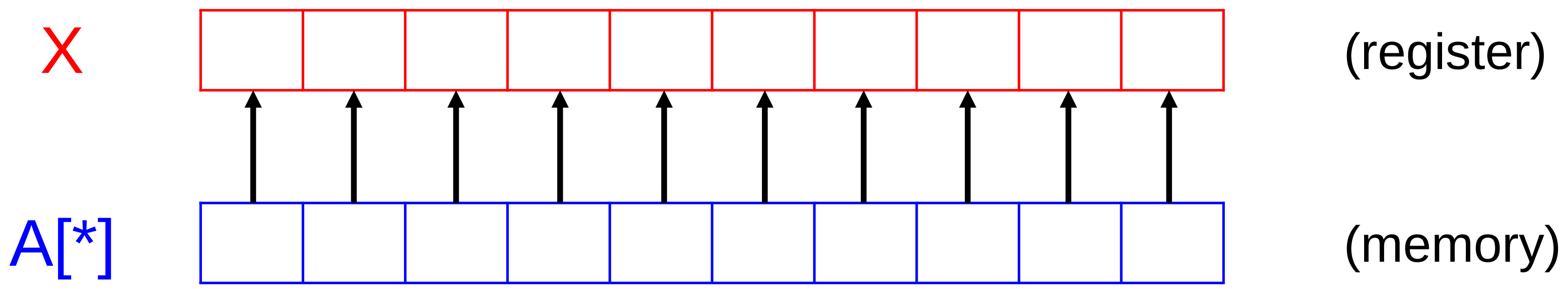
- Parallel algorithm:
  Inspect edges in parallel



GPU as parallel accelerator

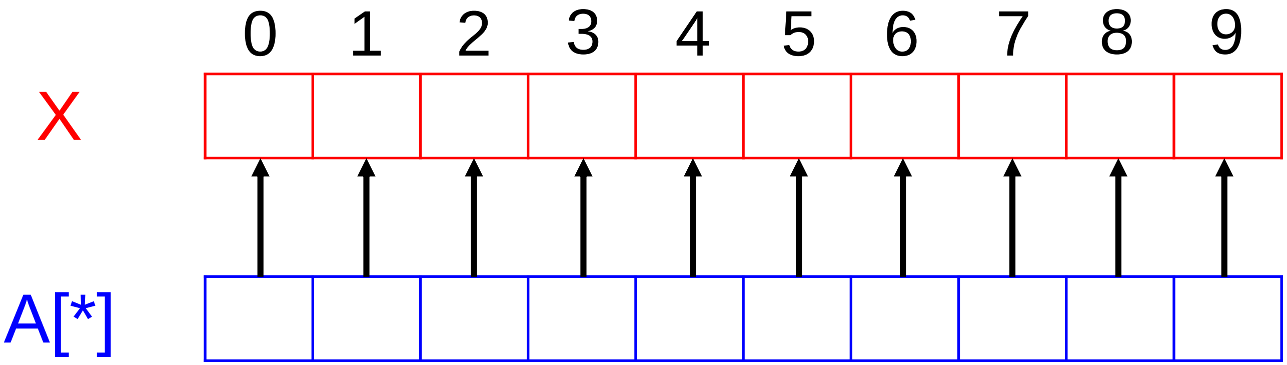
- NVIDIA GPUs : organized in **warps**
  32 threads share one instruction counter

LOAD A[*] INTO X



Warp divergence

Thread 0-9: LOAD A[0-9] INTO X



0 1 2 3 4 5 6 7 8 9

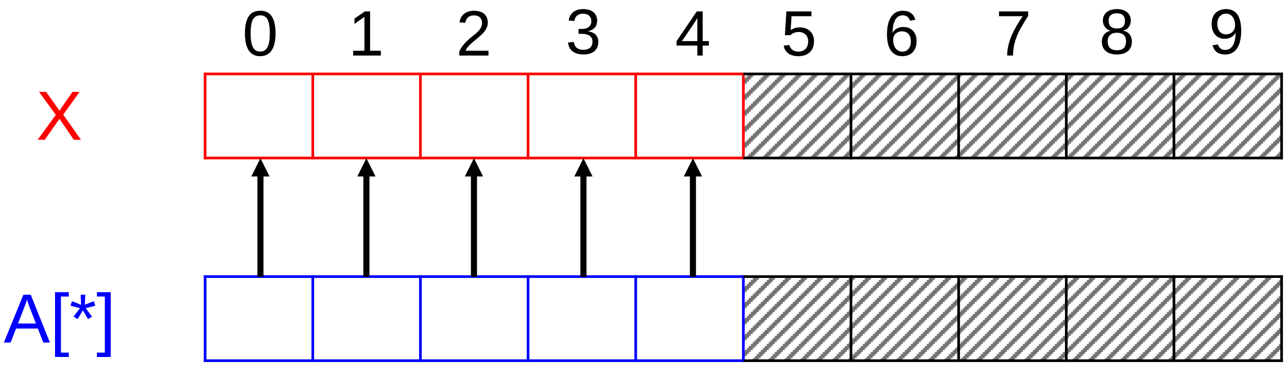
X

A[*]

Warp divergence

Thread 0-4: LOAD A[0-4] INTO X

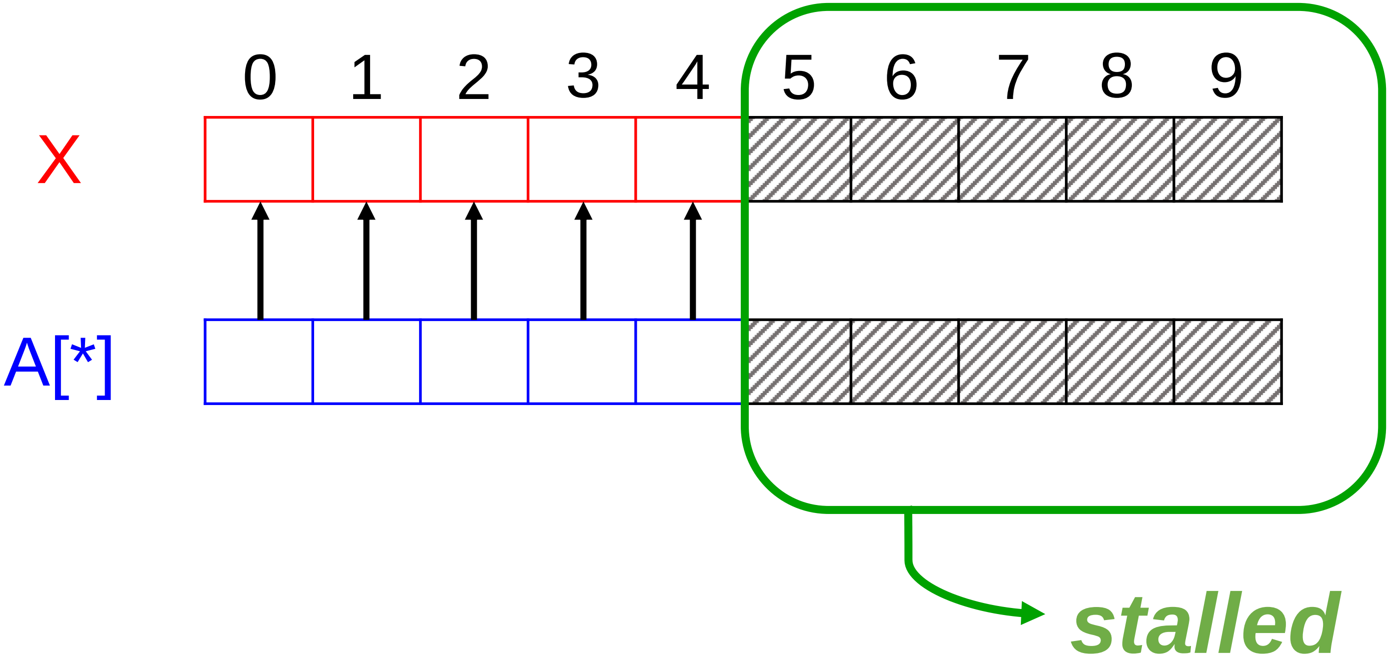
Thread 5-9: NONE



Warp divergence

Thread 0-4: LOAD A[0-4] INTO X

Thread 5-9: NONE

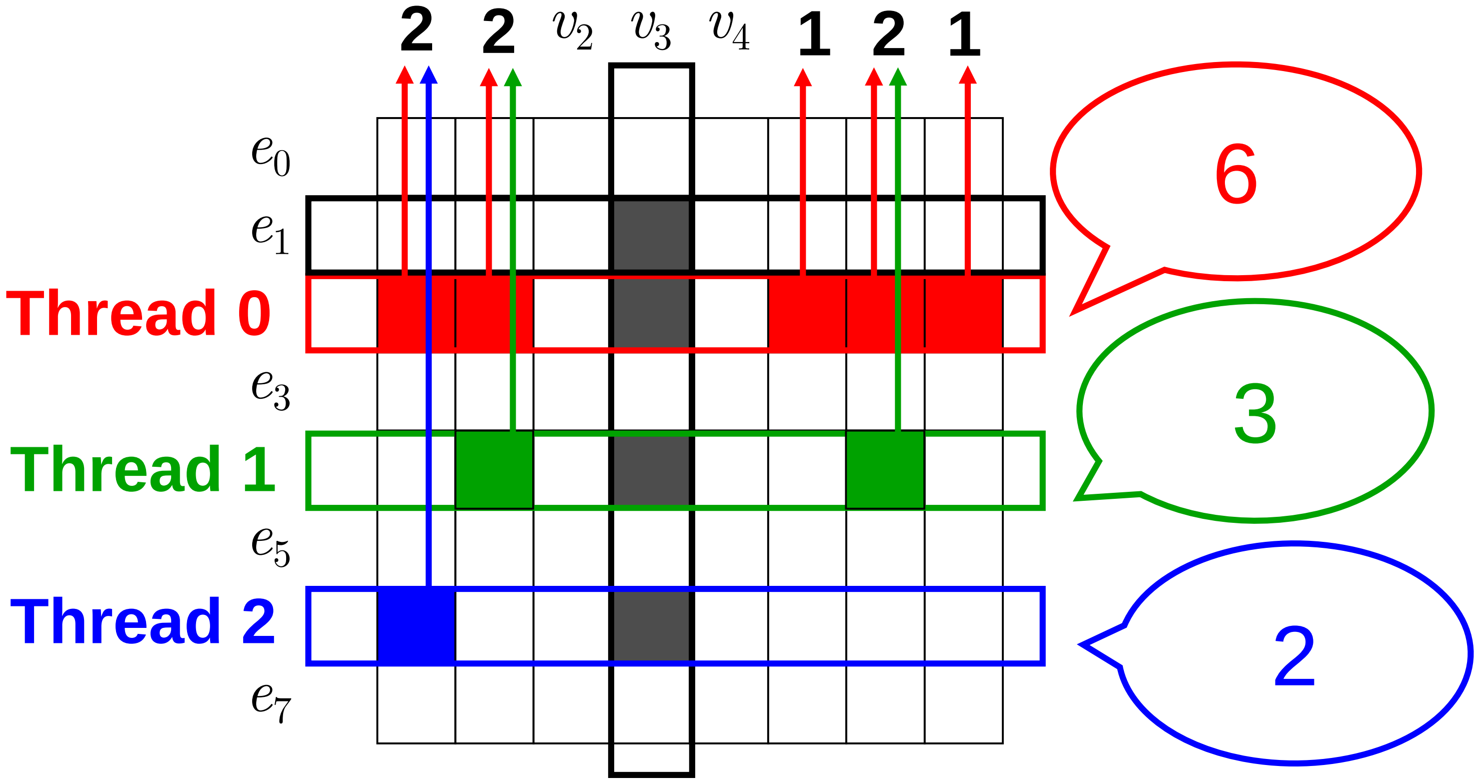


Warp divergence

- Serializes execution
- Caused by **load imbalance**
  - Sparse/irregular data

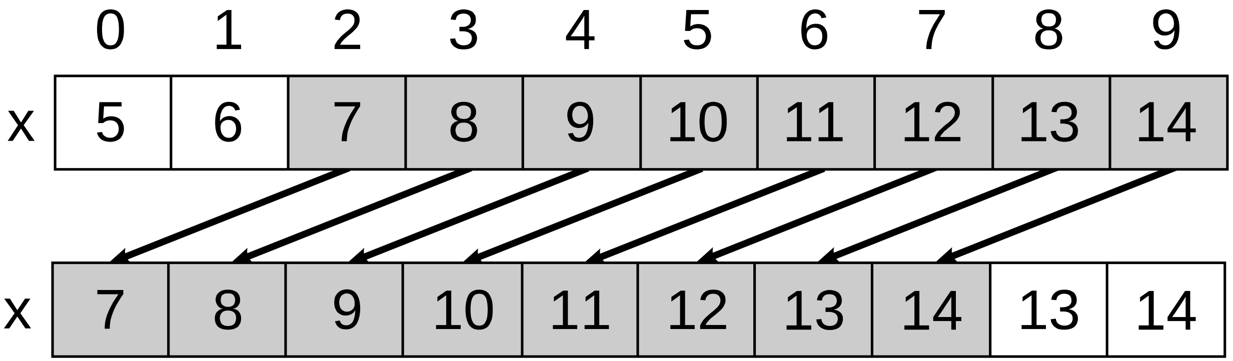
Mondriaan algorithm

- A naïve strategy results in load imbalance
- Nonzero entry = workload



SHFL to the rescue

- Compiler primitive
- Shuffles content of adjacent registers

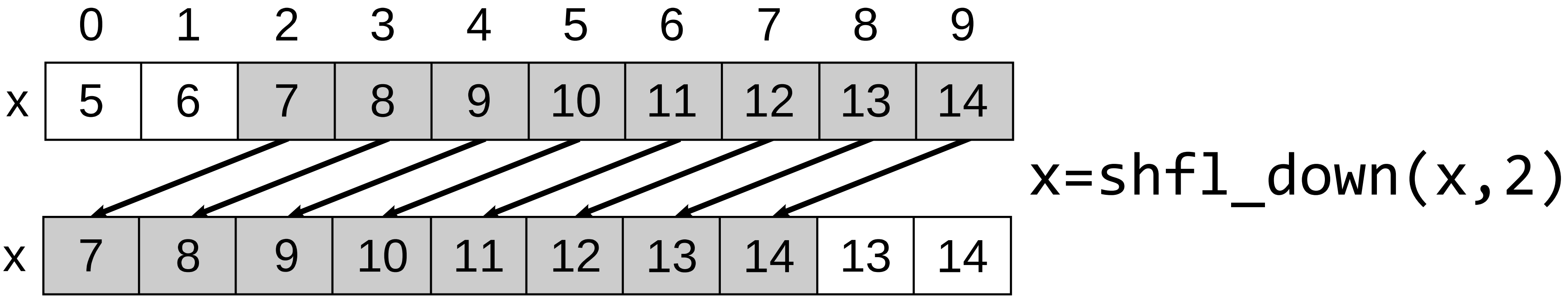


	0	1	2	3	4	5	6	7	8	9
x	5	6	7	8	9	10	11	12	13	14
x	7	8	9	10	11	12	13	14	13	14

```
x=shfl_down(x,2)
```

SHFL to the rescue

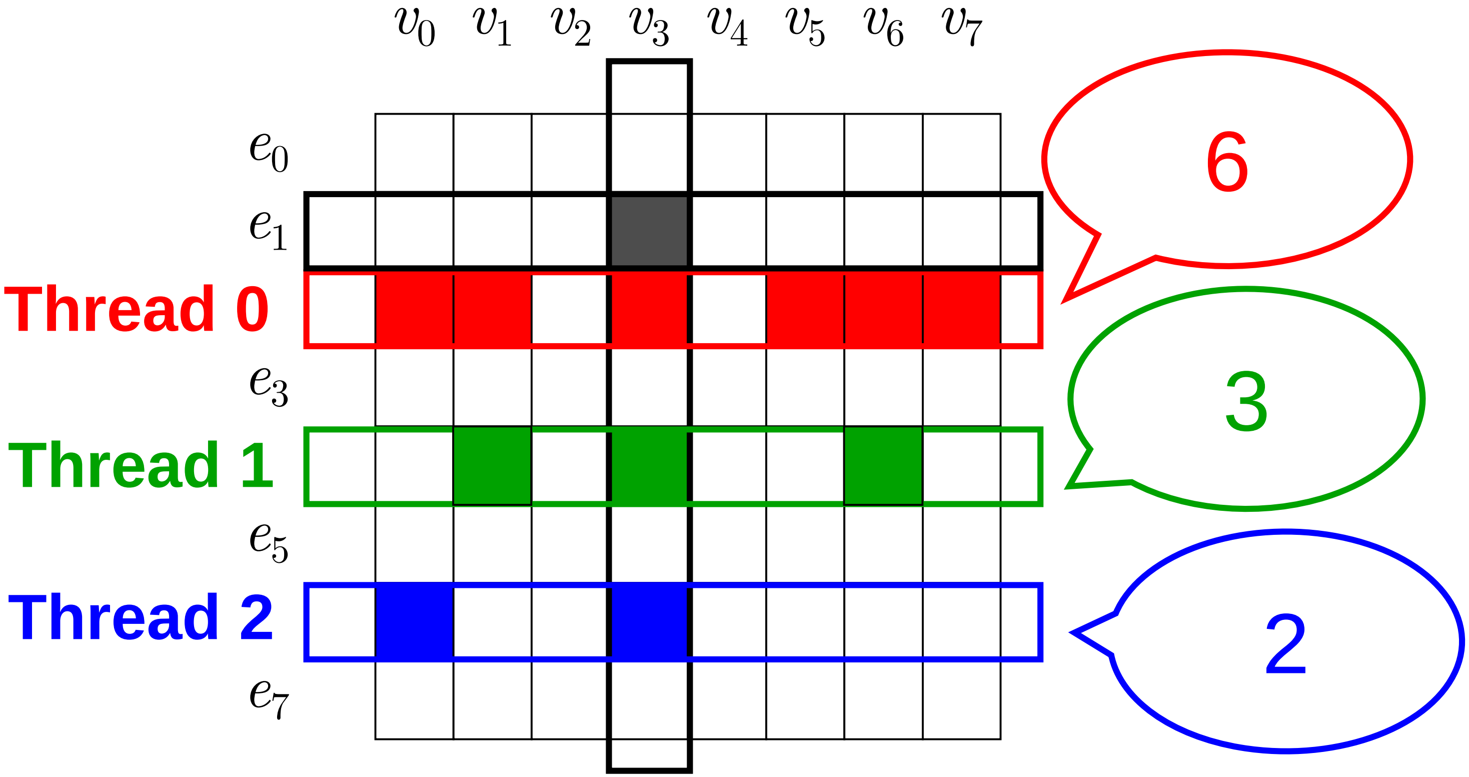
- Compiler primitive
- Shuffles content of adjacent registers
- **Single machine instruction**
- **Warp-synchronous; no sync. needed after**



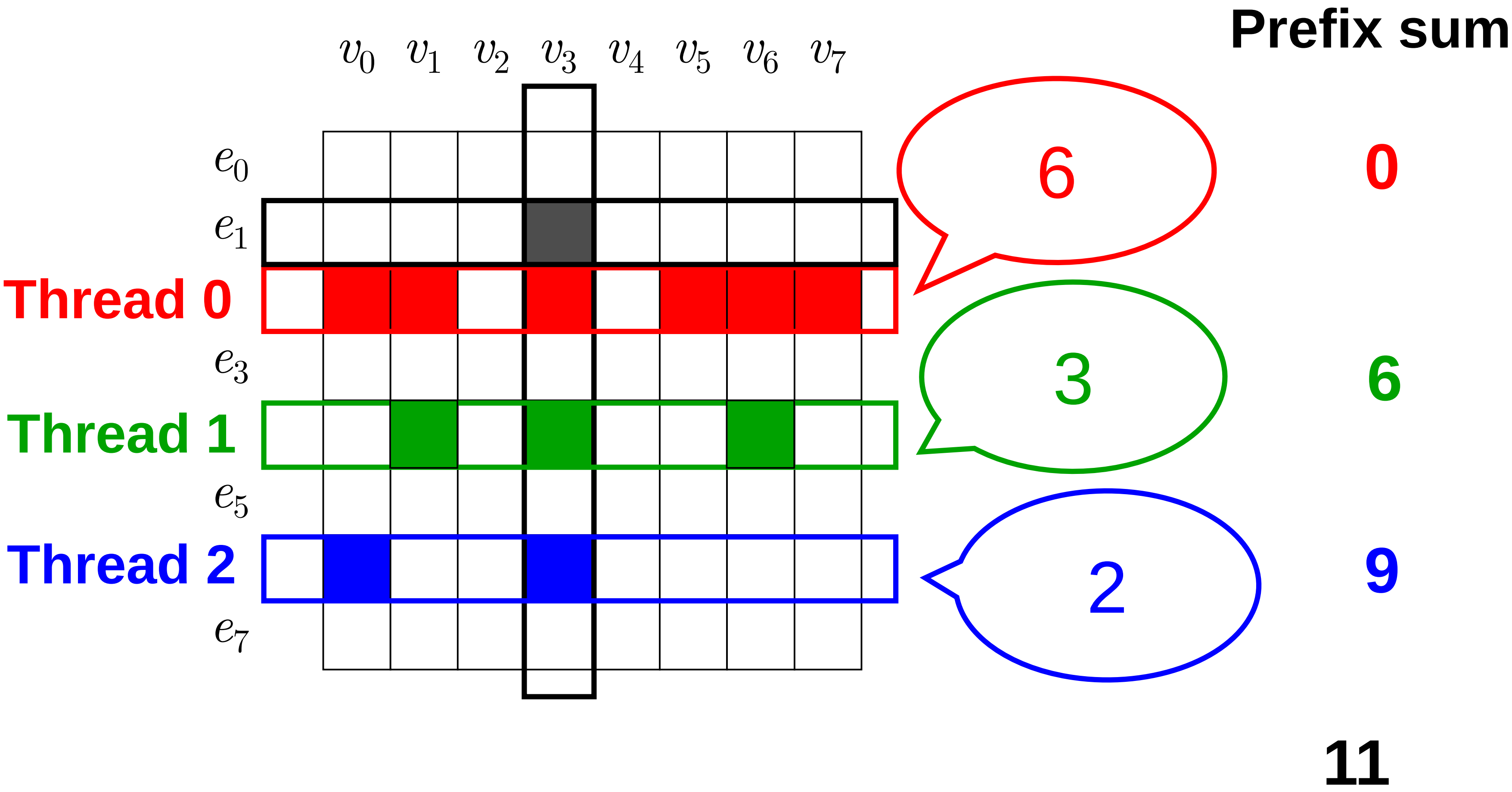
	0	1	2	3	4	5	6	7	8	9
x	5	6	7	8	9	10	11	12	13	14
x	7	8	9	10	11	12	13	14	13	14

```
x=shfl_down(x,2)
```

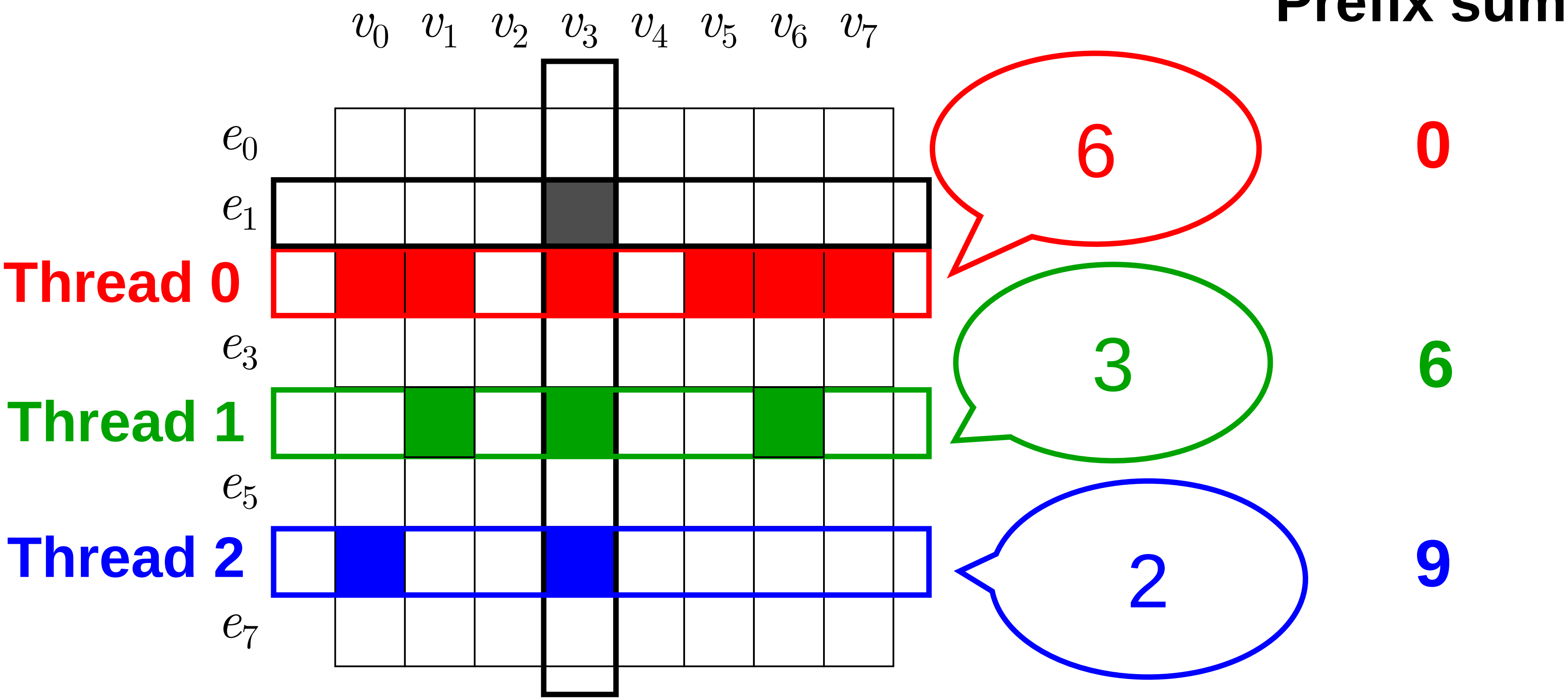
Runtime planning with SHFL



Runtime planning with SHFL



Runtime planning with SHFL



Thread 0

Thread 1

Thread 2

6

3

2

Prefix sum

0

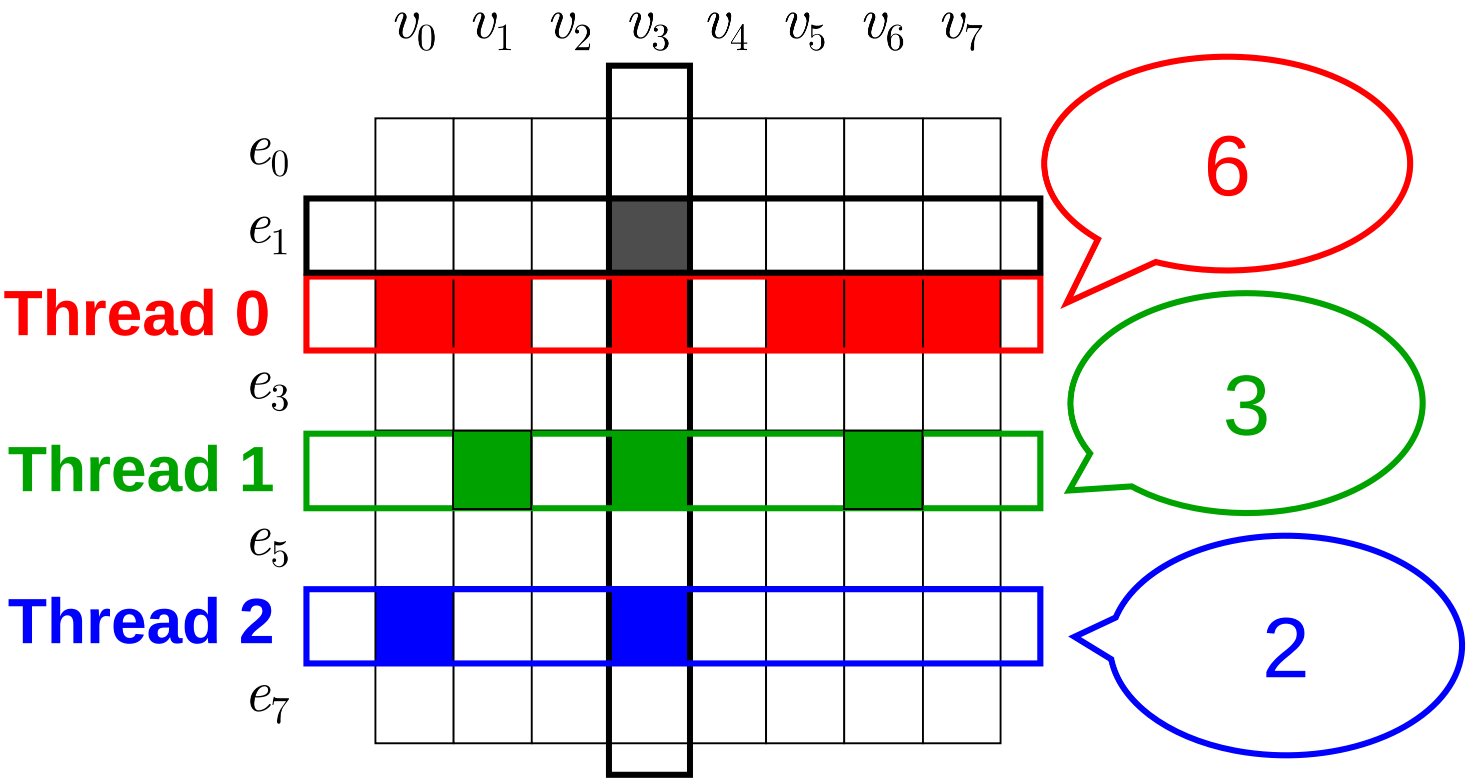
6

9

11

ceil(11 / 3) = **4**

Runtime planning with SHFL



Range

0 – 3

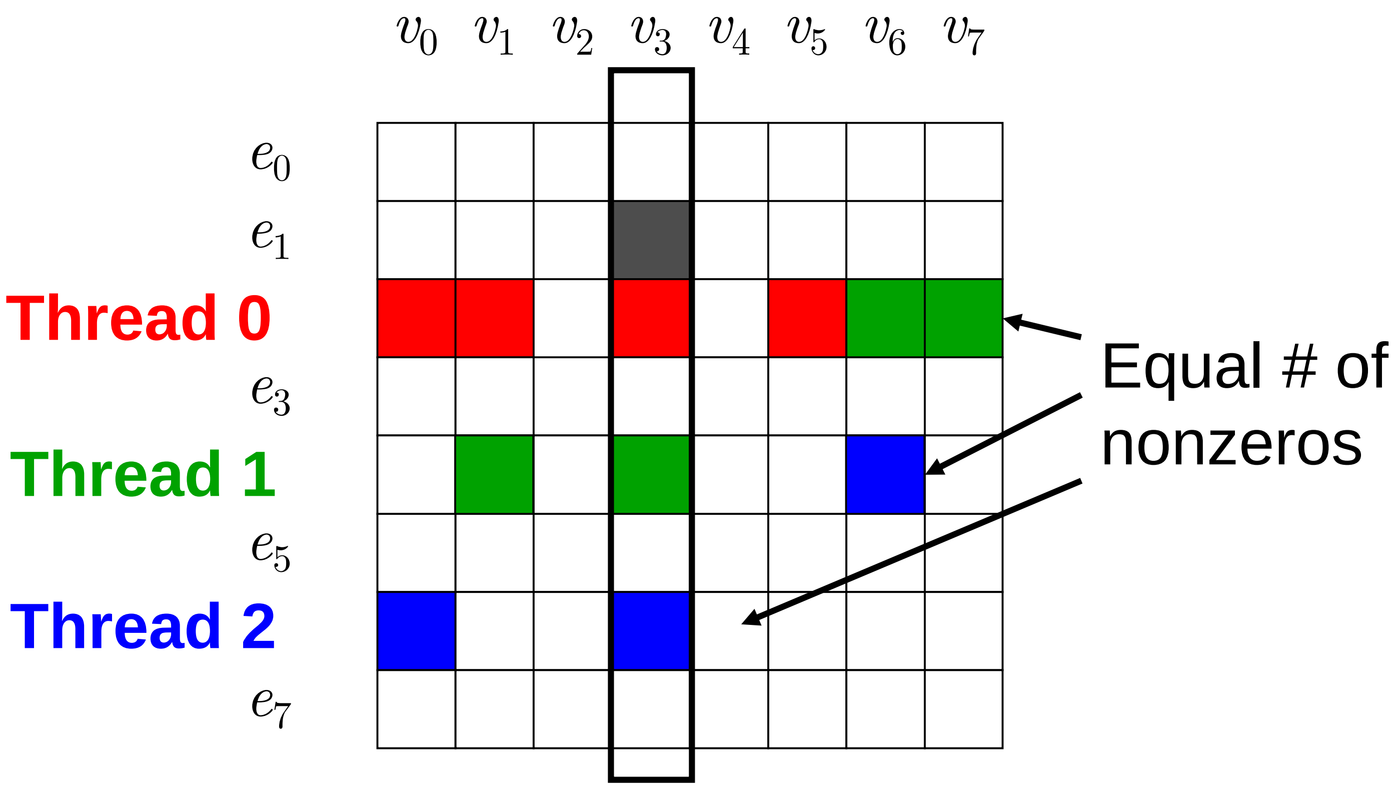
4 – 7

8 – 10

11

ceil(11 / 3) = **4**

Runtime planning with SHFL



Equal # of nonzeros

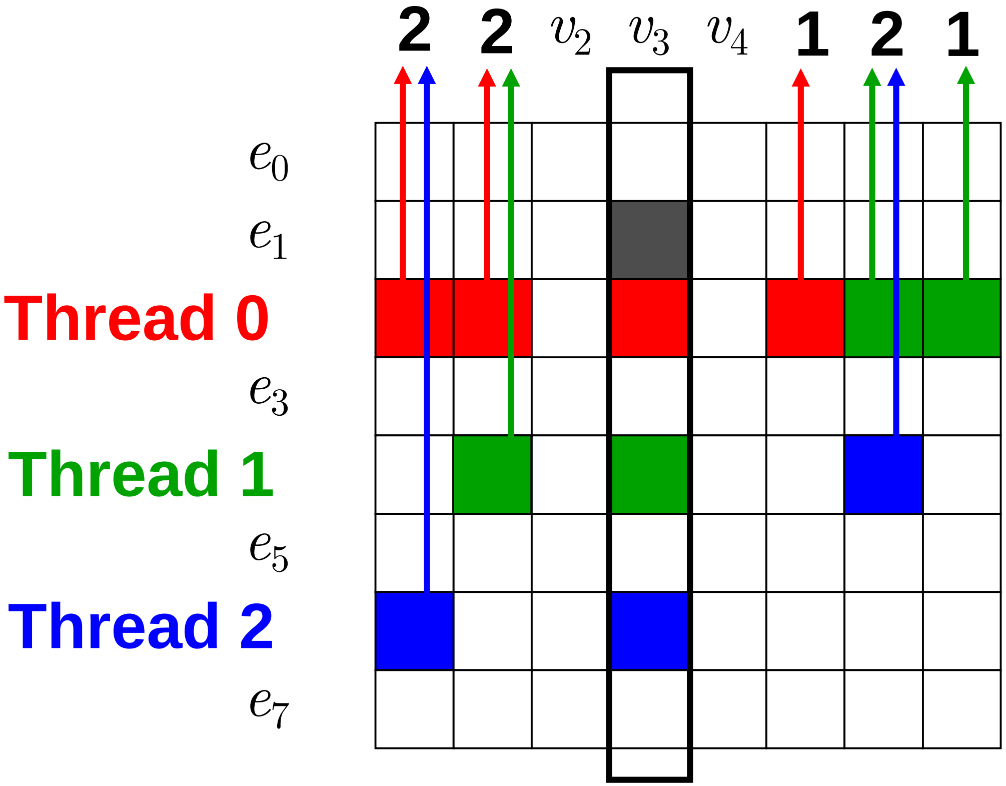
Range

0 – 3

4 – 7

8 – 10

Runtime planning with SHFL



2 2 v_2 v_3 v_4 1 2 1

e_0

e_1

Thread 0

e_3

Thread 1

e_5

Thread 2

e_7

Range

0 – 3

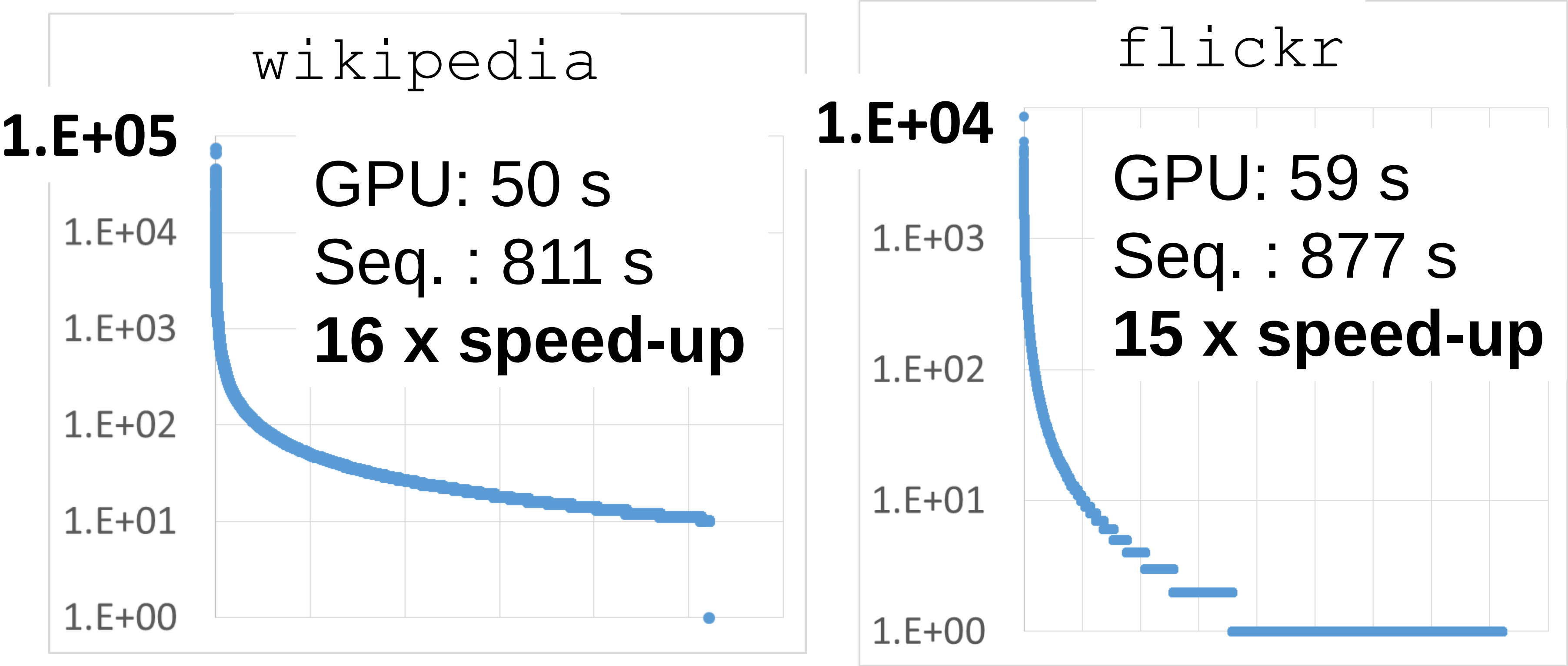
4 – 7

8 – 10

Results

- NVIDIA Tesla K20c (5GB mem)
- Intel Xeon E5-2620
- Reference sequential implementation of Mondriaan

Results: long-tailed distribution

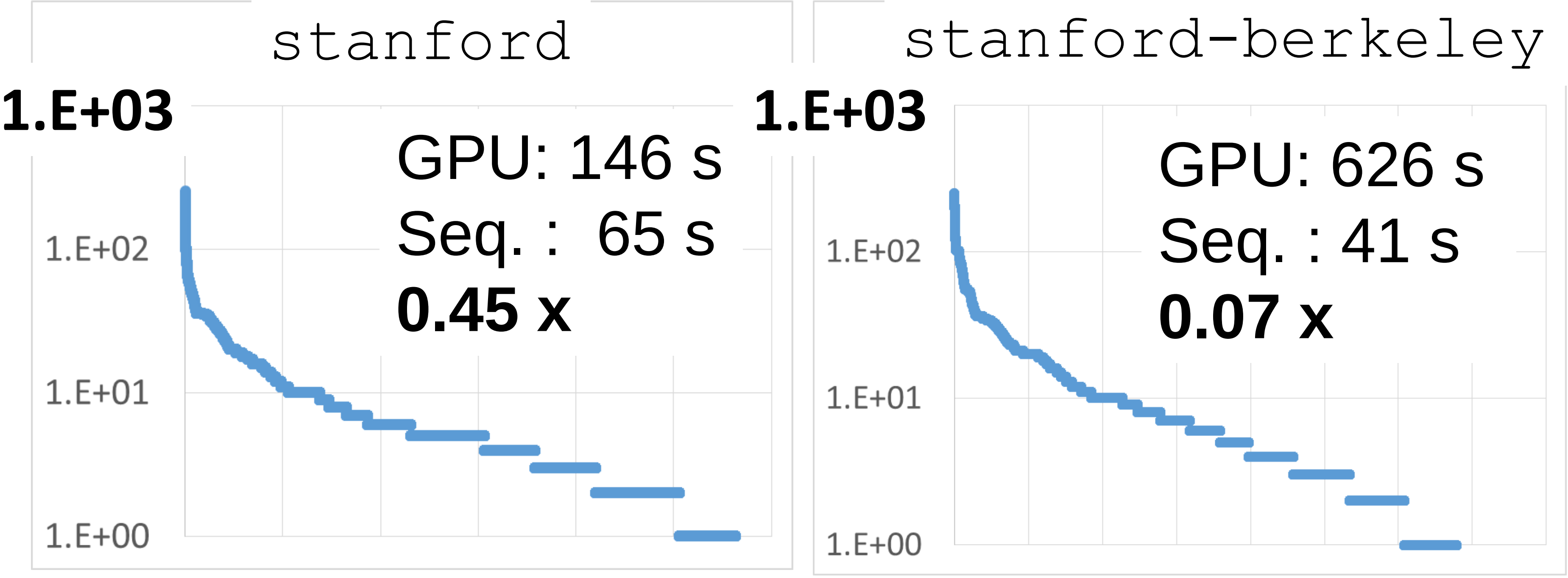


GPU: 50 s
Seq. : 811 s
16 x speed-up

GPU: 59 s
Seq. : 877 s
15 x speed-up

Y-axis: # of nonzeros in columns, descending, log-scale

Results: non-long-tailed distribution



GPU: 146 s
Seq. : 65 s
0.45 x

GPU: 626 s
Seq. : 41 s
0.07 x

Y-axis: # of nonzeros in columns, descending, log-scale

Analysis of results

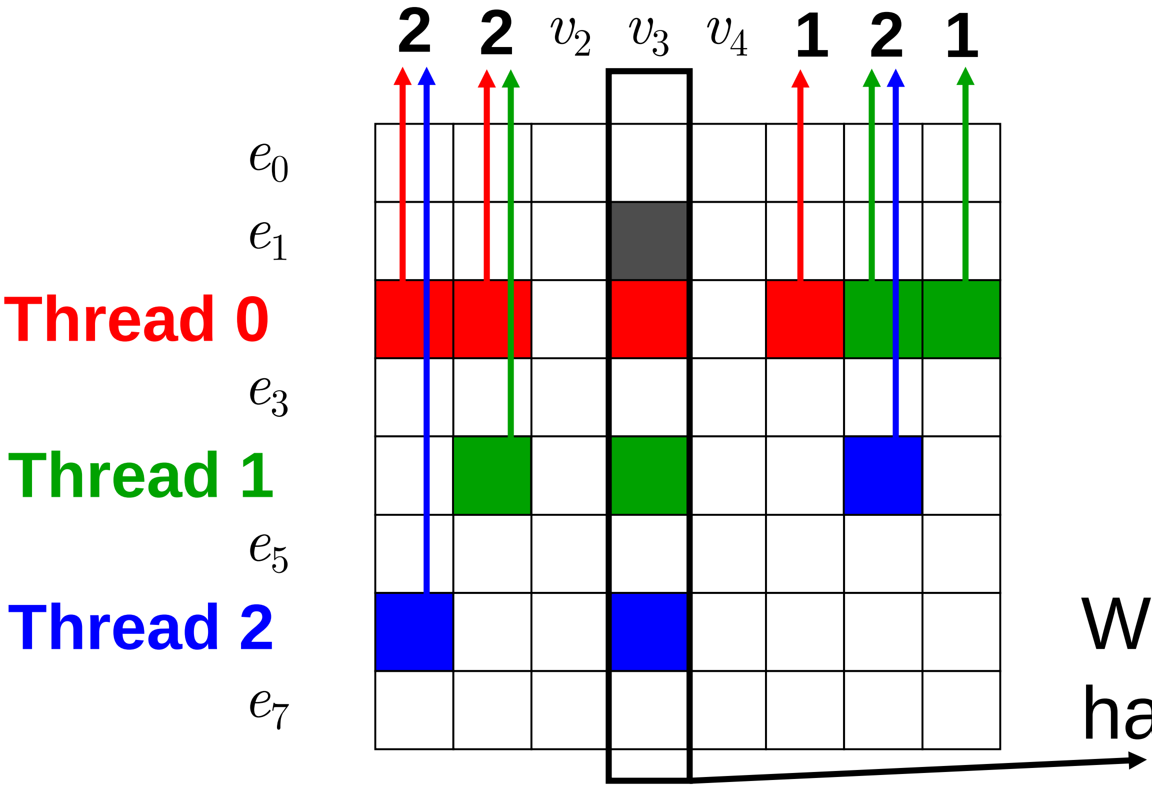
- Good speedup for data with **long-tailed distribution of nonzeros**

Analysis of results

- Good speedup for data with **long-tailed distribution of nonzeros**
- Synthetic data
  - First 1,000 columns : 100,000 nonzeros each
  - Next 699,000 columns : all zero
  - Speedup: 123 x

Analysis of results

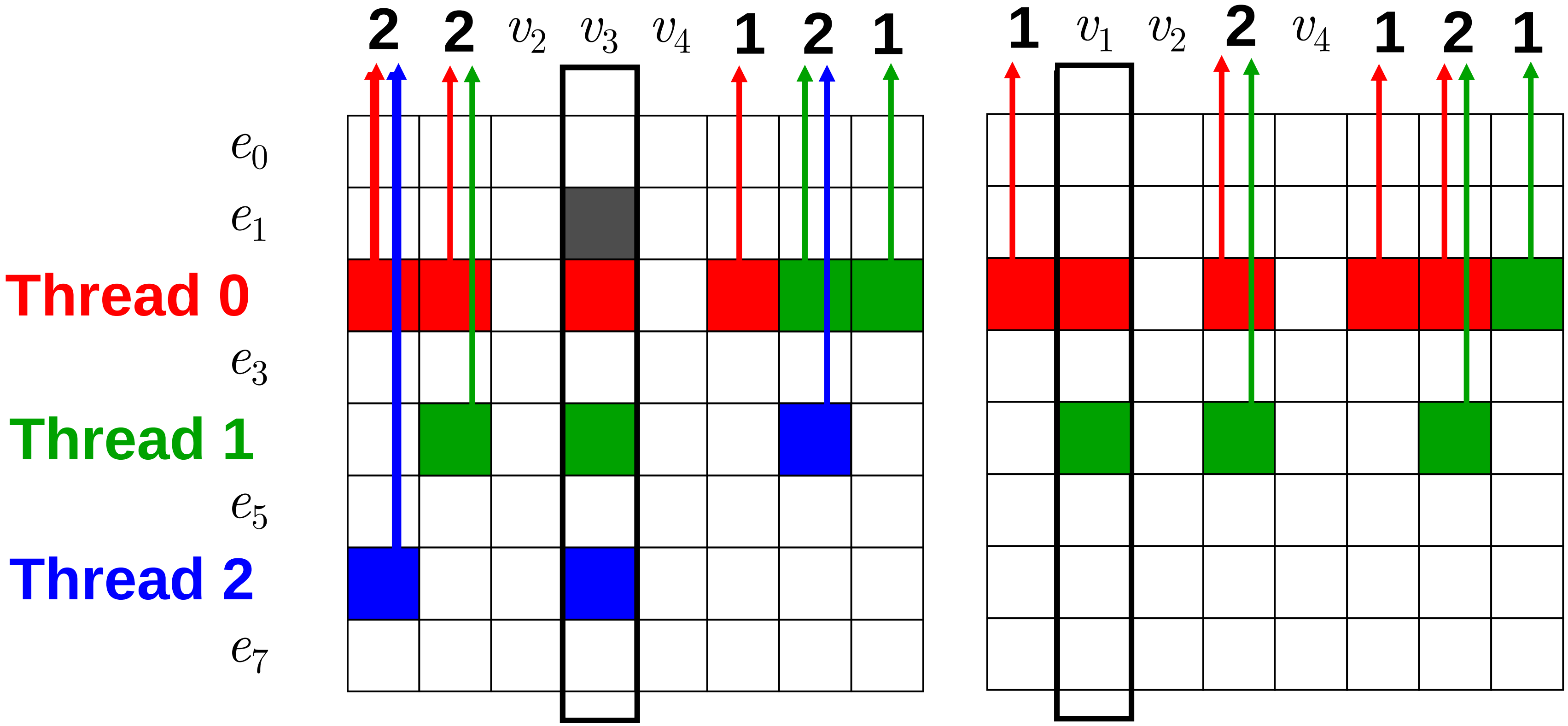
- Most nodes sparsely connected (<< 32 edges)
- Now: one warp, one instance of Mondriaan



What if this column has < 32 nonzeros?

Work in progress

- Pool **multiple instances of Mondriaan** into warp



Conclusion

- Implemented hypergraph algorithm handling arbitrary connectivity patterns
- Explored SHFL for task planning

- Future work: more flexible allocation strategy

Acknowledgment

- Trinity College: Student Research Program
- NVIDIA: CUDA Teaching Center