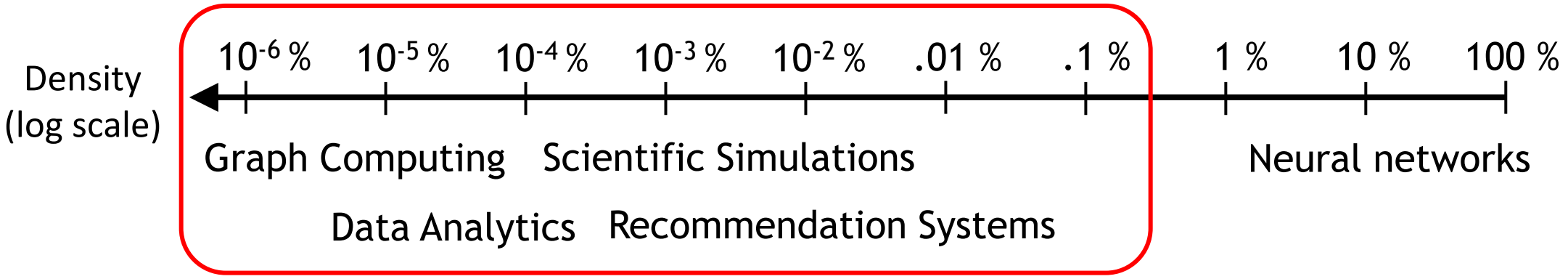
Tailors:
Accelerating Sparse Tensor Algebra by Overbooking Buffer Capacity

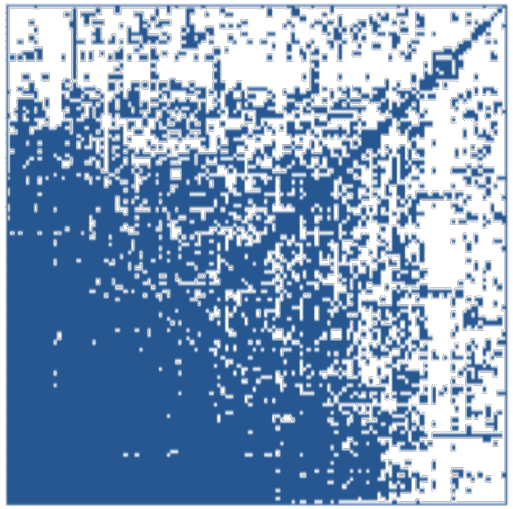
Zi Yu (Fisher) Xue[1], Yannan Nellie Wu[1], Joel S. Emer[1,2], Vivienne Sze[1]

[1]MIT [2]NVIDIA

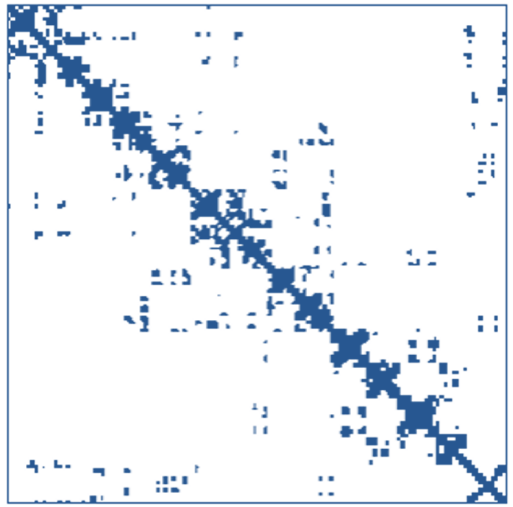
http://emze.csail.mit.edu/tailors

Many applications use highly sparse tensors





367K nonzeros



4.34M nonzeros

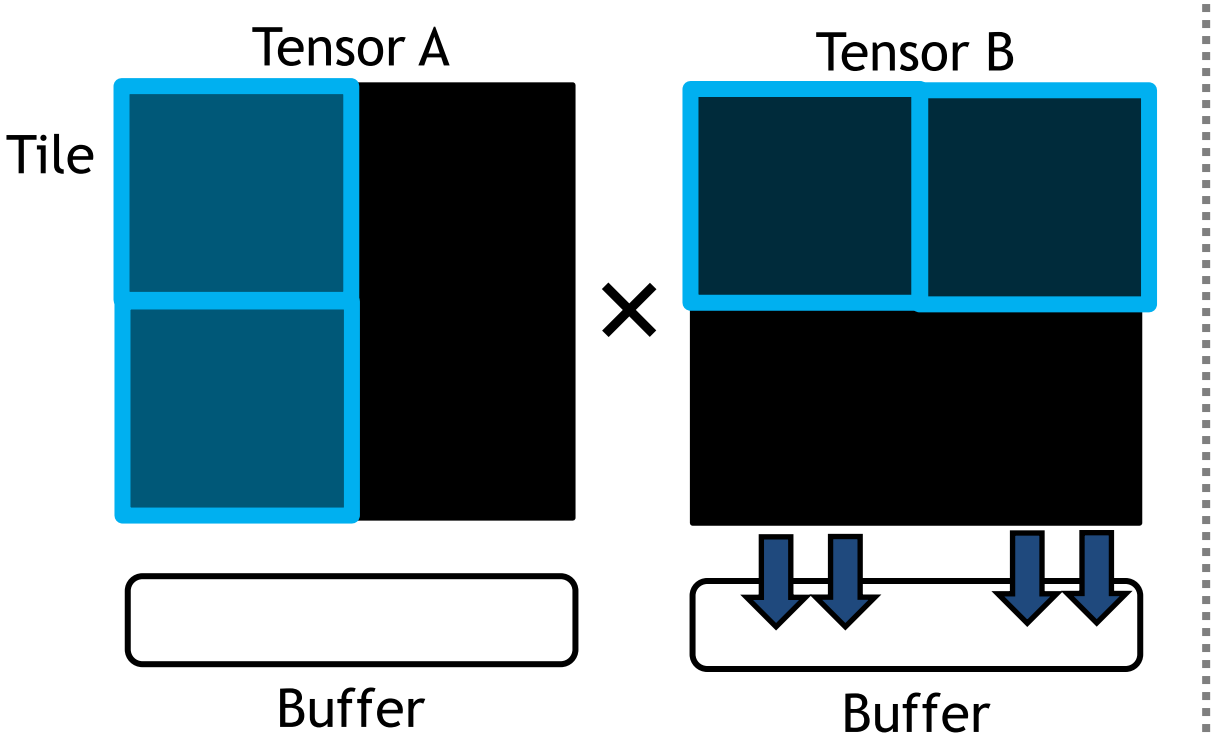
3.20M nonzeros

Zero Values

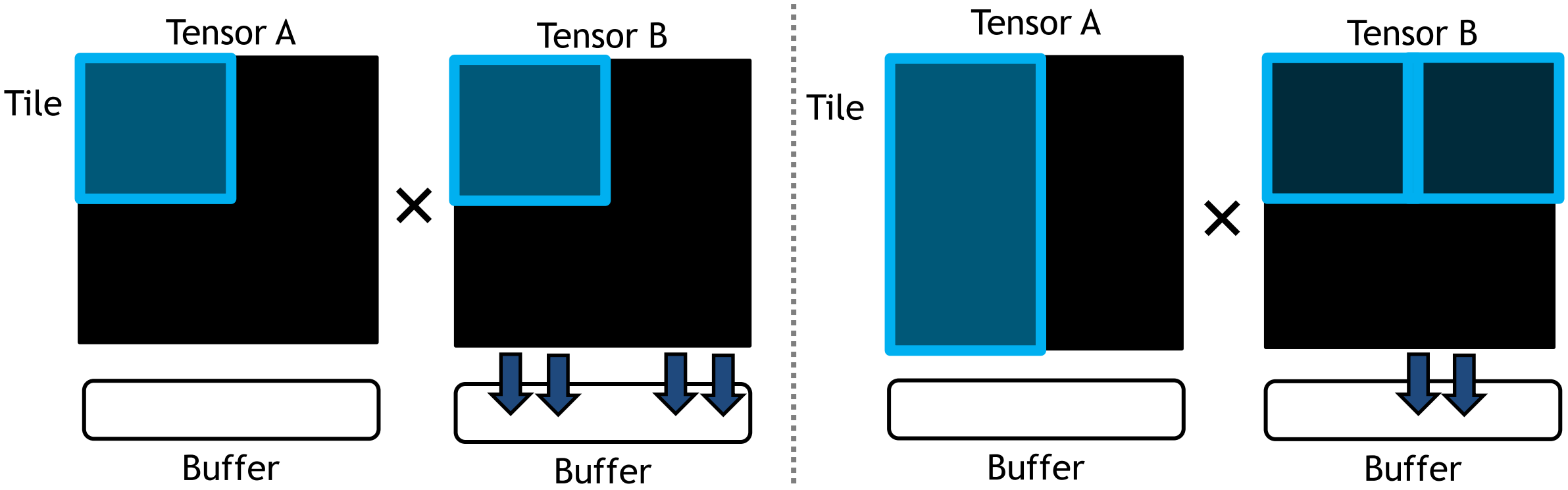
Nonzero Values

[**Hegde**, *MICRO 2019*]

Tiling reduces loads of nonstationary tiles

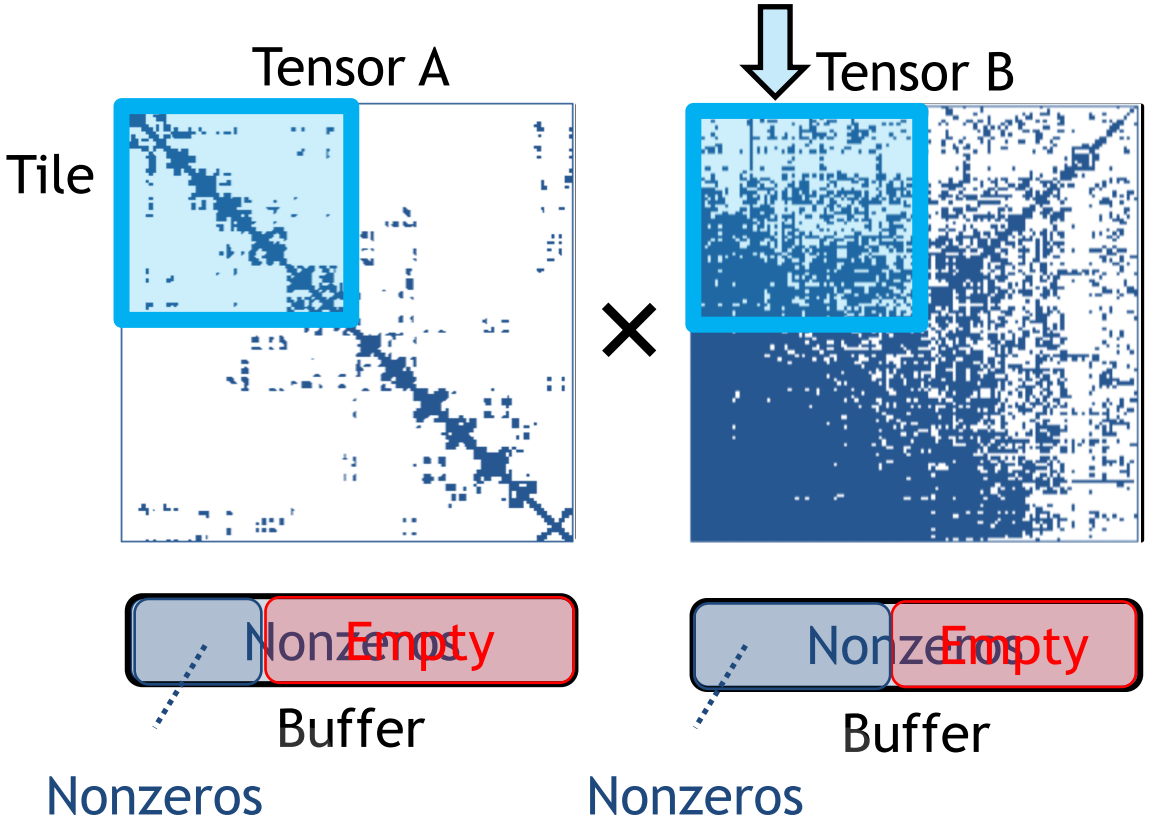


Tiling reduces loads of nonstationary tiles

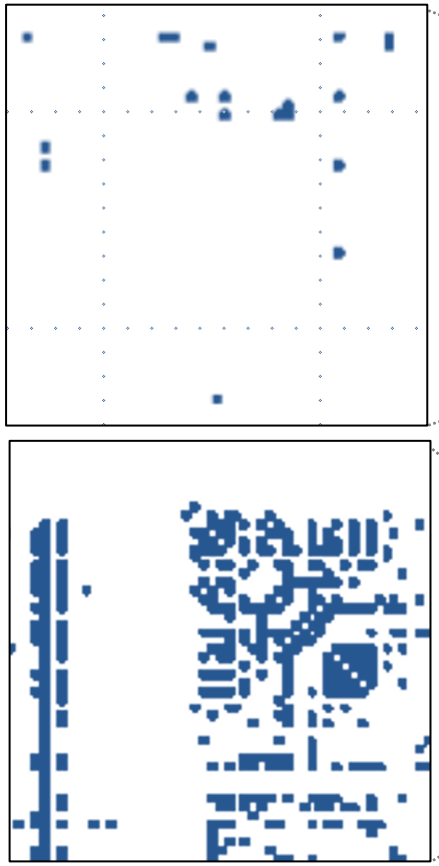


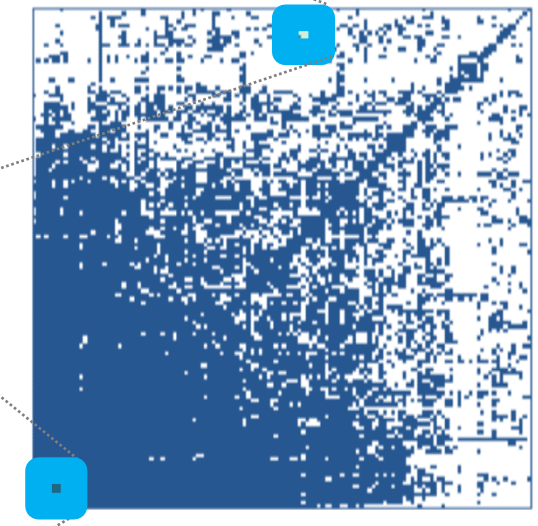
Larger tile sizes → greater data reuse → less DRAM traffic

Tiling sparse tensors is challenging

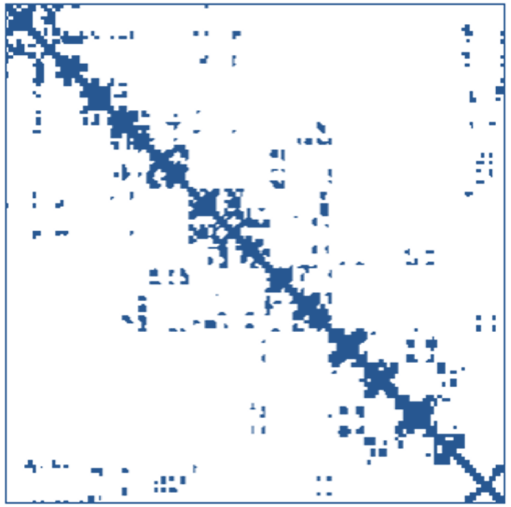


Tensors vary in the *distribution* of sparsity





email-Enron

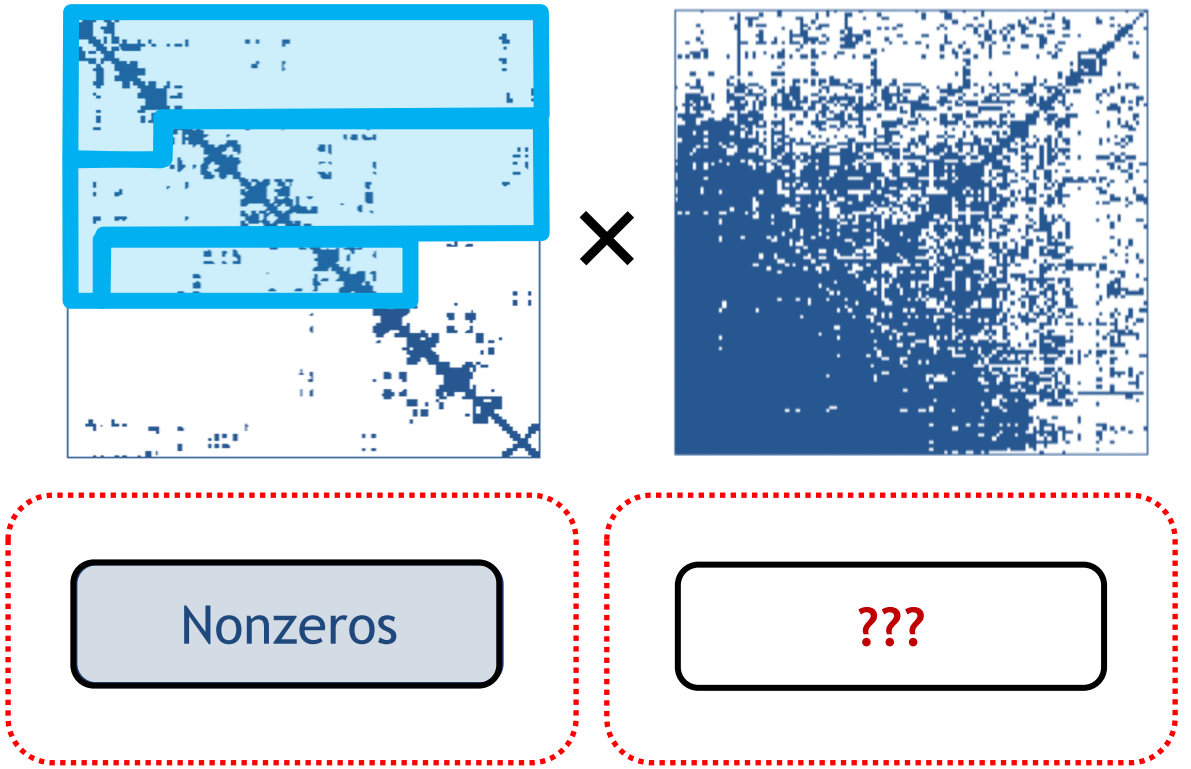


pdb1HYS

amazon0312

Tiling with *uniform occupancy*

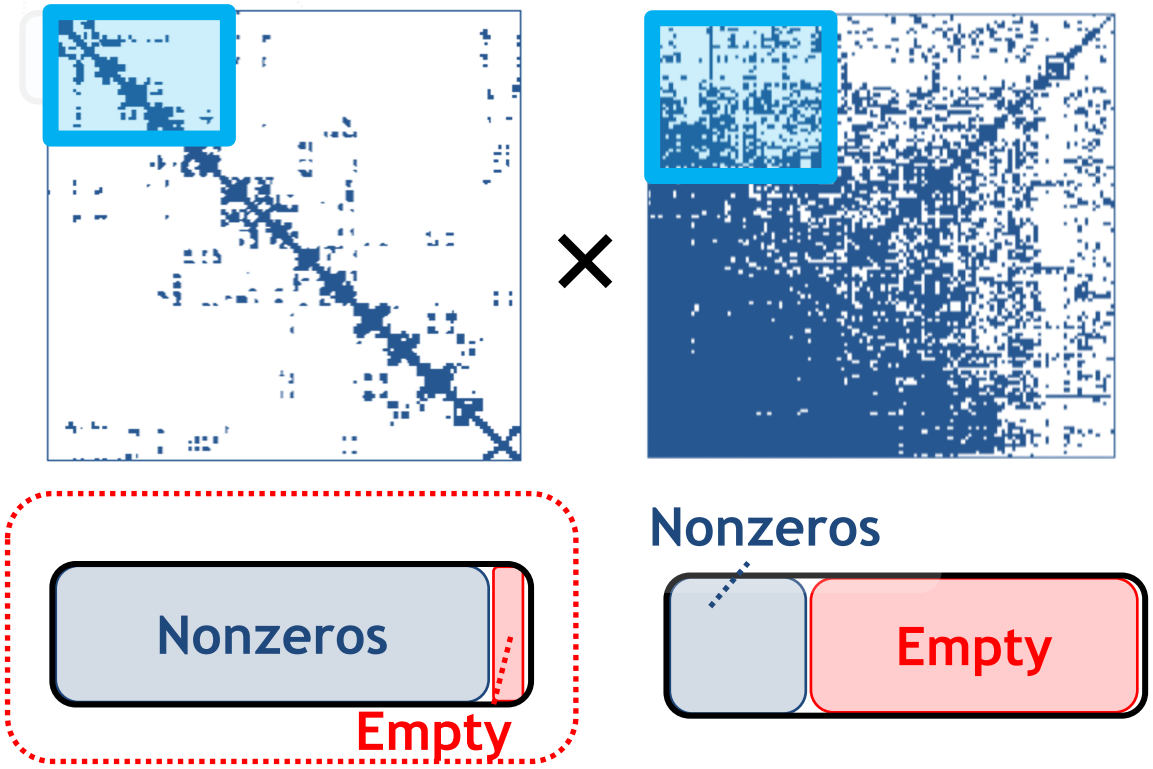
- Always fetch enough nonzeros to fill the buffer (ideal buffer utilization)
- Leads to non-uniform tile shapes (hard to tile other operand)



Tiling with *uniform shape*

+ Always construct tiles with the same shape (easy to tile both operands)

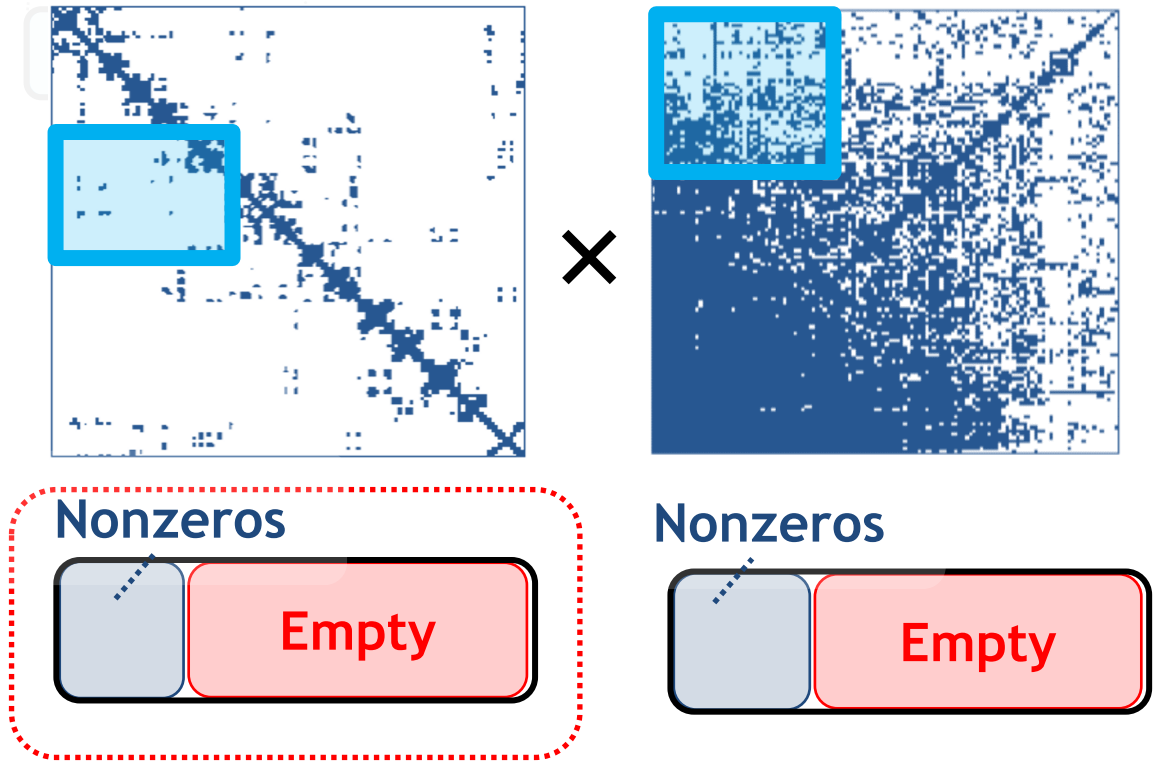
- All tiles must fit within the buffer (low buffer utilization)



Tiling with *uniform shape*

+ Always construct tiles with the same shape (easy to tile both operands)

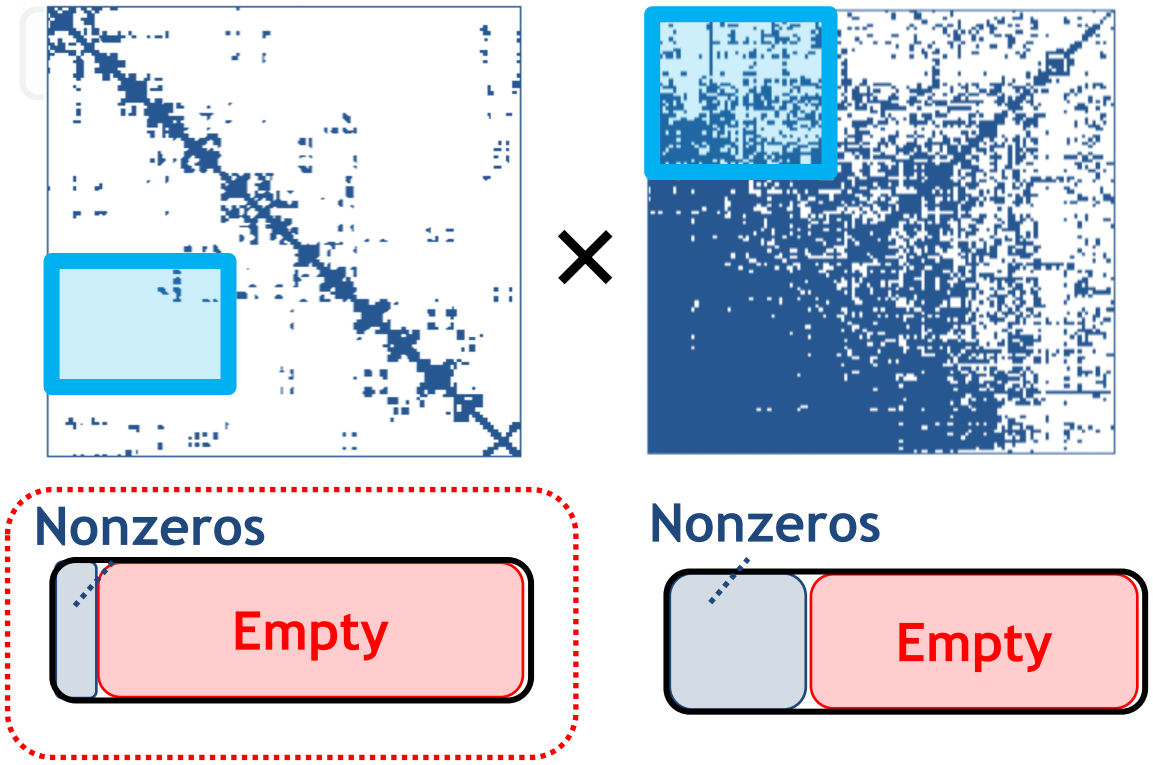
- All tiles must fit within the buffer (low buffer utilization)



Tiling with *uniform shape*

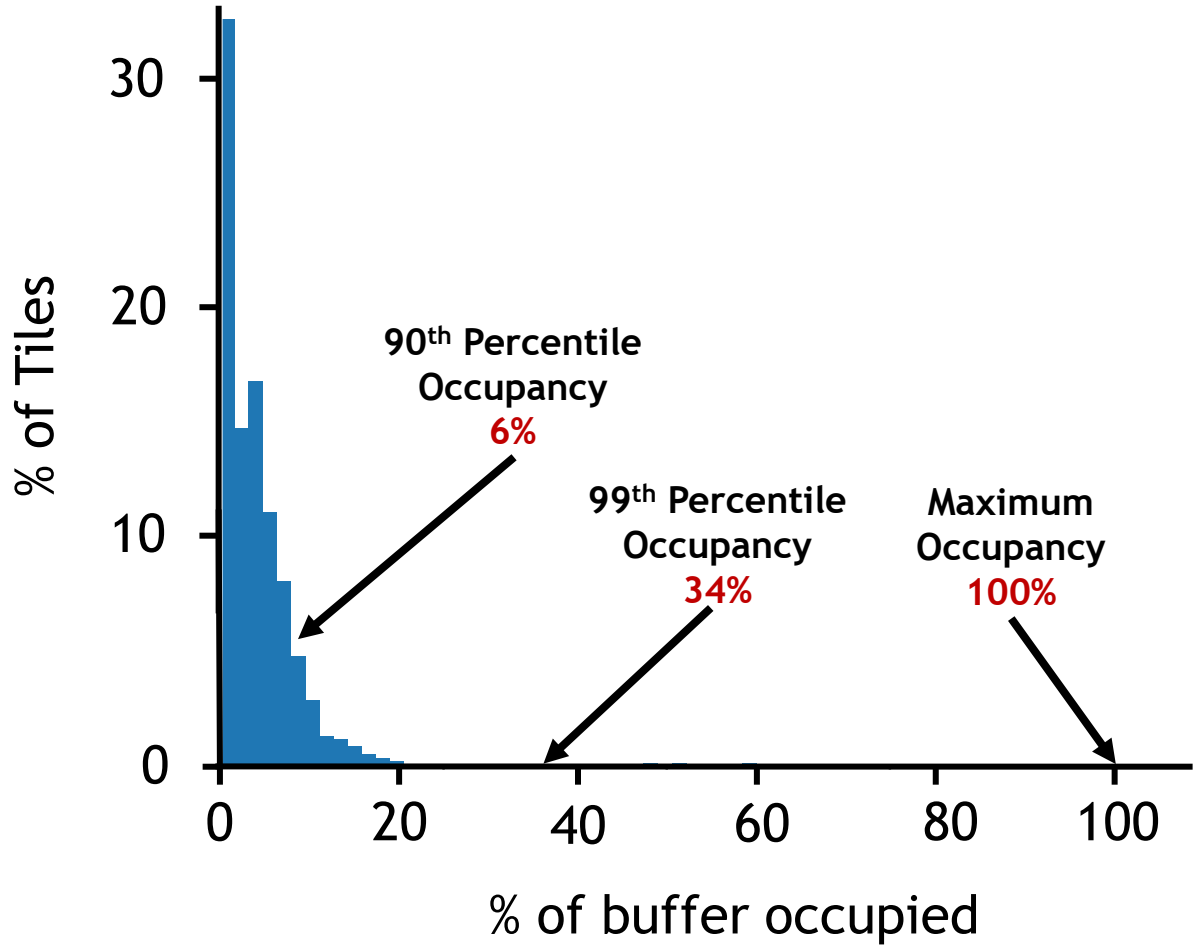
+ Always construct tiles with the same shape (easy to tile both operands)

- All tiles must fit within the buffer (low buffer utilization)



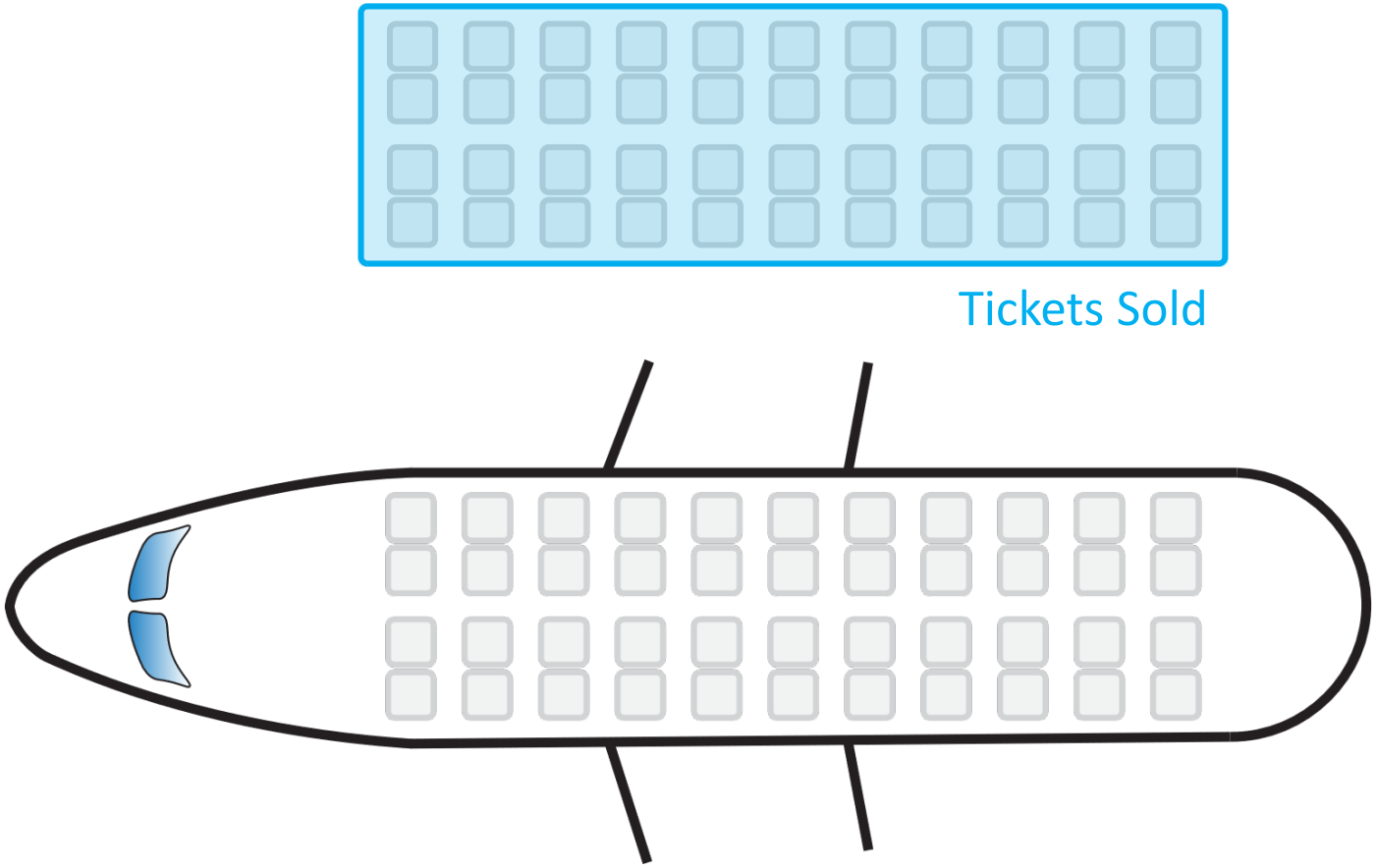
Uniform shape tiling has poor buffer utilization

- Constructing tiles based on the maximum tile occupancy may overprovision buffer space
- Most tiles do not fully occupy the buffer

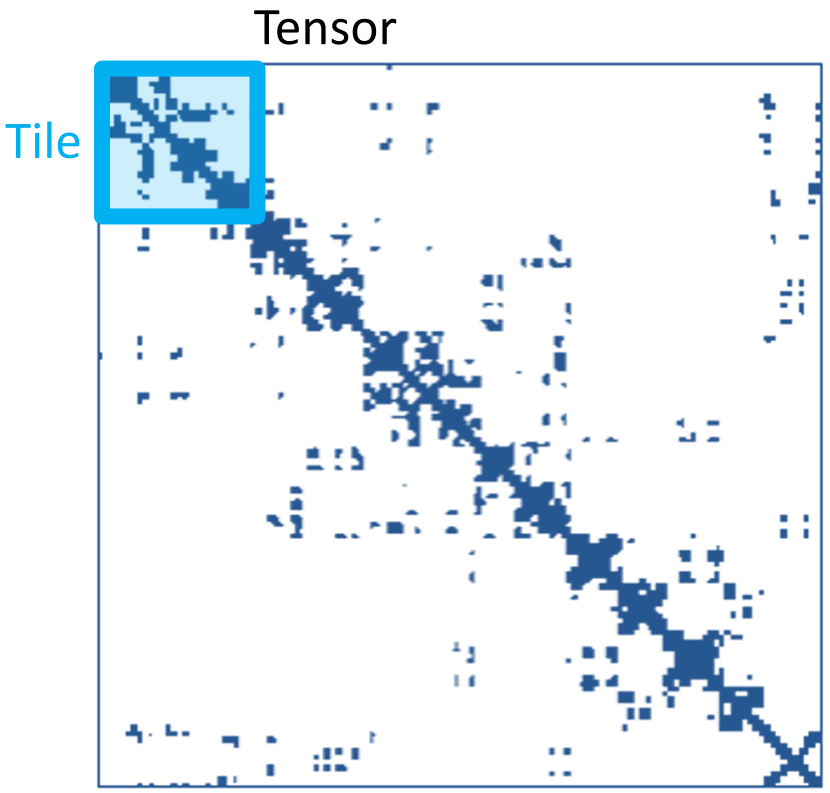


What if we could **overbook** buffer capacity?

Sparsity leads to poor utilization

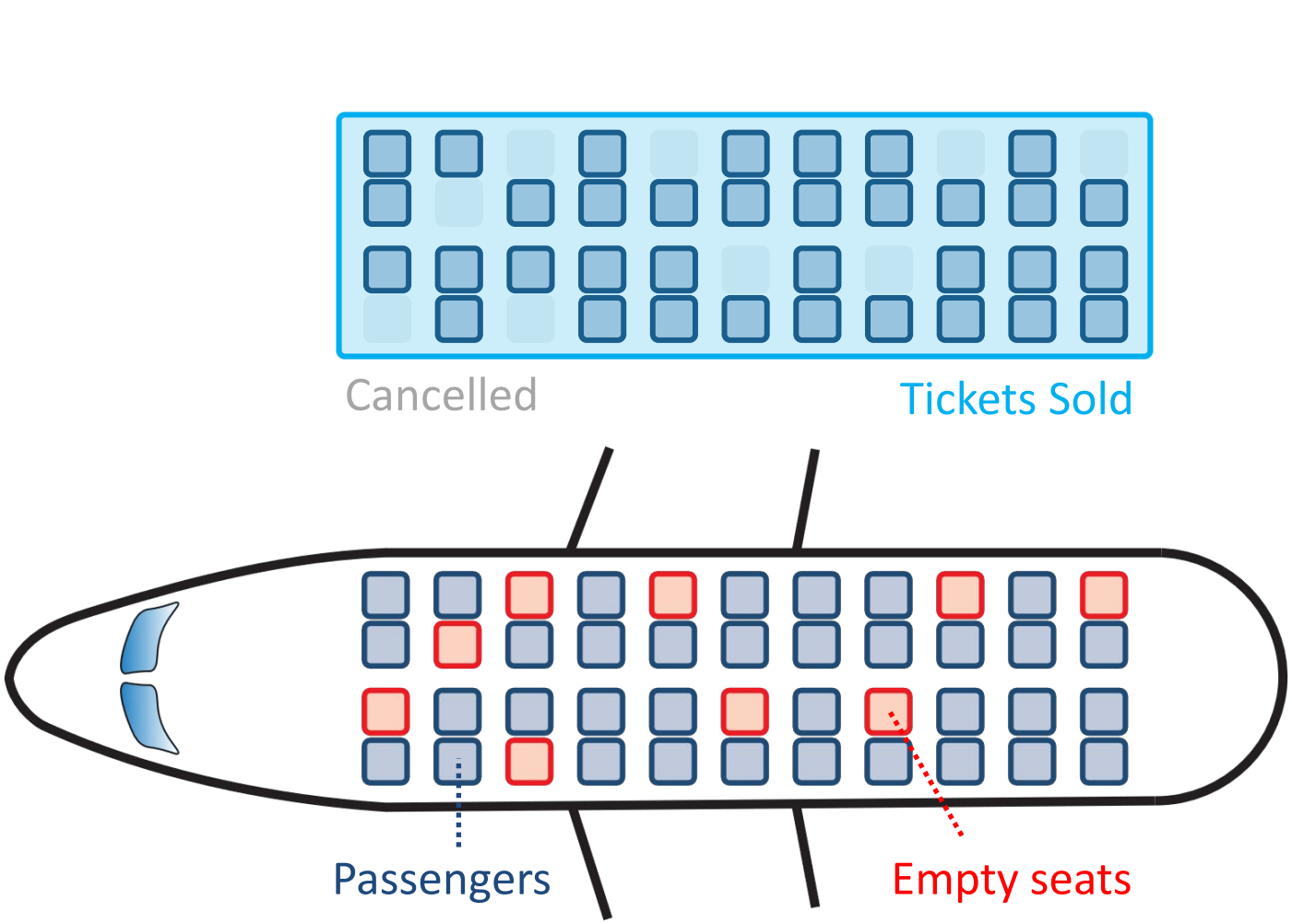


Sparsity leads to poor utilization

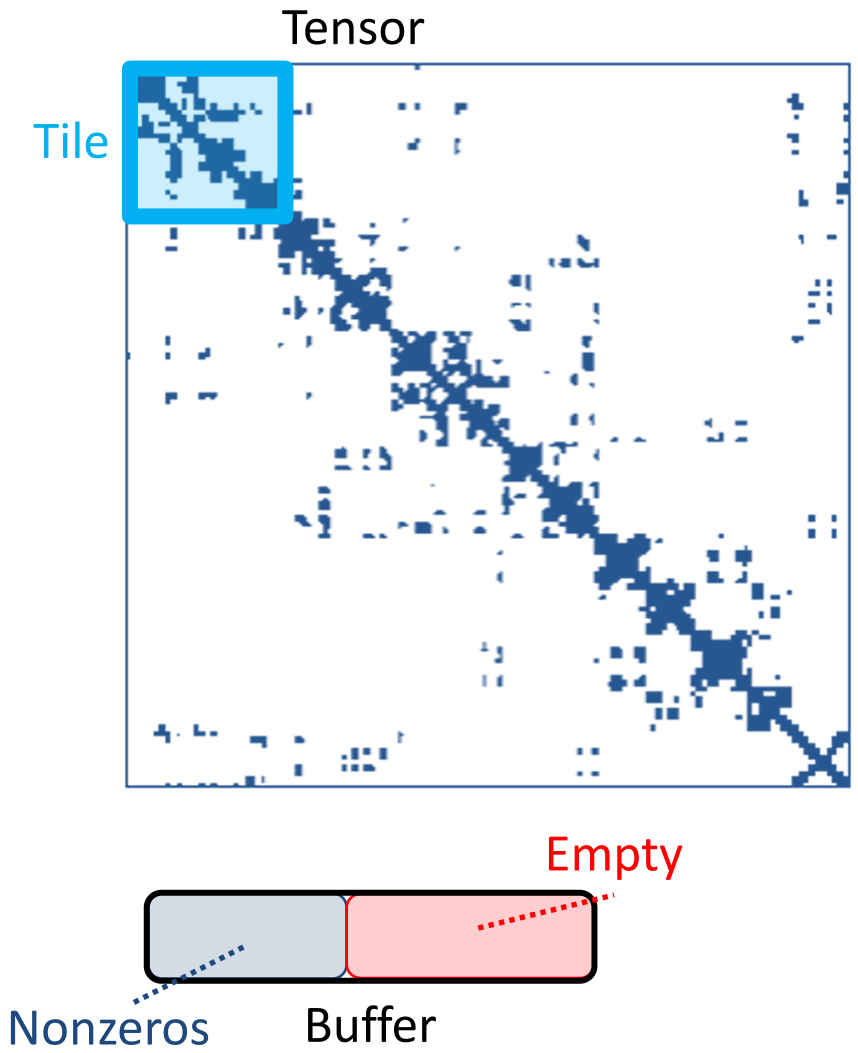


Buffer

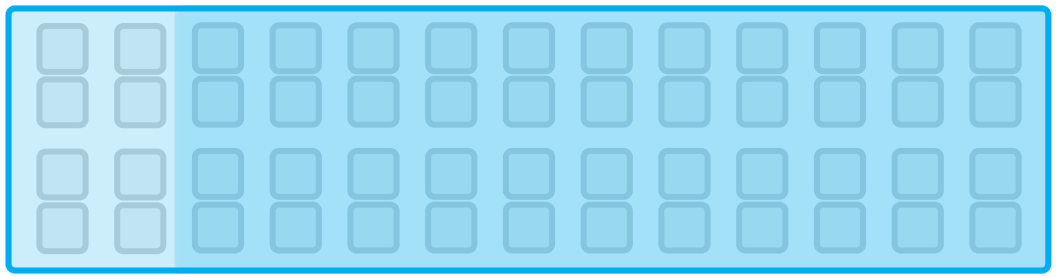
Sparsity leads to poor utilization



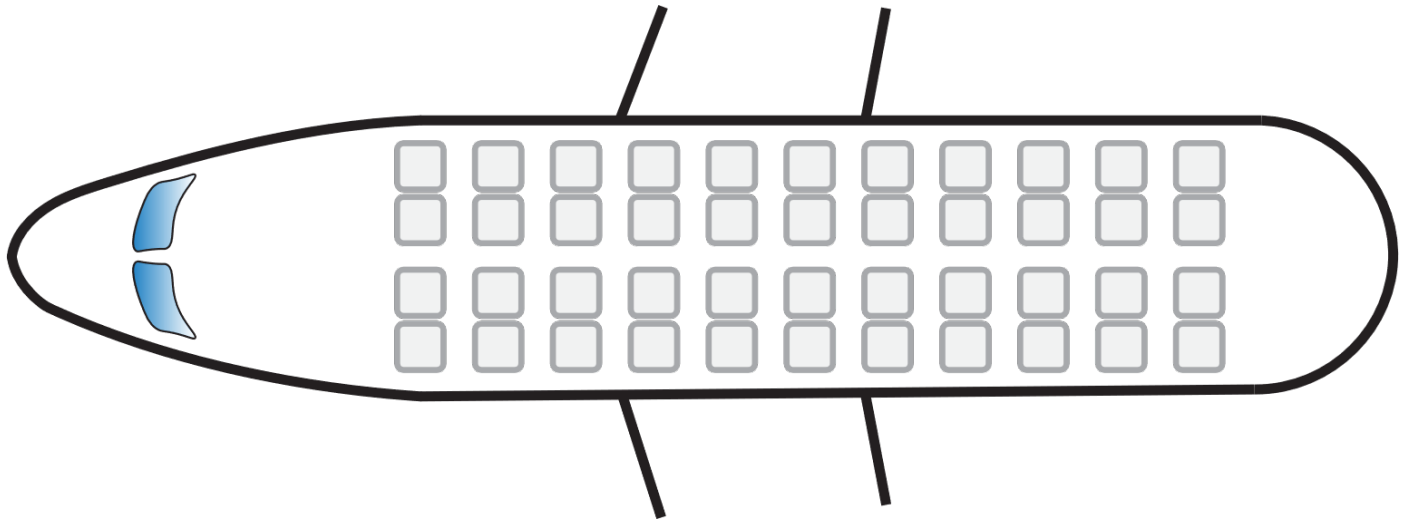
Sparsity leads to poor utilization



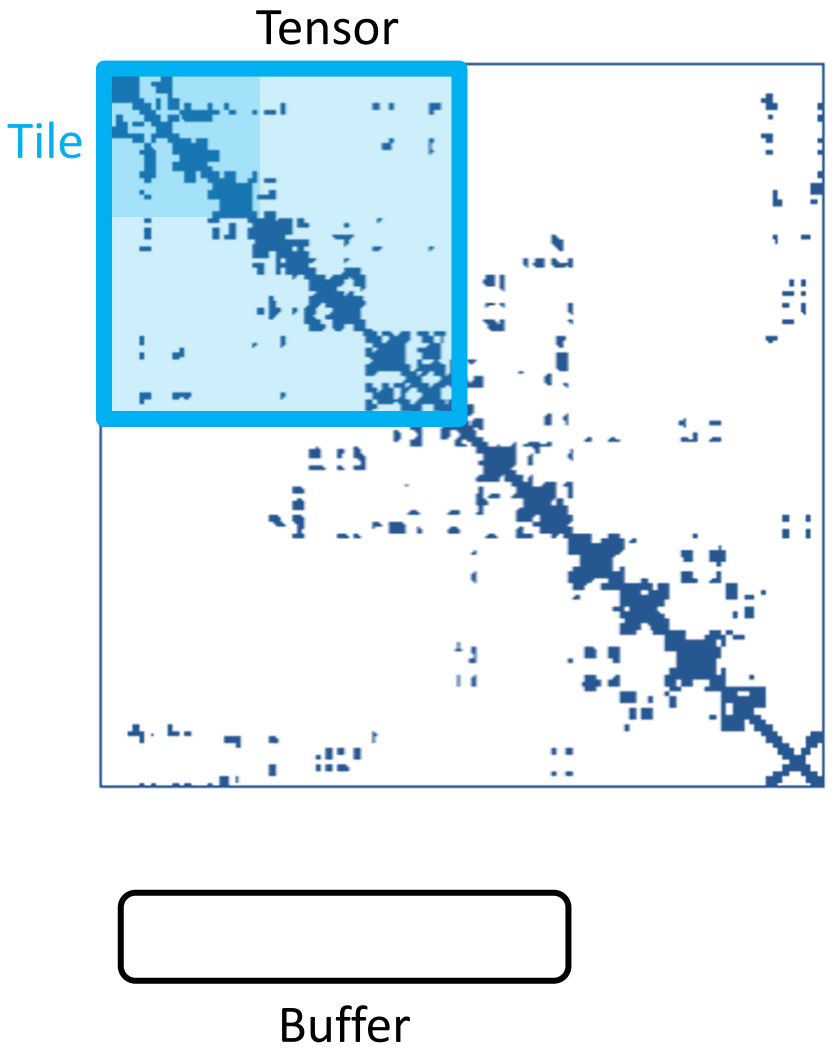
Overbooking improves utilization



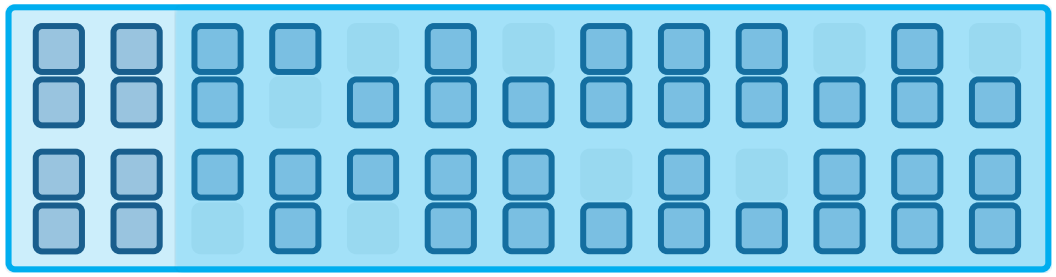
Tickets Sold

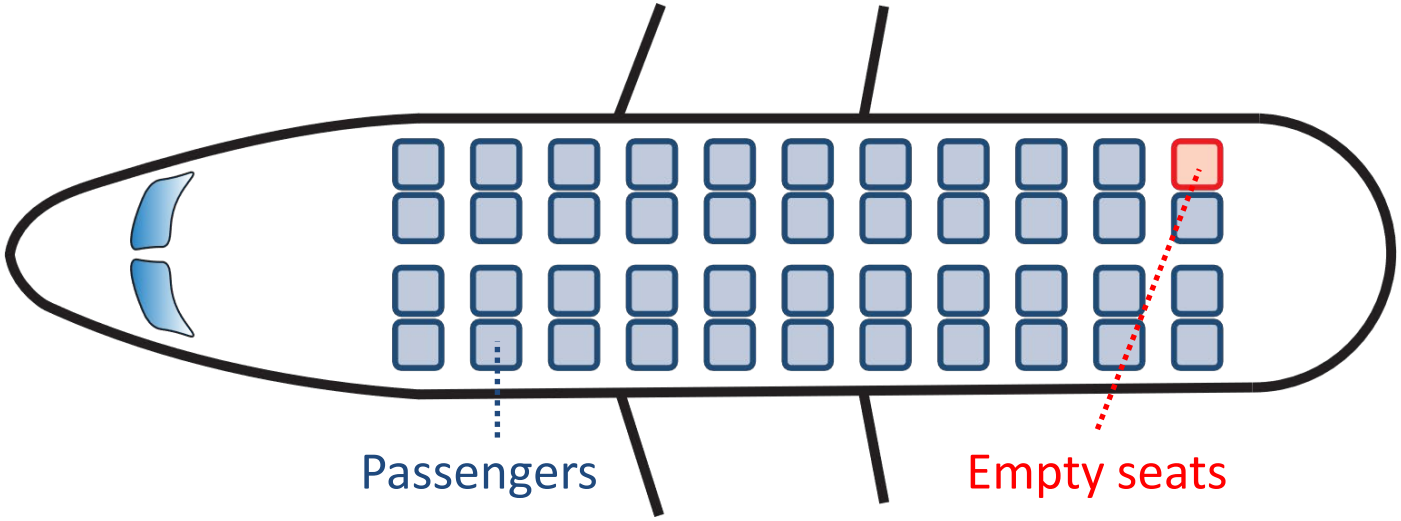


Overbooking improves utilization

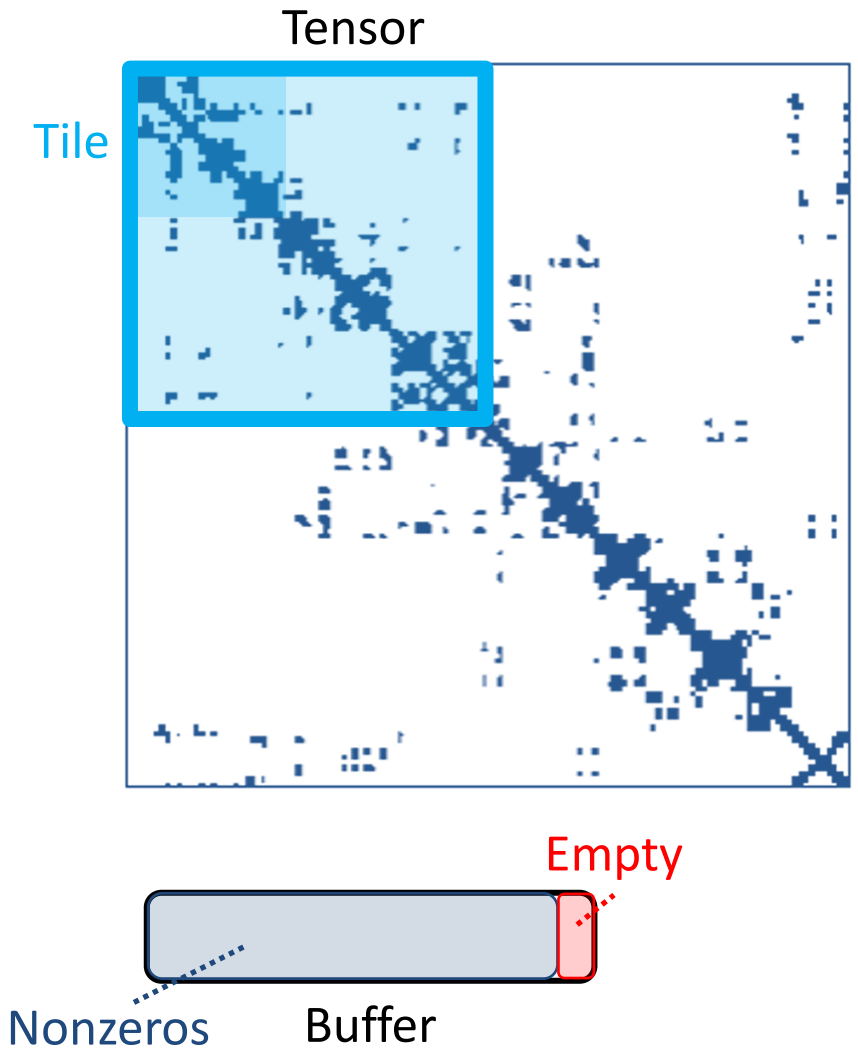


Overbooking improves utilization

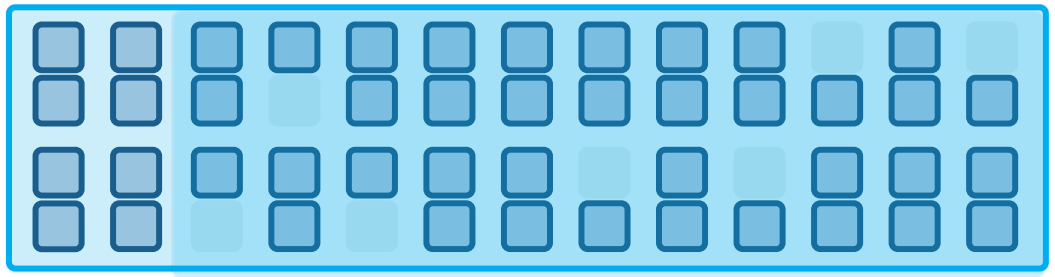


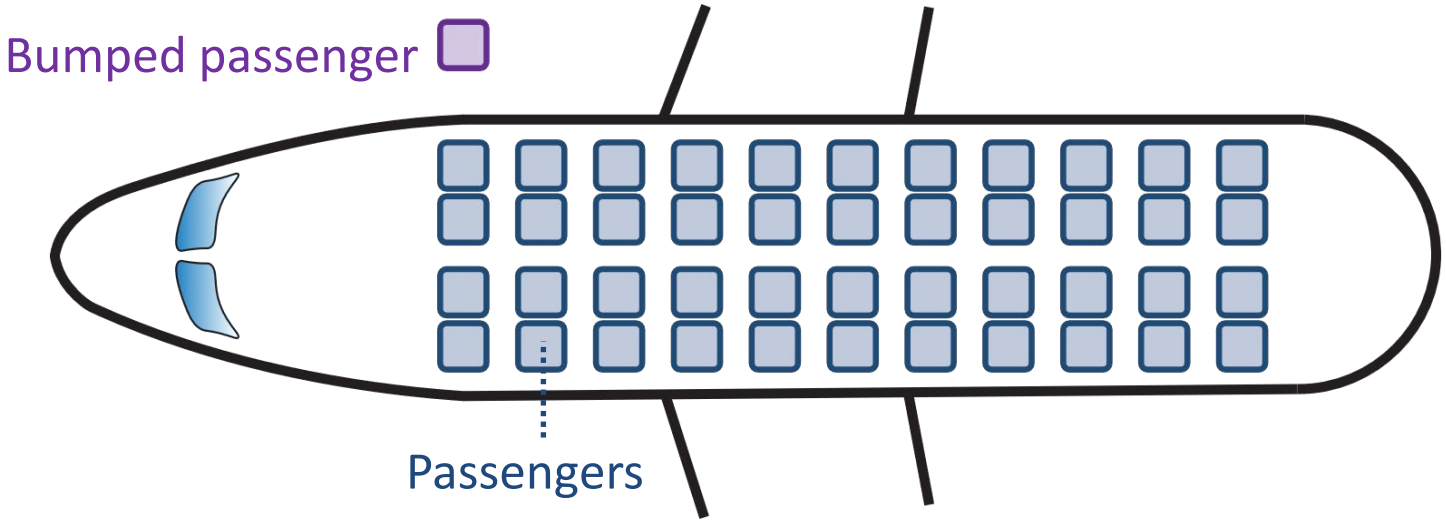


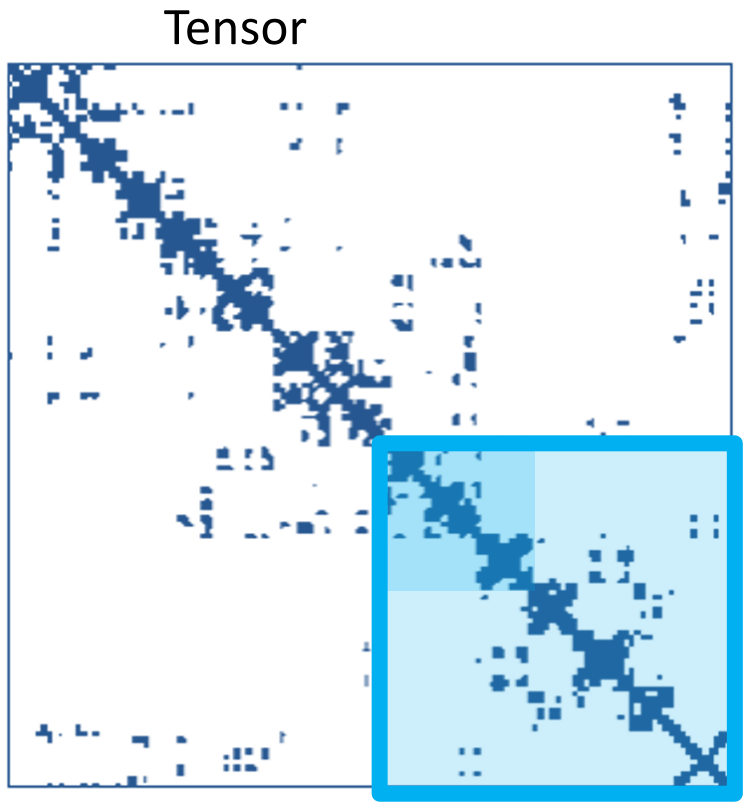
Overbooking improves utilization

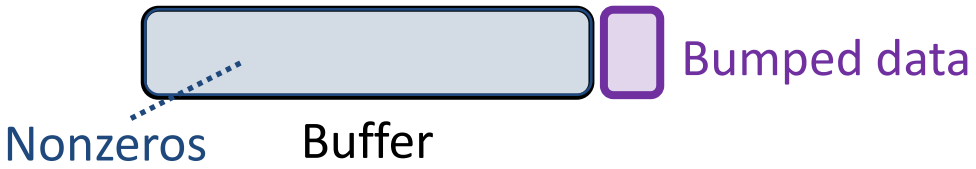


Overbooking improves utilization

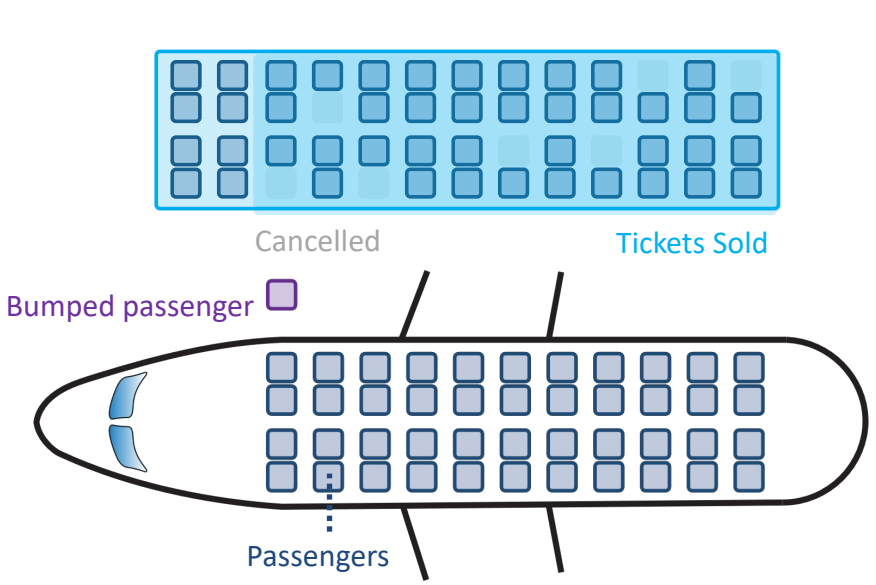


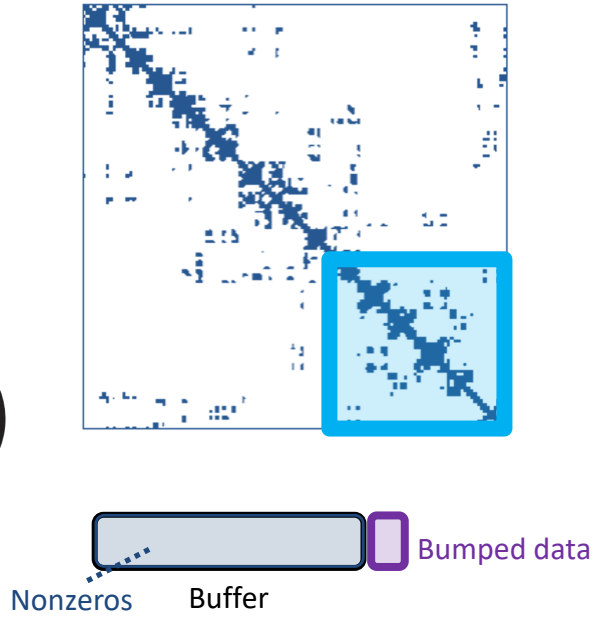






Overbooking improves utilization





How do we deal with the bumped data?

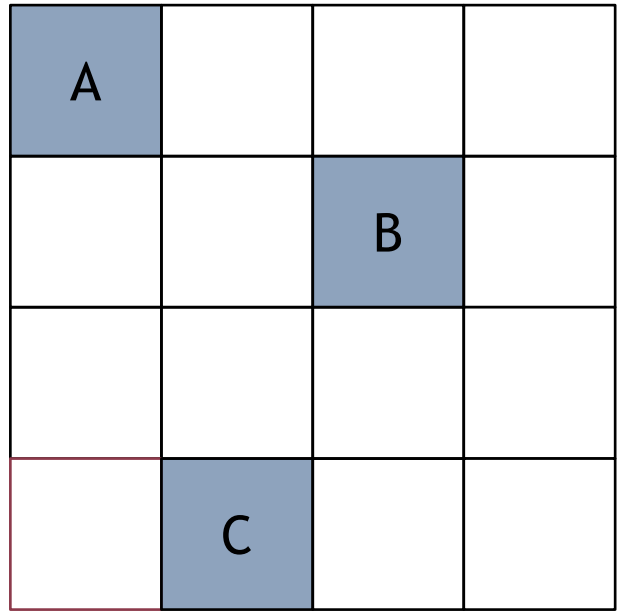
Tailors

How do we determine how much to overbook?

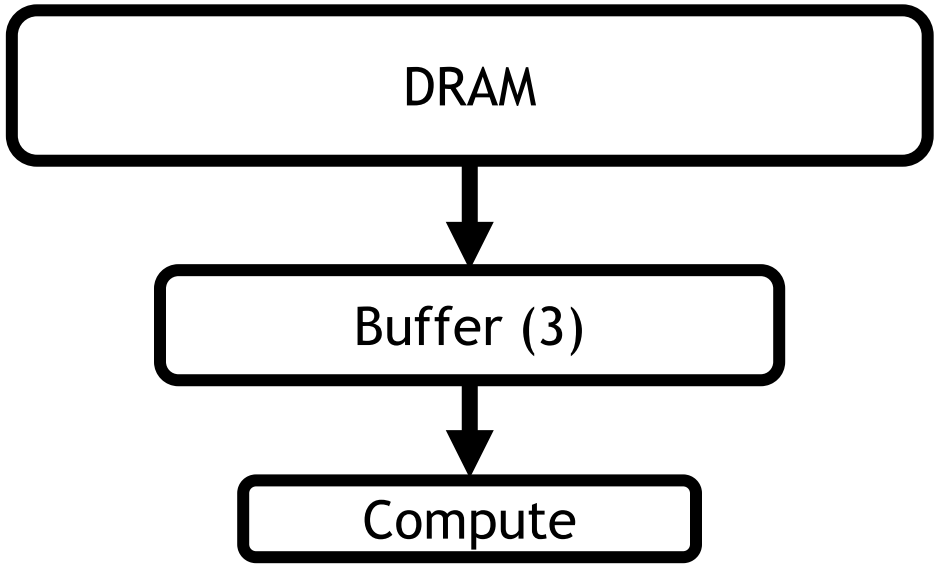
Swiftiles

Managing unbumped data

Objective: minimize DRAM traffic
by maximizing data reuse

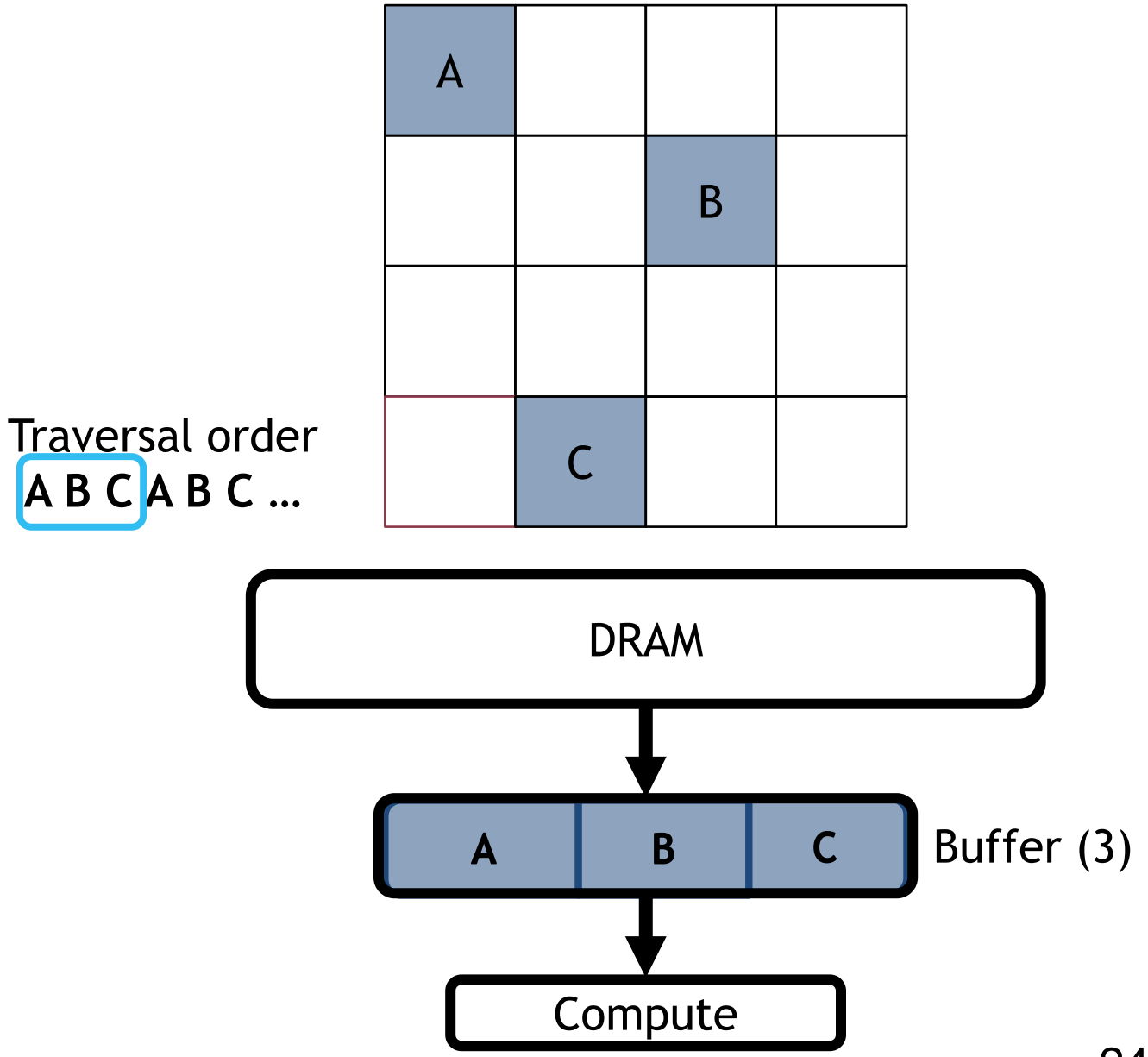


Traversal order
A B C A B C …



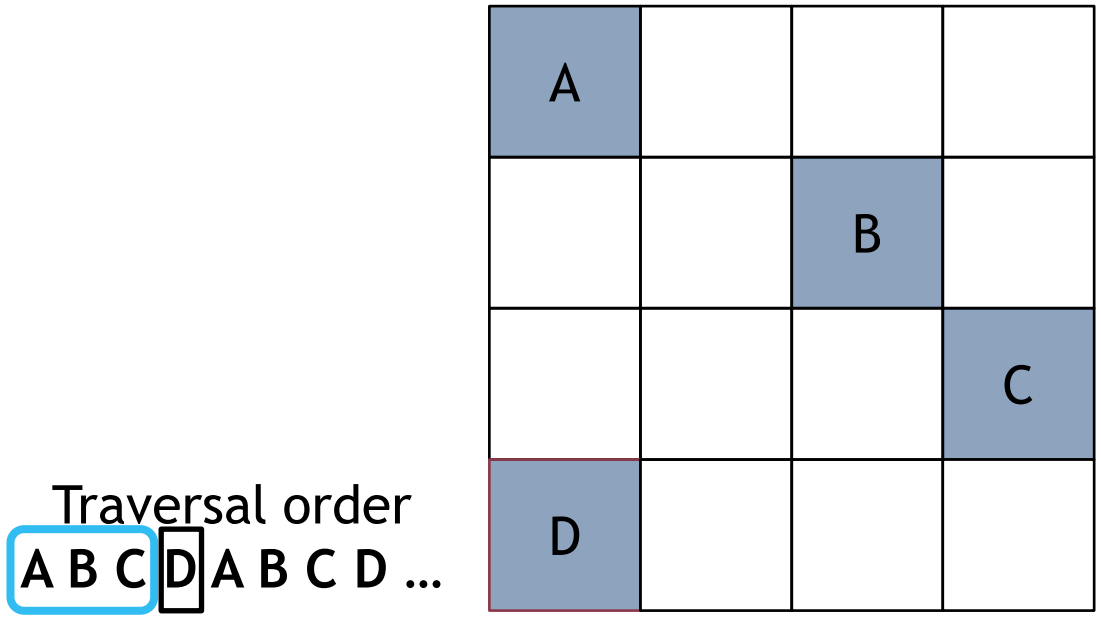
Managing unbumped data with buffets

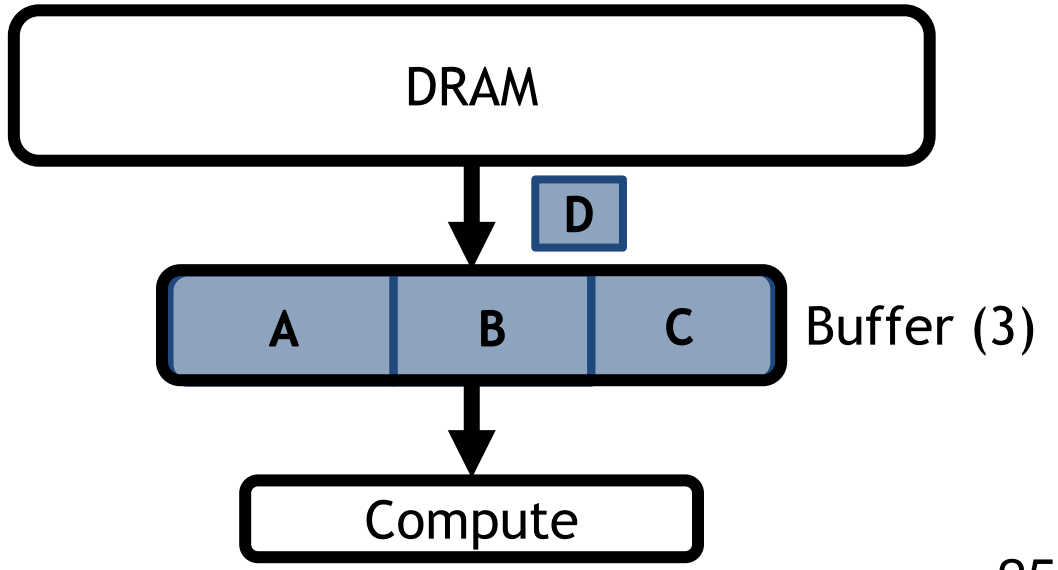
- Buffets are used to orchestrate data in a number of domain-specific accelerators
- Buffets operate on a sliding window of data



Managing bumped data with buffets

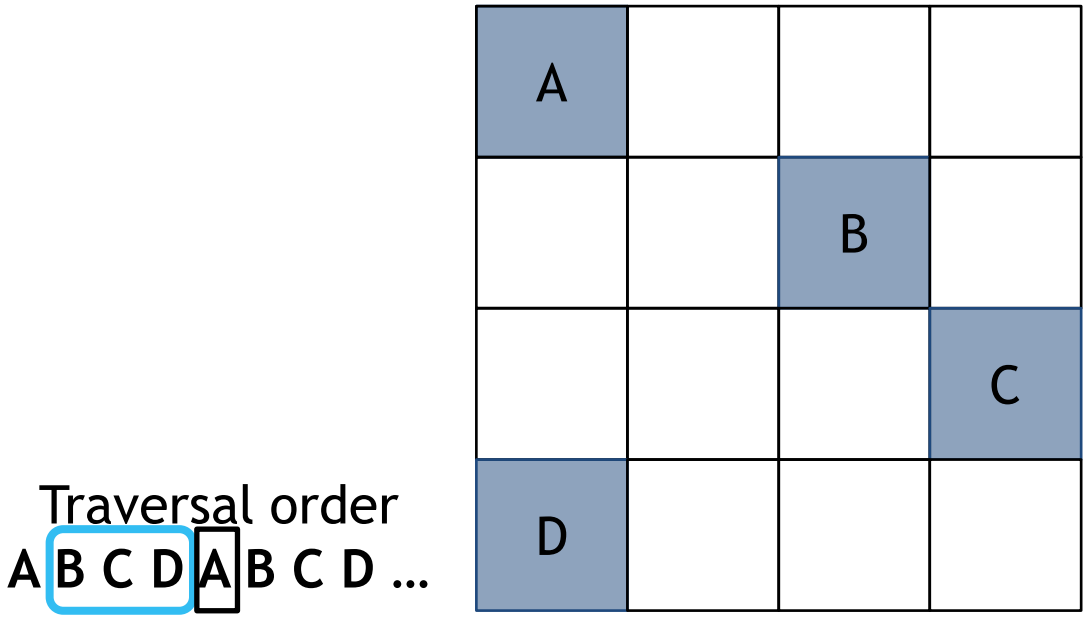
- Buffets operate on a sliding window of data
- Sliding window leads to poor data reuse when data is bumped

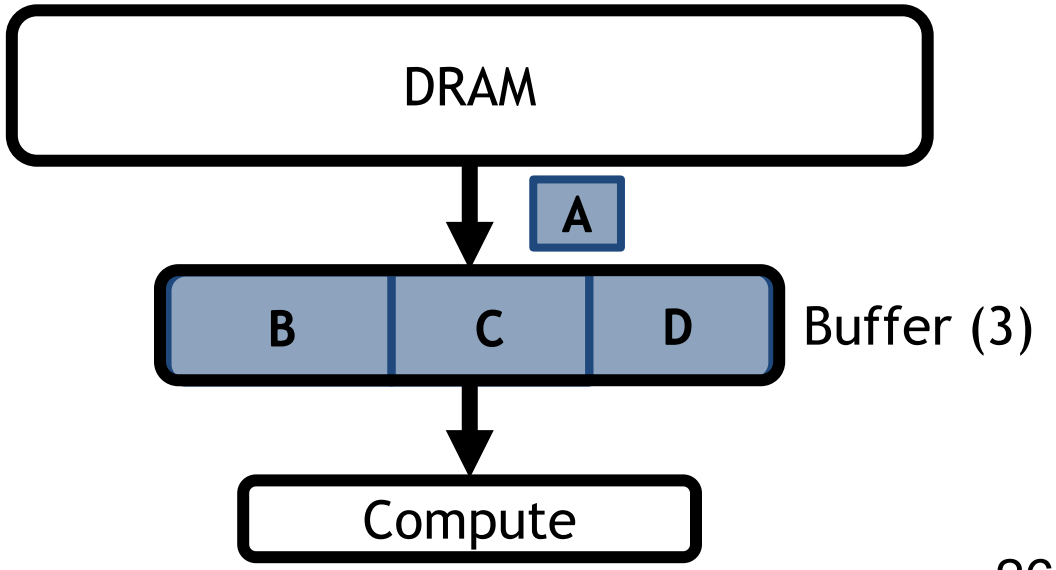




Managing bumped data with buffets

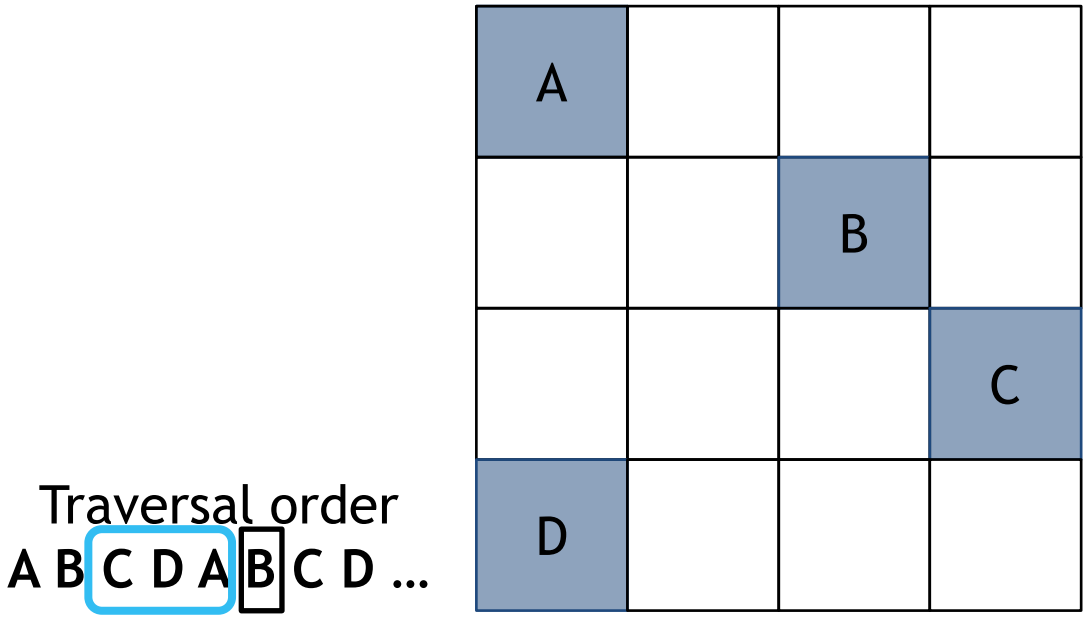
- Buffets operate on a sliding window of data
- Sliding window leads to poor data reuse when data is bumped

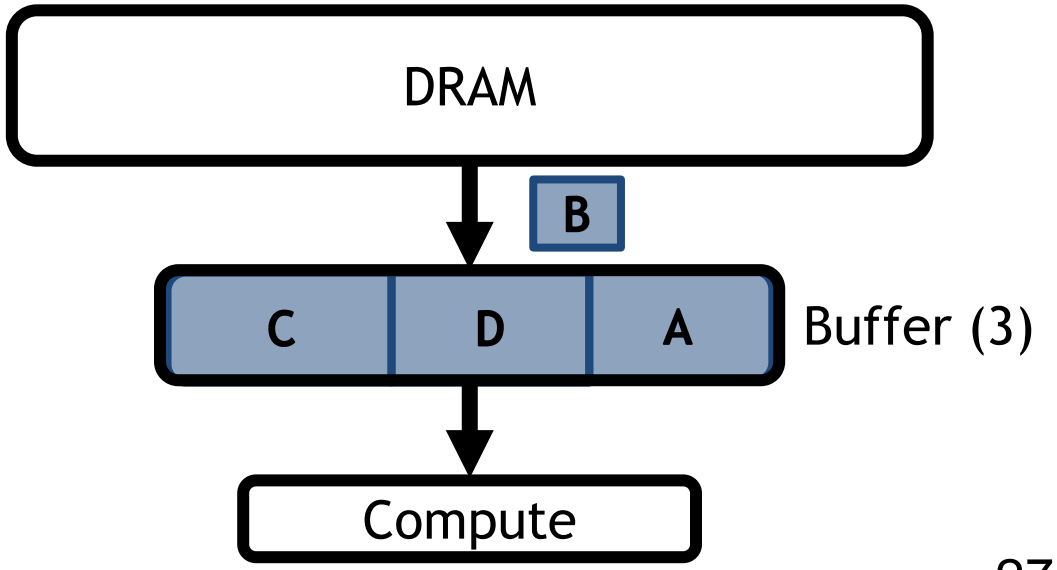




Managing bumped data with buffets

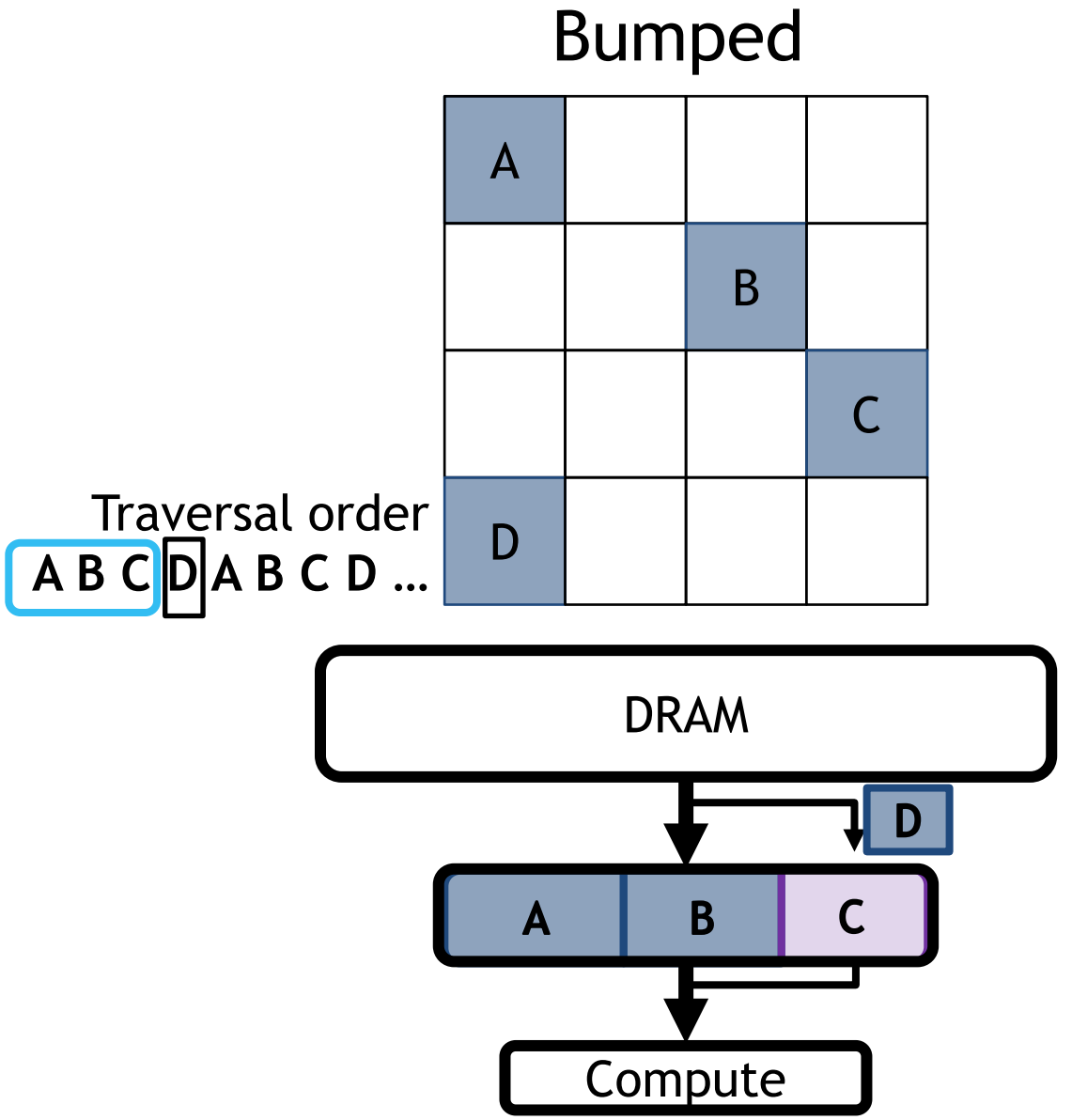
- Buffets operate on a sliding window of data
- Sliding window leads to poor data reuse when data is bumped





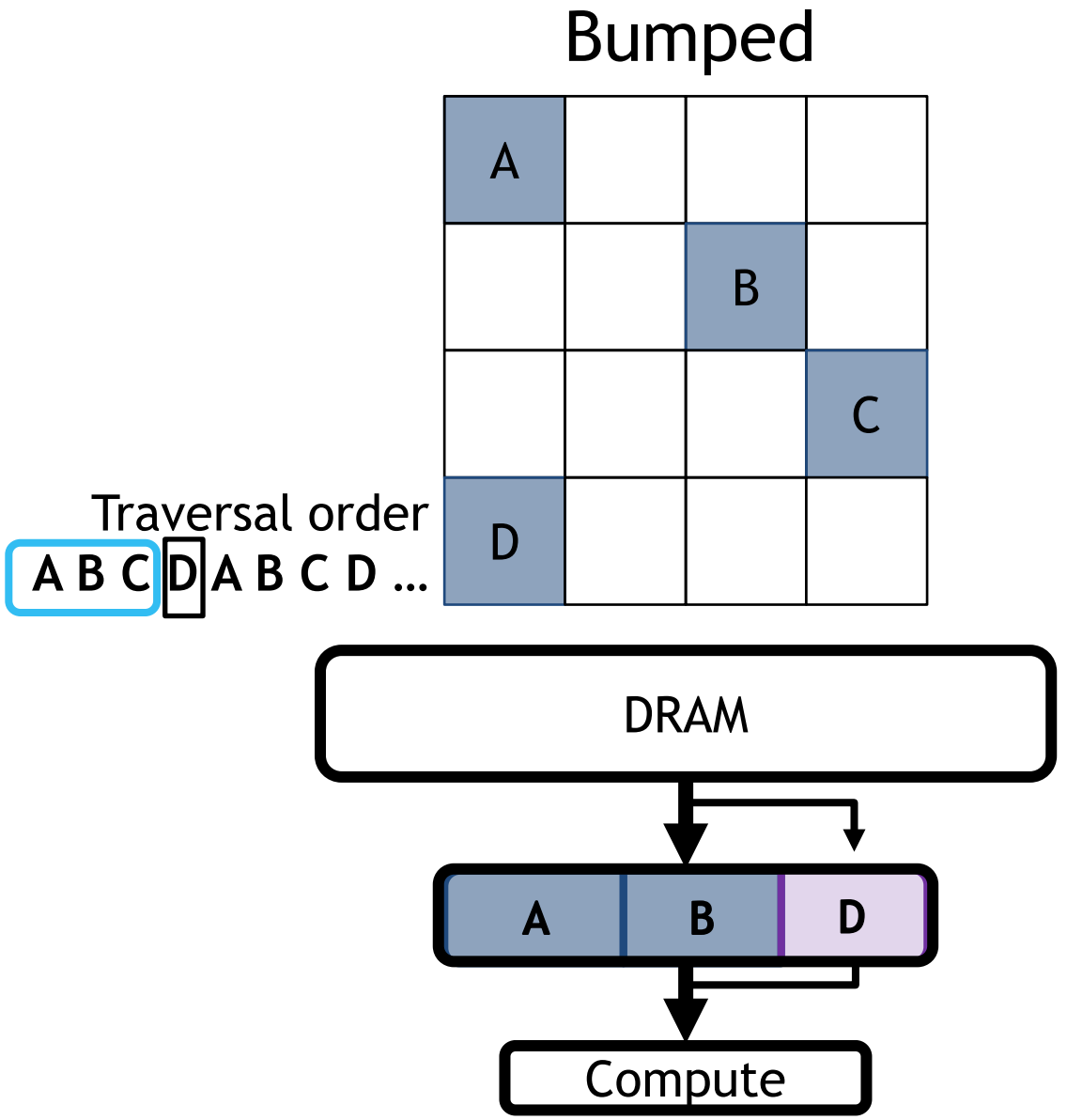
Tailors: Tail Overbooked Buffers for overbooked data

- Tailors dynamically splits the buffer when overbooked to stream bumped data through the tail of the buffer
- Lose reuse for bumped data, but not for unbumped data

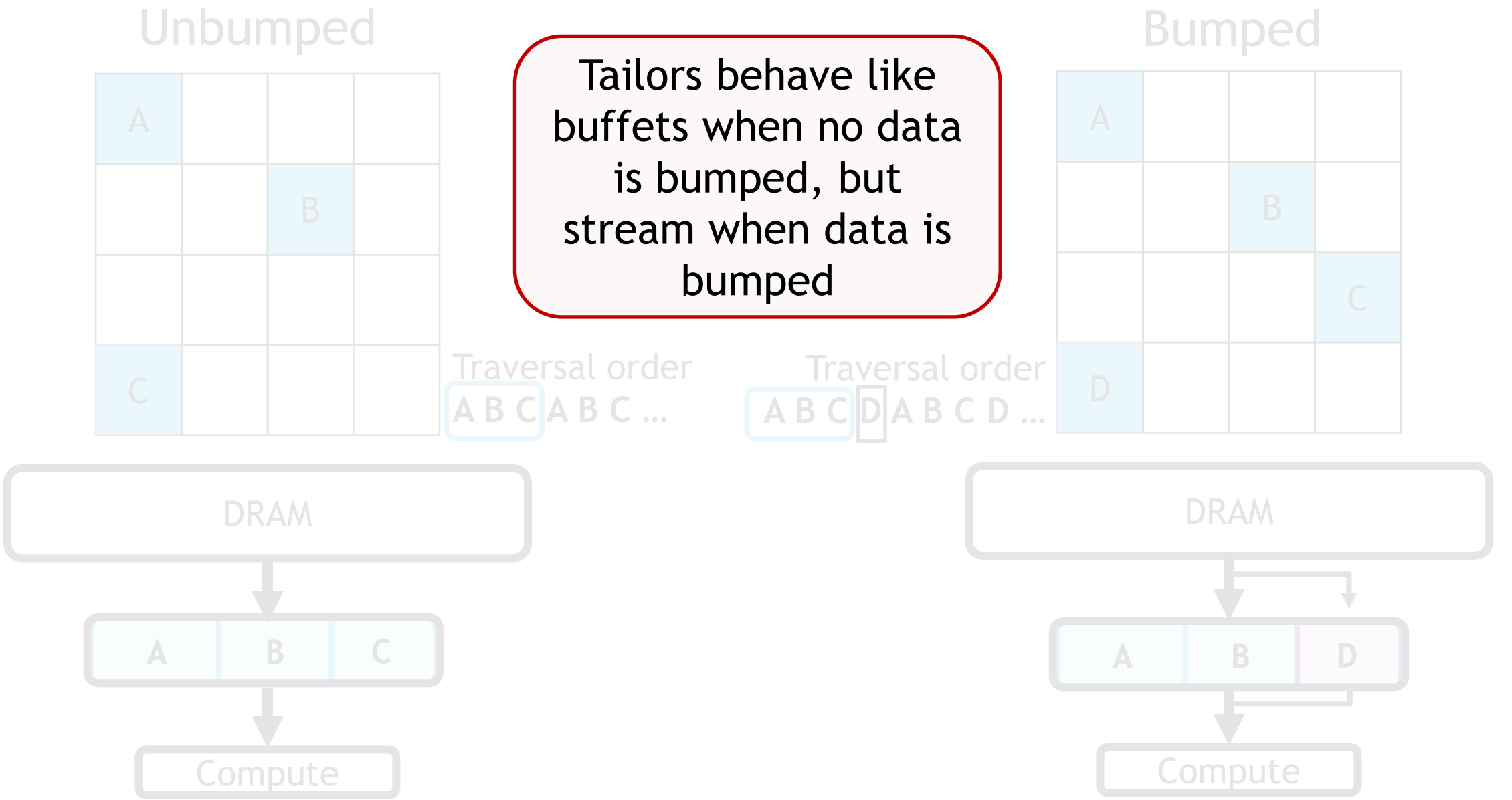


Tailors: Tail Overbooked Buffers for overbooked data

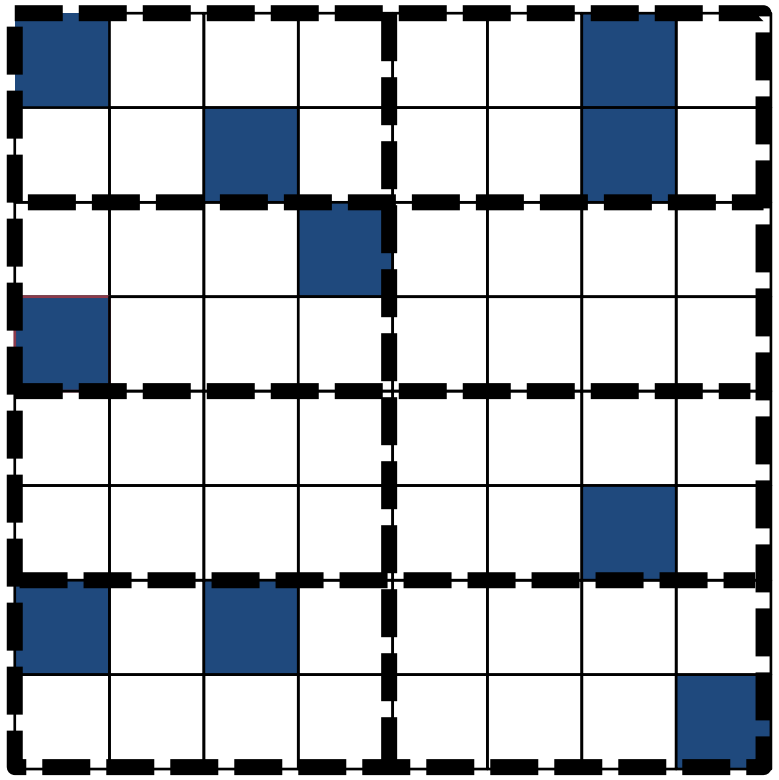
- Tailors dynamically splits the buffer when overbooked to stream bumped data through the tail of the buffer
- Lose reuse for bumped data, but not for unbumped data



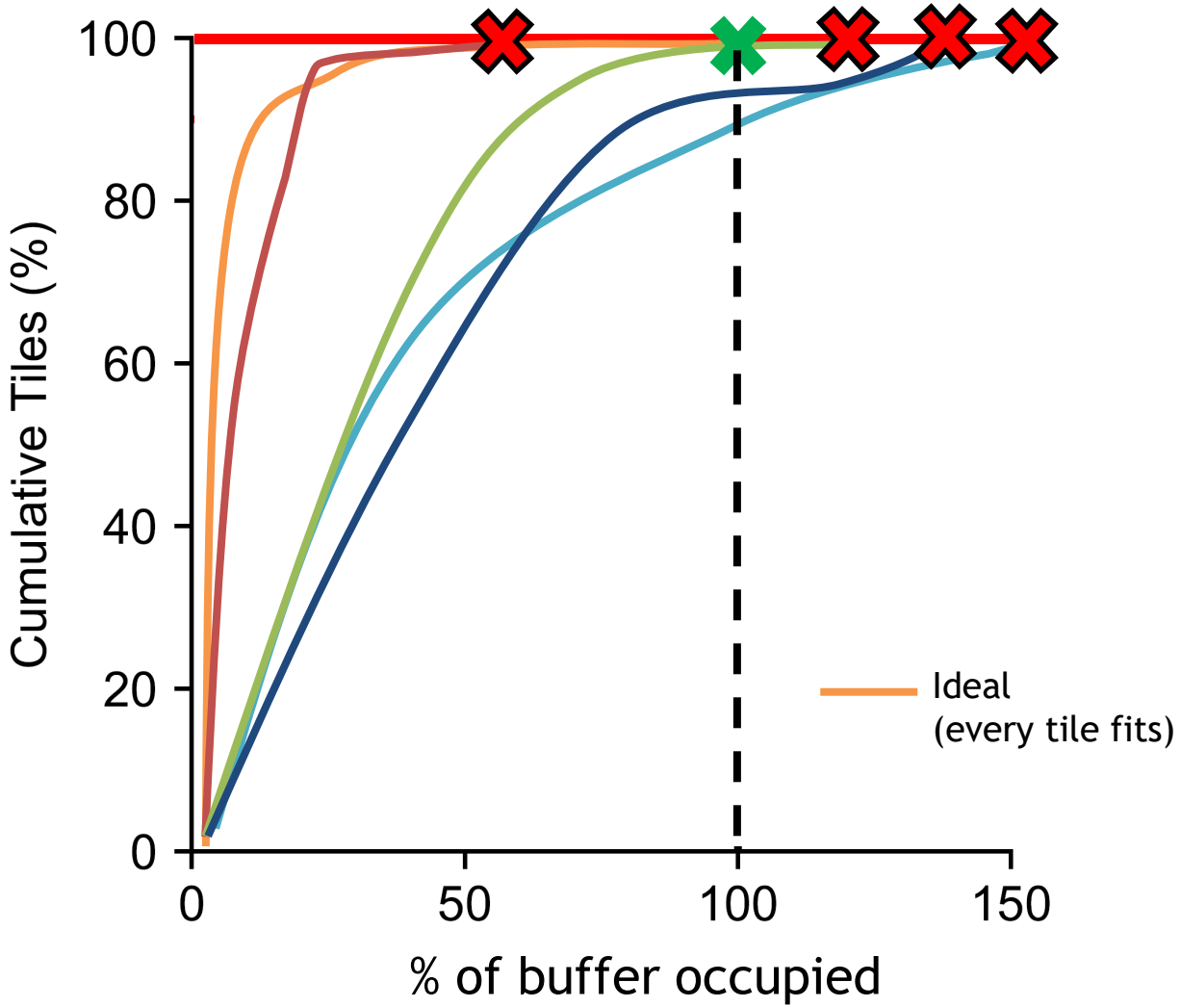
Tailors: Tail Overbooked Buffers for overbooked data



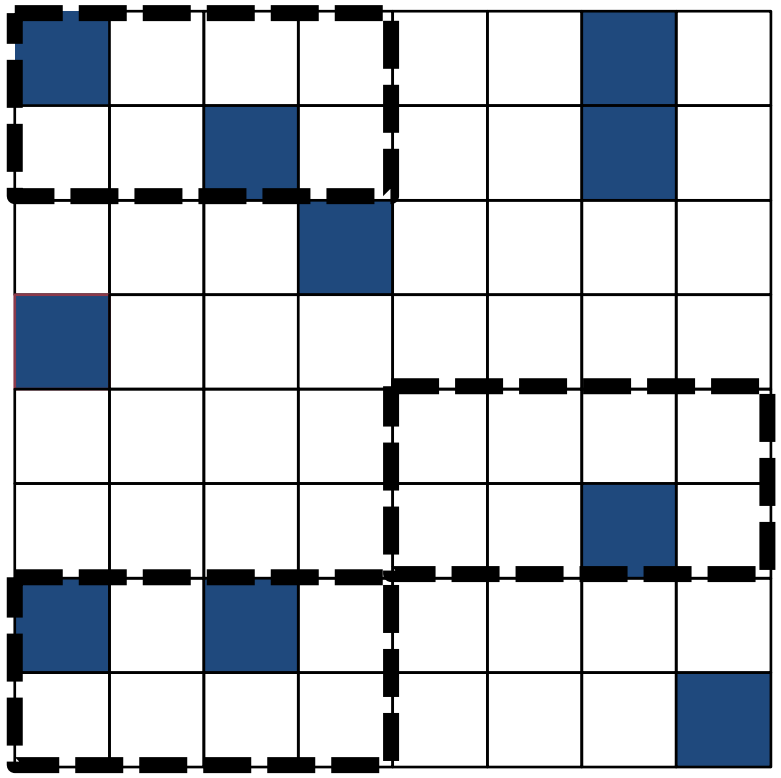
Determining the size of a tile is challenging



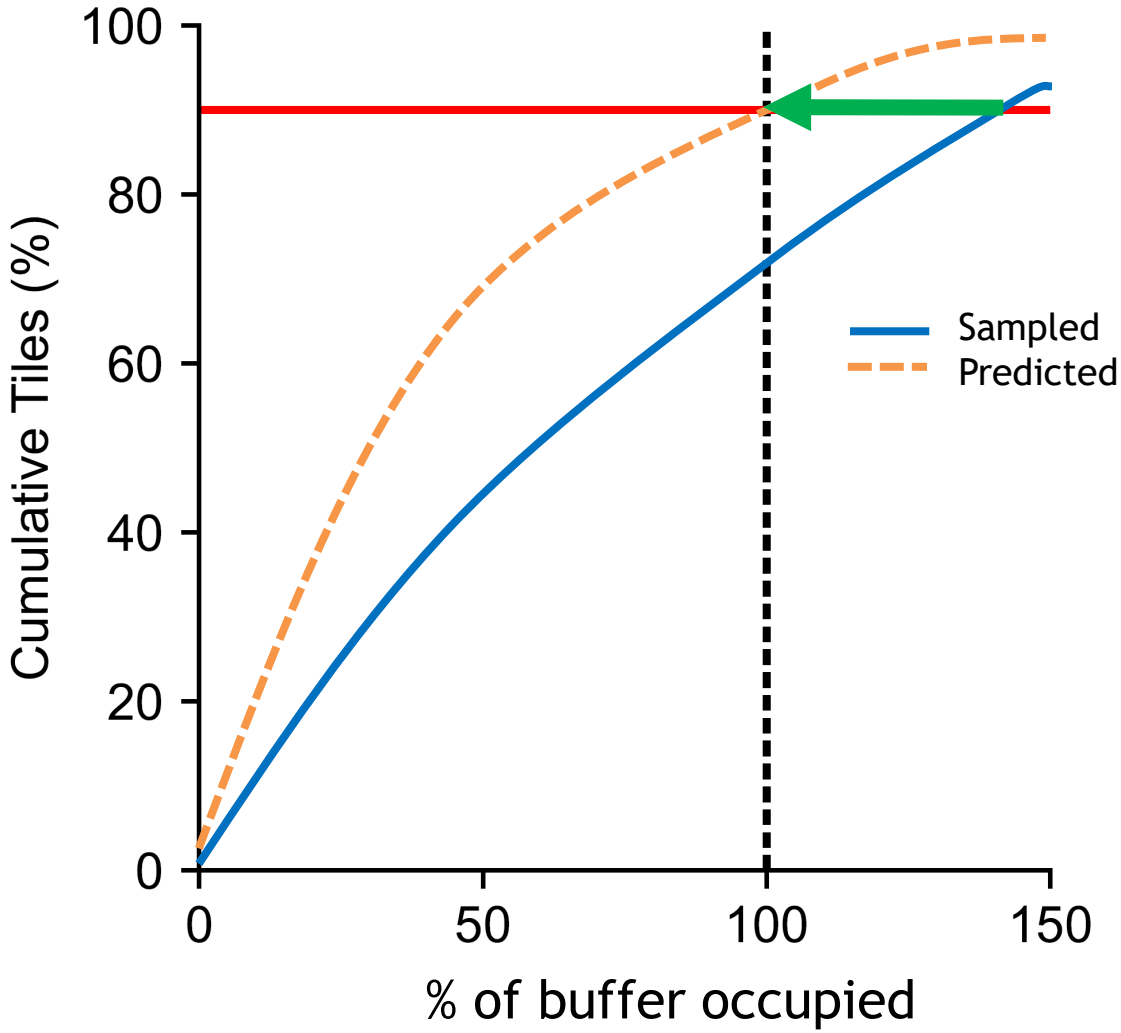
Full traversal to find maximum occupancy



Swiftiles: A swift tiling algorithm for overbooking



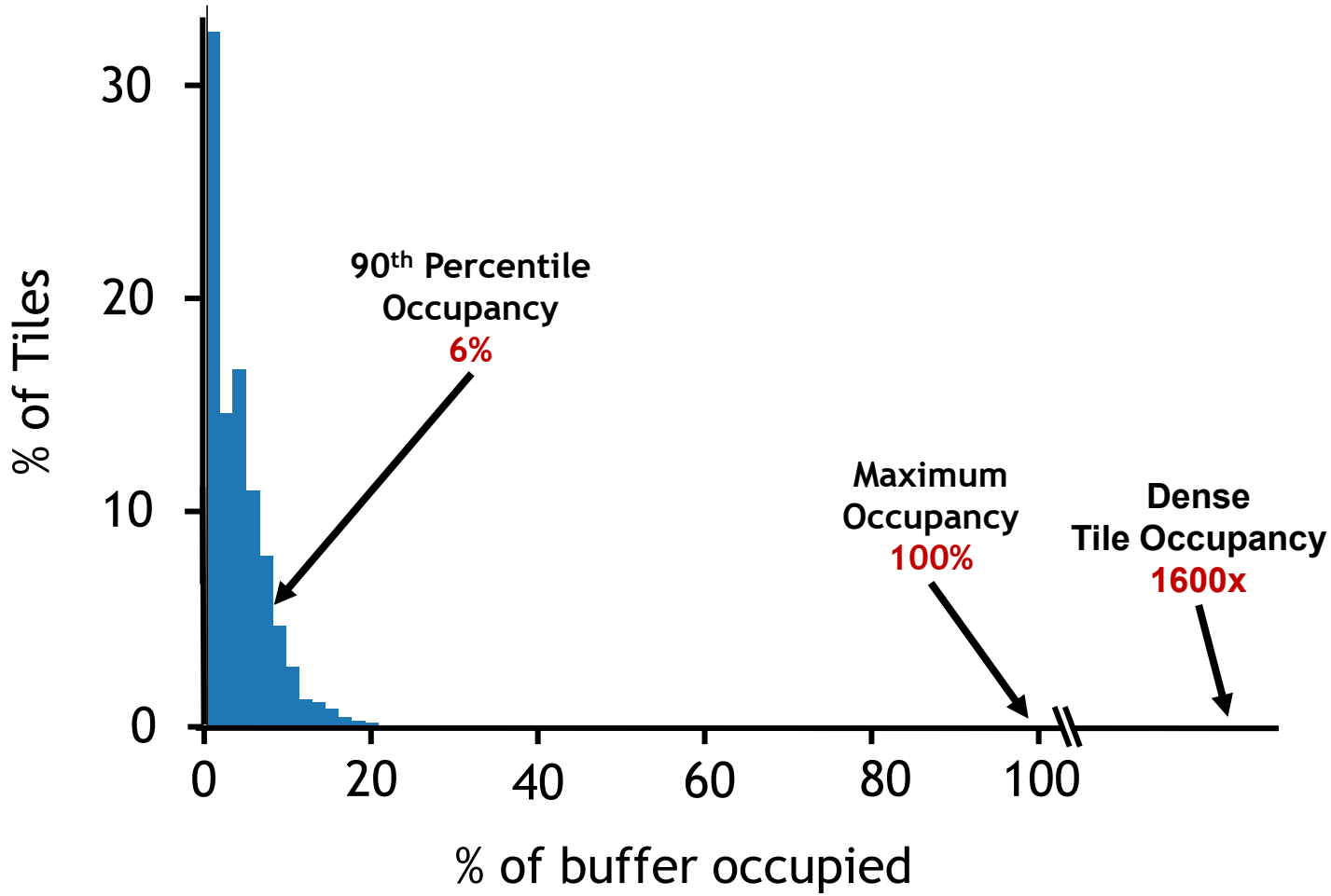
Random sampling to generate approximate occupancy distribution



Scale distribution to buffer size

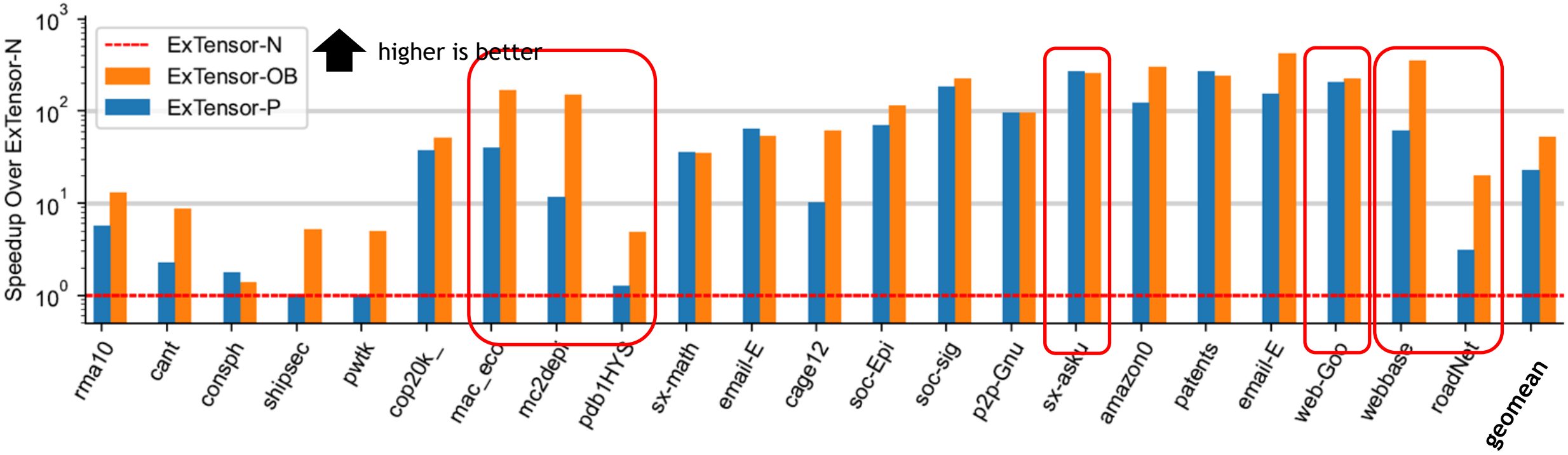
Evaluation against other tiling options

- ExTensor-Naive
  - No knowledge of tile occupancy, so must tile assuming dense tiles
- ExTensor-Prescient
  - Uses the maximum tile size where all tiles still fit in the buffer
- **ExTensor-Overbooking**
  - Tailors+Swiftiles where 90% of tiles fit in buffer



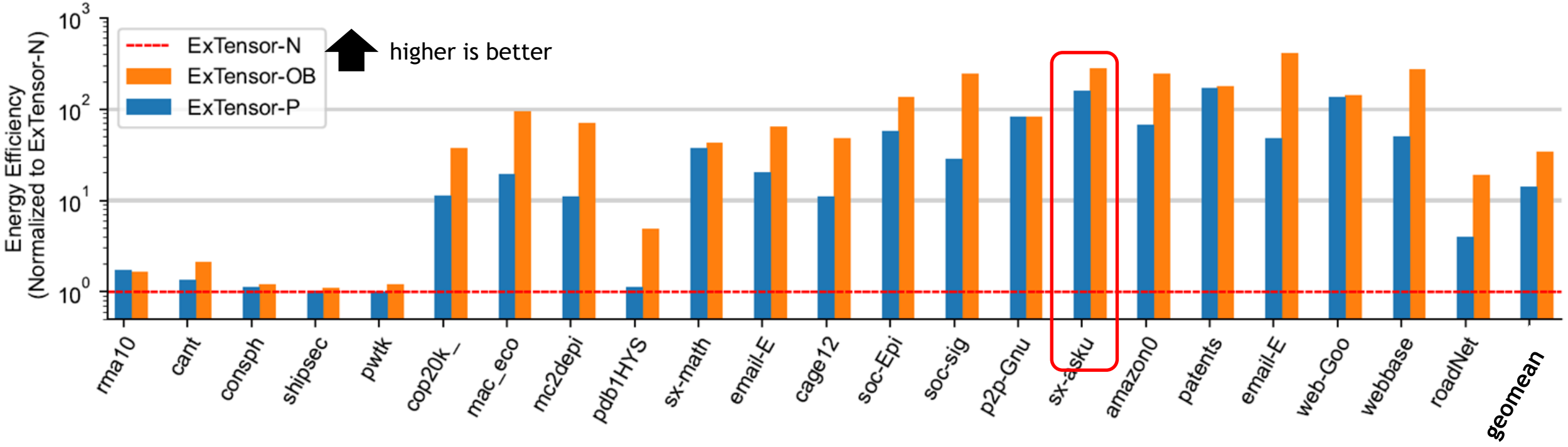
[Hegde, *MICRO 2019*]

Results - Speedup over ExTensor-N



52.7x speedup over ExTensor-N, 2.3x over ExTensor-P

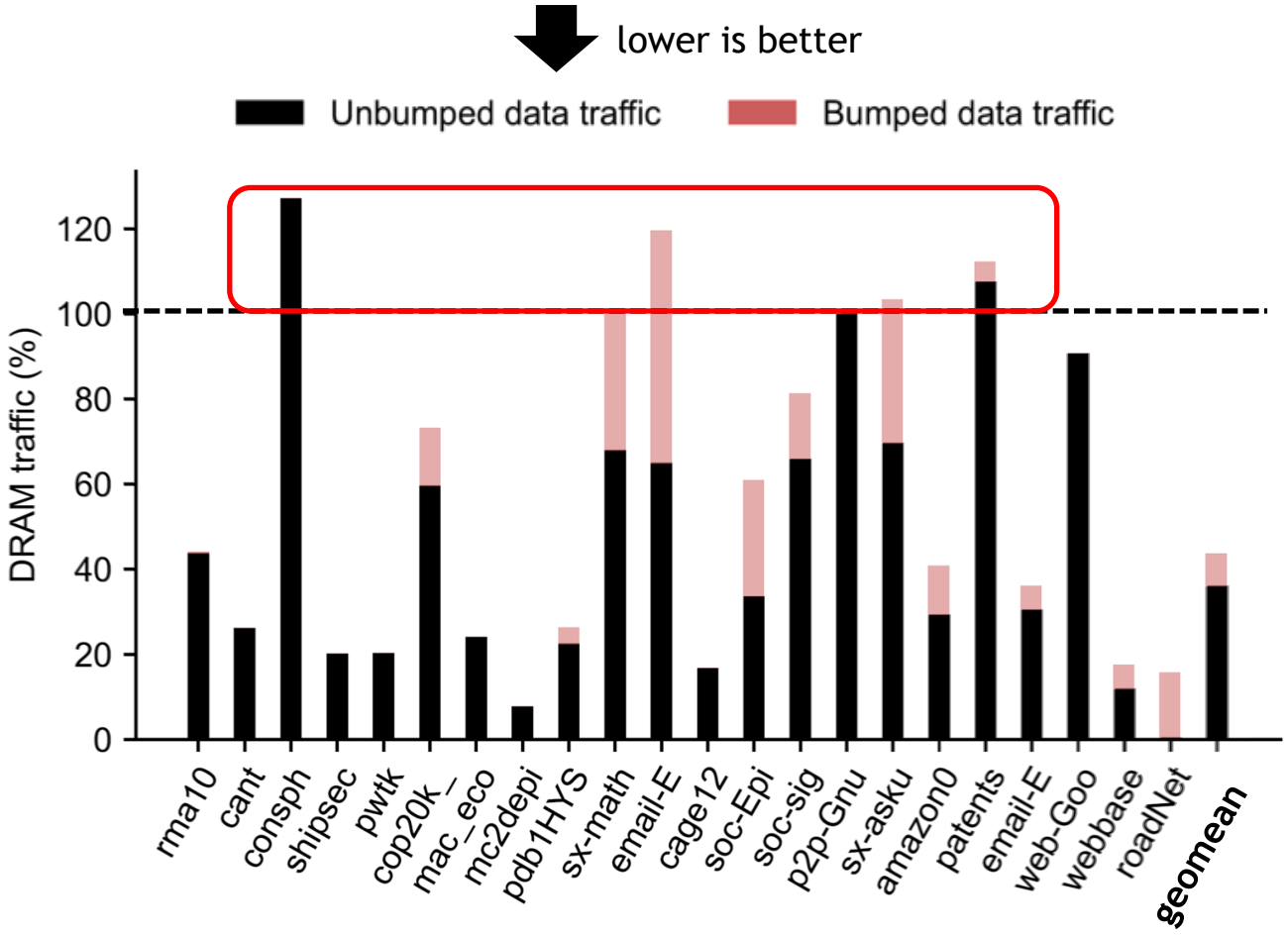
Results – Energy Efficiency relative to ExTensor-N



22.5x more energy efficient than ExTensor-N, 2.5x more than ExTensor-P

Results – DRAM traffic compared to ExTensor-P

- Bumped data has no data reuse while unbumped data does
- Increase in DRAM traffic due to streaming bumped data is offset by reduced DRAM traffic from larger tiles



Key Takeaways

http://emze.csail.mit.edu/tailors

Overbooking

Intentionally under-provisioning buffer capacity can improve buffer utilization and reduce DRAM traffic

Tailors

Dynamically splitting buffer for bumped data maintains data reuse without additional area

Swiftiles

Samples tensor to estimate tile size with low preprocessing cost

This work was funded in part by the MIT AI Hardware Program