An Accelerated Procedure for Hypergraph Coarsening on the GPU

Lin Cheng, Hyunsu Cho, and Peter Yoon

Trinity College

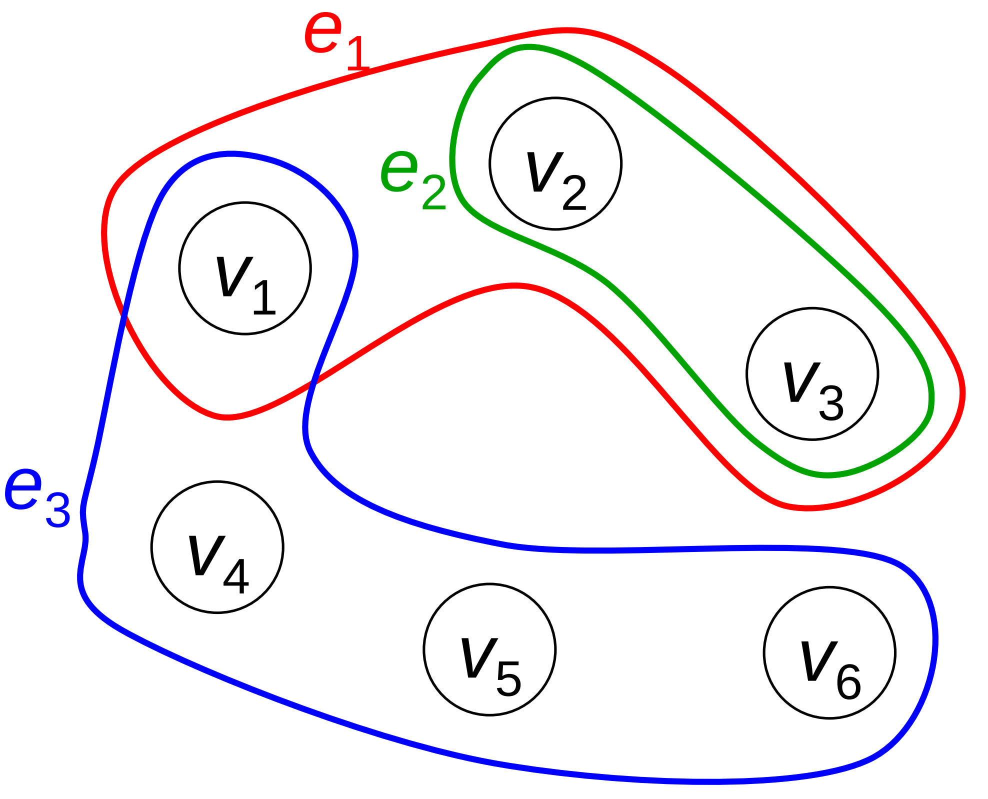
Hartford, CT, USA

Outline

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

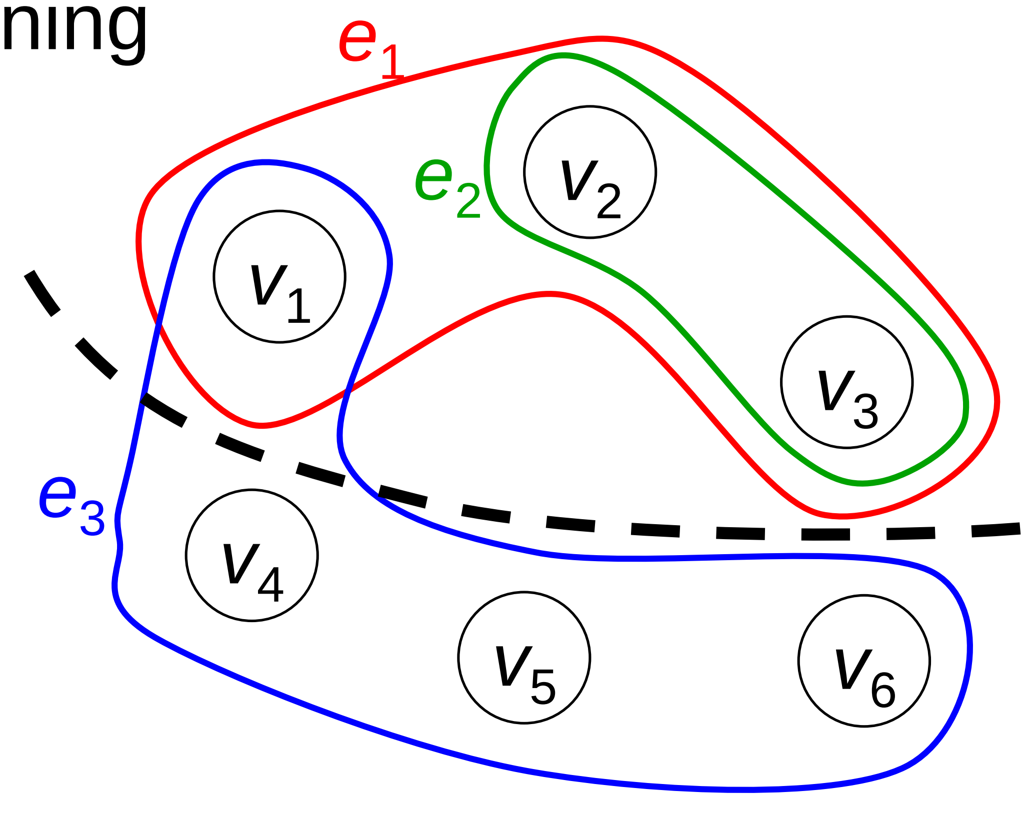
Hypergraph

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



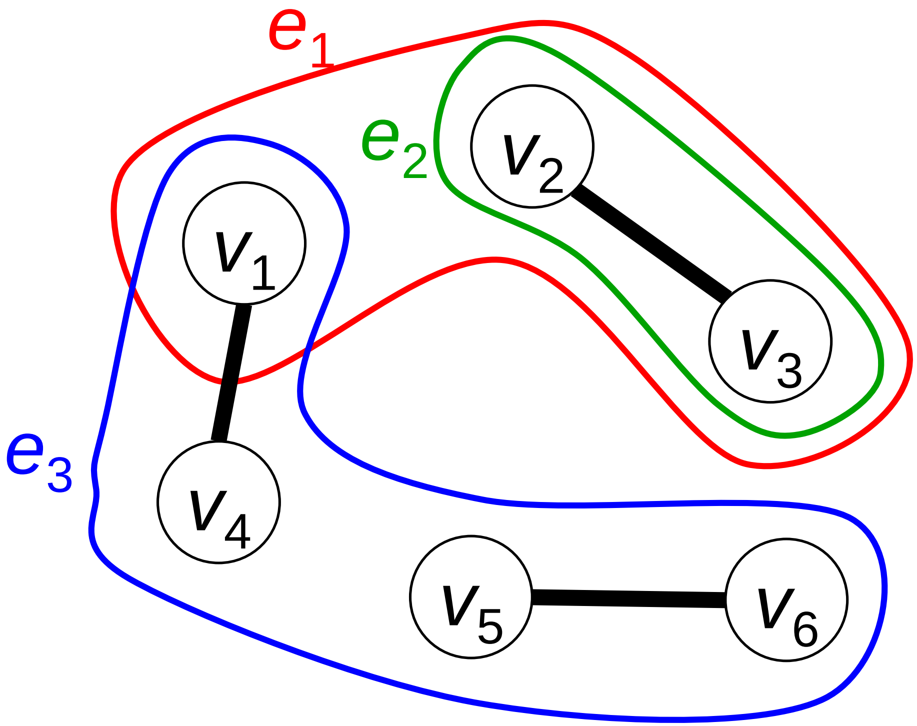
Hypergraph

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



Hypergraph coarsening

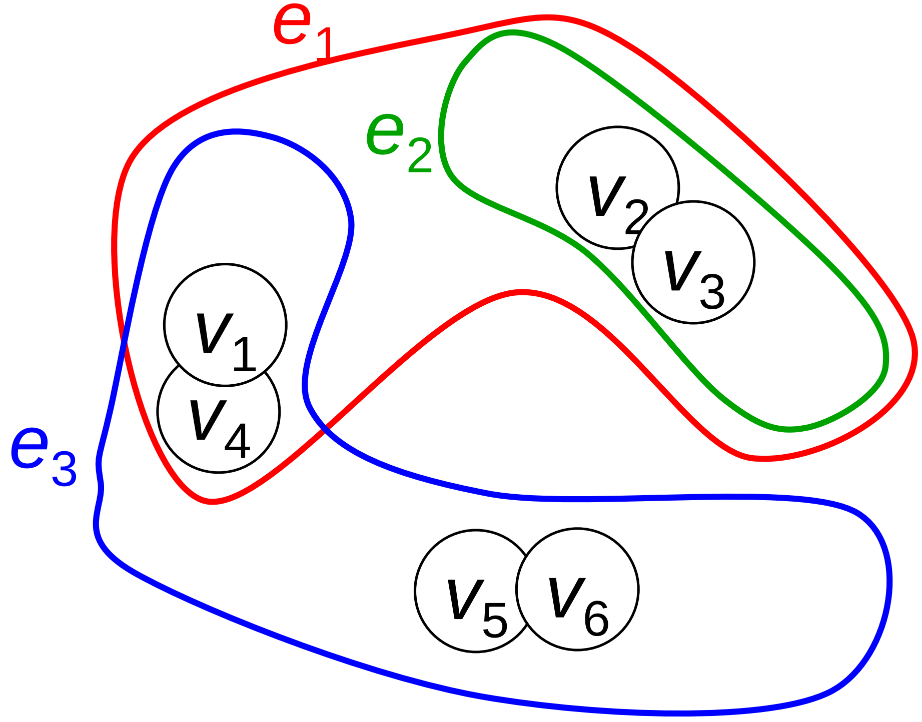
- Heuristic: reduce # nodes by fusing



6 nodes

Hypergraph coarsening

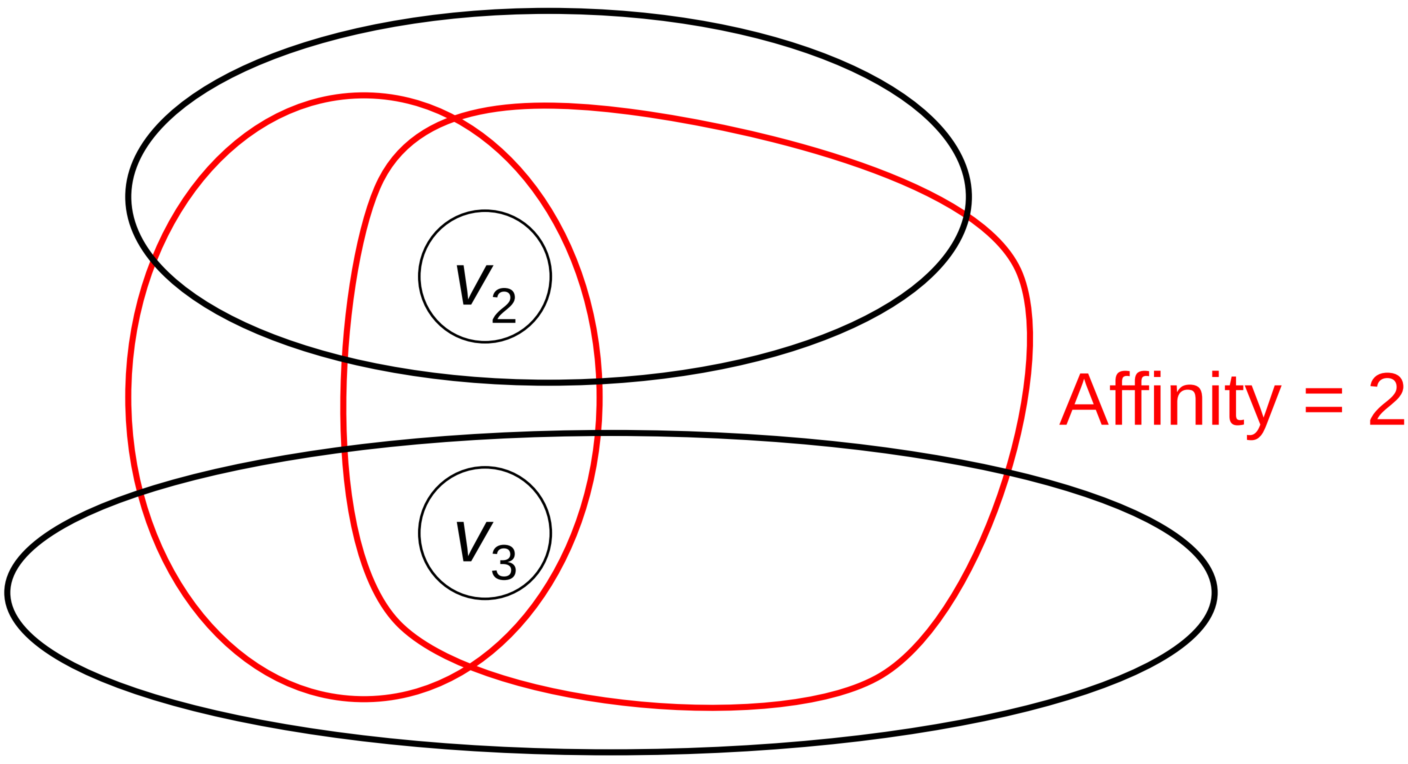
- Heuristic: reduce # nodes by fusing



3 nodes

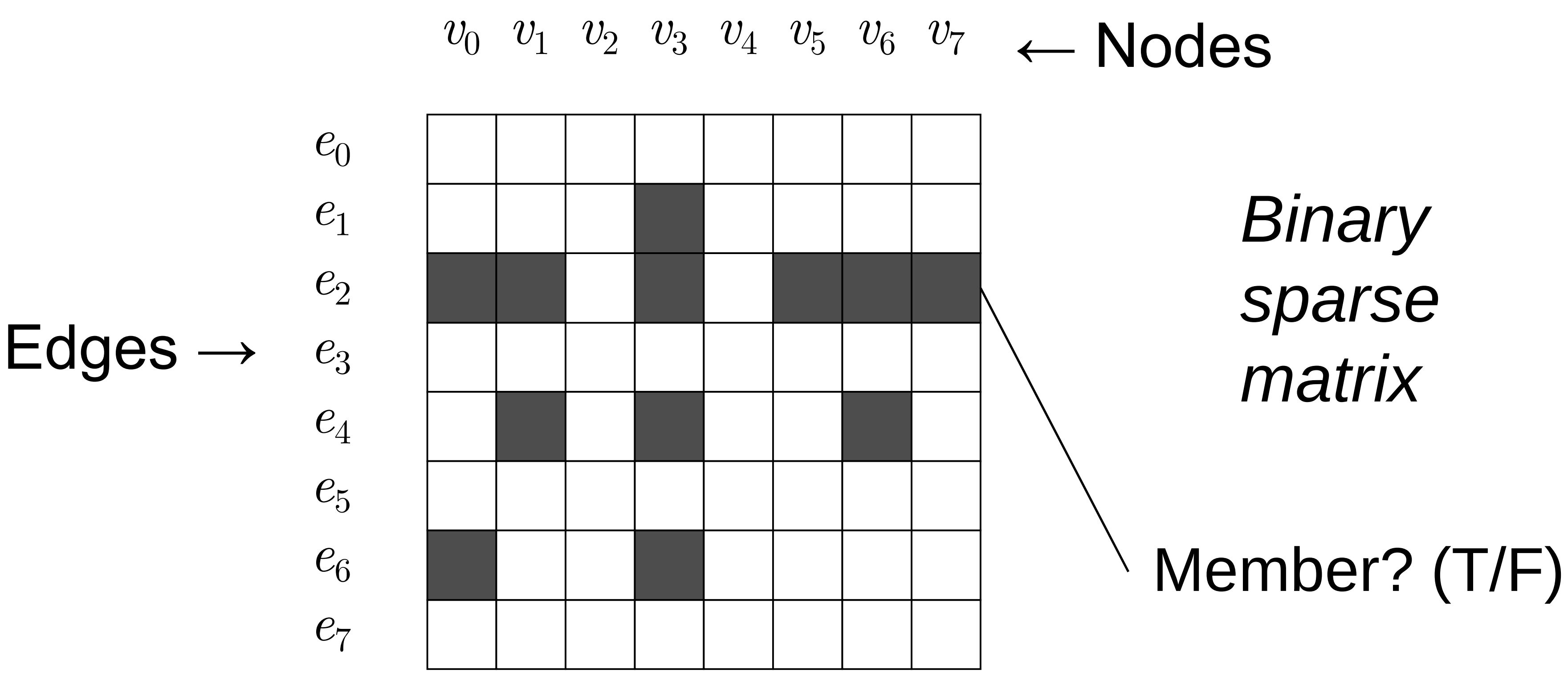
Mondriaan algorithm

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



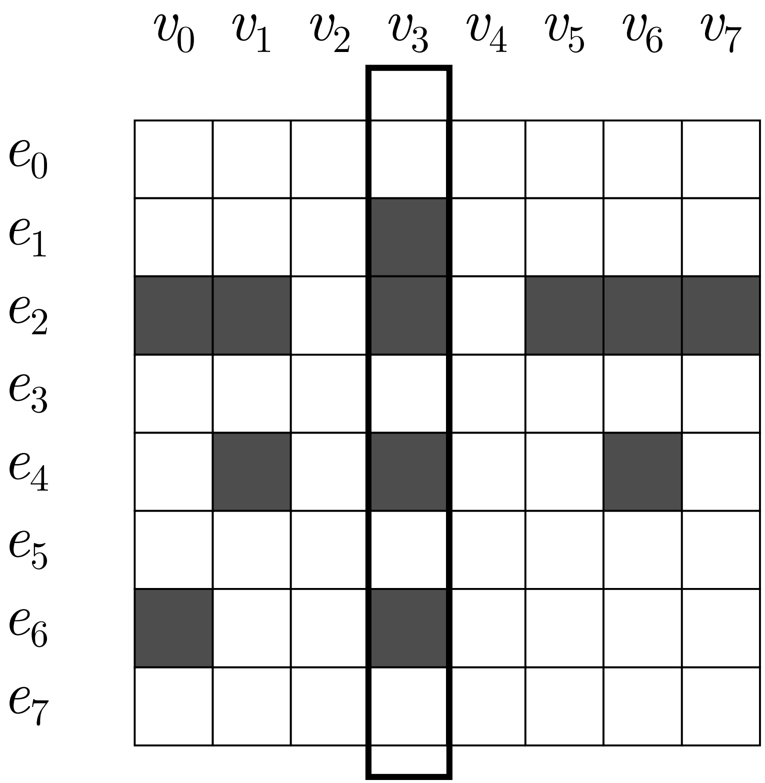
Mondriaan algorithm

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



Mondriaan algorithm

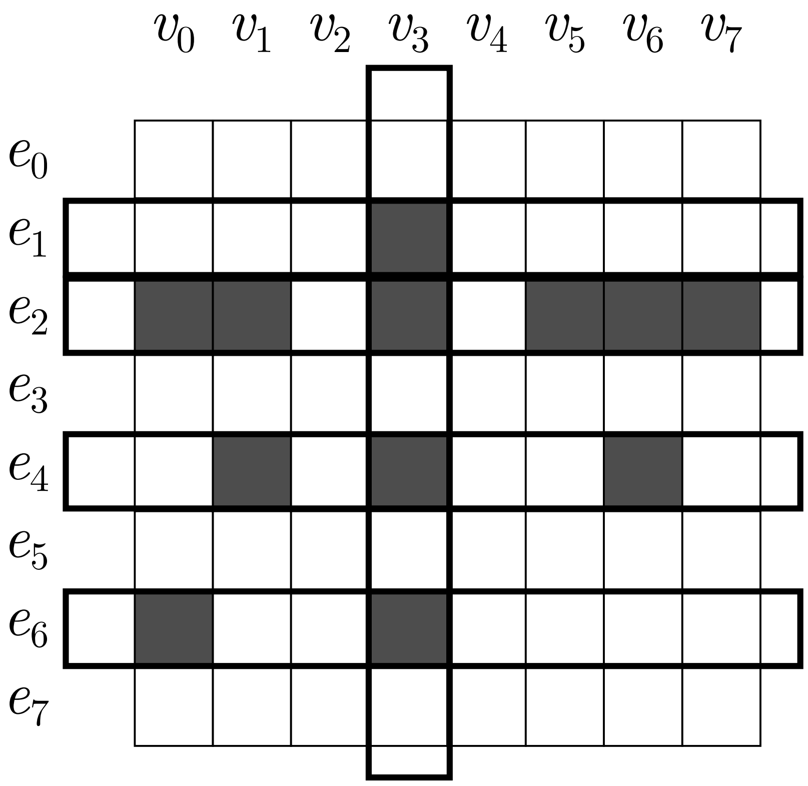
- Given a node, find most “similar” neighbor
- **Similarity = # hyperedges containing both**



1. Find edges containing v_3

Mondriaan algorithm

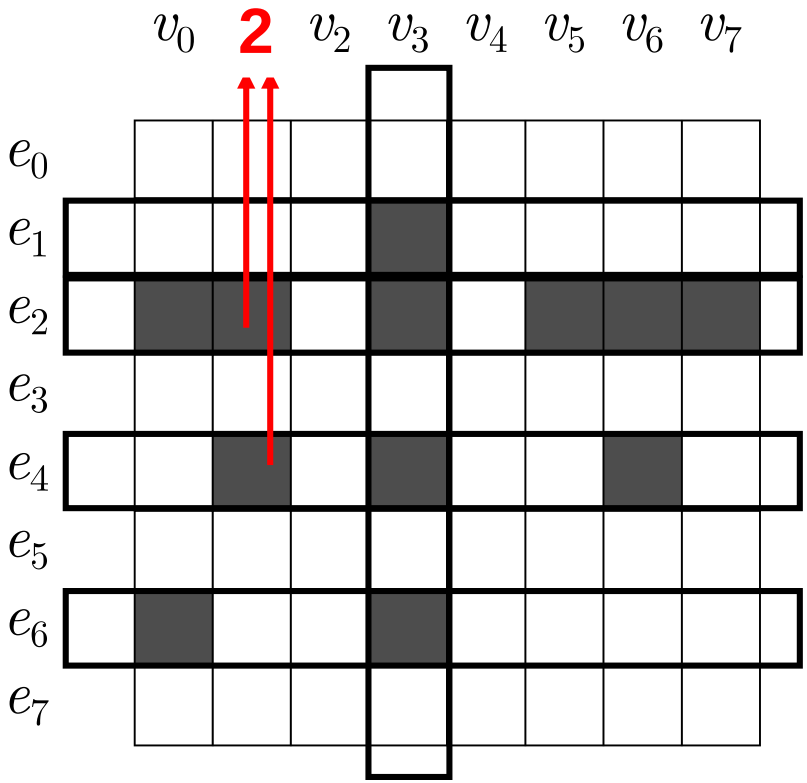
- Given a node, find most “similar” neighbor
- **Similarity = # hyperedges containing both**



1. Find edges containing v_3
2. Collect nonzeros

Mondriaan algorithm

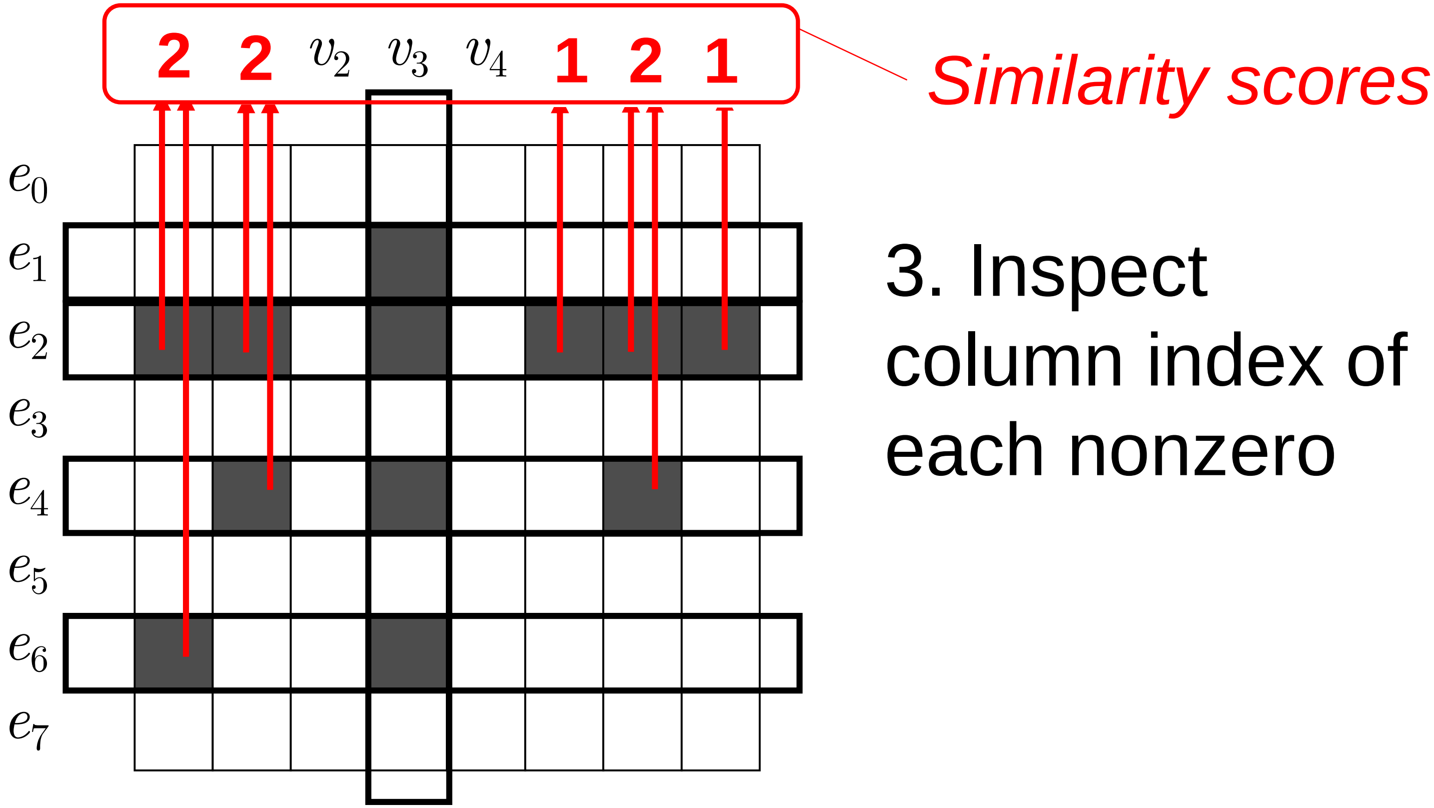
- Given a node, find most “similar” neighbor
- **Similarity = # hyperedges containing both**



3. Inspect column index of each nonzero

Mondriaan algorithm

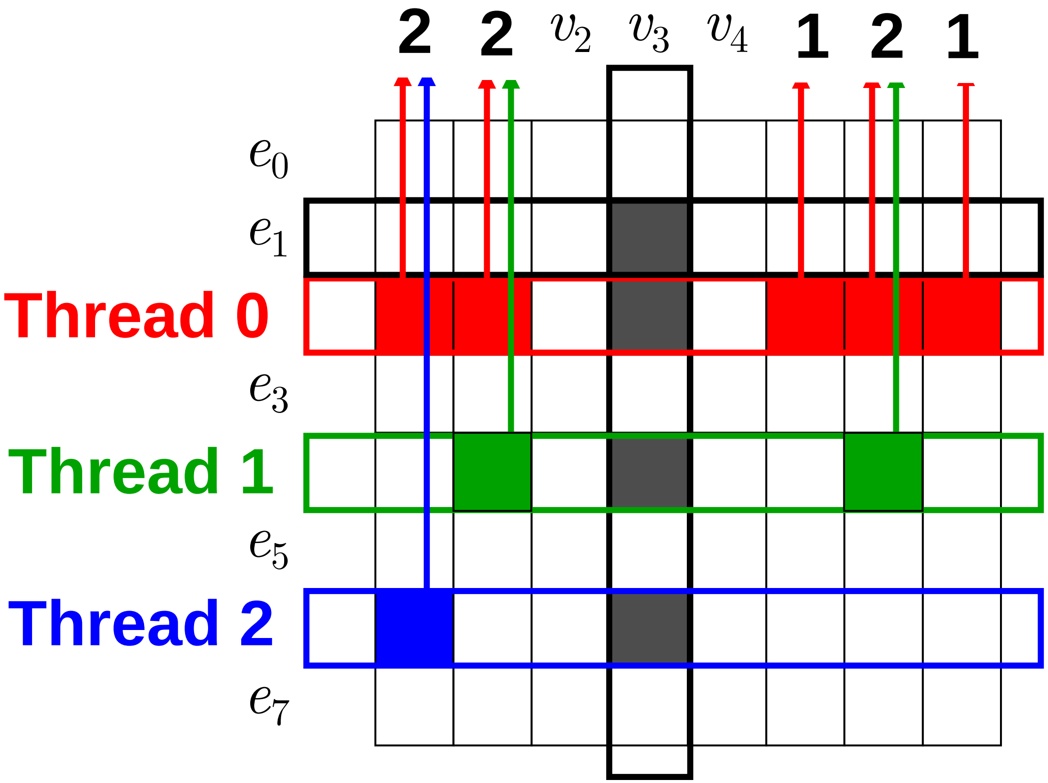
- Given a node, find most “similar” neighbor
- **Similarity = # hyperedges containing both**



3. Inspect column index of each nonzero

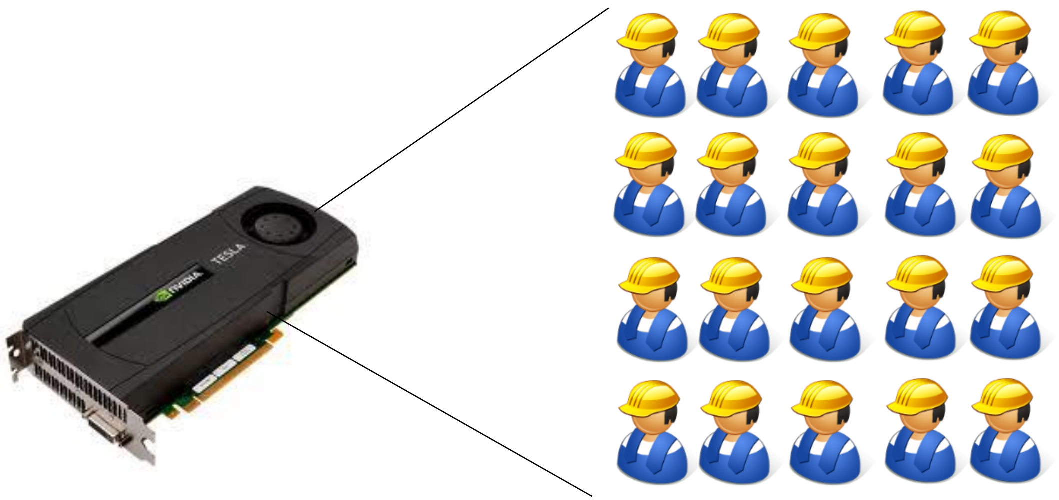
Mondriaan algorithm

- Parallel algorithm:
  Inspect edges in parallel



GPU as parallel accelerator

- General-purpose computation
- Massively parallel; many-core

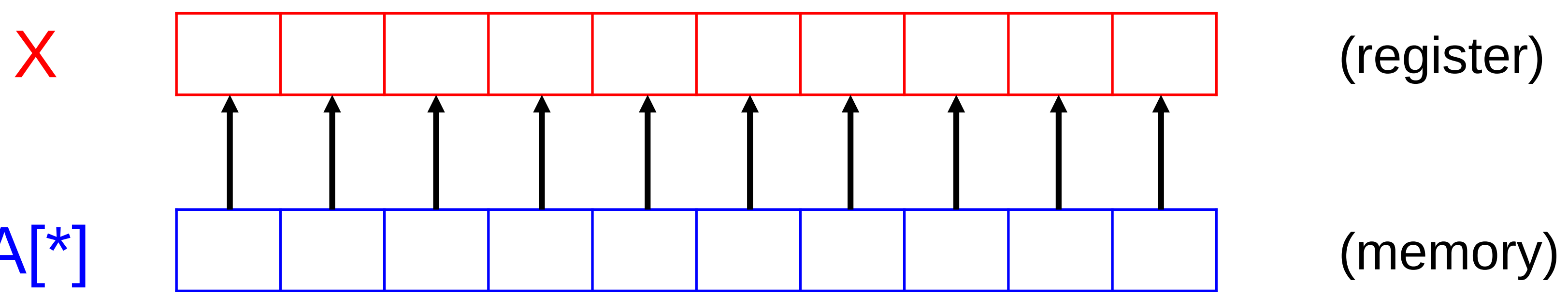


GPU as parallel accelerator

- **SIMD** (Single-Instruction-Multiple-Data)
- NVIDIA GPUs : organized in **warps**

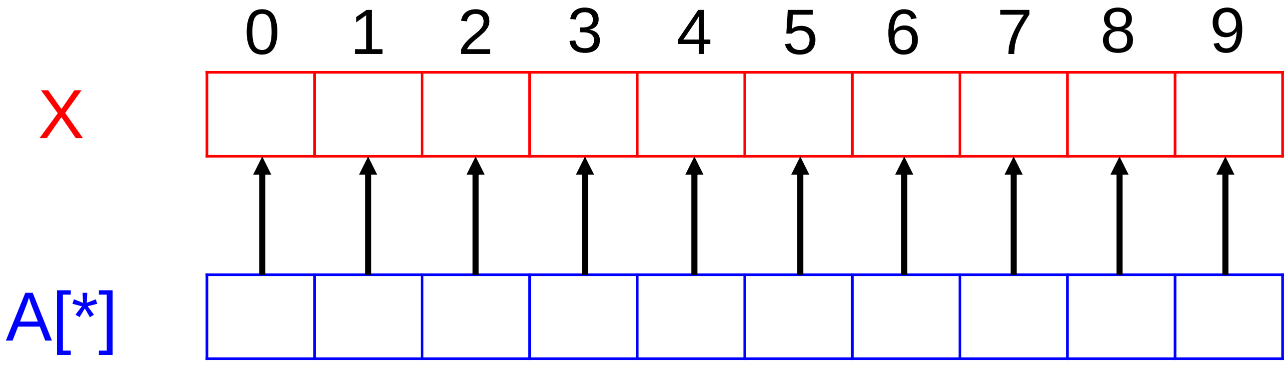
  32 threads share one instruction counter

LOAD A[*] INTO X



Warp divergence

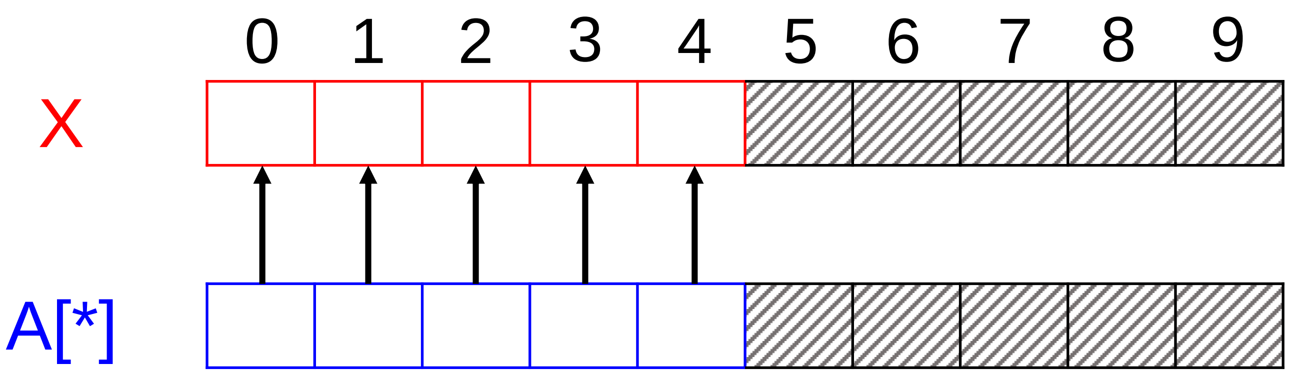
Thread 0-9: LOAD A[0-9] INTO X



Warp divergence

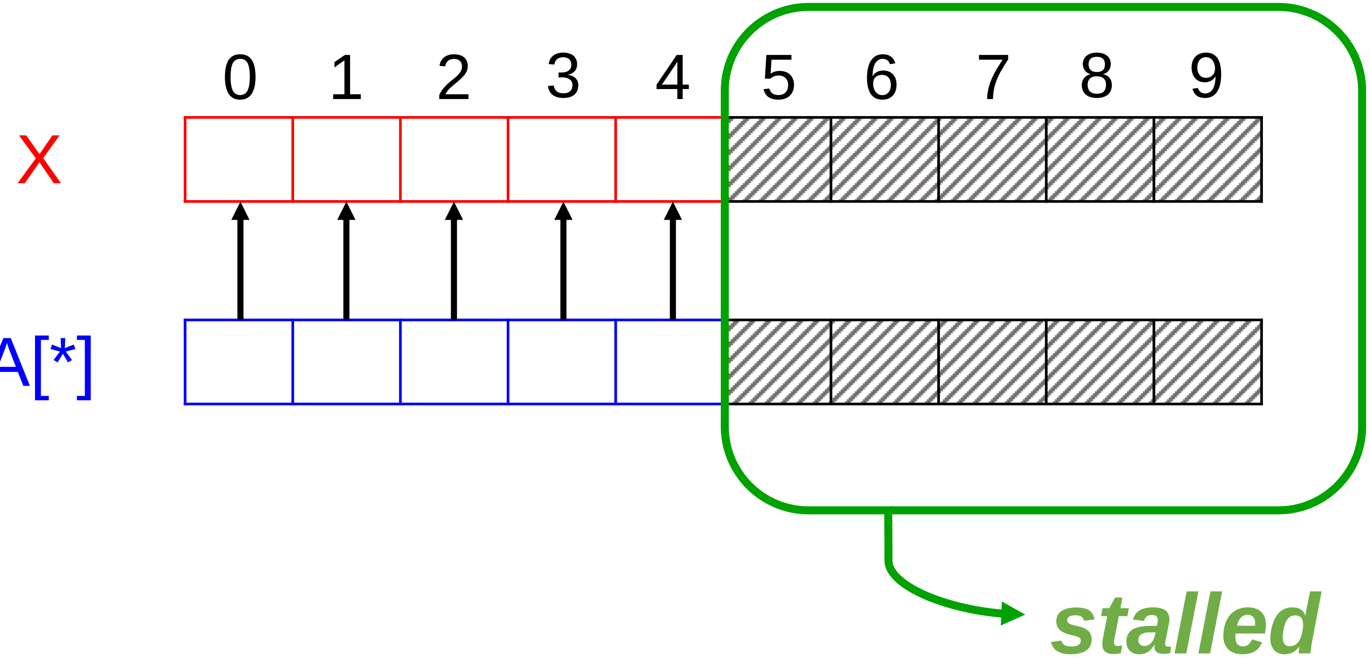
Thread 0-4: LOAD A[0-4] INTO X

Thread 5-9: NONE



Warp divergence

Thread 0-4: LOAD A[0-4] INTO X
Thread 5-9: NONE

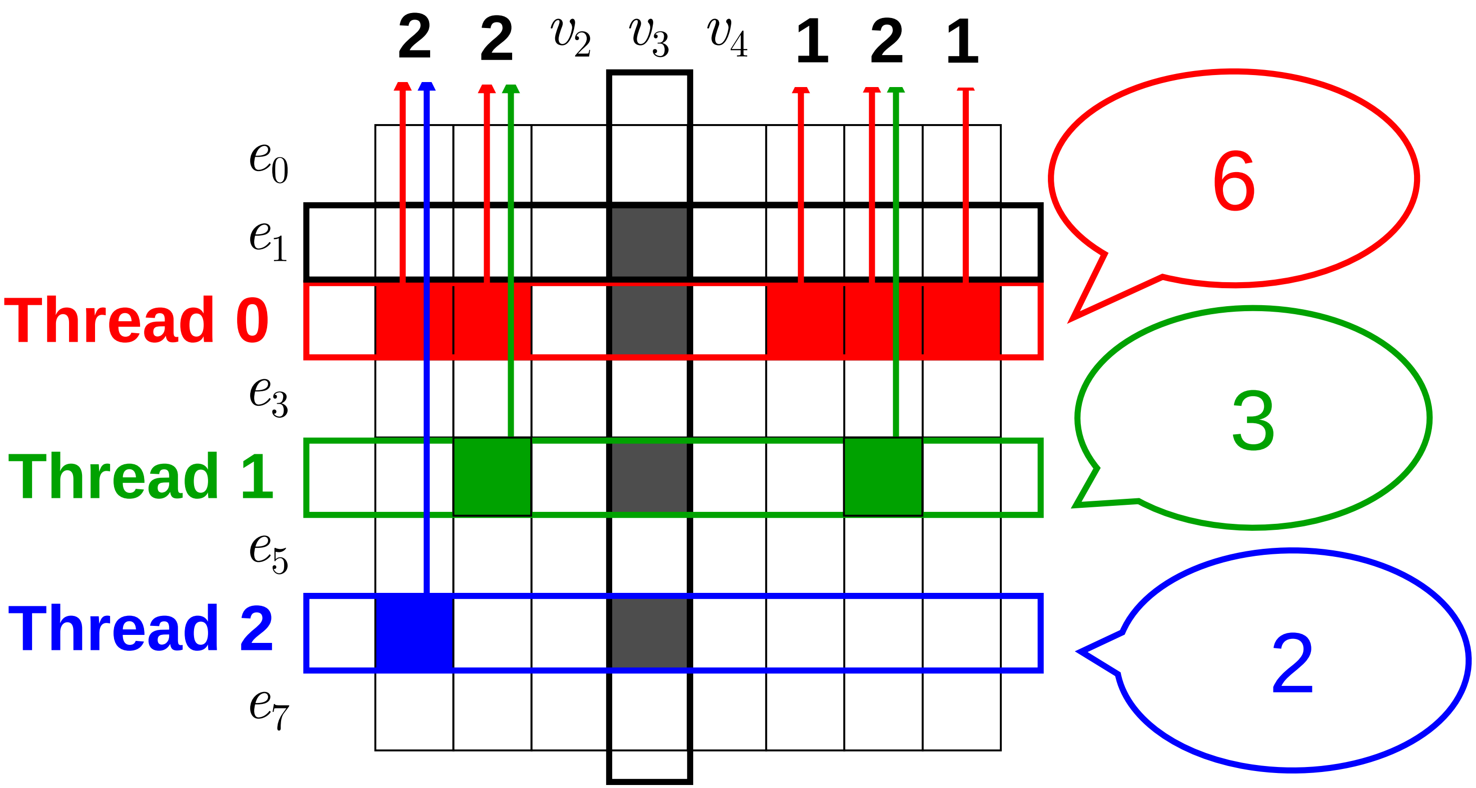


Warp divergence

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

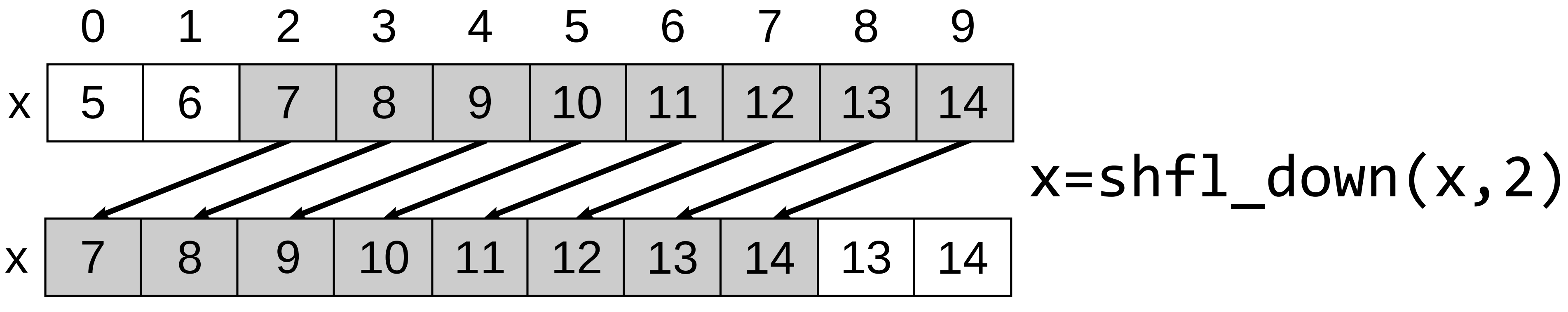
Mondriaan algorithm

- A naïve strategy results in load imbalance



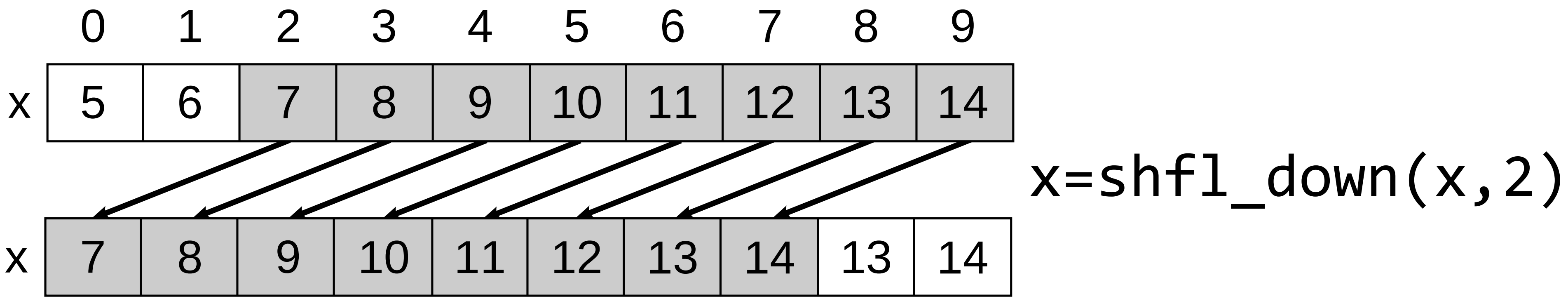
SHFL to the rescue

- Compiler primitive
- Shuffles content of adjacent registers



SHFL to the rescue

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



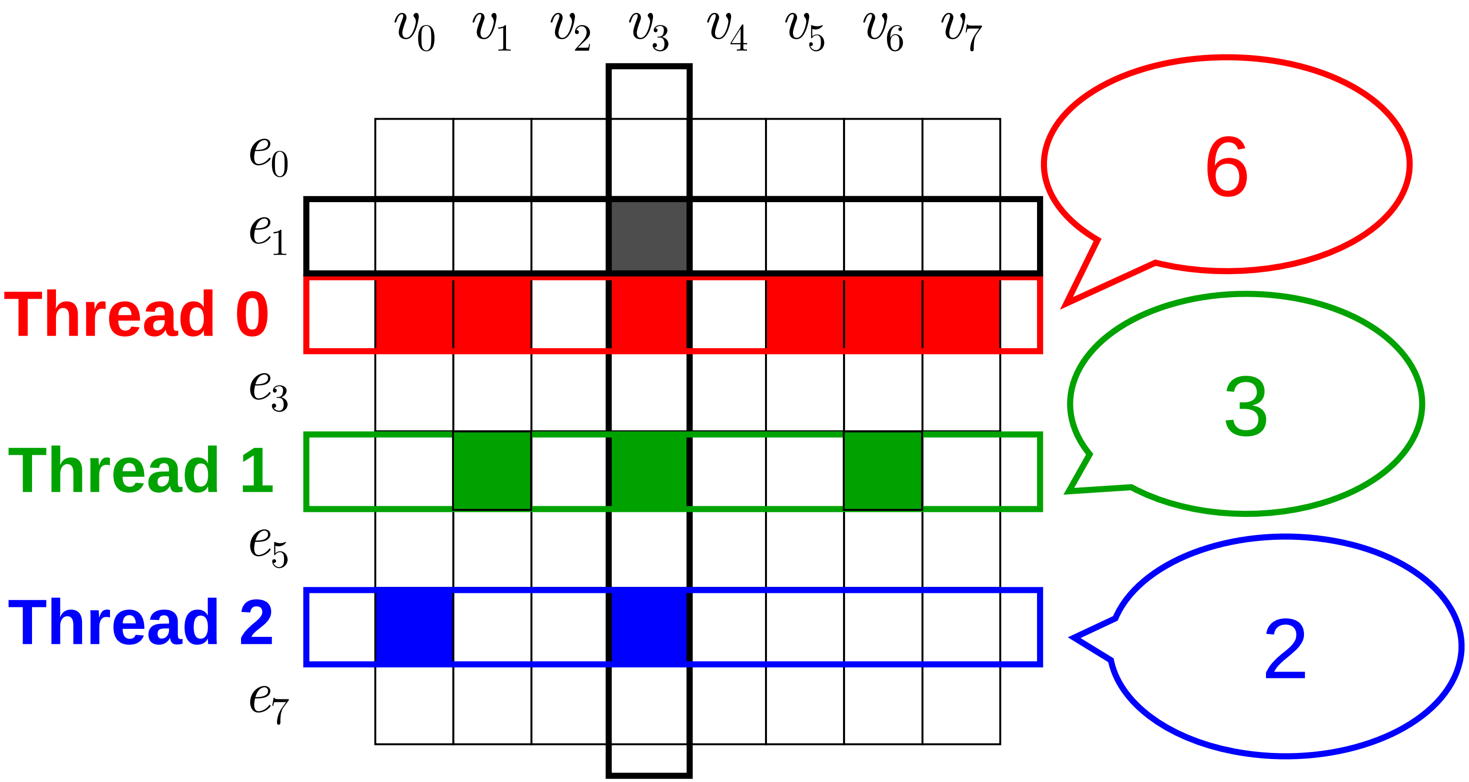
SHFL to the rescue

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

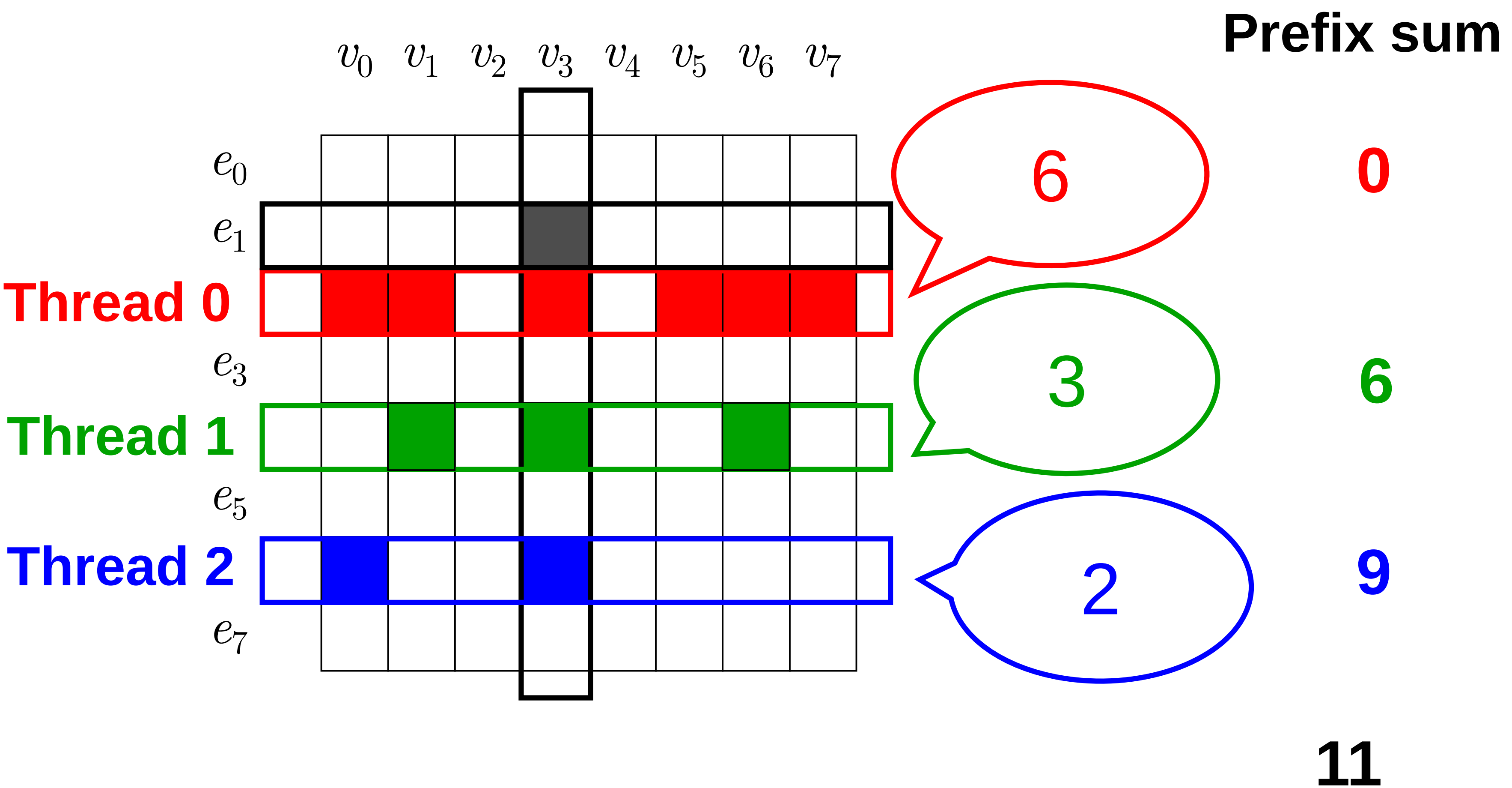
- CUB Library: http://nvlabs.github.io/cub/

  warp-wide reduction, prefix sum

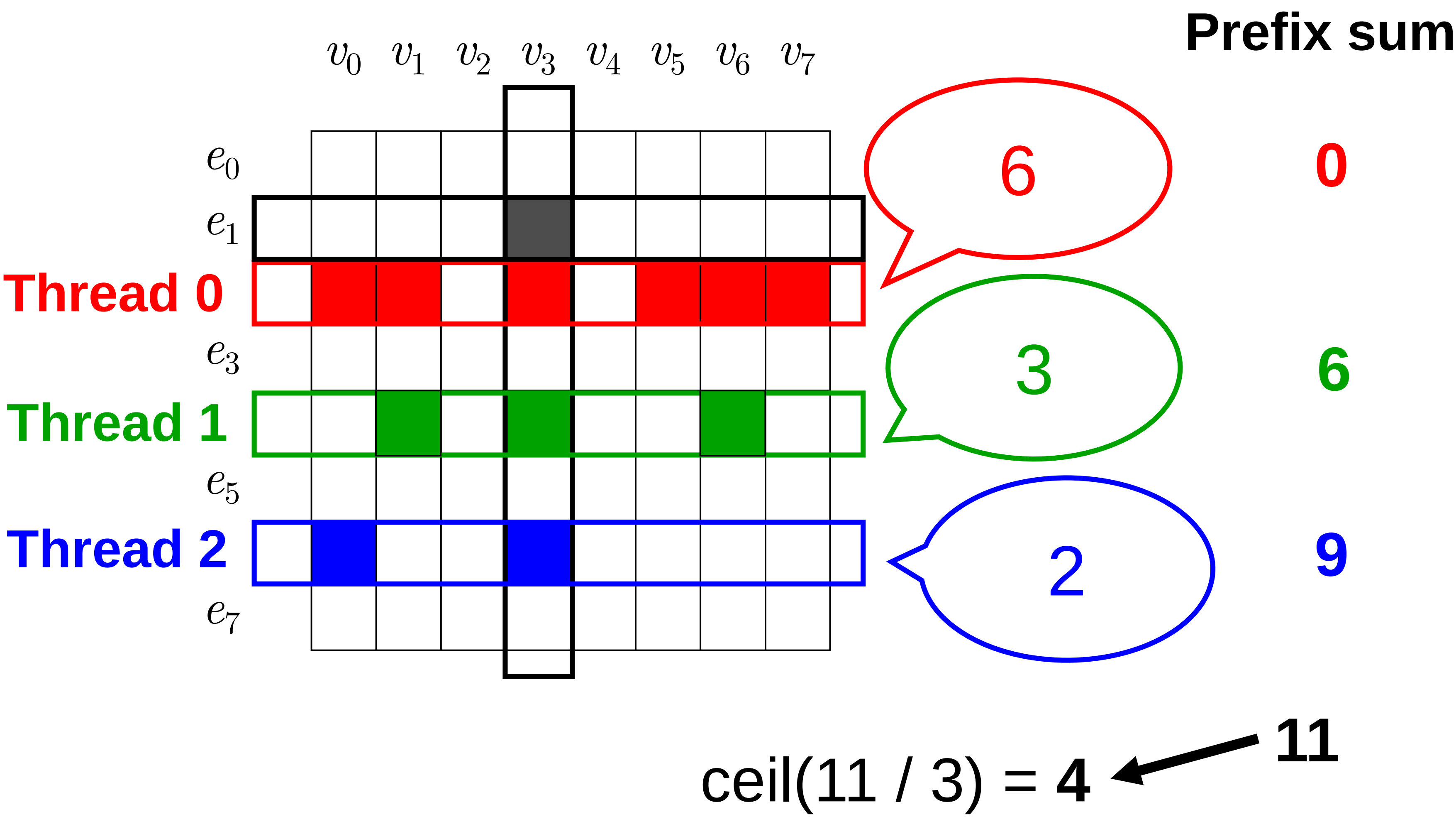
Runtime planning with SHFL



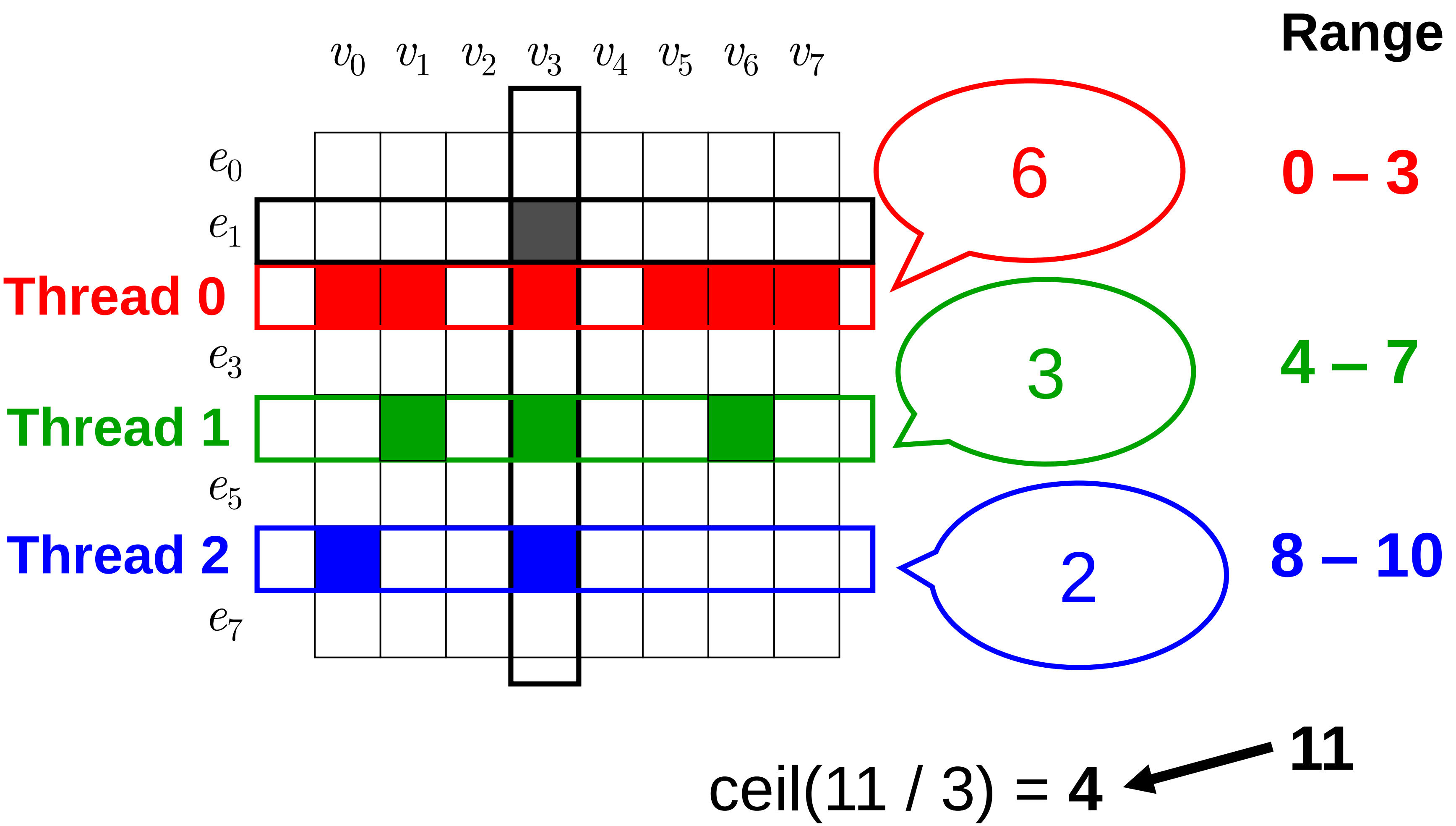
Runtime planning with SHFL



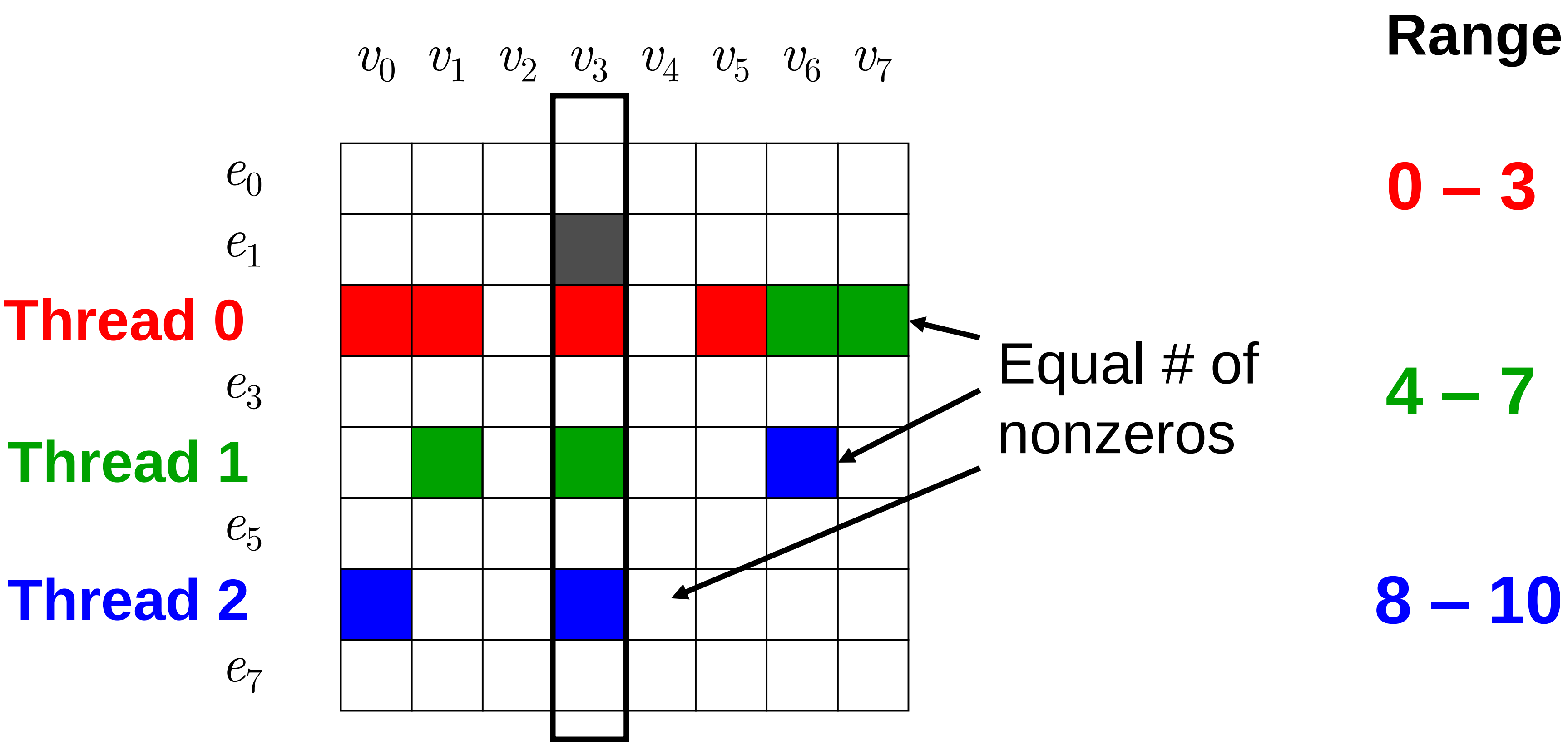
Runtime planning with SHFL



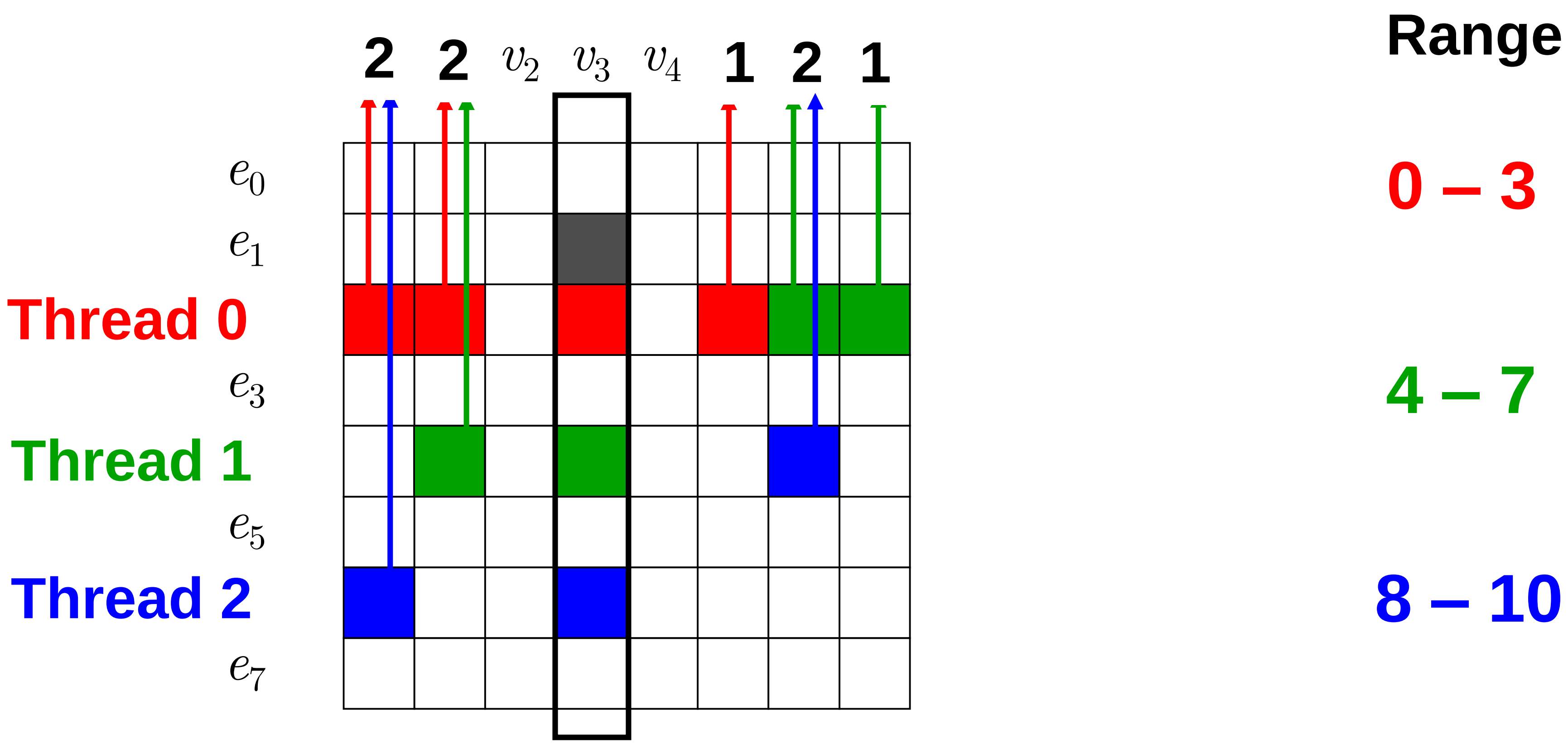
Runtime planning with SHFL



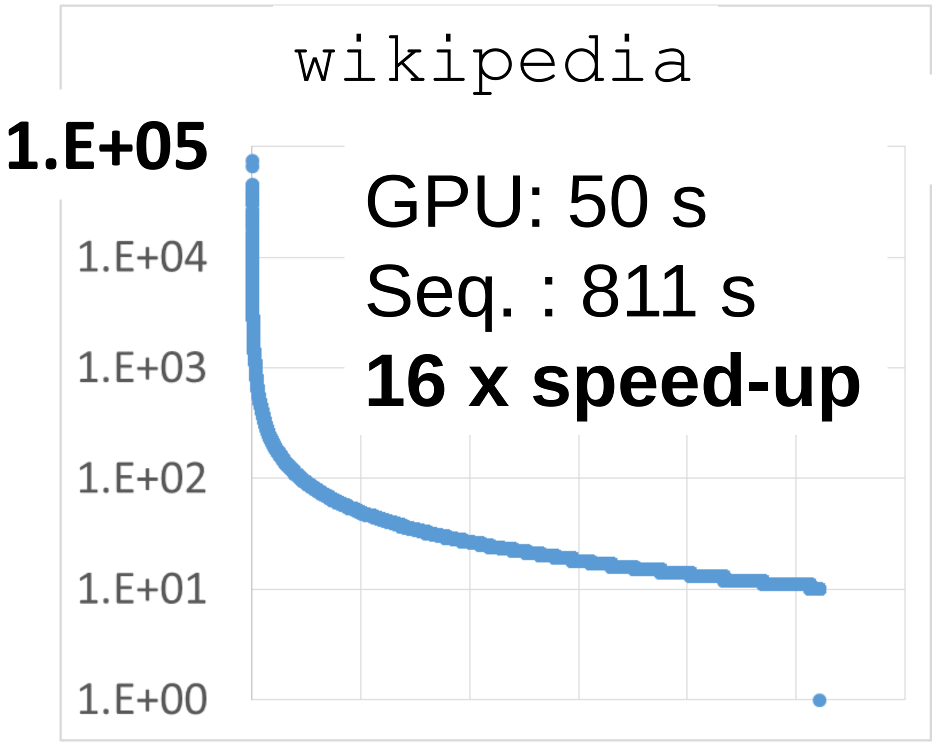
Runtime planning with SHFL

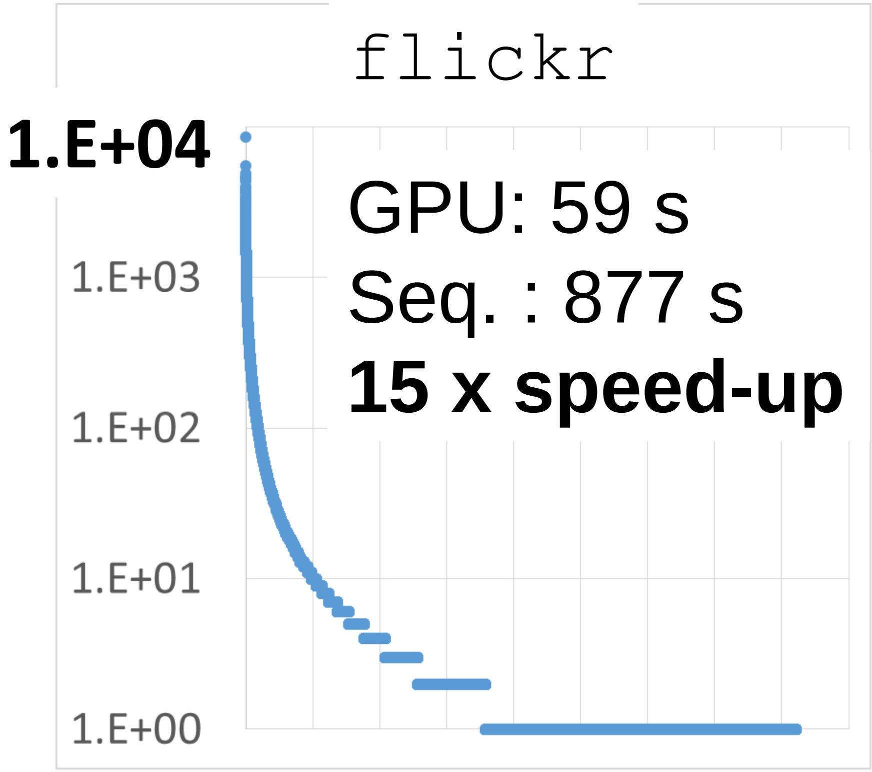


Runtime planning with SHFL



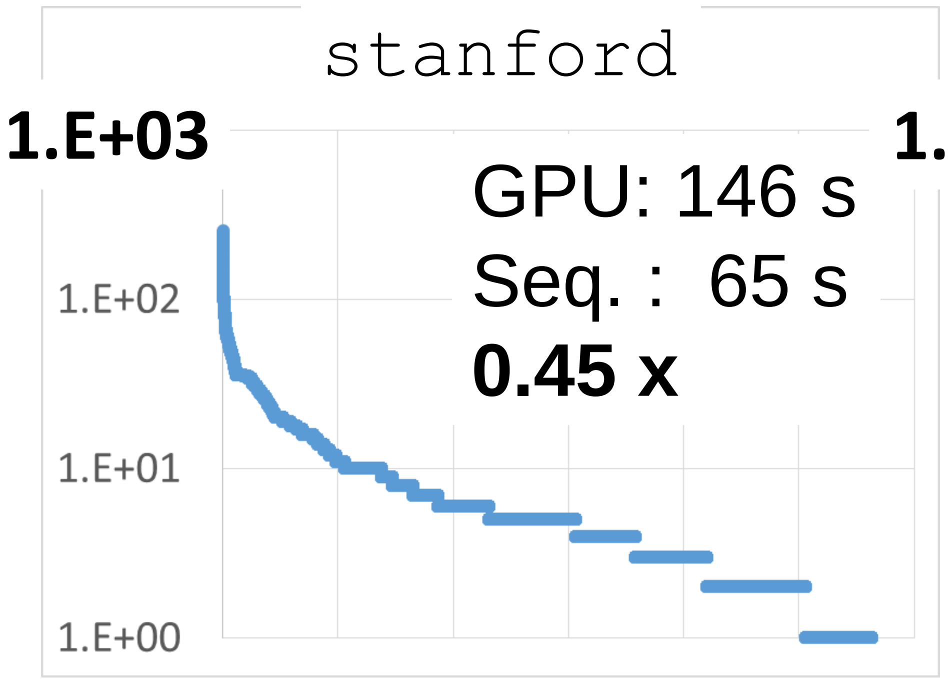
Results: long-tailed distribution

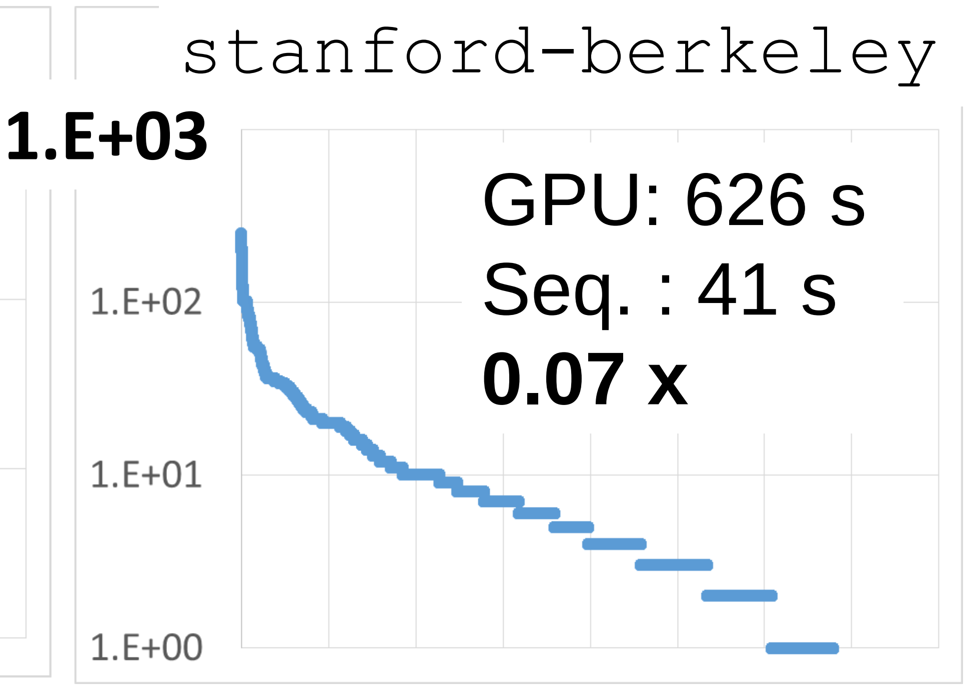




Y-axis: # of nonzeros in columns, descending, log-scale

Results: non-long-tailed distribution





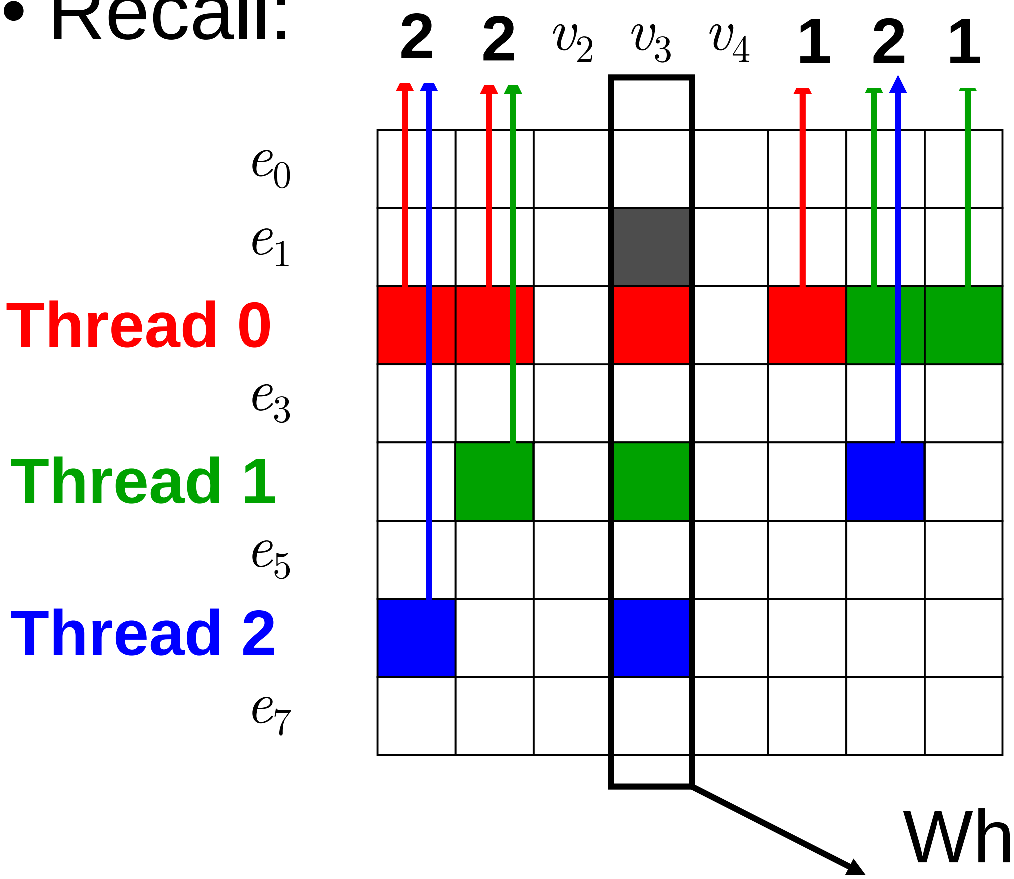
Y-axis: # of nonzeros in columns, descending, log-scale

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

- Recall:



What if this column has < 32 nonzeros?

Conclusion

- Novel technique for task planning using SHFL
- Implementation of hypergraph algorithm handling arbitrary connectivity patterns

Acknowledgment

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