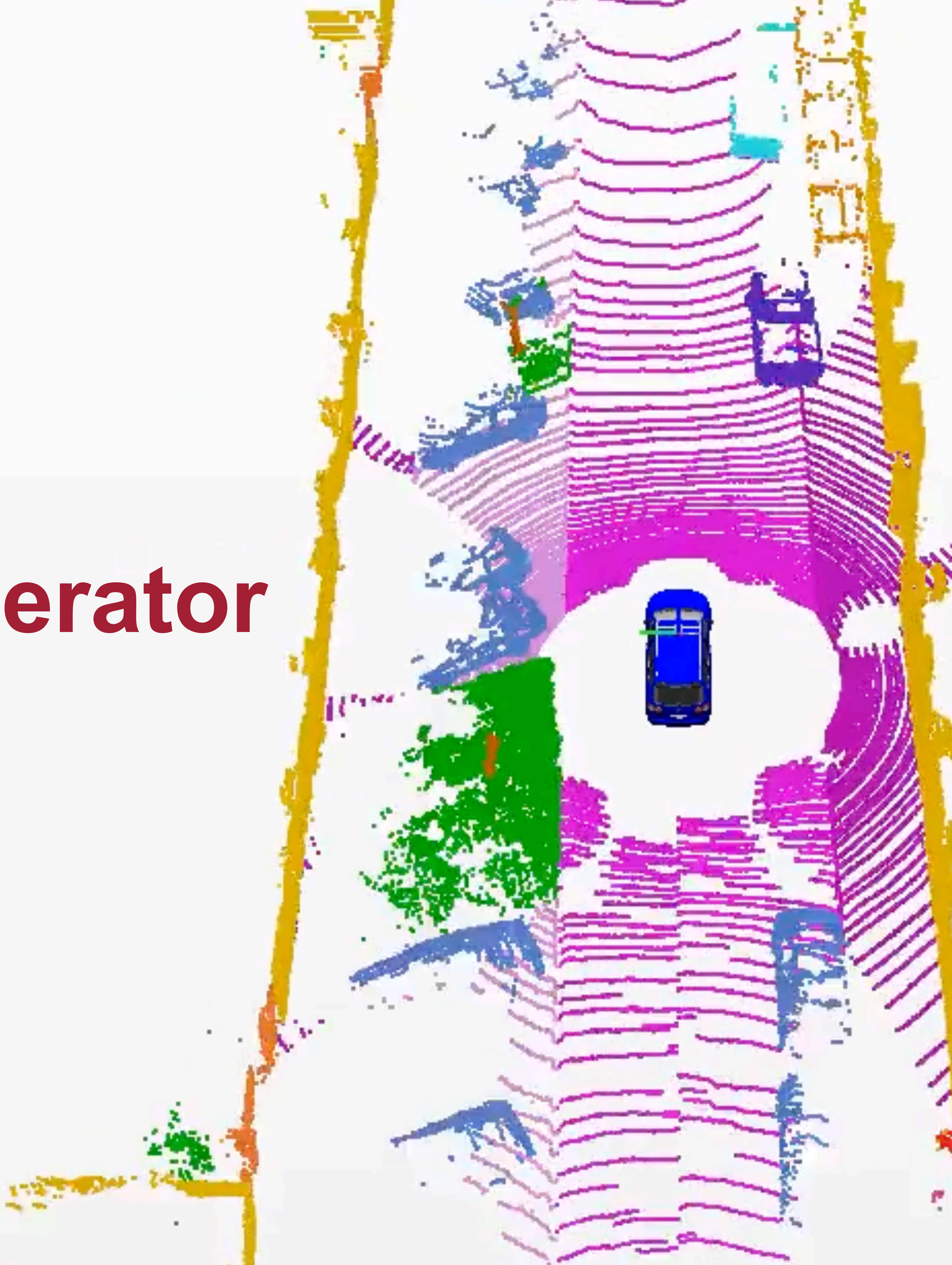
PointAcc
Efficient Point Cloud Accelerator

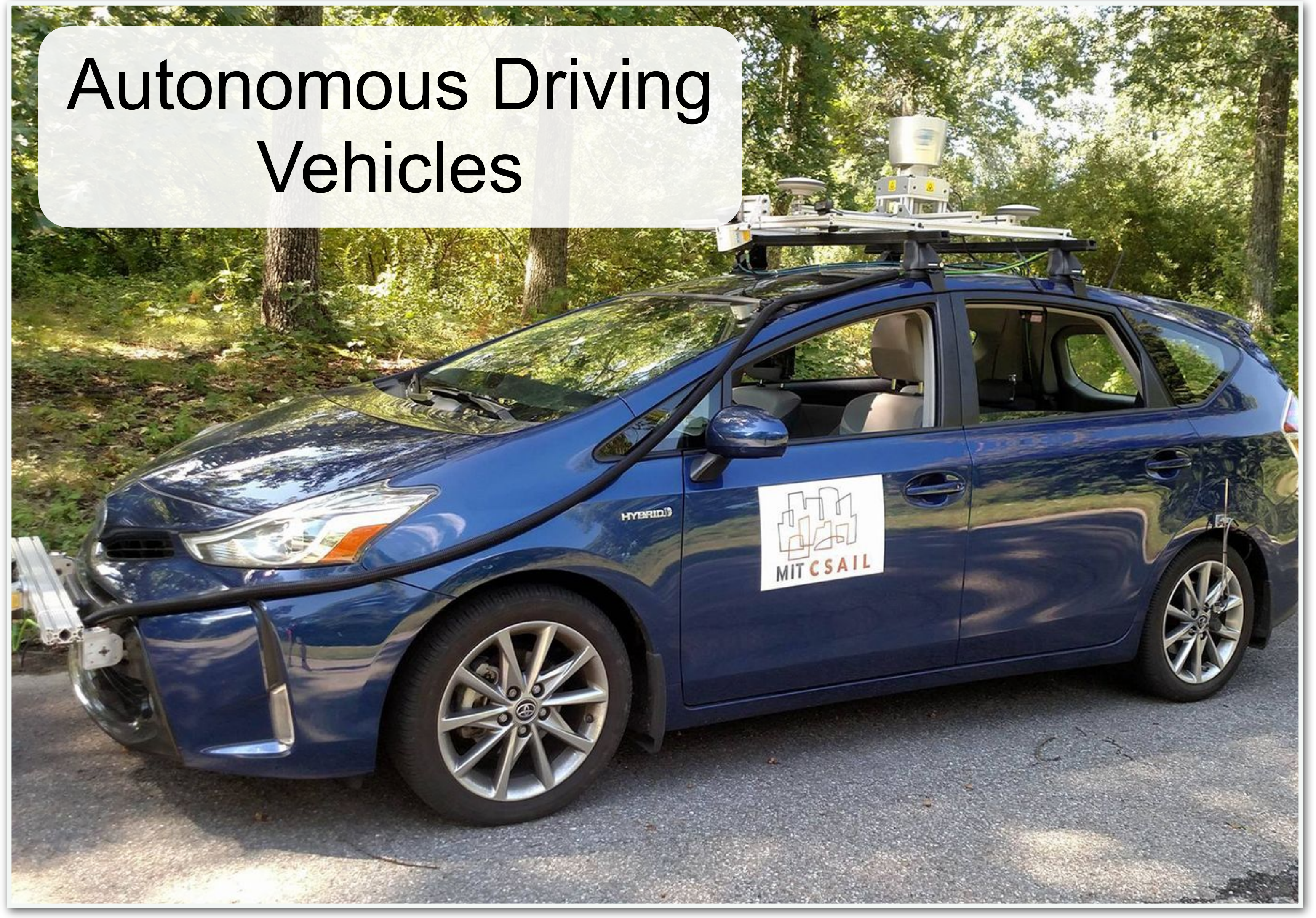
Yujun Lin, Zhekai Zhang, Haotian Tang,
Hanrui Wang, and Song Han

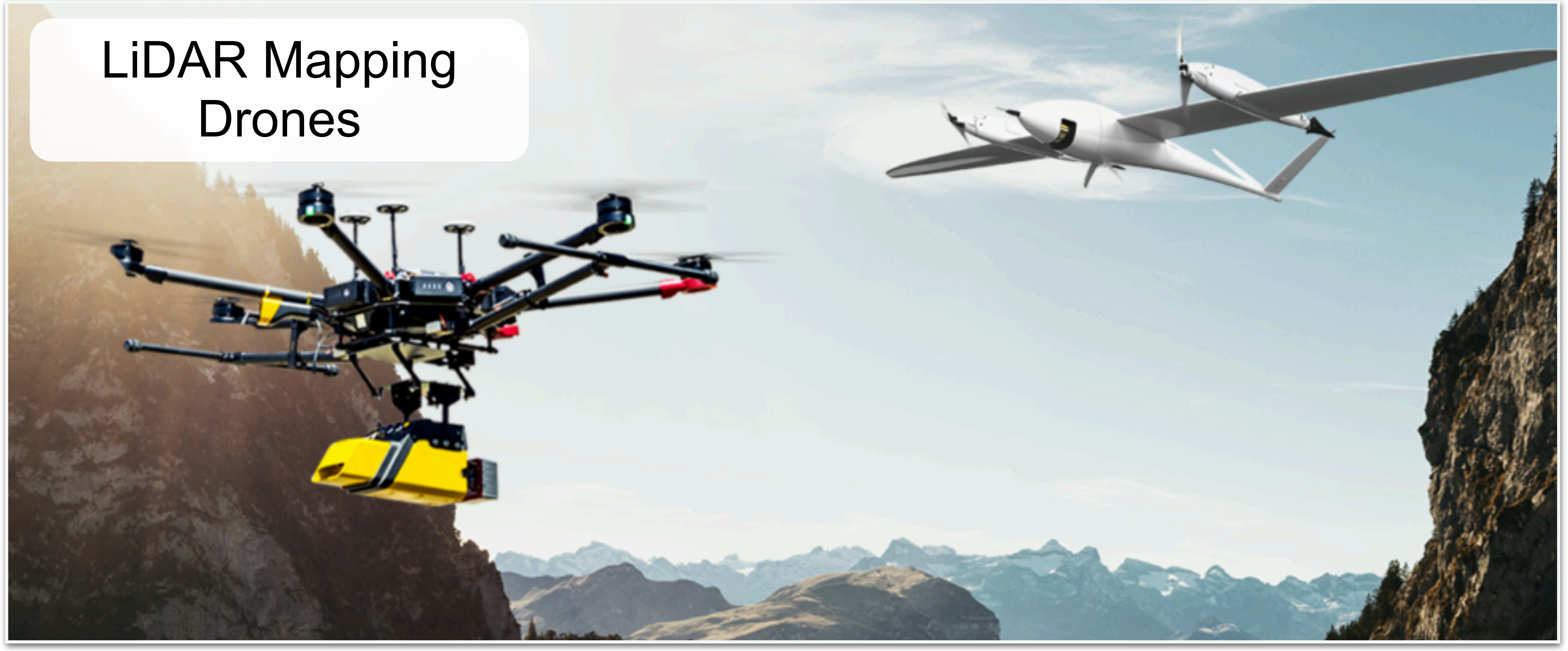


https://hanlab.mit.edu/projects/pointacc
video source: http://www.semantic-kitti.org/


2D Image
3D Point Cloud
video source: http://www.cvlibs.net/datasets/kitti/

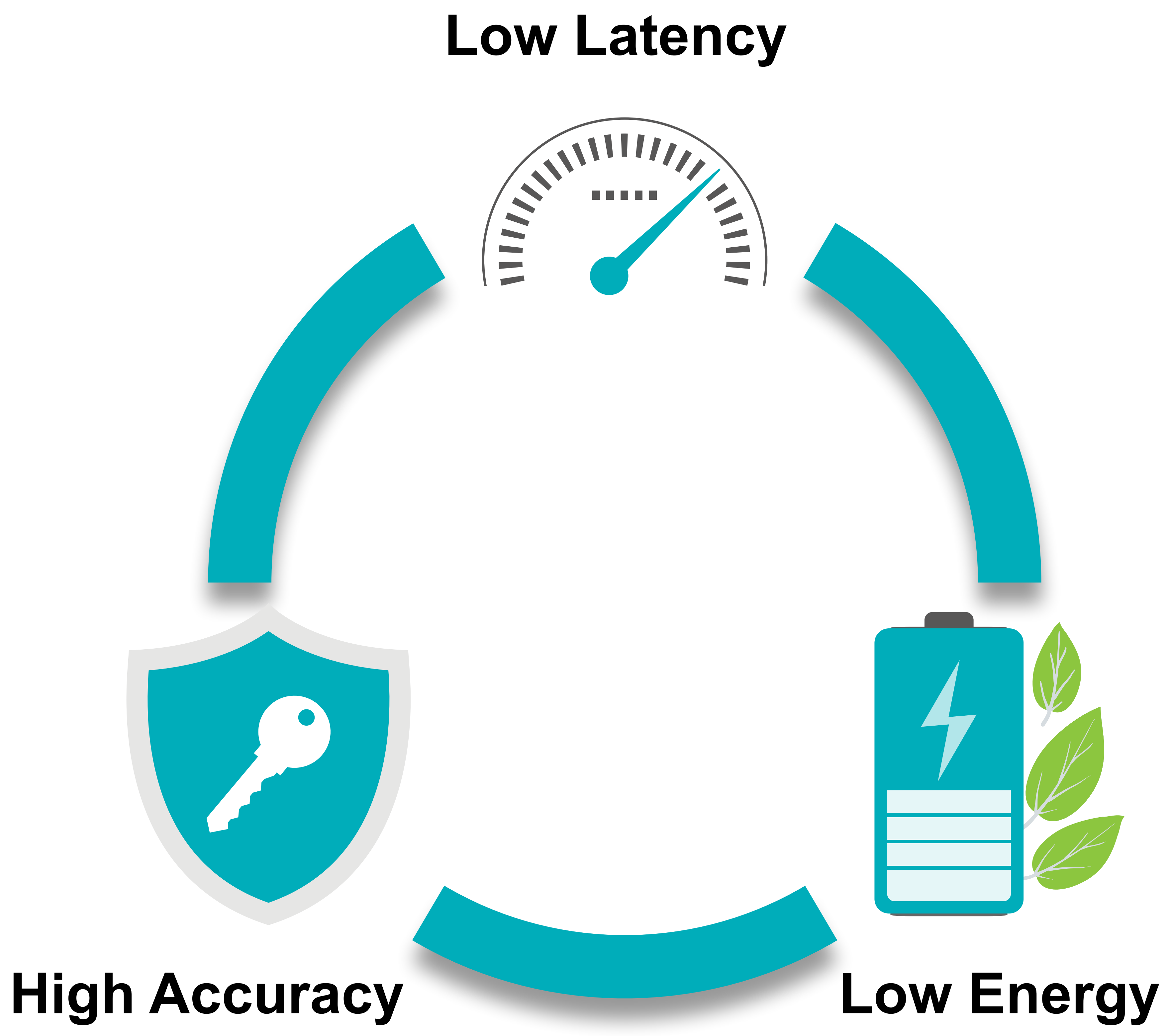
Point Cloud Deep Learning are Everywhere





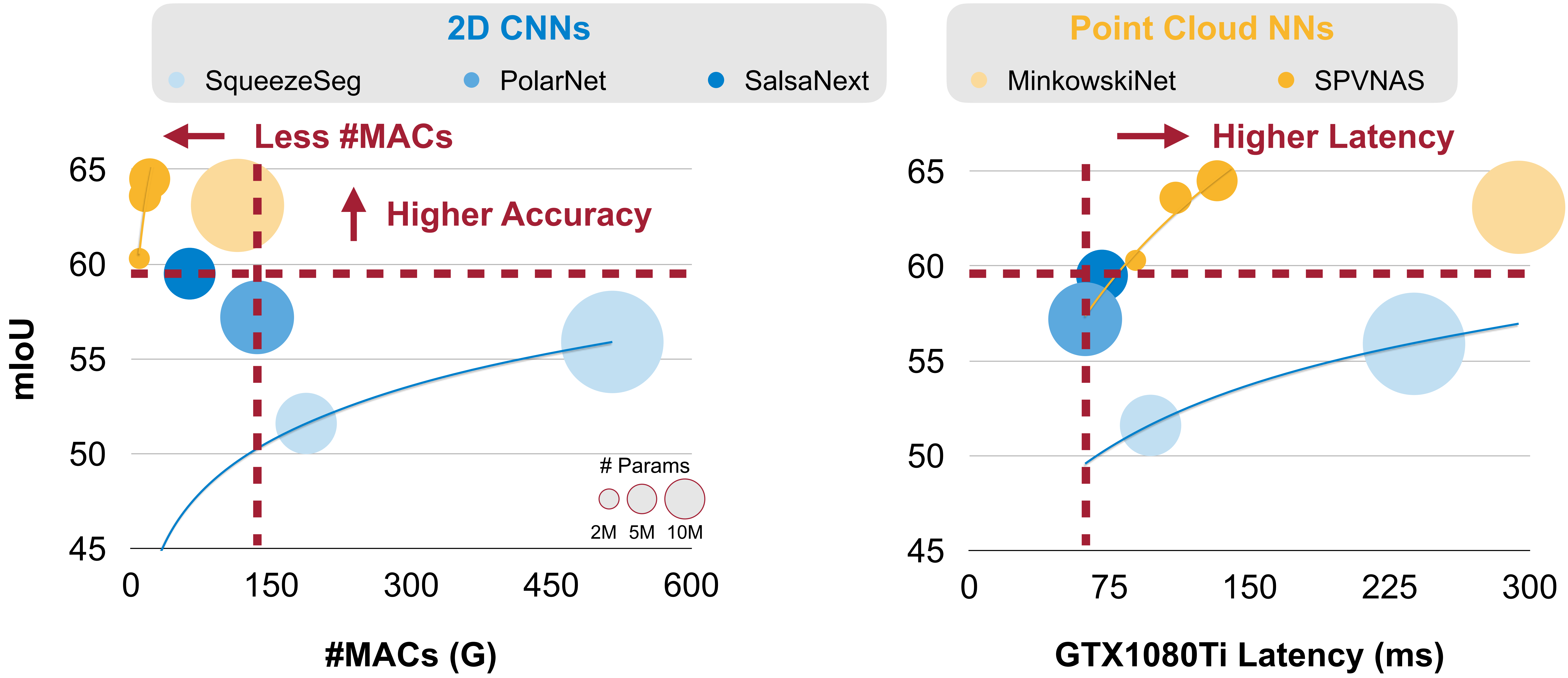
Efficiency and Safety are Important

Safety



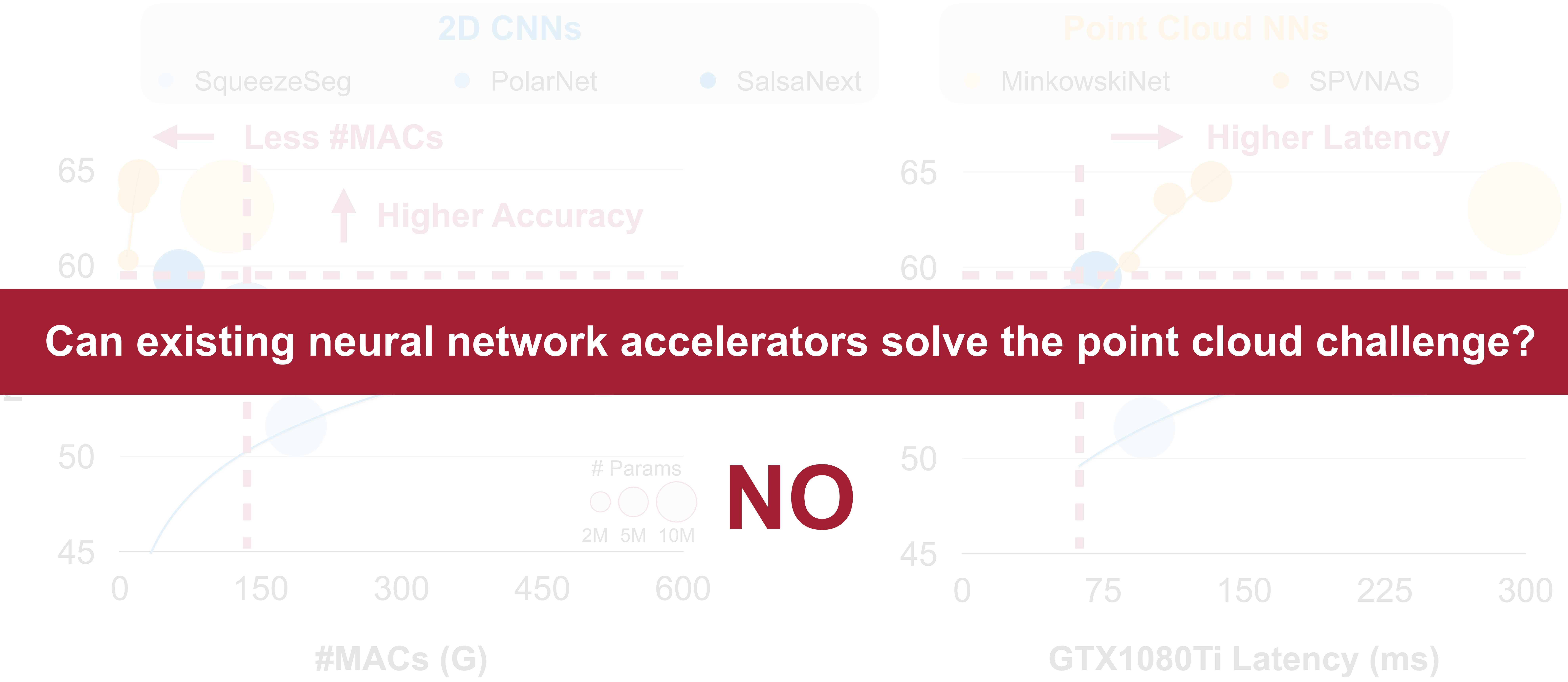
Efficiency

Efficiency and Safety are Important



Point Cloud Networks have **higher accuracy**, but **cannot run efficiently** on today's GPUs (7× less #MACs but 1.3× slower).

Efficiency and Safety are Important



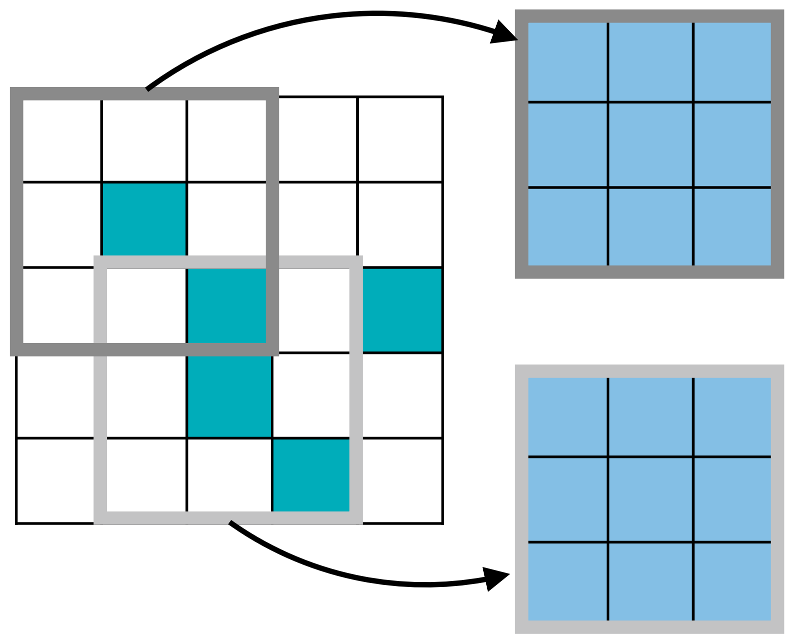
Can existing neural network accelerators solve the point cloud challenge?

NO

Point Cloud Networks have **higher accuracy**, but **cannot run efficiently** on today's GPUs (7× less #MACs but 1.3× slower).

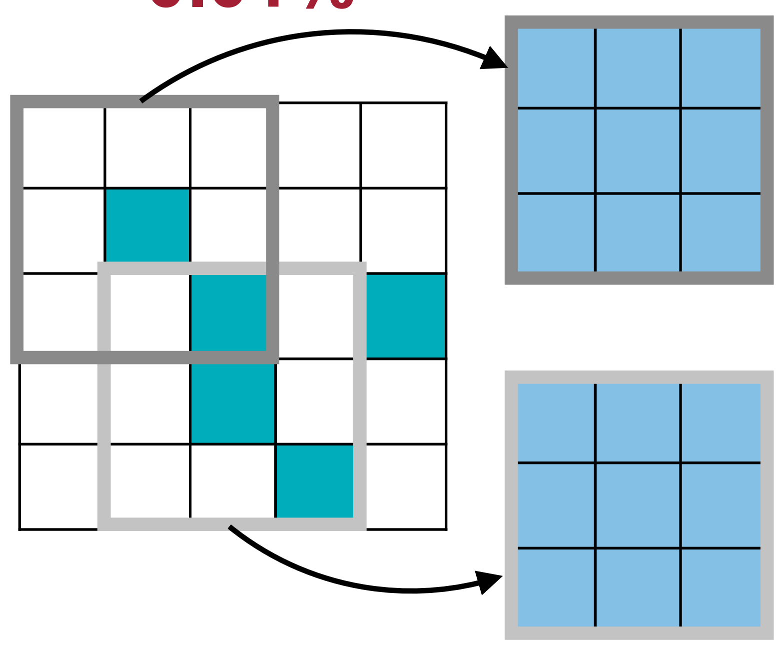
Point Cloud Convolution is Different from Convention

Conventional Convolution

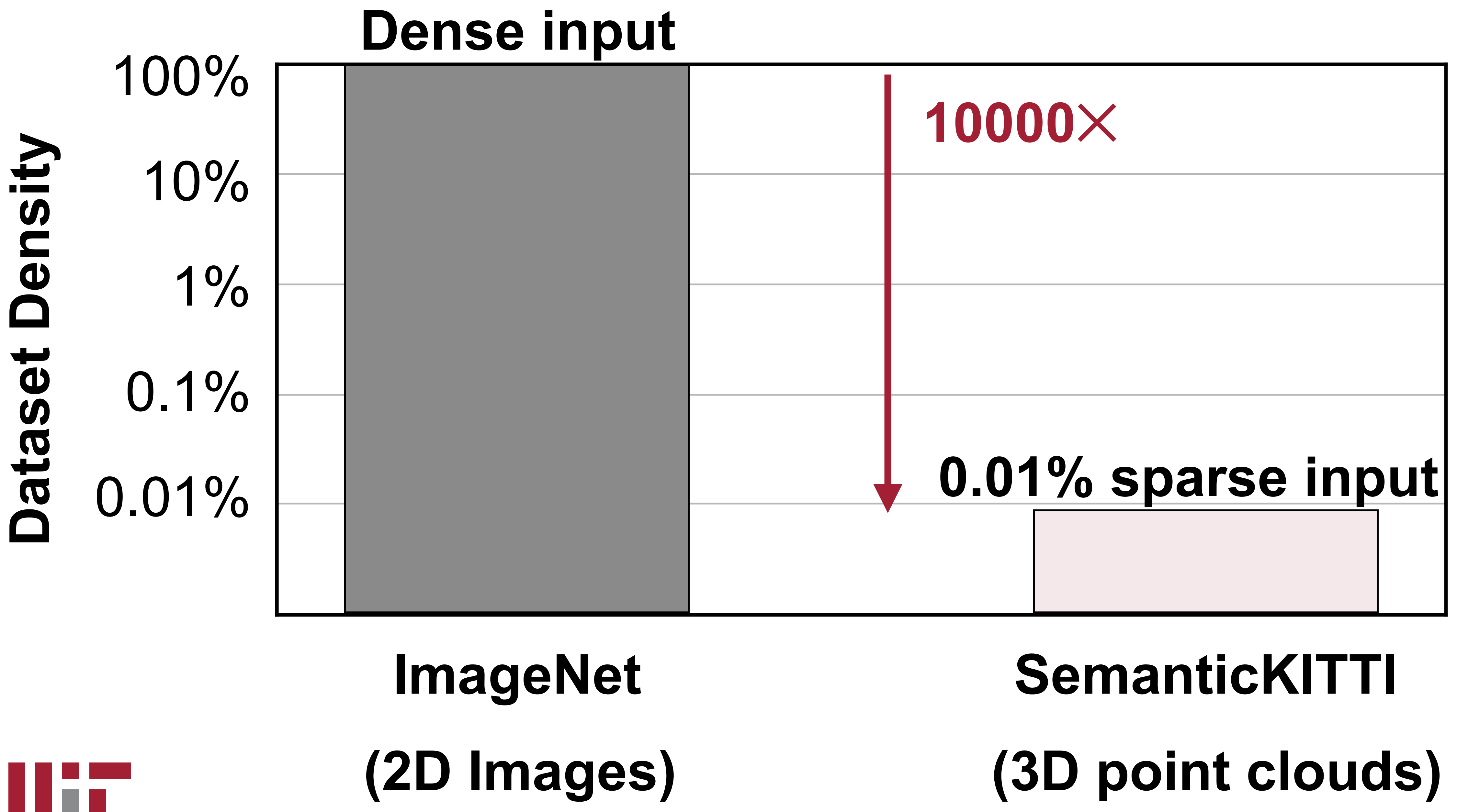


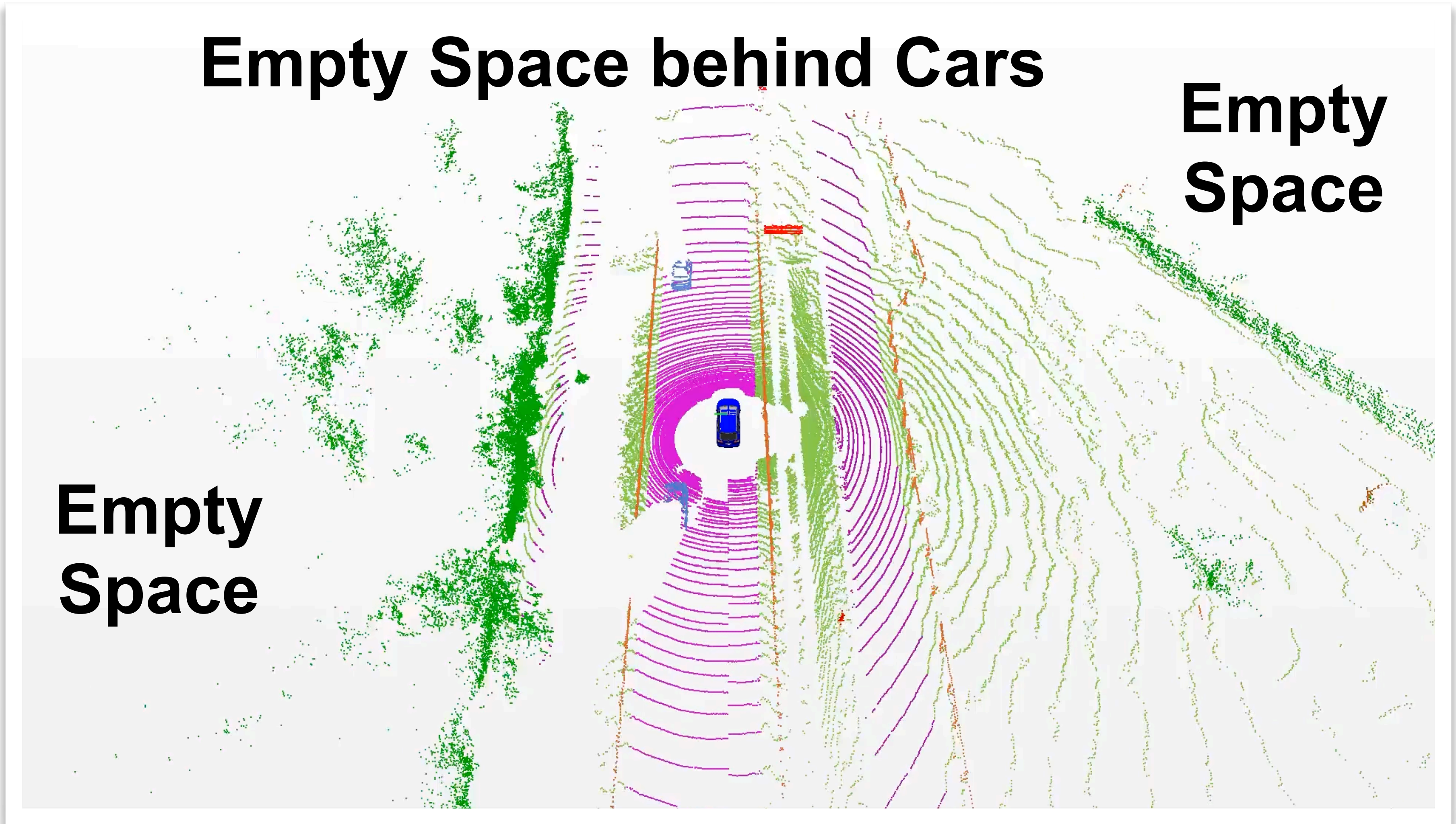
Input sparsity
is from ReLU

Point Cloud Convolution



Input sparsity is
from the distribution
in physical space

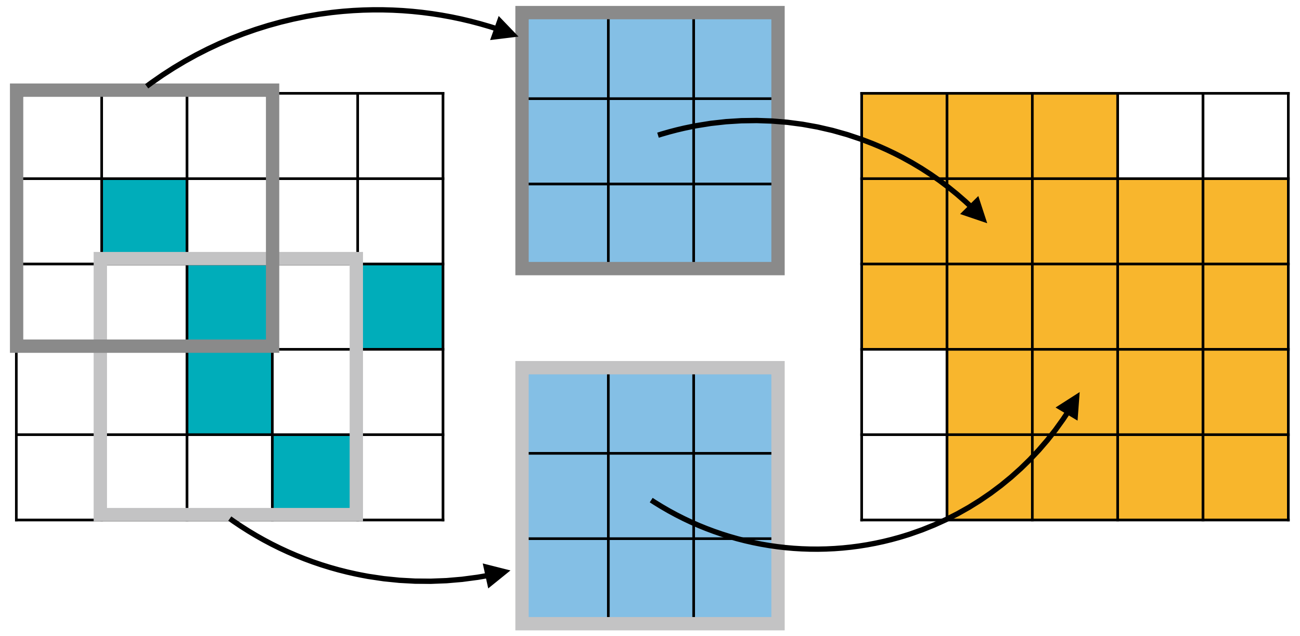




video source: http://www.semantic-kitti.org/

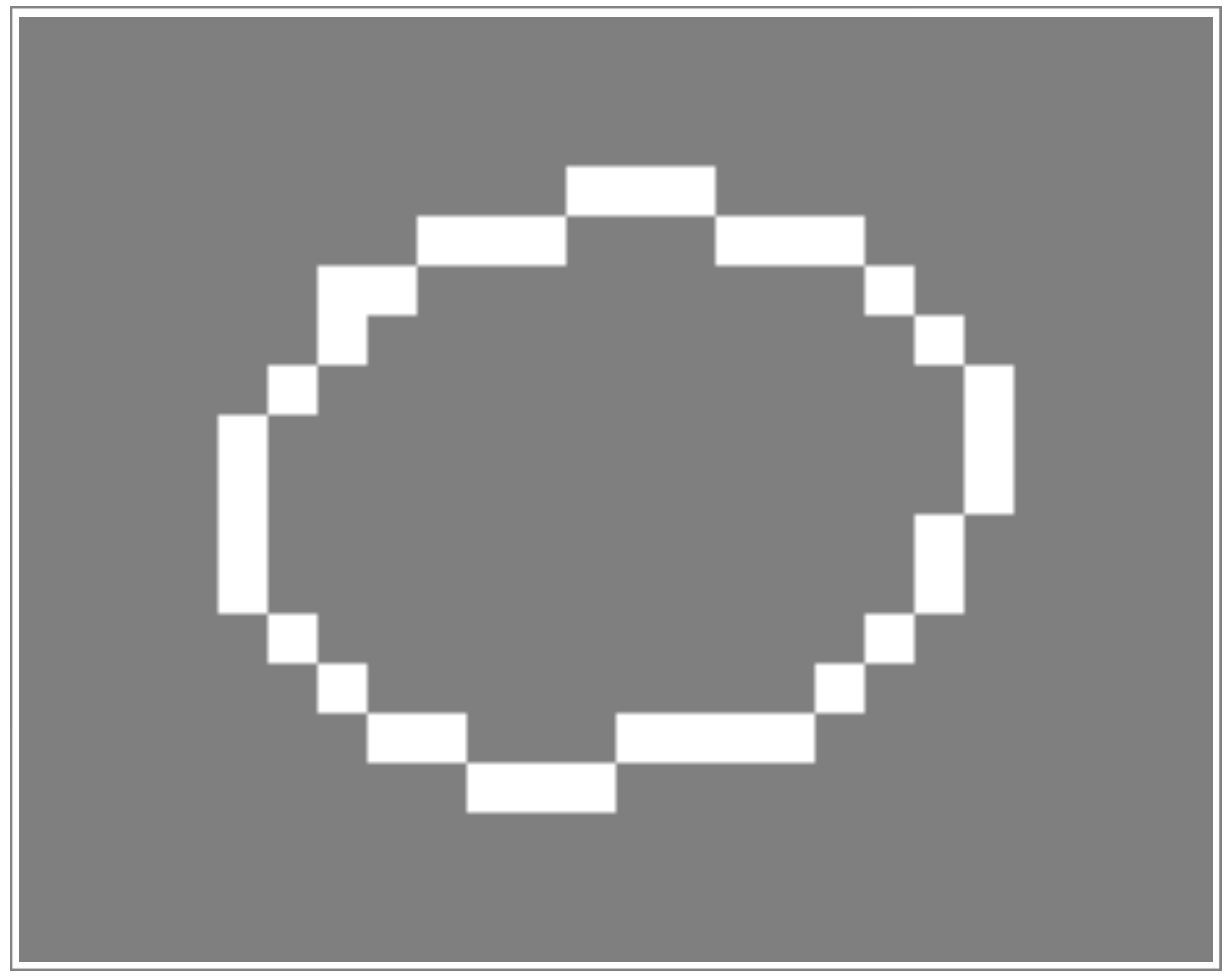
Point Cloud Convolution is Different from Convention

Conventional Convolution

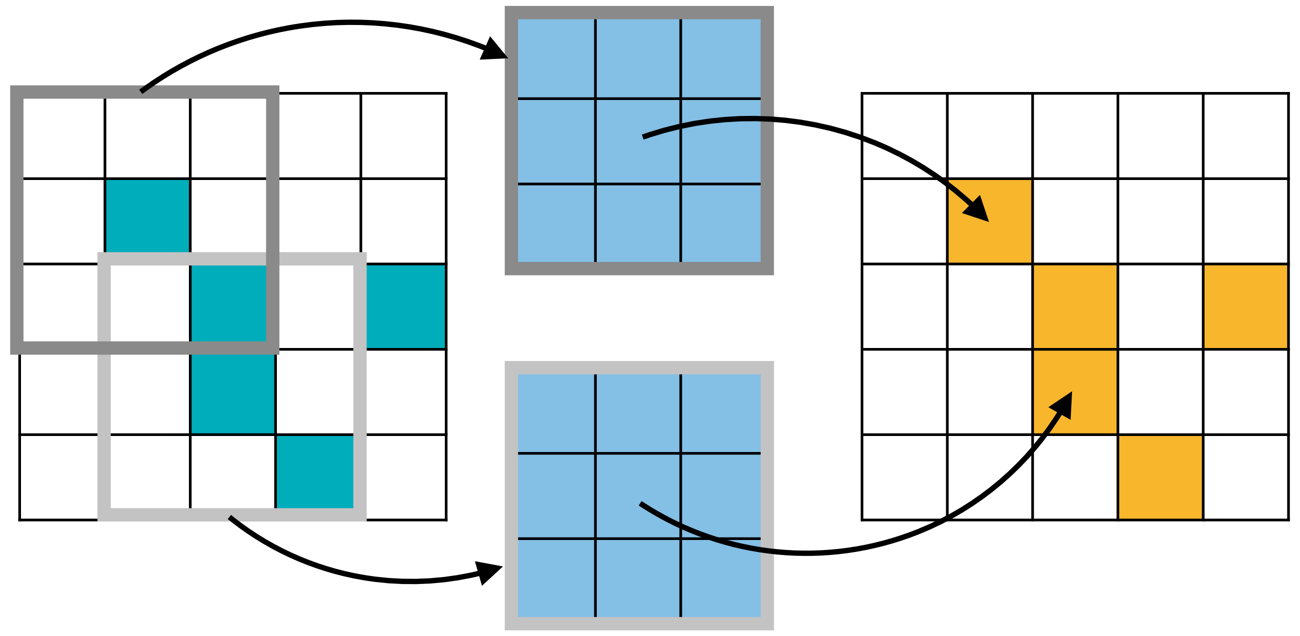


Input sparsity is from ReLU

Nonzeros will dilate

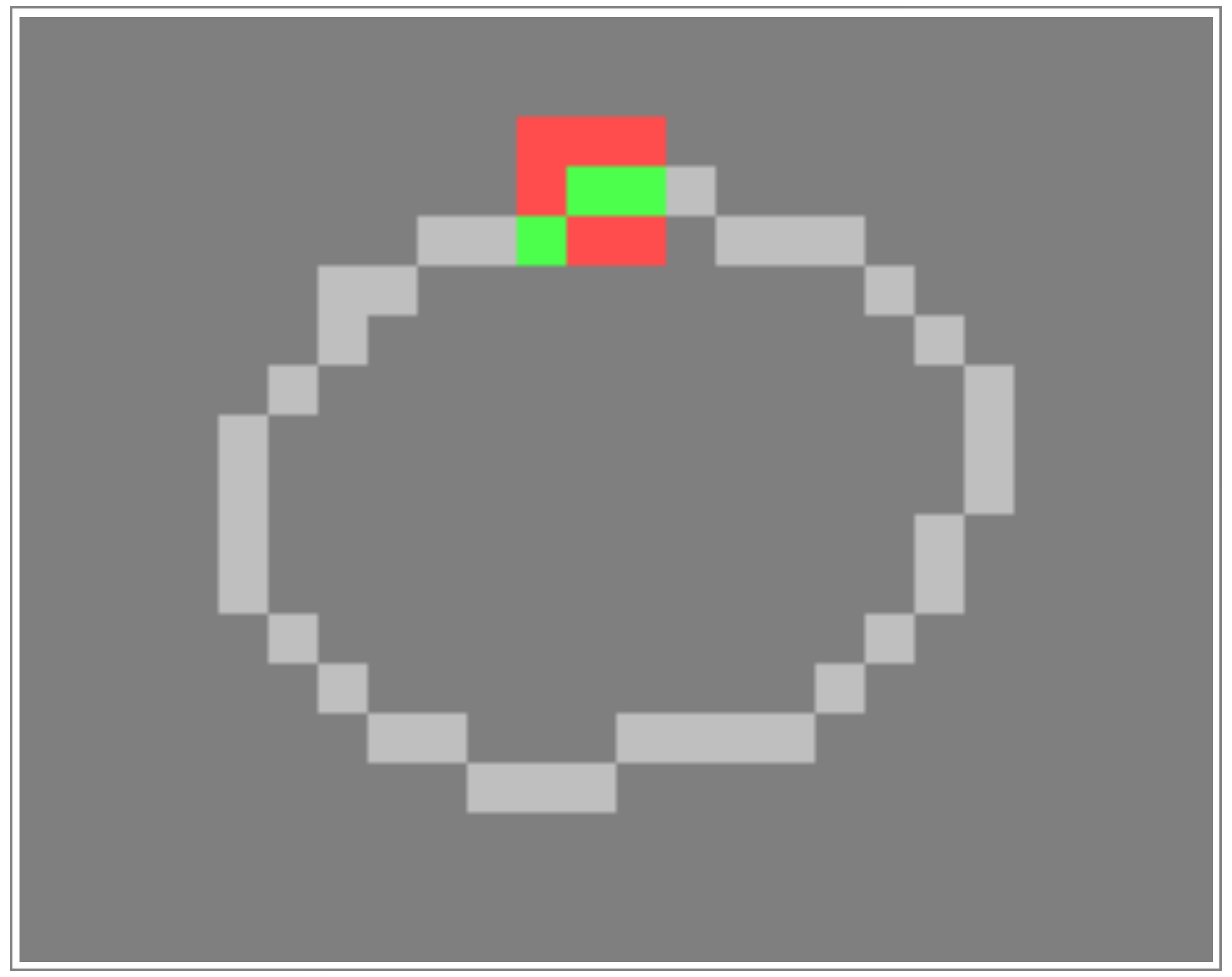


Point Cloud Convolution



Input sparsity is from the distribution in physical space

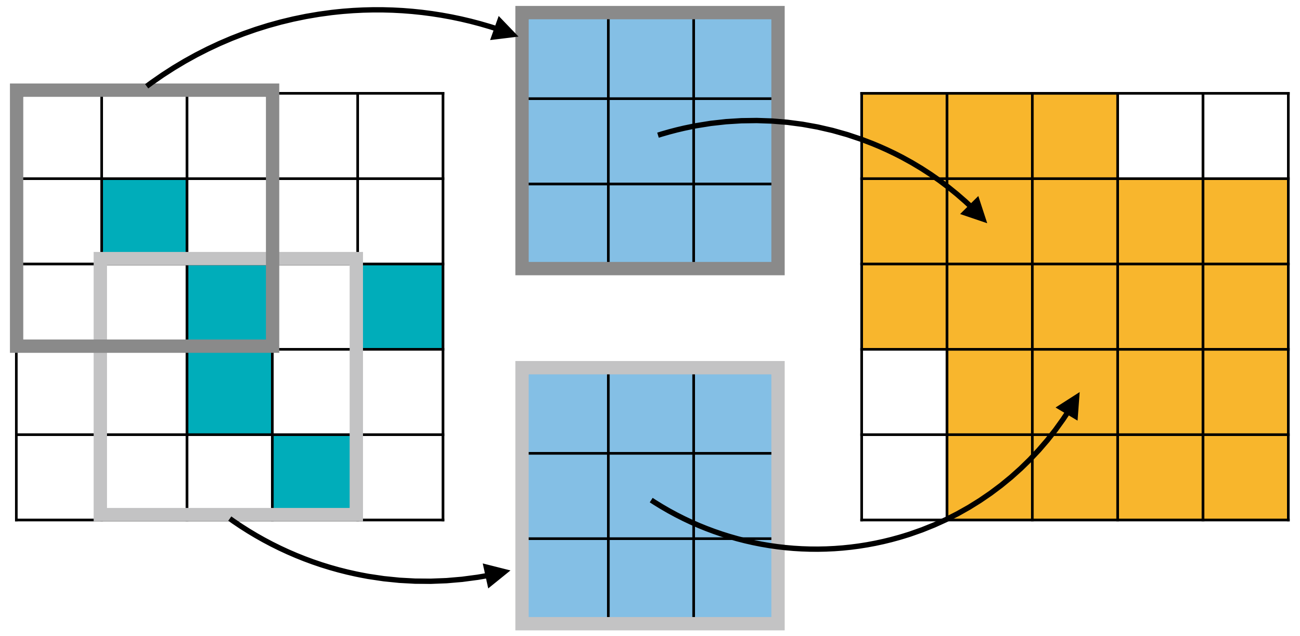
Nonzeros will not dilate



video source: https://github.com/facebookresearch/SparseConvNet

Point Cloud Convolution is Different from Convention

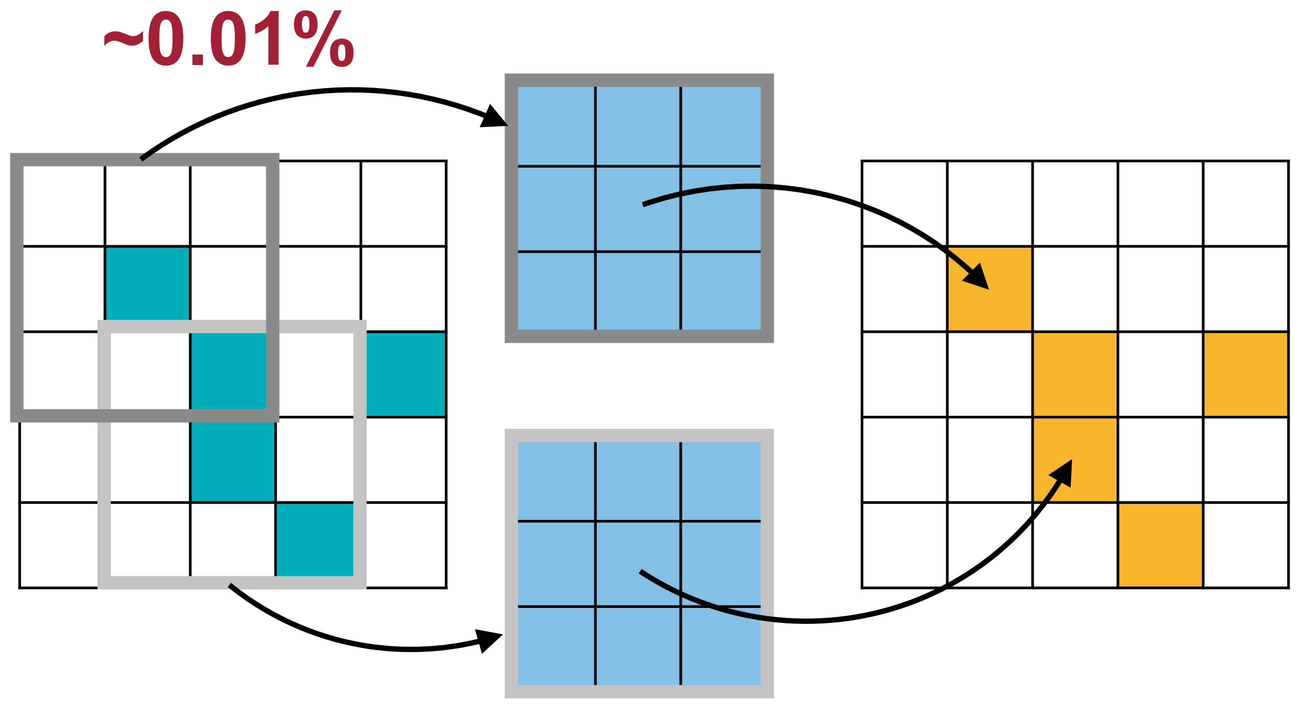
Conventional Convolution



Input sparsity
is from ReLU

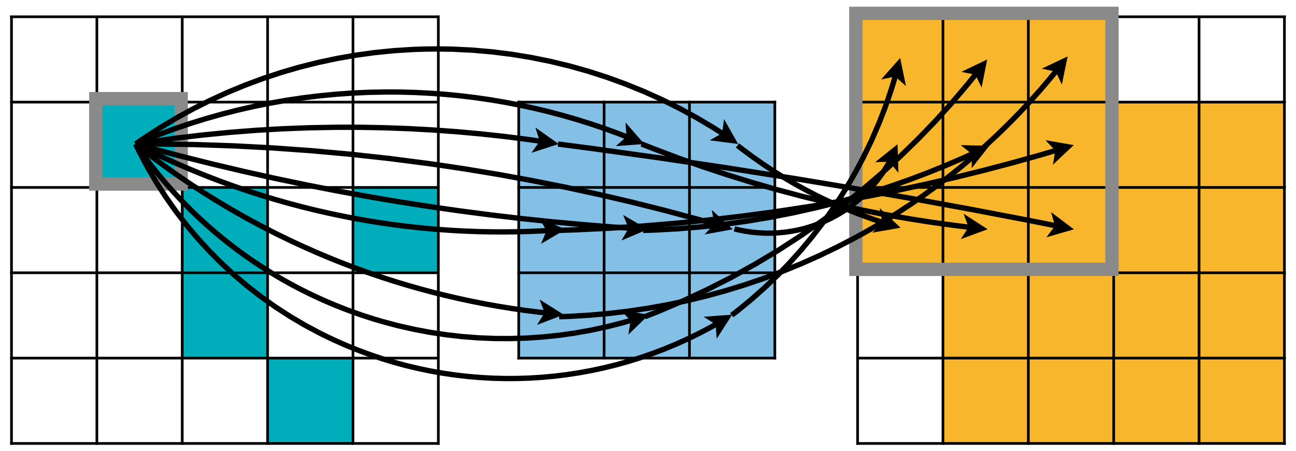
Nonzeros
will dilate

Point Cloud Convolution

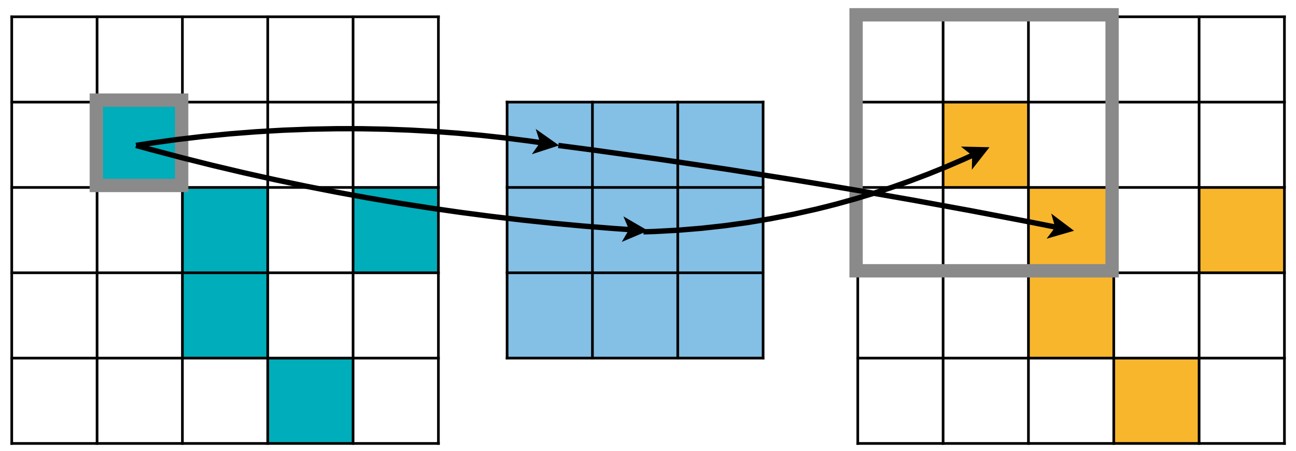


Input sparsity is
from the distribution
in physical space

Nonzeros
will not dilate

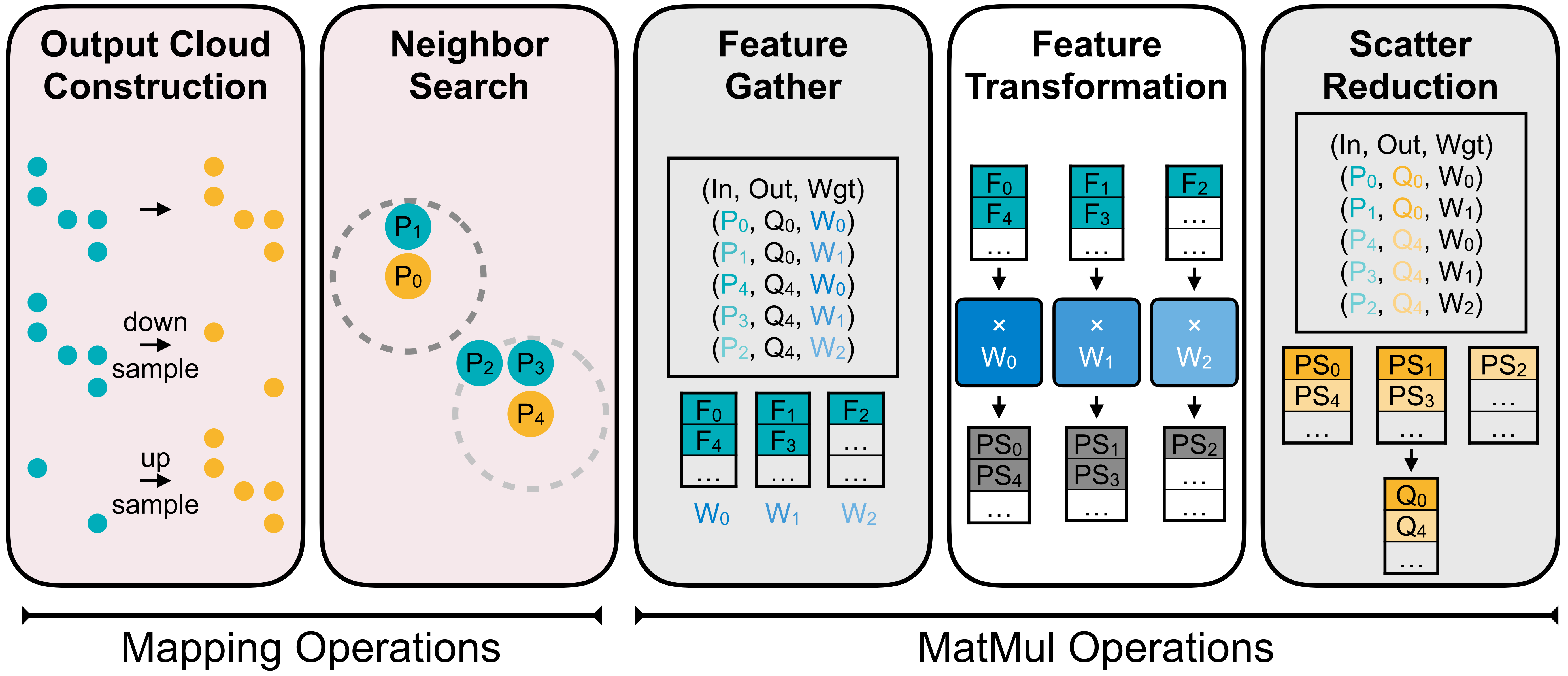


Each nonzero input is multiplied
with all nonzero weights



Each nonzero input is not
multiplied with all nonzero weights

Point Cloud Convolution is Different from Convention



Sparsity in Physical Space
Input sparsity is from the point distribution in physical space

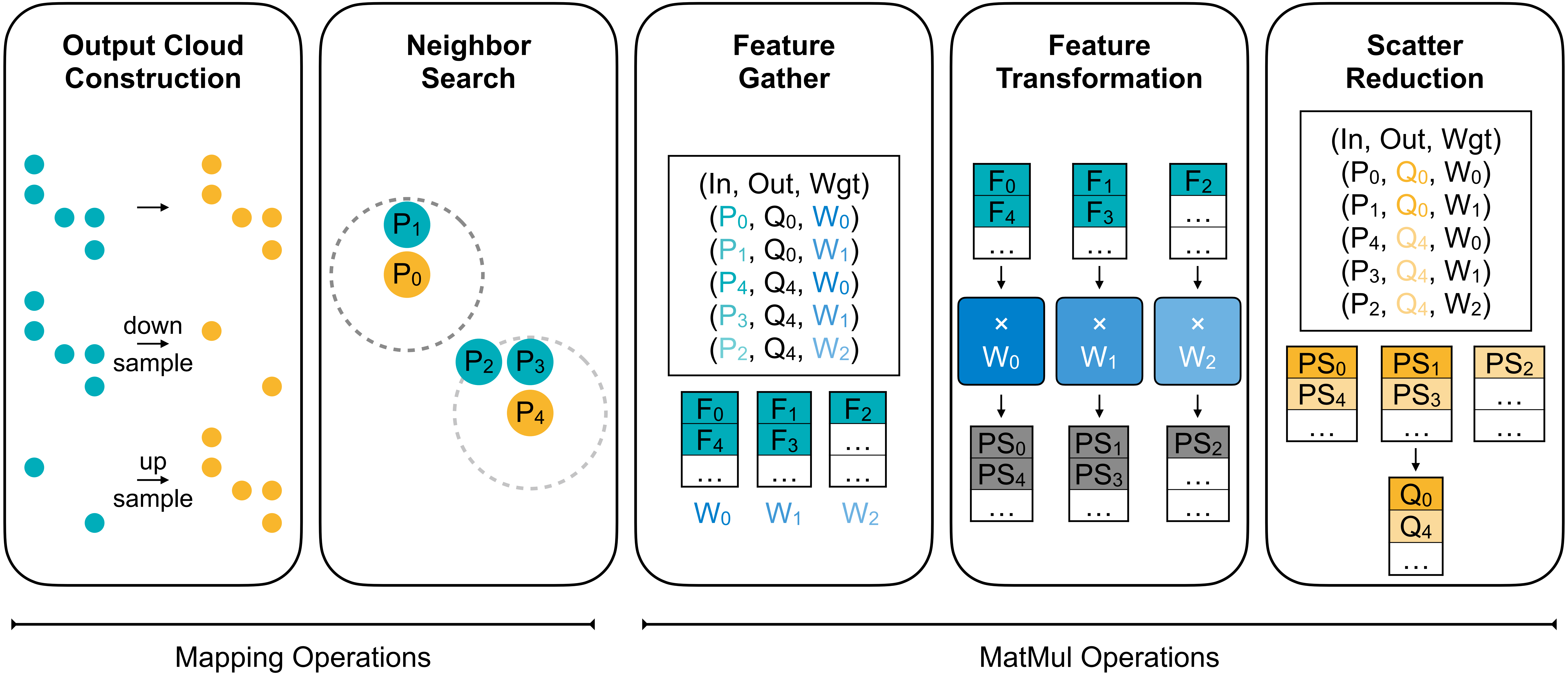
Nonzeros will not dilute

Irregular and sparse computation pattern
Each nonzero input are not multiplied with all nonzero weights

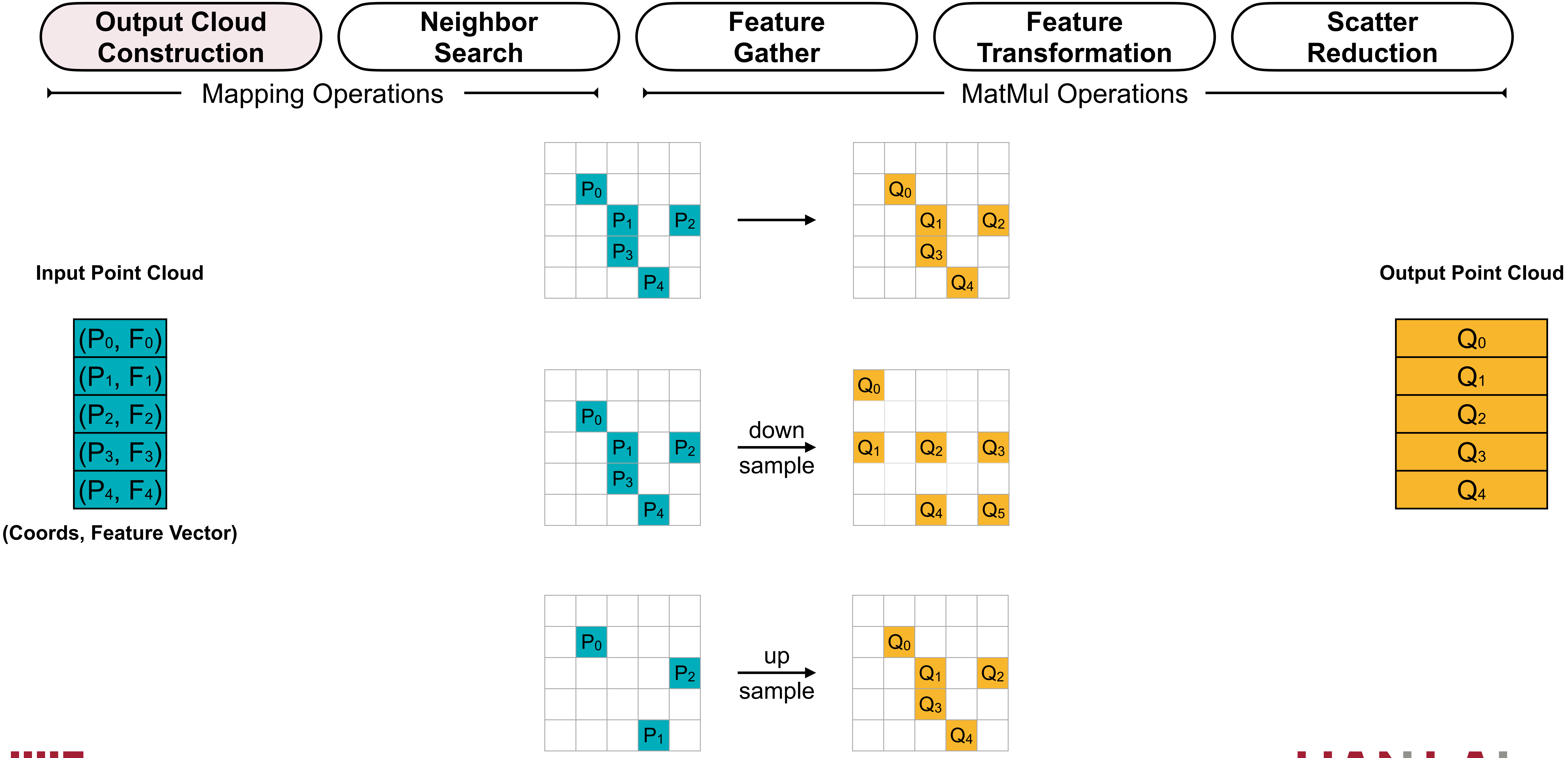
New operations to find neighbors
Mapping operations

Explicit data movement overhead
Gather features and Scatter psums

Introduction to Point Cloud Convolution



Step 1: Build Output Point Cloud



Step 2: Search Neighbors

Output Cloud Construction → **Neighbor Search** → **Feature Gather** → **Feature Transformation** → **Scatter Reduction**

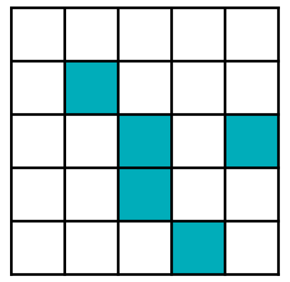
Mapping Operations (Output Cloud Construction, Neighbor Search) | MatMul Operations (Feature Gather, Feature Transformation, Scatter Reduction)

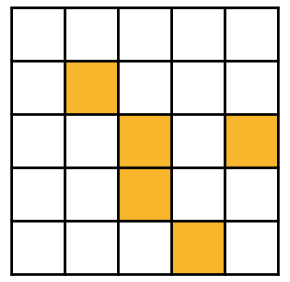
For each *output* point, we find the *input* points in its neighborhood and generate **(Input Point, Output Point, Weight Index)** maps

Input Point Cloud

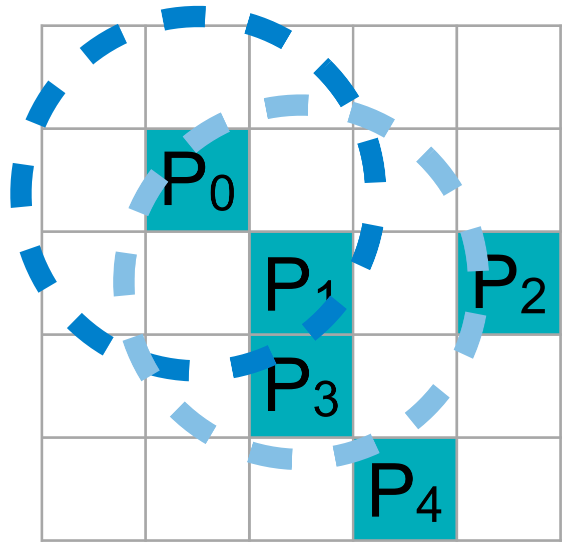
(P_0, F_0)
(P_1, F_1)
(P_2, F_2)
(P_3, F_3)
(P_4, F_4)

(Coords, Feature Vector)

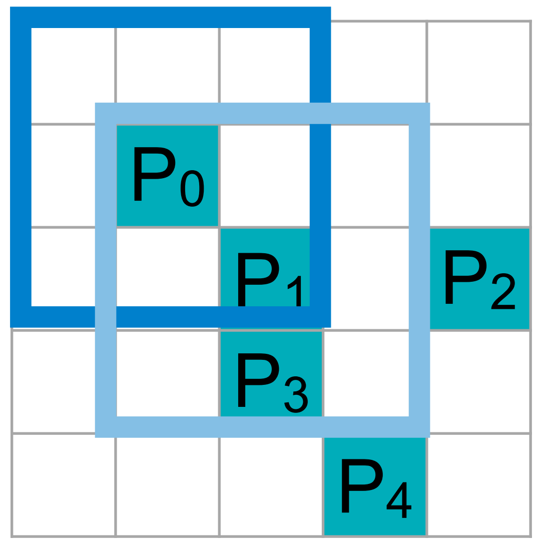




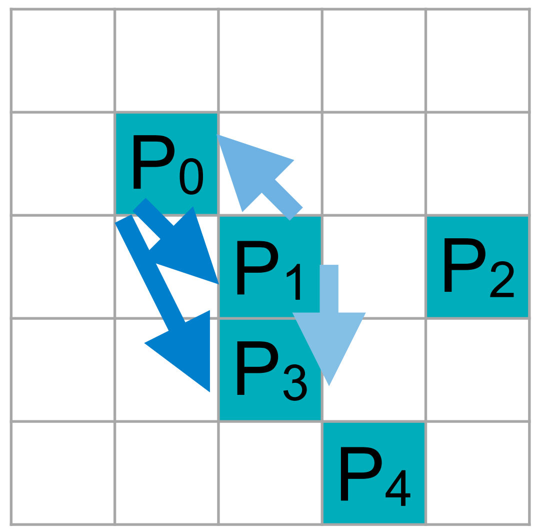
Neighborhood Shape		
Ball	**Cube**	**Nearest Neighbor**
Ball Query	Kernel Mapping	k-Nearest-Neighbors



(In, Out, Wgt)
(P_0, Q_0, W_0)
(P_1, Q_0, W_1)
(P_0, Q_1, W_0)
(P_3, Q_1, W_1)
…



(In, Out, Wgt)
$(P_0, Q_0, W_{0,0})$
$(P_1, Q_0, W_{1,1})$
$(P_0, Q_1, W_{-1,-1})$
$(P_1, Q_1, W_{0,0})$
$(P_3, Q_1, W_{1,0})$



(In, Out, Wgt)
(P_0, Q_0, W_0)
(P_1, Q_0, W_1)
(P_3, Q_0, W_2)
(P_1, Q_1, W_0)
(P_3, Q_1, W_1)

Output Point Cloud

Q_0
Q_1
Q_2
Q_3
Q_4

Step 3: Gather Input Features

Step 4: Transform Input Features

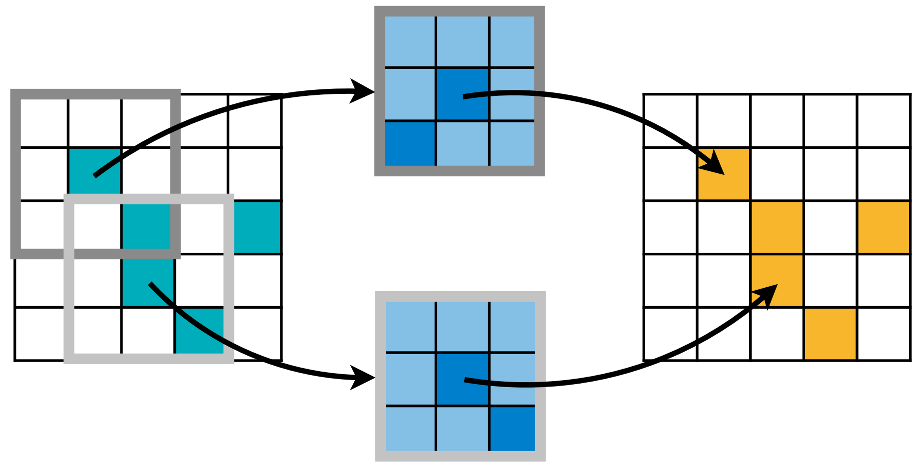
Output Cloud Construction | **Neighbor Search** | **Feature Gather** | **Feature Transformation** | **Scatter Reduction**

Mapping Operations (Output Cloud Construction, Neighbor Search) — MatMul Operations (Feature Gather, Feature Transformation, Scatter Reduction)

Input Point Cloud

(P_0, F_0)
(P_1, F_1)
(P_2, F_2)
(P_3, F_3)
(P_4, F_4)

(Coords, Feature Vector)



Maps (In, Out, Wgt)
(P_0, Q_1, $W_{-1,-1}$)
(P_3, Q_4, $W_{-1,-1}$)
(P_1, Q_3, $W_{-1,0}$)
(P_0, Q_0, $W_{0,0}$)
(P_1, Q_1, $W_{0,0}$)
(P_2, Q_2, $W_{0,0}$)
(P_3, Q_3, $W_{0,0}$)
(P_4, Q_4, $W_{0,0}$)
(P_3, Q_1, $W_{1,0}$)
(P_1, Q_0, $W_{1,1}$)
(P_4, Q_3, $W_{1,1}$)

Gather By Weight

$[F_0; F_3] \times W_{-1,-1} = [Psum_1; Psum_4]$

$[F_1] \times W_{-1,0} = [Psum_3]$

$[F_0; F_1; F_2; F_3; F_4] \times W_{0,0} = [Psum_0; Psum_1; Psum_2; Psum_3; Psum_4]$

$[F_3] \times W_{1,0} = [Psum_1]$

$[F_1; F_4] \times W_{1,1} = [Psum_0; Psum_3]$

p-by-*ic*   *ic*-by-*oc*   *p*-by-*oc* × **n**

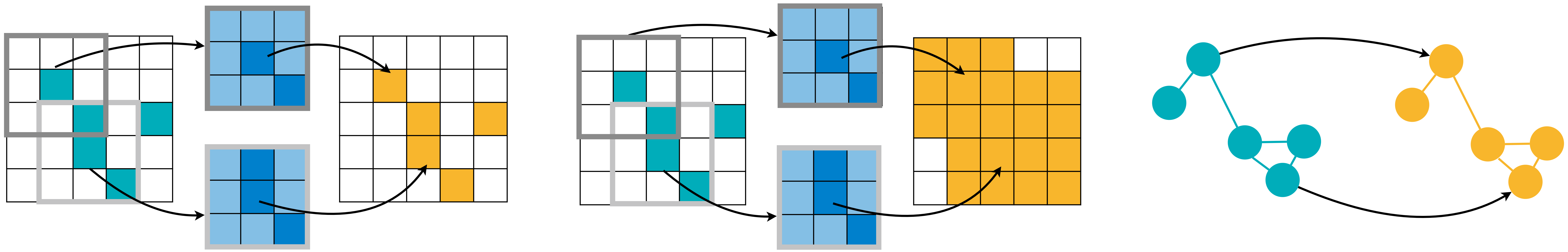
Matrix-Matrix Multiplication

Output Point Cloud

Q_0
Q_1
Q_2
Q_3
Q_4

Step 5: Scatter and Reduce Partial Sums

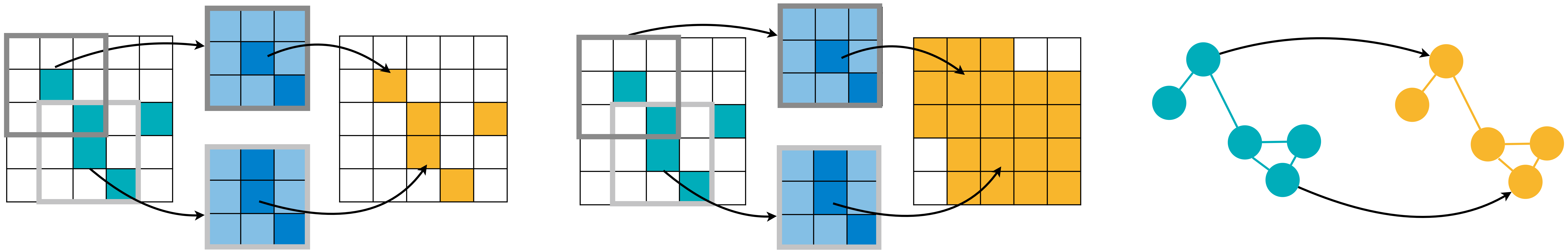
Point Cloud Deep Learning is Different



	Point Cloud NN	CNN	Graph CNN
source of sparsity	the distribution of points in physical space	ReLU and weight pruning	-
input sparsity	**nonzeros do not dilate**; input sparsity **keeps low**	nonzeros dilate; input sparsity quickly reduces	-

- Extreme low utilization on existing sparse CNN accelerators

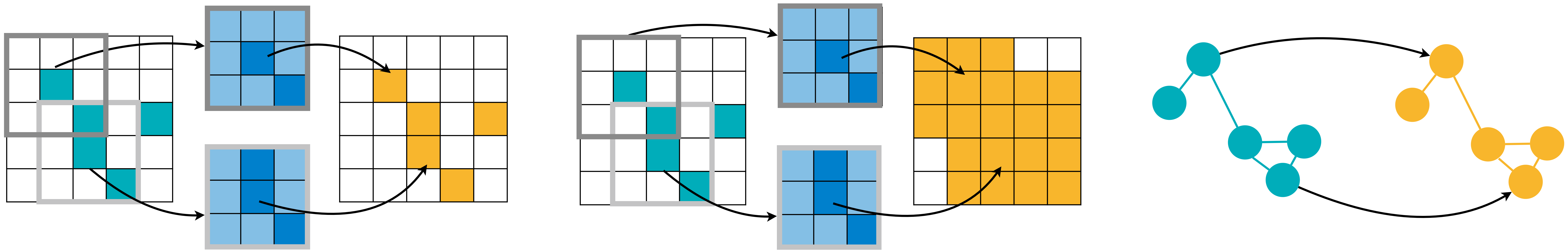
Point Cloud Deep Learning is Different



	Point Cloud NN	CNN	Graph CNN
source of sparsity	the distribution of points in physical space	ReLU and weight pruning	-
input sparsity	**nonzeros do not dilate**; input sparsity **keeps low**	nonzeros dilate; input sparsity quickly reduces	-
receptive field	both outputs' coords and neighbors need to be **explicitly calculated**	outputs' and neighbors' coords are inferred by pointer arithmetic	neighbors are given from the adjacency matrix

- Existing NN accelerators do not support mapping ops

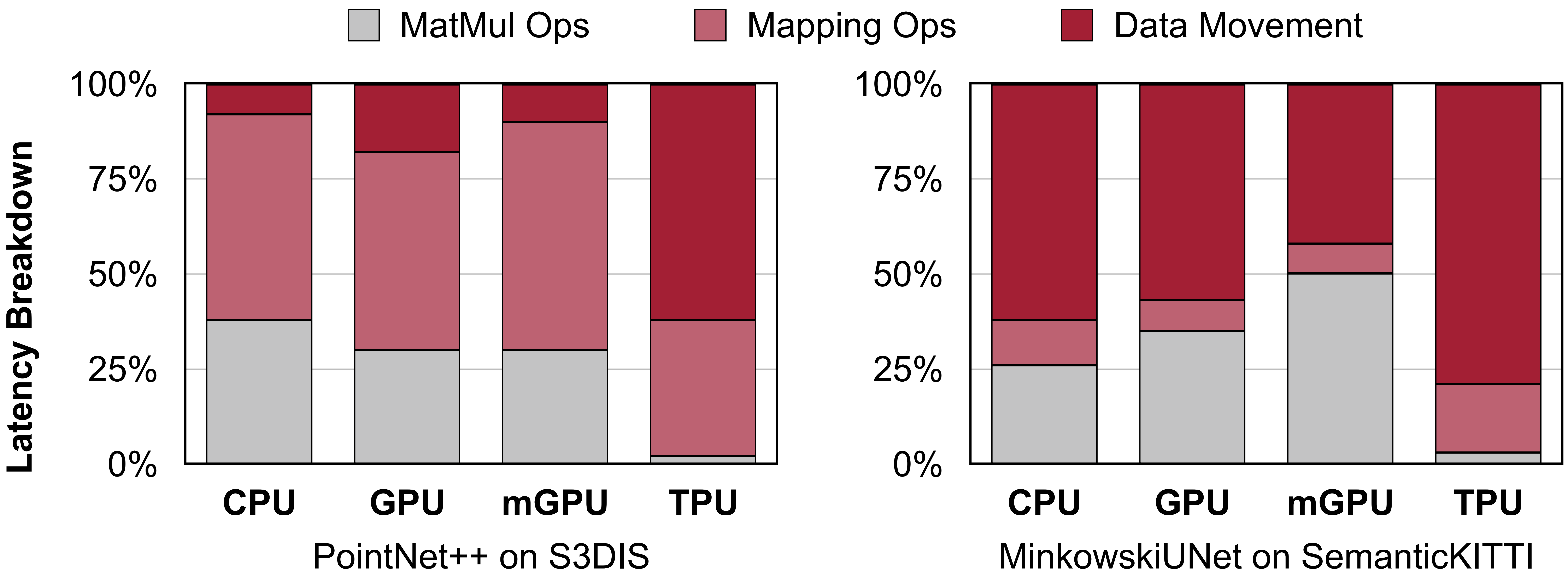
Point Cloud Deep Learning is Different



	Point Cloud NN	CNN	Graph CNN
source of sparsity	the distribution of points in physical space	ReLU and weight pruning	-
input sparsity	**nonzeros do not dilate**; input sparsity **keeps low**	nonzeros dilate; input sparsity quickly reduces	-
receptive field	both outputs' coords and neighbors need to be **explicitly calculated**	outputs' and neighbors' coords are inferred by pointer arithmetic	neighbors are given from the adjacency matrix
weights	weights can be shared or **different** for neighbors	weight are different for neighbors	weights are shared among neighbors
data movement	require **both gather and scatter**	no explicit gather or scatter	require either gather or scatter

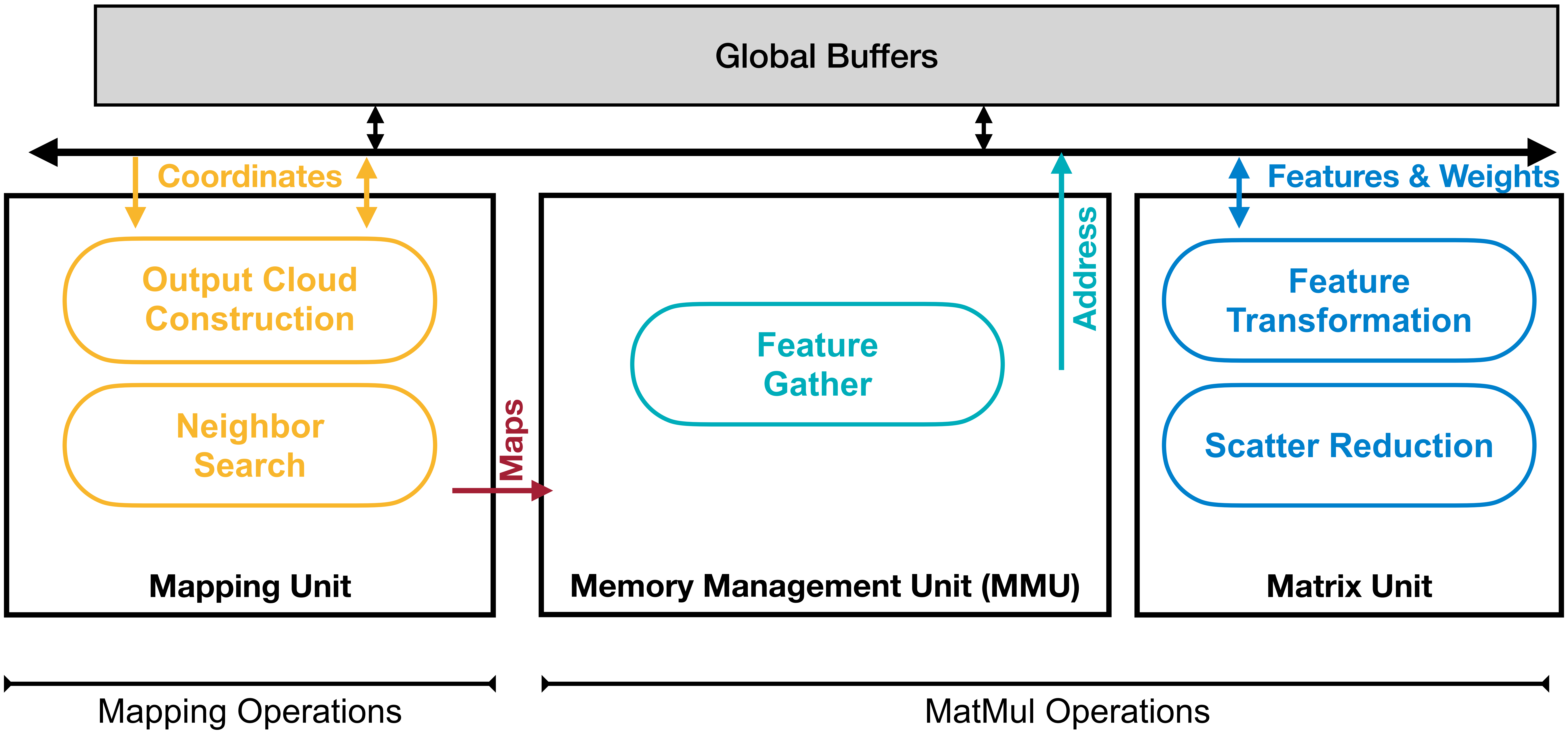
- GCN accelerators and previous PointNet accelerator Mesorasi do not support

Challenge of Point Cloud Deep Learning

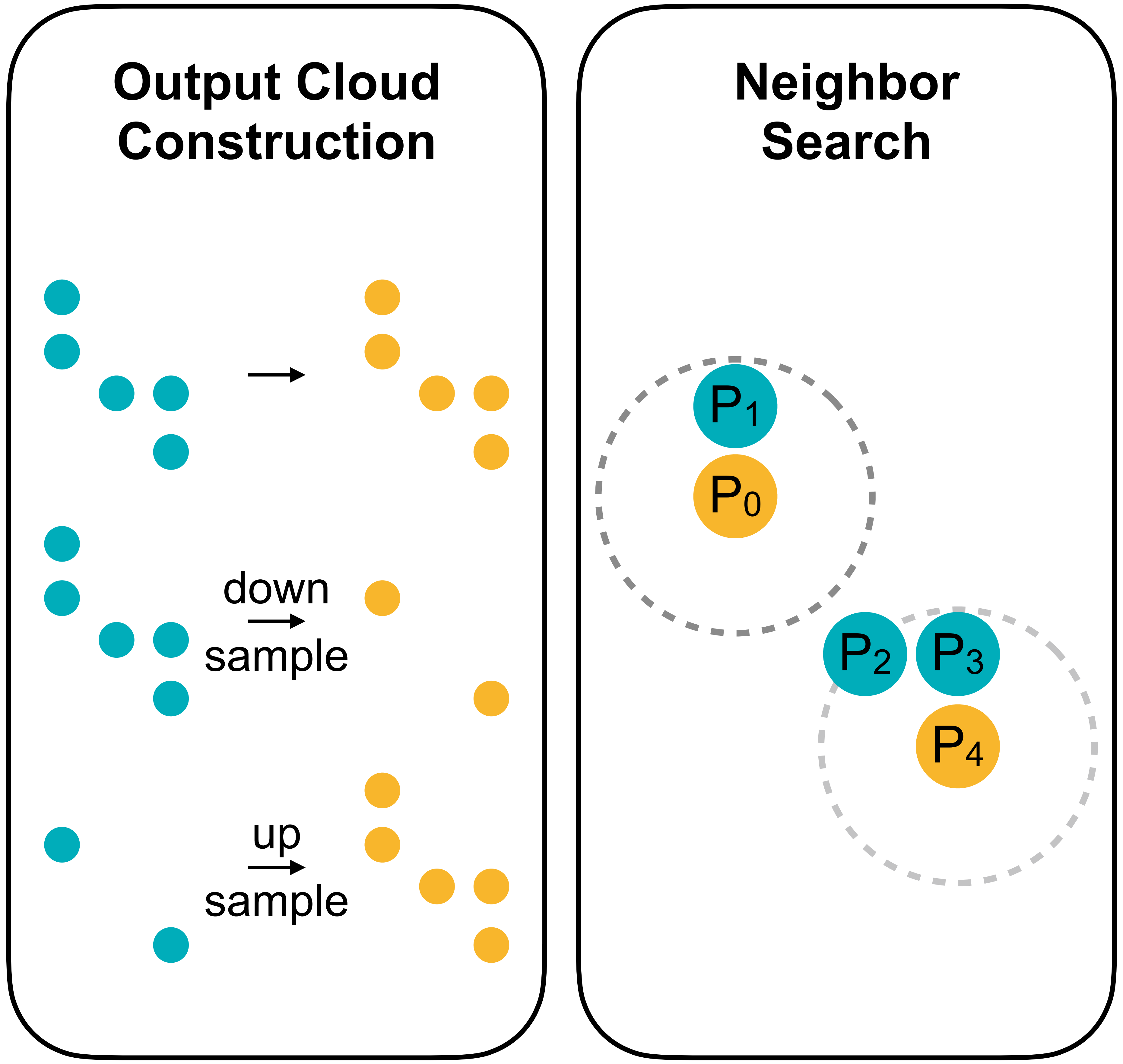


- Due to extreme sparsity, mapping operations and data movement overhead together take up **>50%** of the total runtime latency
- Due to unsupported mapping ops, data movement between co-processors (CPU and TPU) worsens the bottleneck

PointAcc: Efficient Point Cloud Accelerator



Mapping Unit: Diverse Mapping Ops in One Versatile Arch



Goal of Mapping Ops:

Generate Map (input point, output point, weight index)

Key Observation:

Maps are constructed based on the ***comparison*** among distances

Mapping Unit: Diverse Mapping Ops in One Versatile Arch

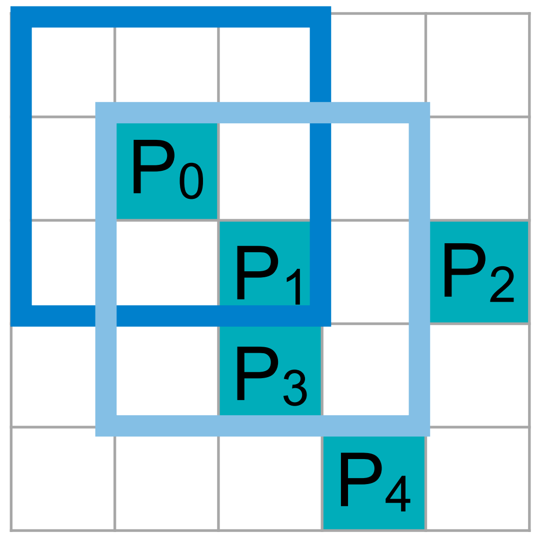
Goal of Mapping Ops: Generate Map (input point, output point, weight index)

Key Observation: Maps are constructed based on the ***comparison*** among distances

Kernel Mapping

- comparison op: not greater than, $|\Delta x| <= 1, |\Delta y| <= 1, |\Delta z| <= 1$
- query the hash table of input point cloud for each output point

```
For Q in O={Q0, Q1, Q2, …}:
  Find P in I, s.t. |Px−Qx| <= 1 && |Py−Qy| <= 1 && |Pz−Qz| <= 1
```



(In, Out, Wgt)
(P_0, Q_0, $W_{0,0}$)
(P_1, Q_0, $W_{1,1}$)
(P_0, Q_1, $W_{-1,-1}$)
(P_1, Q_1, $W_{0,0}$)
…

```
For W in {W-1,-1, W-1,0, W1,0, …}:
  For Q in O={Q0, Q1, Q2, …}:
    Query Q+δW in HashTable(I)
```

- require on-chip memory as large as 160MB
- cannot be parallelized efficiently

Mapping Unit: Diverse Mapping Ops in One Versatile Arch

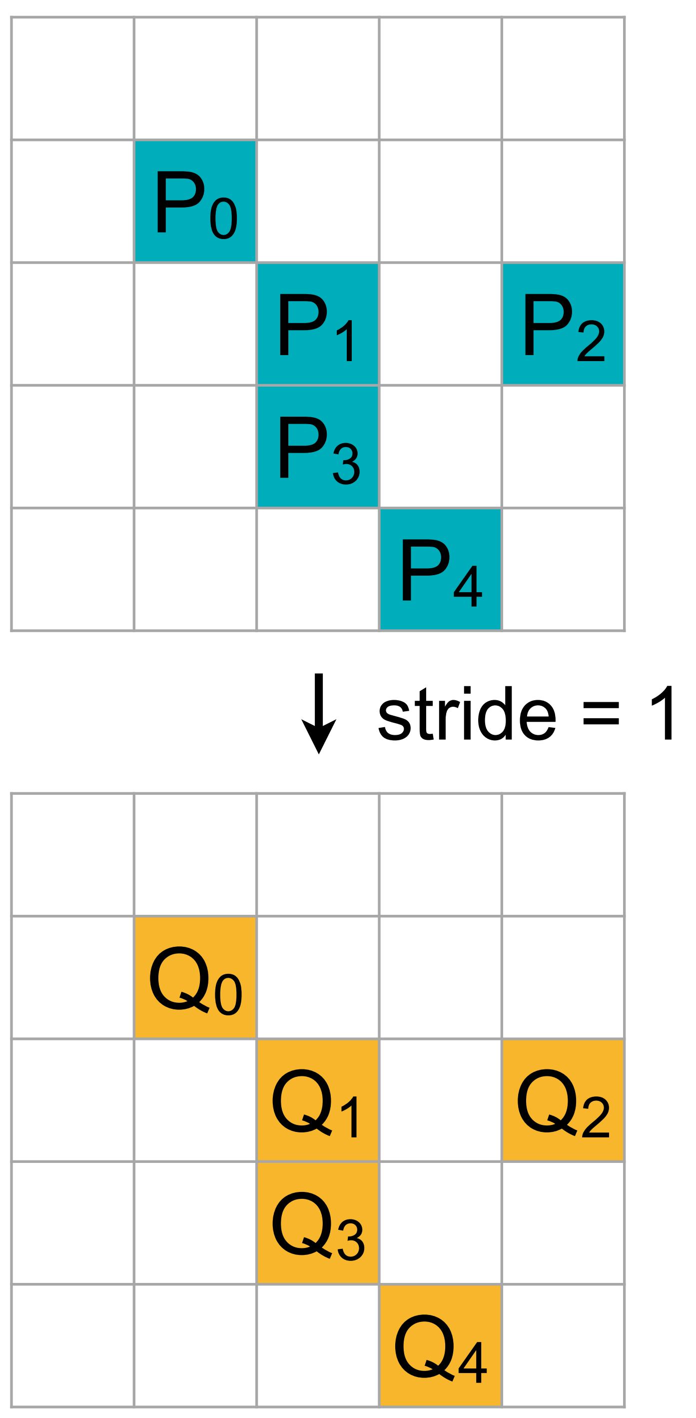
Goal of Mapping Ops: Generate Map (input point, output point, weight index)

Key Observation: Maps are constructed based on the ***comparison*** among distances

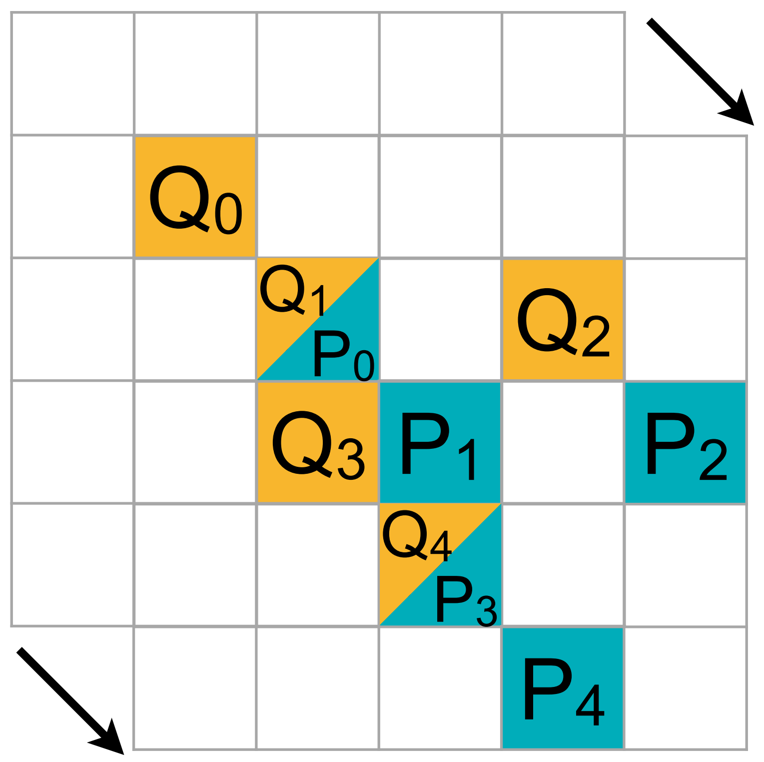
Kernel Mapping

- comparison op: not greater than, $|\Delta x| <= 1, |\Delta y| <= 1, |\Delta z| <= 1$
- ~~query the hash table of input point cloud for each output point~~
- coordinates intersection for each neighbor position
- $\rightarrow$ parallezable merge-sort and equal comparison

Input Point Cloud



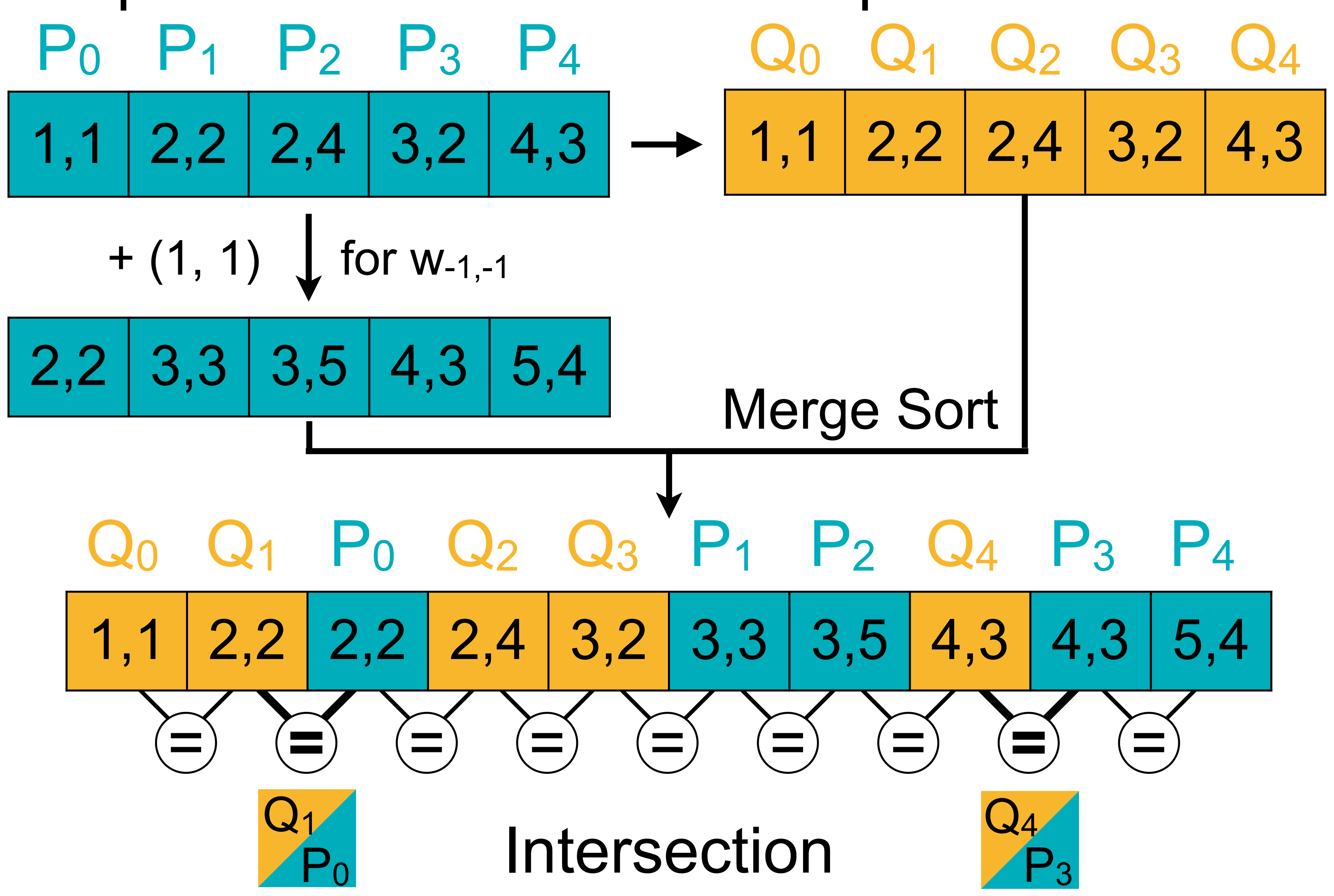
Output Point Cloud



Shift Input for $W_{-1,-1}$

$W_{-1,-1}$	$W_{-1,0}$	$W_{-1,1}$
$W_{0,-1}$	$W_{0,0}$	$W_{0,1}$
$W_{1,-1}$	$W_{1,0}$	$W_{1,1}$

(In, Out, Wgt)
(P_0, Q_1, $W_{-1,-1}$)
(P_3, Q_4, $W_{-1,-1}$)



Mapping Unit: Diverse Mapping Ops in One Versatile Arch

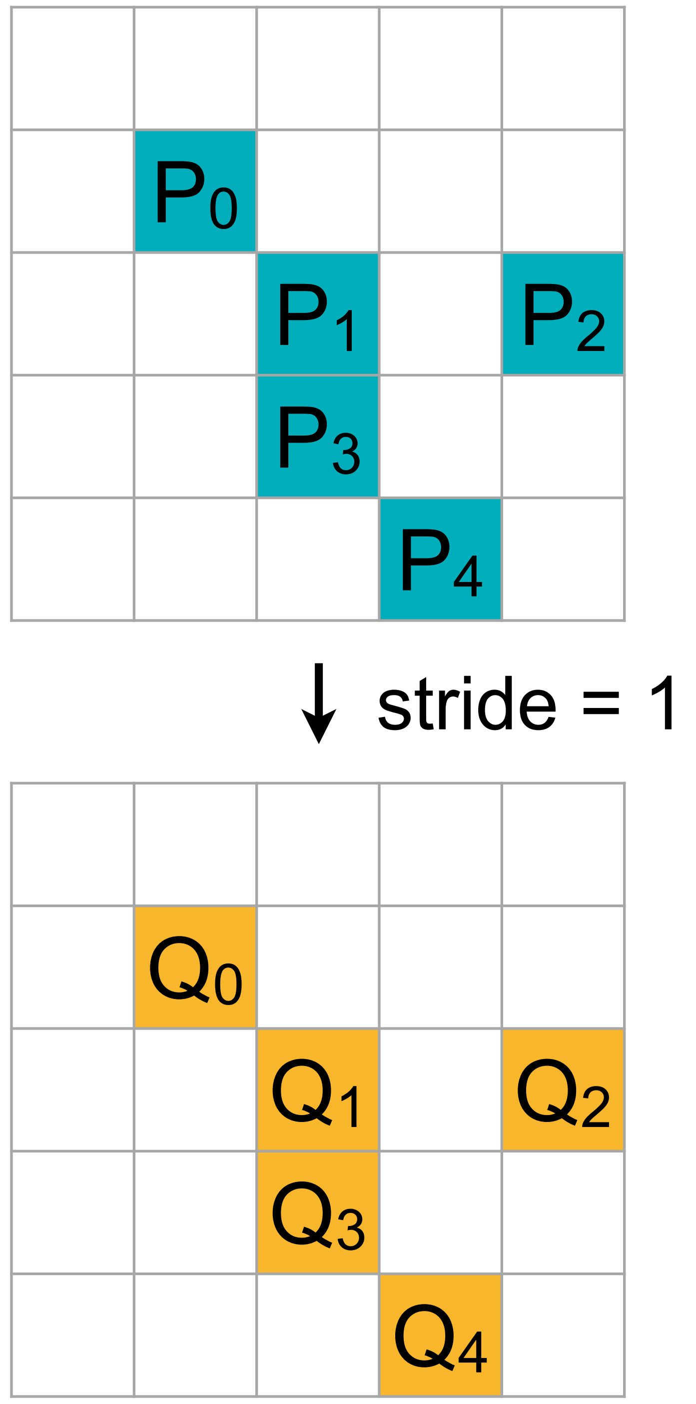
Goal of Mapping Ops: Generate Map (input point, output point, weight index)

Key Observation: Maps are constructed based on the ***comparison*** among distances

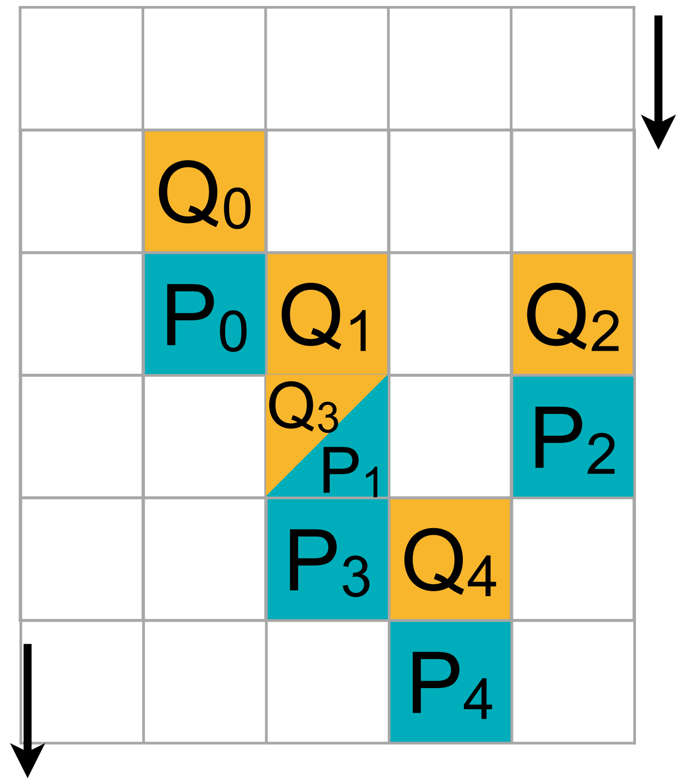
Kernel Mapping

- comparison op: not greater than, $|\Delta x| <= 1, |\Delta y| <= 1, |\Delta z| <= 1$
- ~~query the hash table of input point cloud for each output point~~
- coordinates intersection for each neighbor position
- $\rightarrow$ parallezable merge-sort and equal comparison

Input Point Cloud



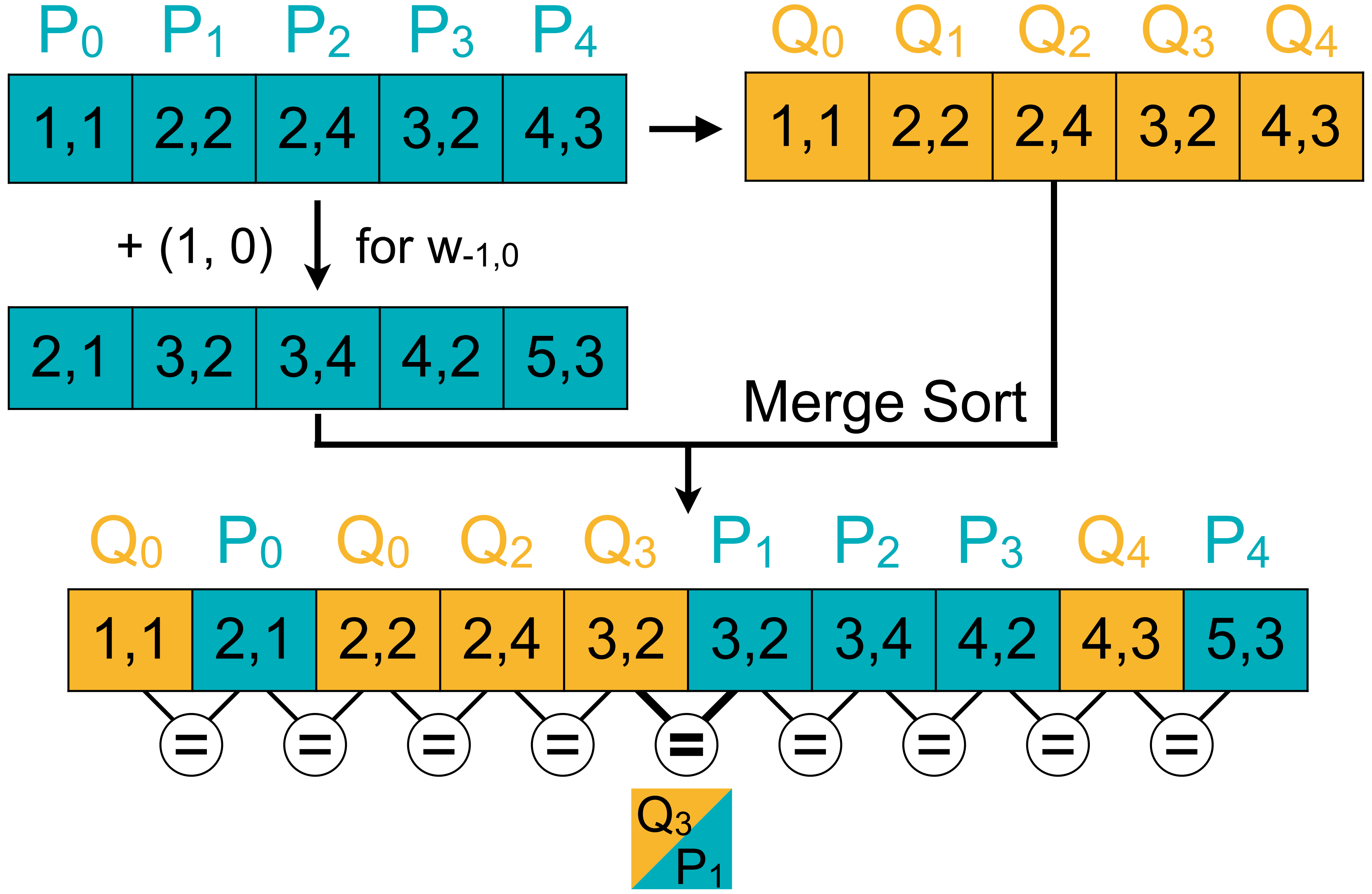
Output Point Cloud



Shift Input for $W_{-1,0}$

$W_{-1,-1}$	$W_{-1,0}$	$W_{-1,1}$
$W_{0,-1}$	$W_{0,0}$	$W_{0,1}$
$W_{1,-1}$	$W_{1,0}$	$W_{1,1}$

(In, Out, Wgt)
(P_0, Q_1, $W_{-1,-1}$)
(P_3, Q_4, $W_{-1,-1}$)
(P_1, Q_3, $W_{-1,0}$)



Mapping Unit: Diverse Mapping Ops in One Versatile Arch

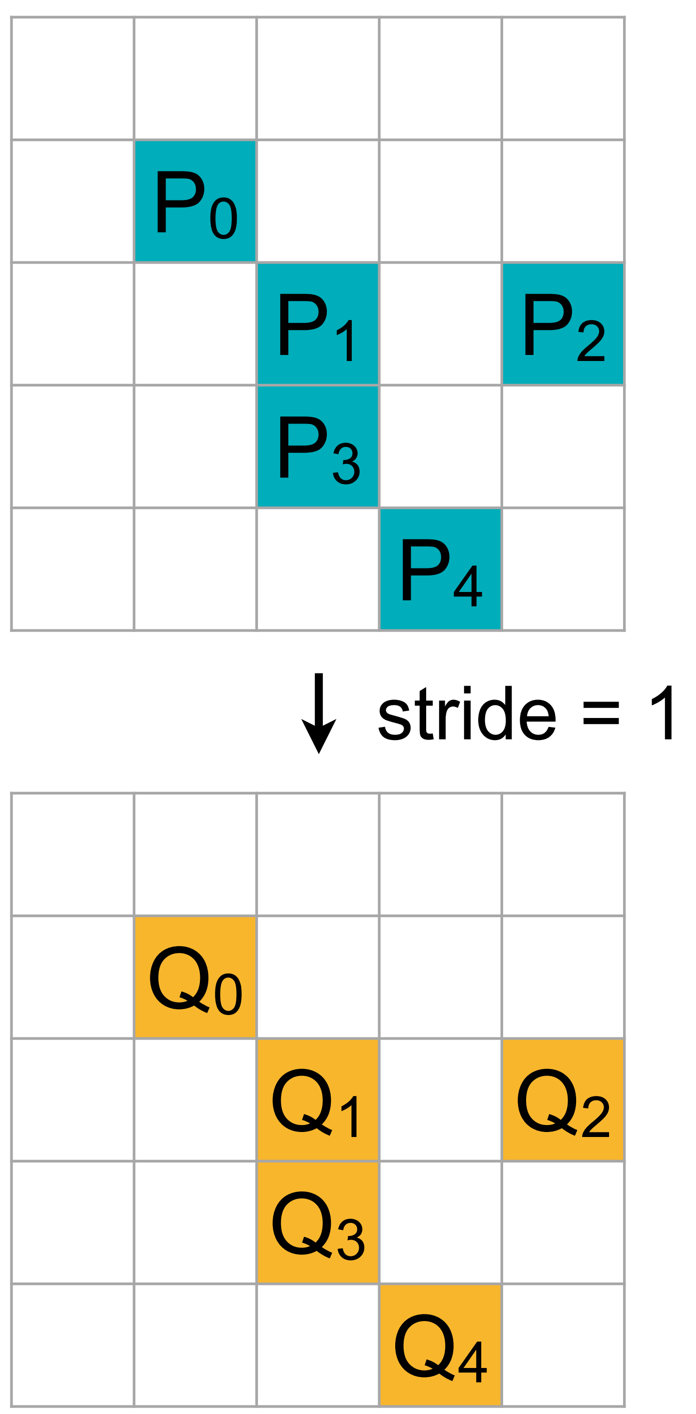
Goal of Mapping Ops: Generate Map (input point, output point, weight index)

Key Observation: Maps are constructed based on the ***comparison*** among distances

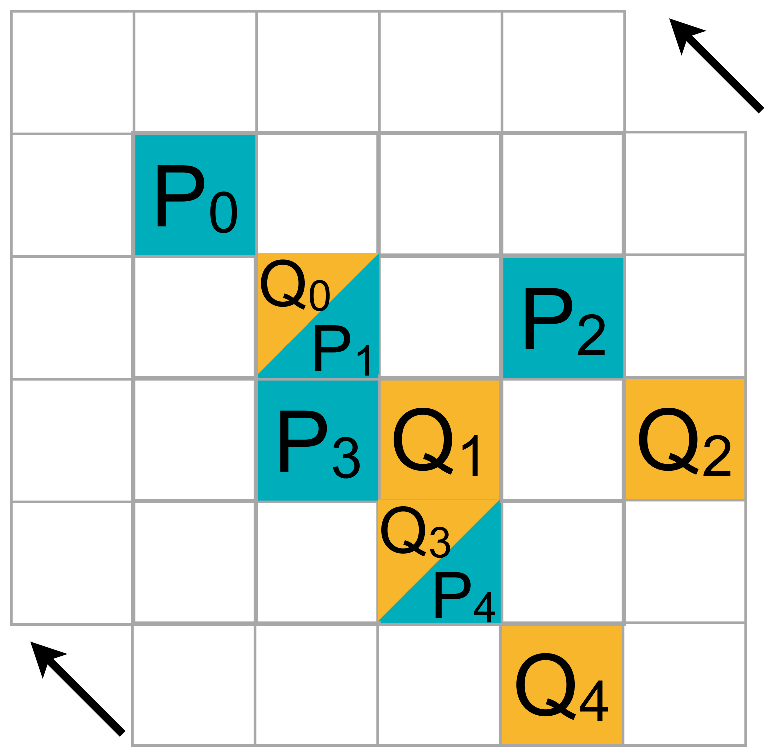
Kernel Mapping

- comparison op: not greater than, $|\Delta x| <= 1, |\Delta y| <= 1, |\Delta z| <= 1$
- ~~query the hash table of input point cloud for each output point~~
- coordinates intersection for each neighbor position
- $\rightarrow$ parallezable merge-sort and equal comparison

Input Point Cloud



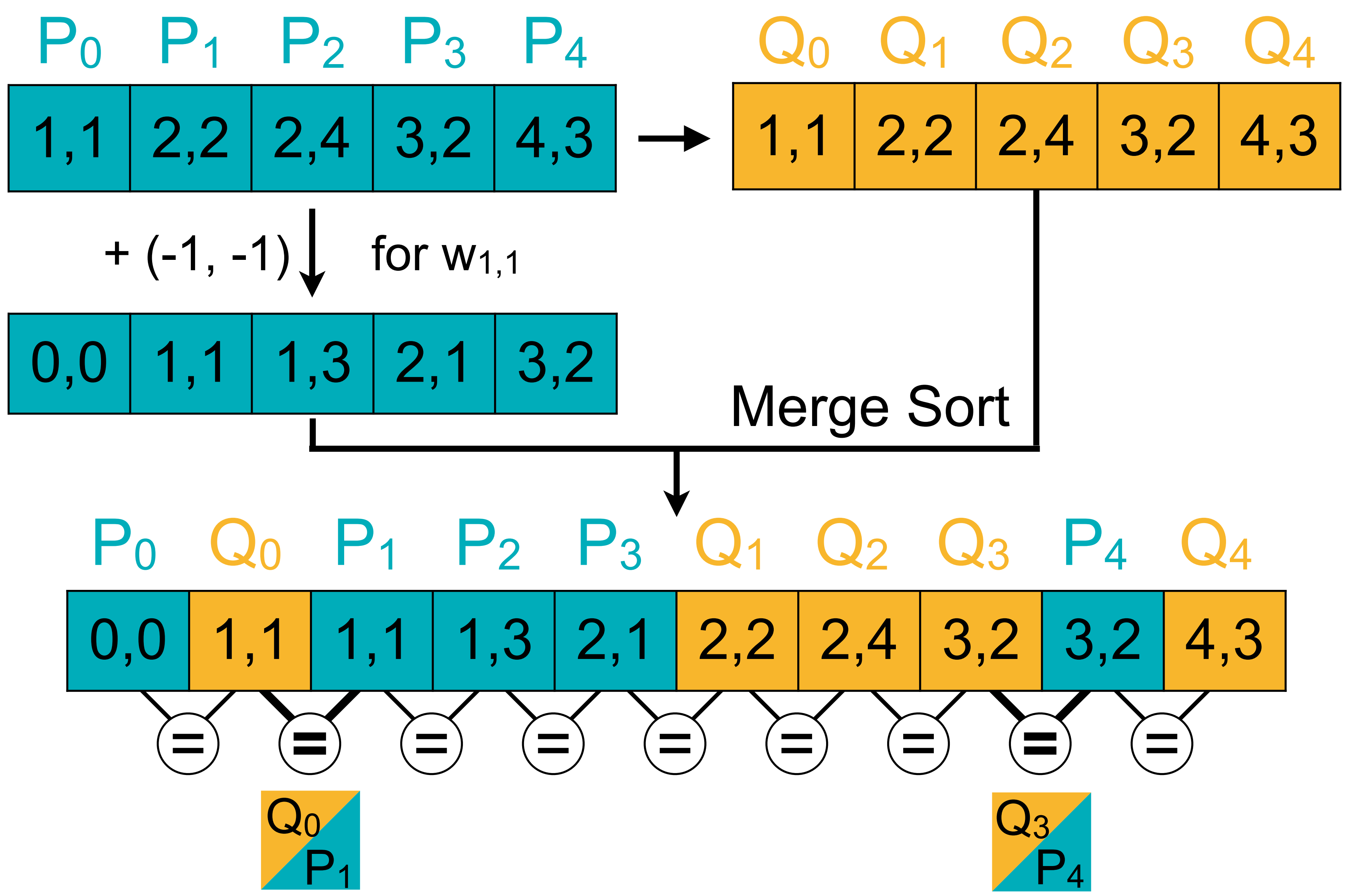
Output Point Cloud



Shift Input for $W_{1,1}$

$W_{-1,-1}$	$W_{-1,0}$	$W_{-1,1}$
$W_{0,-1}$	$W_{0,0}$	$W_{0,1}$
$W_{1,-1}$	$W_{1,0}$	$W_{1,1}$

(In, Out, Wgt)
(P_0, Q_1, $W_{-1,-1}$)
(P_3, Q_4, $W_{-1,-1}$)
…
(P_1, Q_0, $W_{1,1}$)
(P_4, Q_3, $W_{11,1}$)



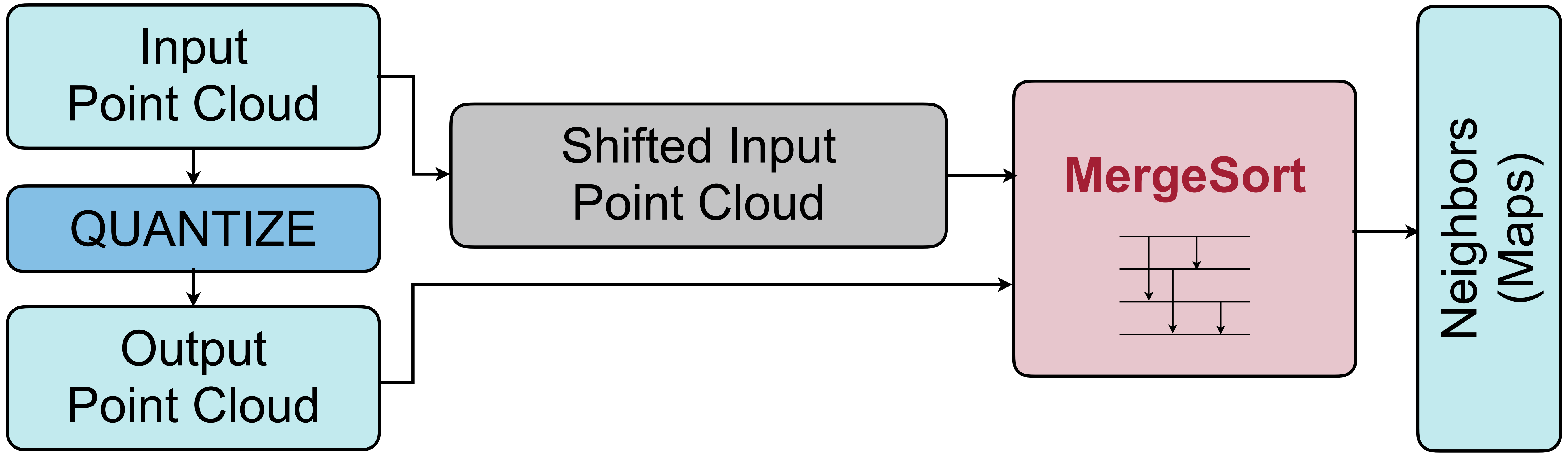
Mapping Unit: Diverse Mapping Ops in One Versatile Arch

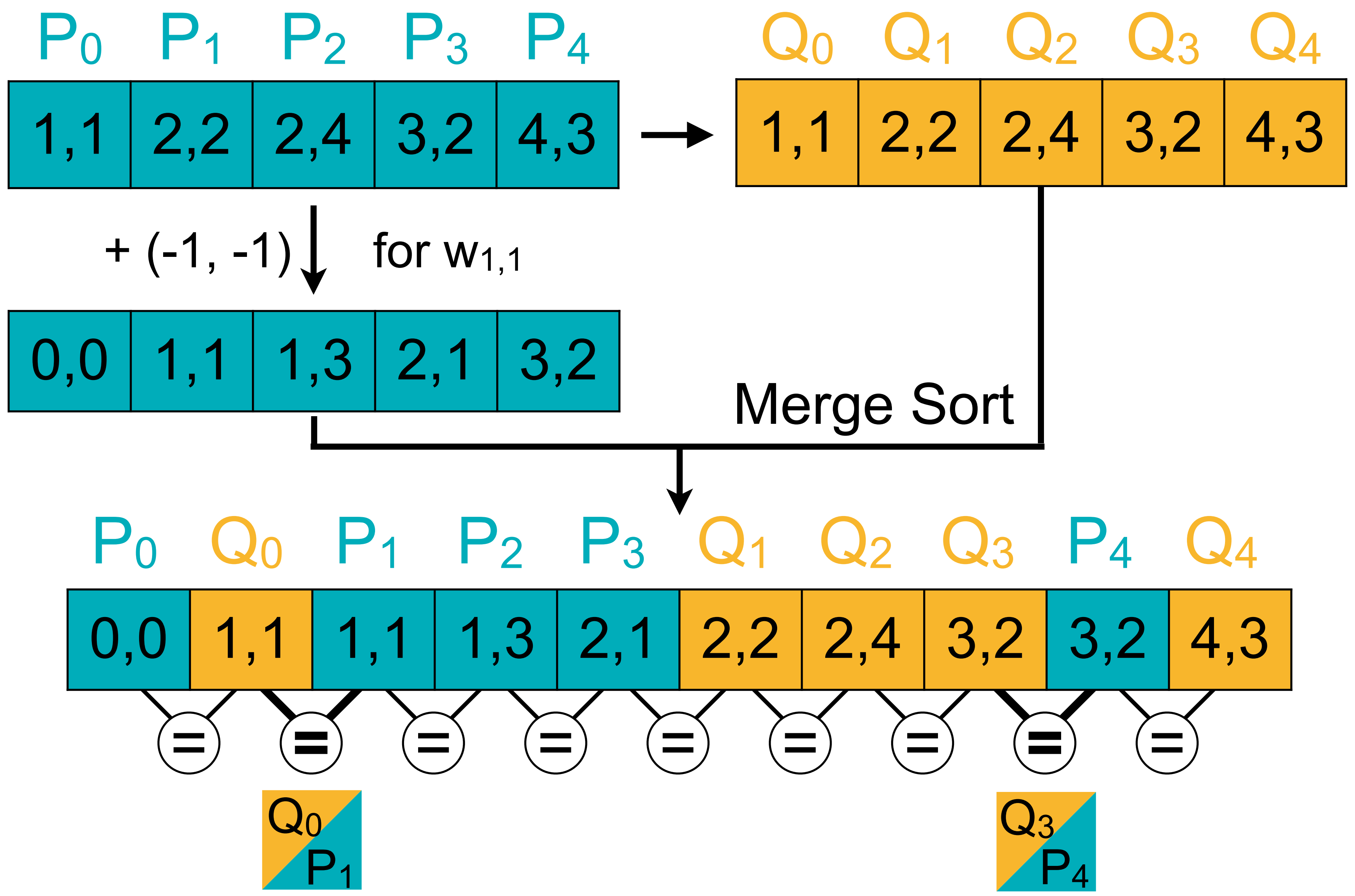
Goal of Mapping Ops: Generate Map (input point, output point, weight index)

Key Observation: Maps are constructed based on the ***comparison*** among distances

Kernel Mapping

- comparison op: not greater than, $|\Delta x| <= 1, |\Delta y| <= 1, |\Delta z| <= 1$
- ~~query the hash table of input point cloud for each output point~~
- coordinates intersection for each neighbor position
- $\rightarrow$ parallezable merge-sort and equal comparison





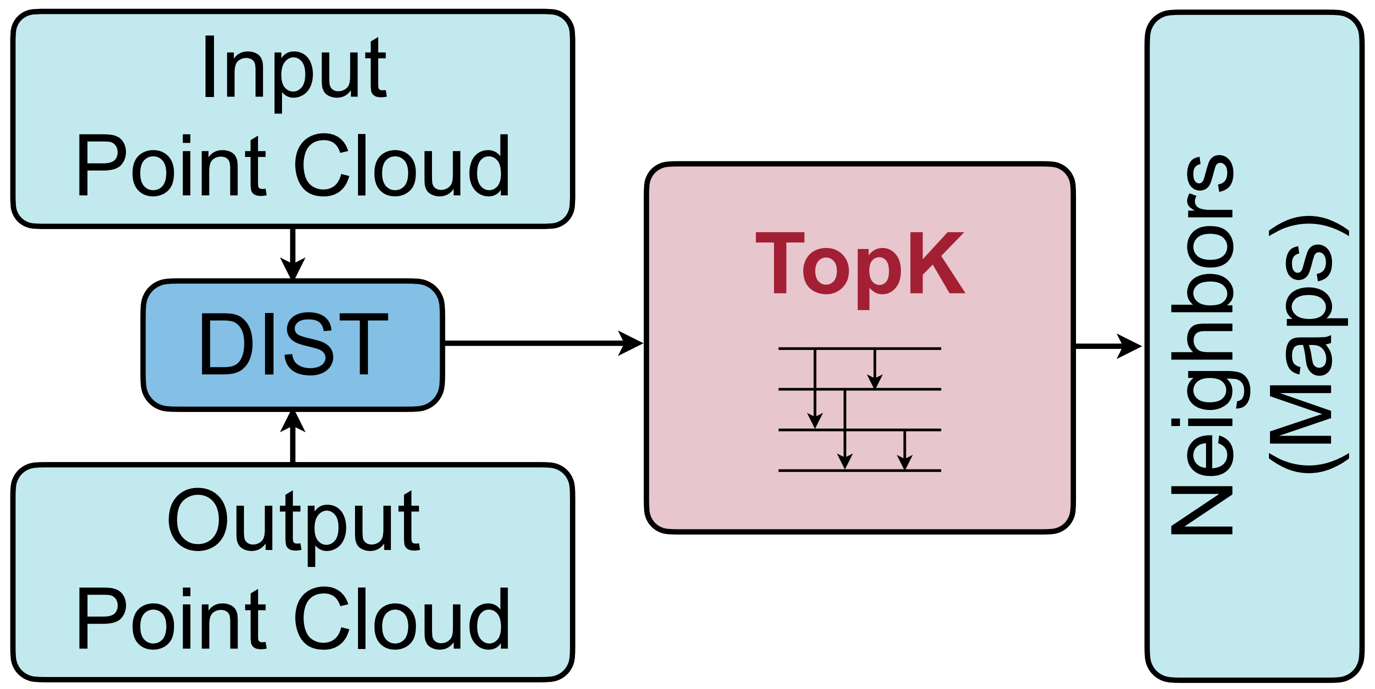
Mapping Unit: Diverse Mapping Ops in One Versatile Arch

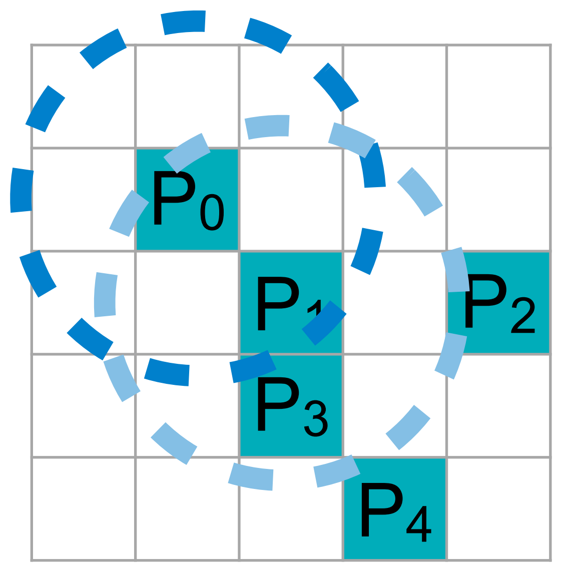
Goal of Mapping Ops: Generate Map (input point, output point, weight index)

Key Observation: Maps are constructed based on the ***comparison*** among distances

k-Nearest-Neighbor / Ball Query

- comparison op: TopK
- ball query filters out the outsider in the nearest neighbors

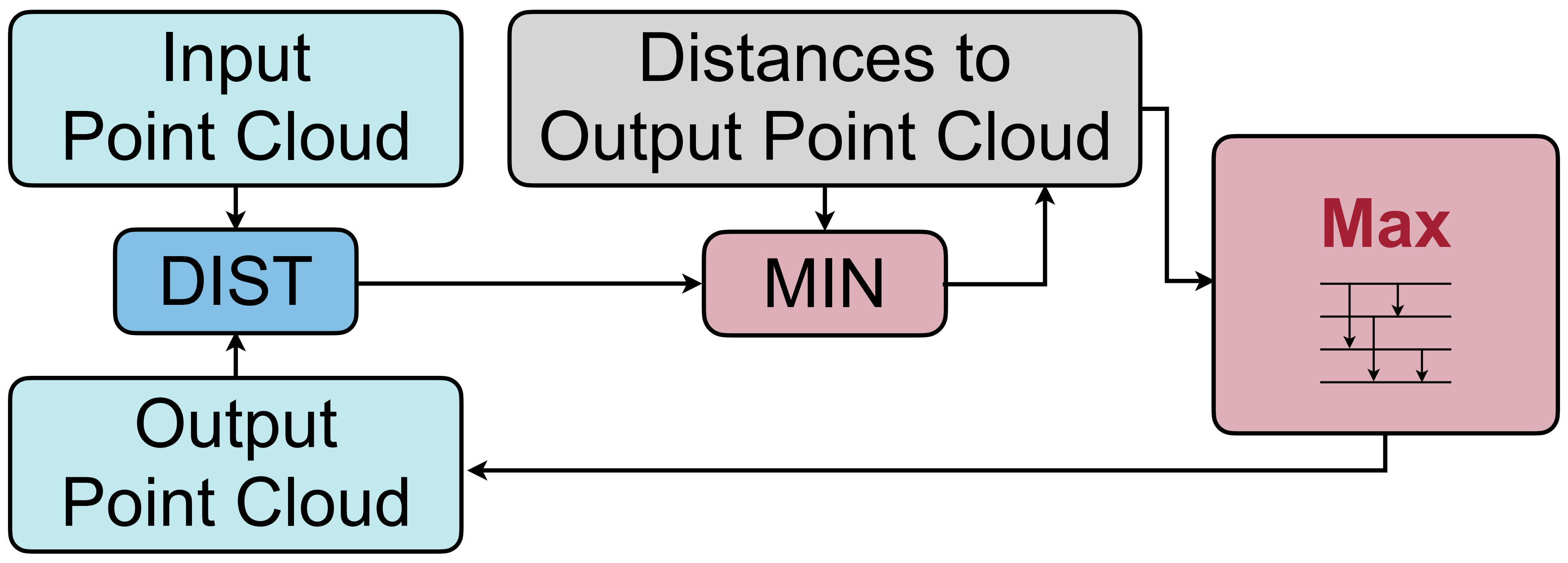




(In, Out, Wgt)
(P_0, Q_0, W_0)
(P_1, Q_0, W_1)
(P_3, Q_0, W_2)
(P_1, Q_1, W_0)
(P_3, Q_1, W_1)

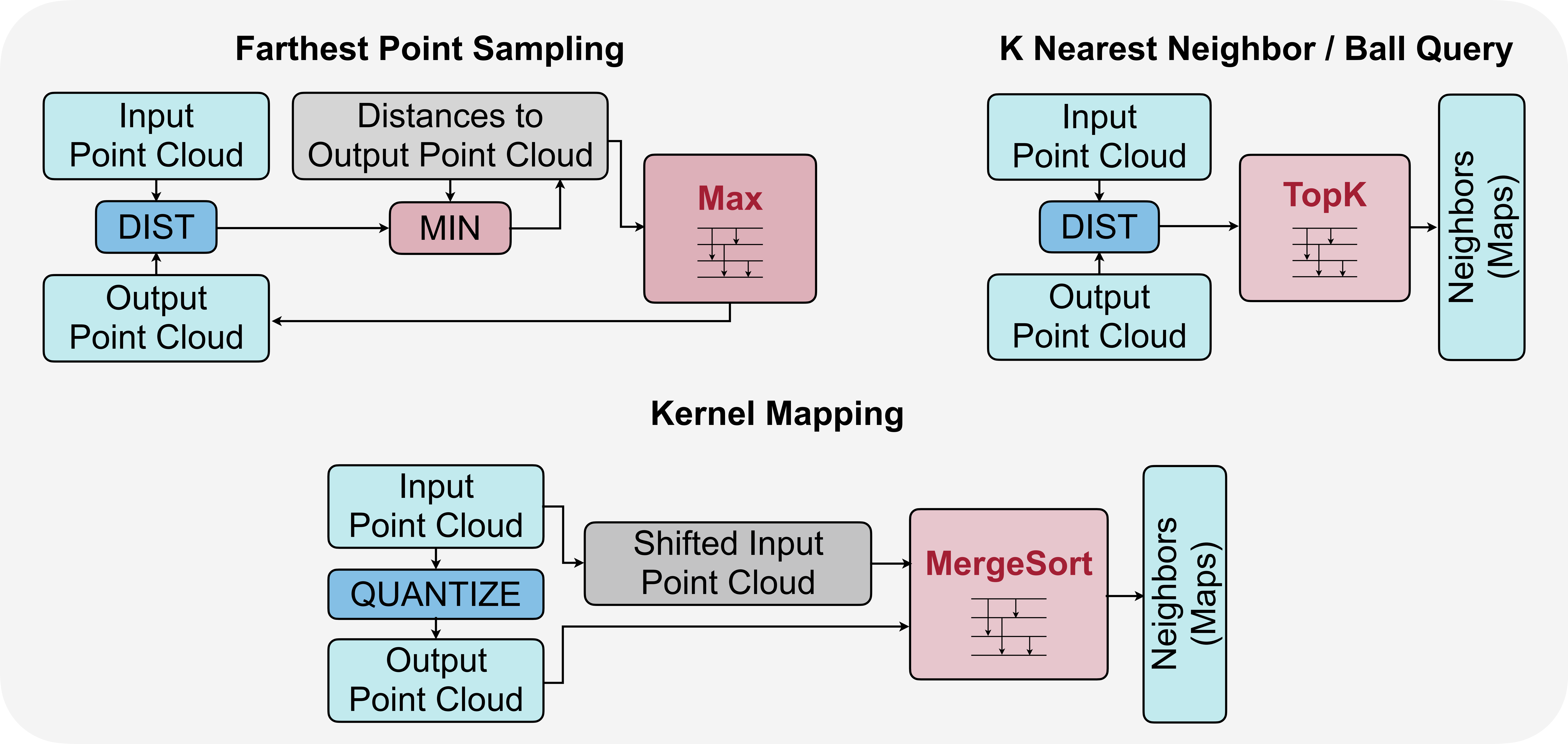
Farthest Point Sampling

- comparison op: ArgMax / Max

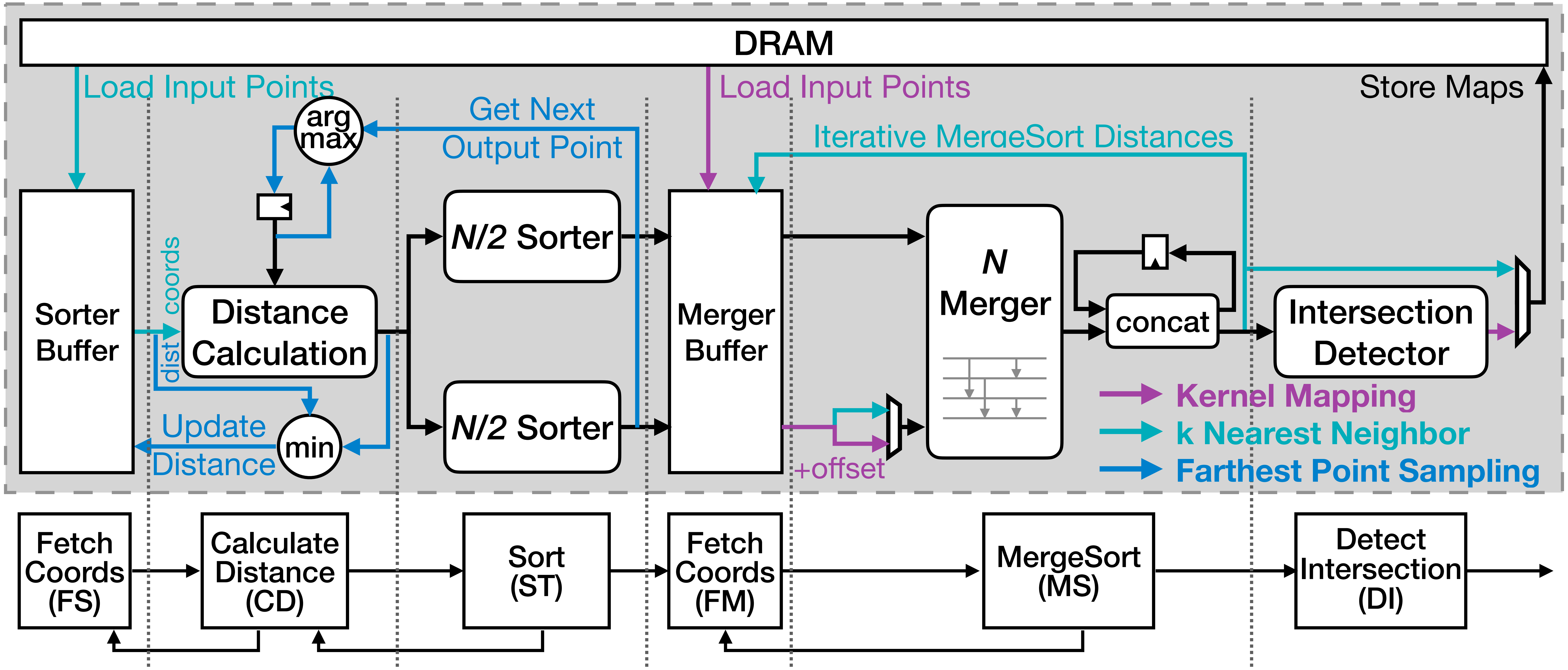


Mapping Unit: Diverse Mapping Ops in One Versatile Arch

- **Goal of Mapping Ops:** Generate Map (input point, output point, weight index)
- **Key Observation:** Maps are constructed based on the ***comparison*** among distances

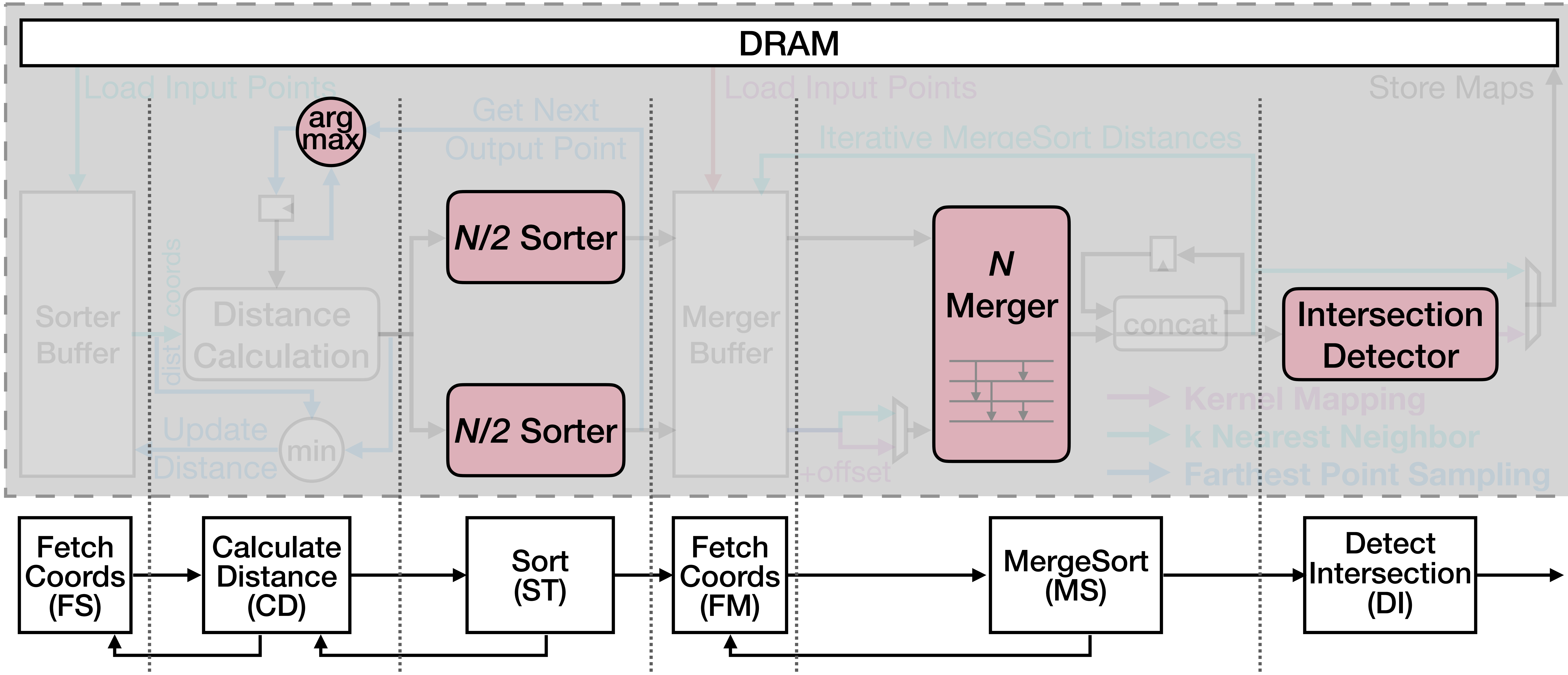


Mapping Unit: Diverse Mapping Ops in One Versatile Arch



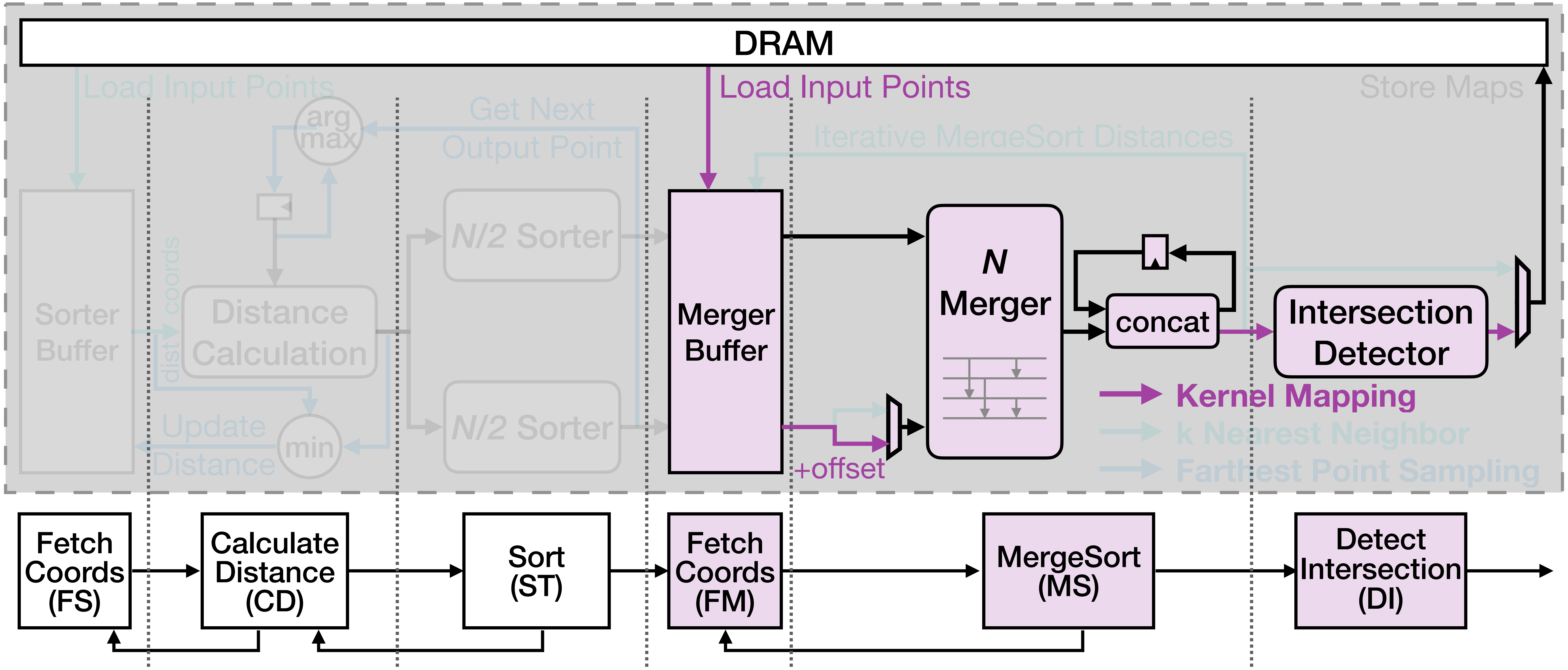
Mapping Unit: Diverse Mapping Ops in One Versatile Arch

- Main component for parallel comparison: sorters, merger



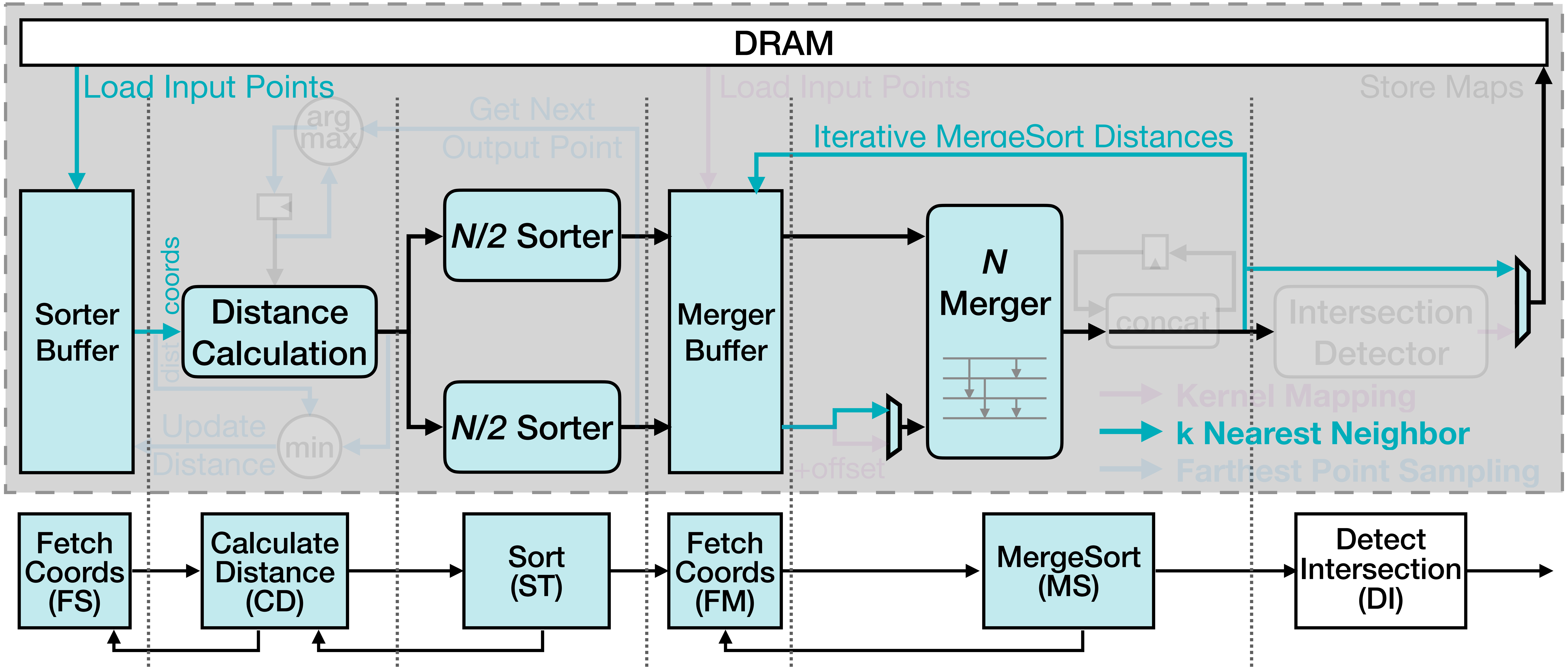
Mapping Unit: Diverse Mapping Ops in One Versatile Arch

- Data flow when running kernel mapping (*i.e.*, the neighborhood shape is cube)



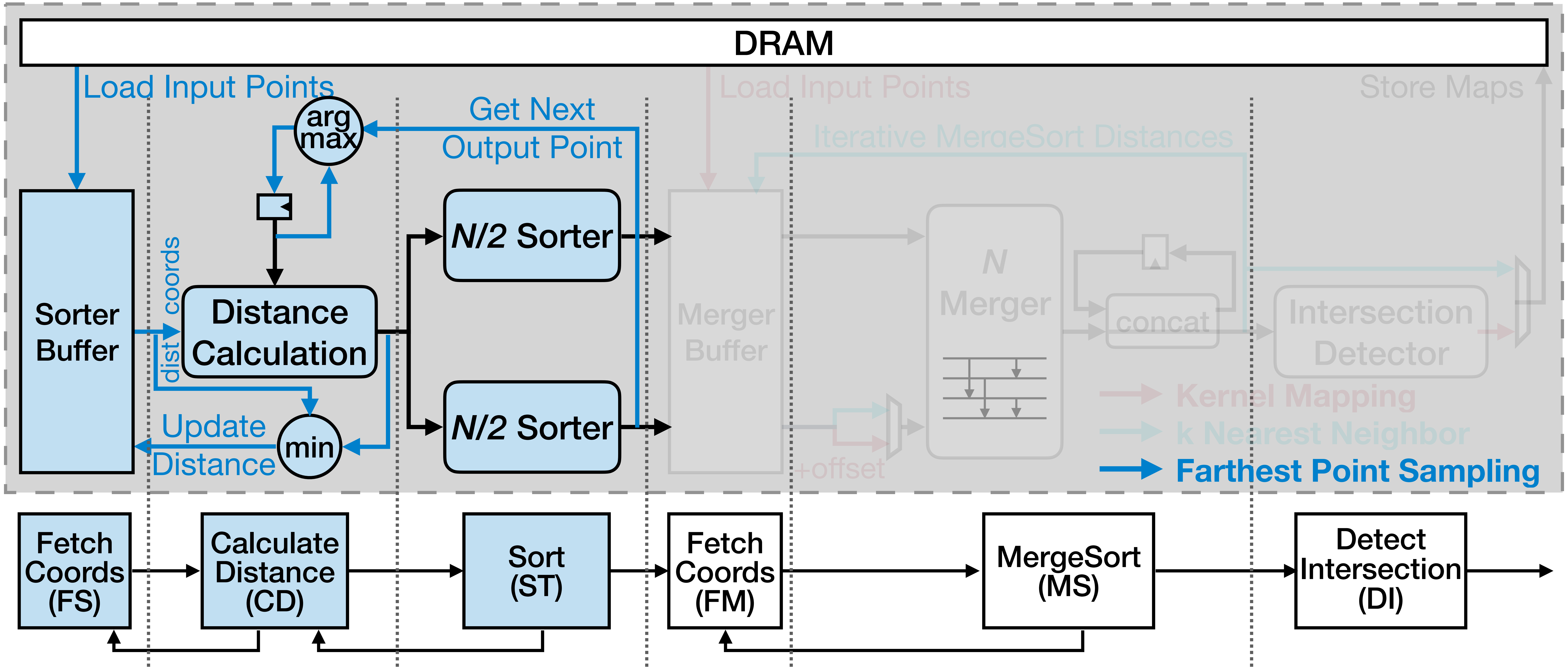
Mapping Unit: Diverse Mapping Ops in One Versatile Arch

- Data flow when running k-nearest-neighbor / ball query (*i.e.*, the neighborhood shape is ball)



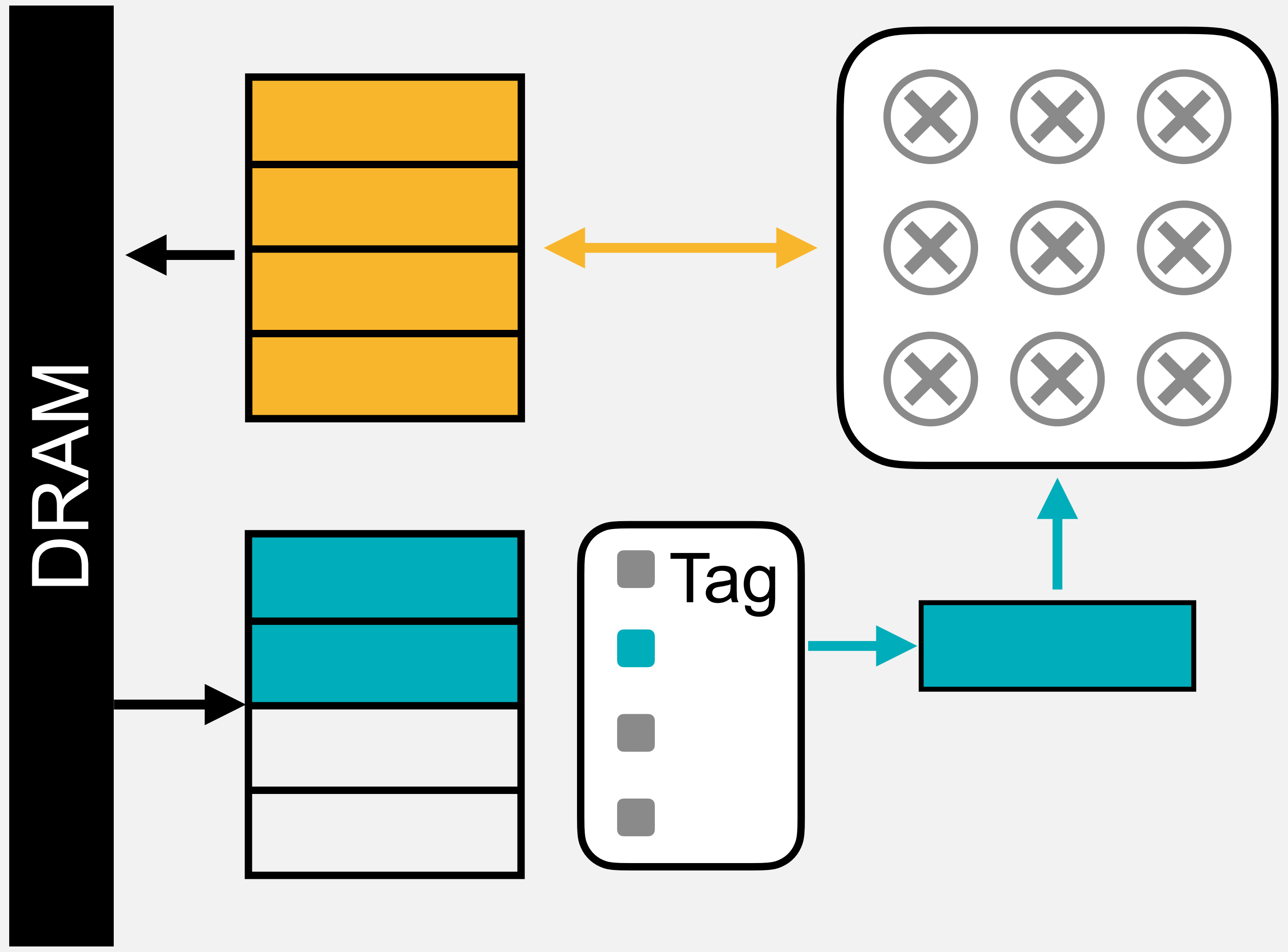
Mapping Unit: Diverse Mapping Ops in One Versatile Arch

- Data flow when running farthest point sampling



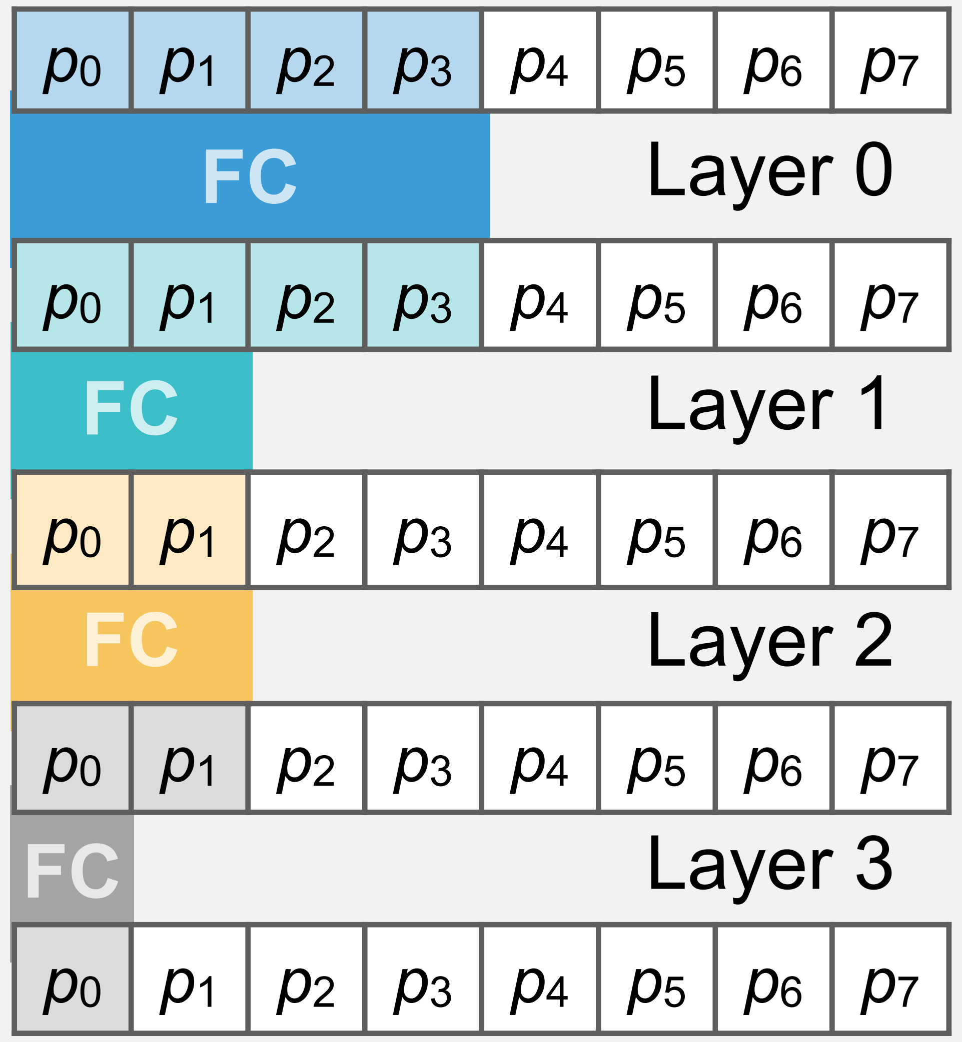
Flexible Memory Management

Sparse Computation



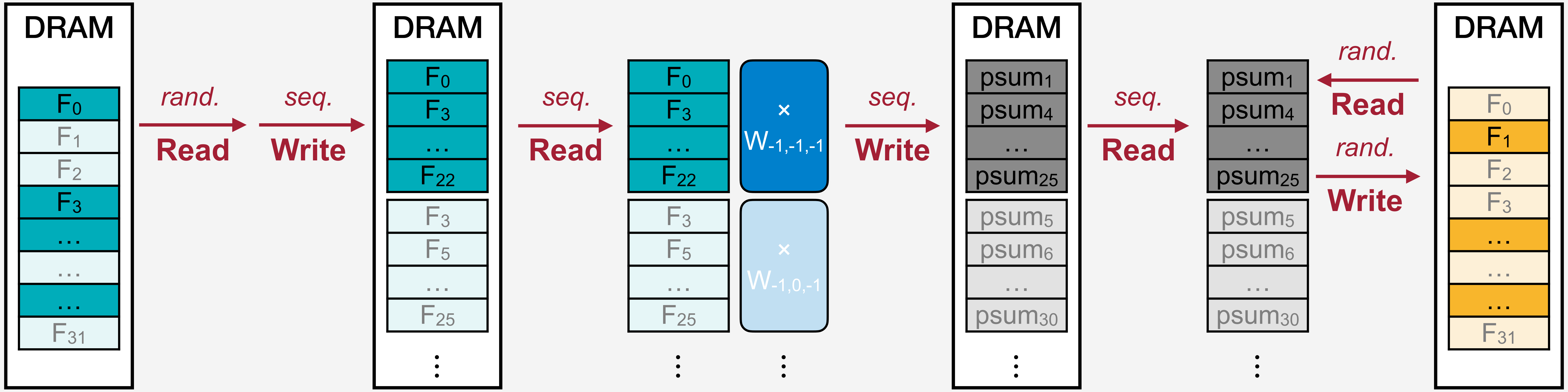
Streaming Computation with Caching

Dense Computation



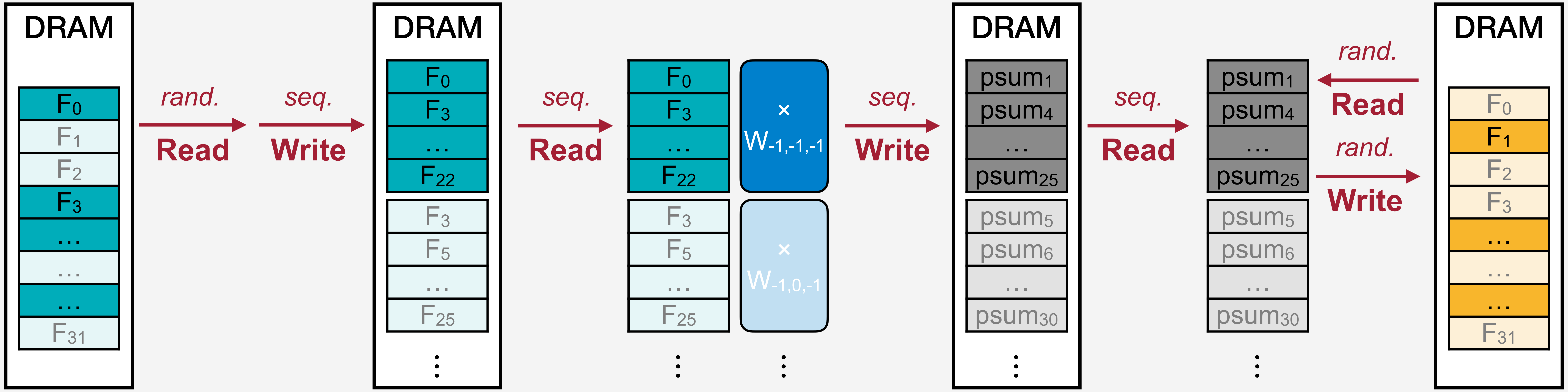
Temporal Layer Fusion

Sparse MatMul Operations

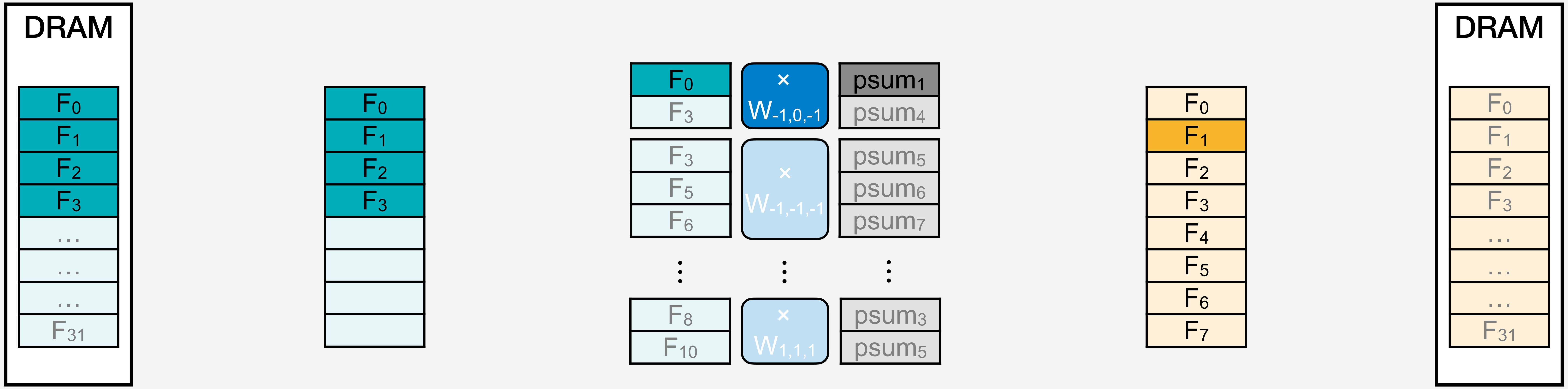


the state-of-the-art GPU implementation

Streaming Sparse MatMul Operations



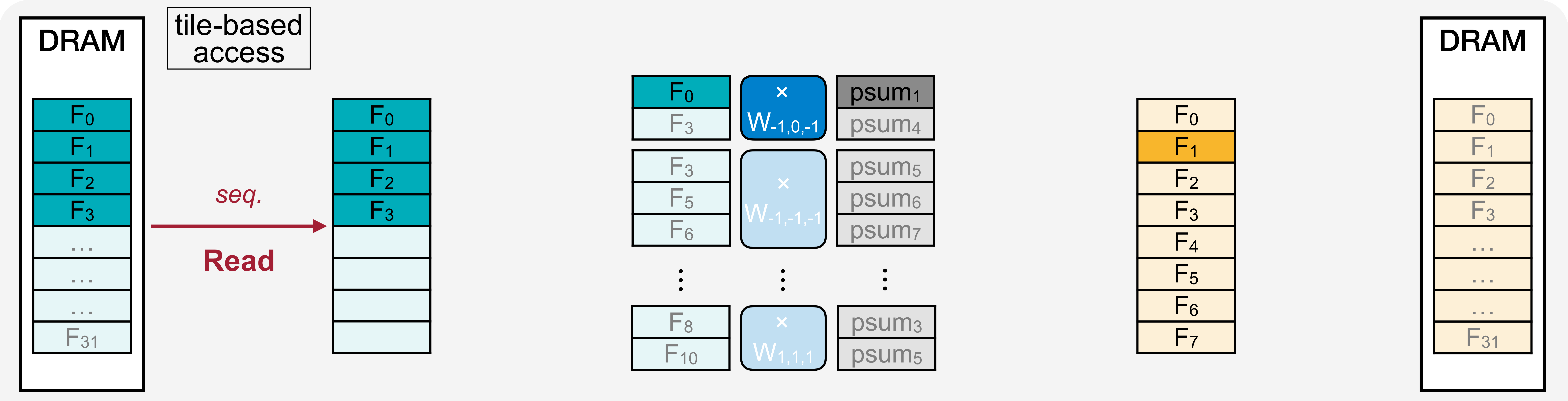
the state-of-the-art GPU implementation



PointAcc implementation

Streaming Sparse MatMul Operations

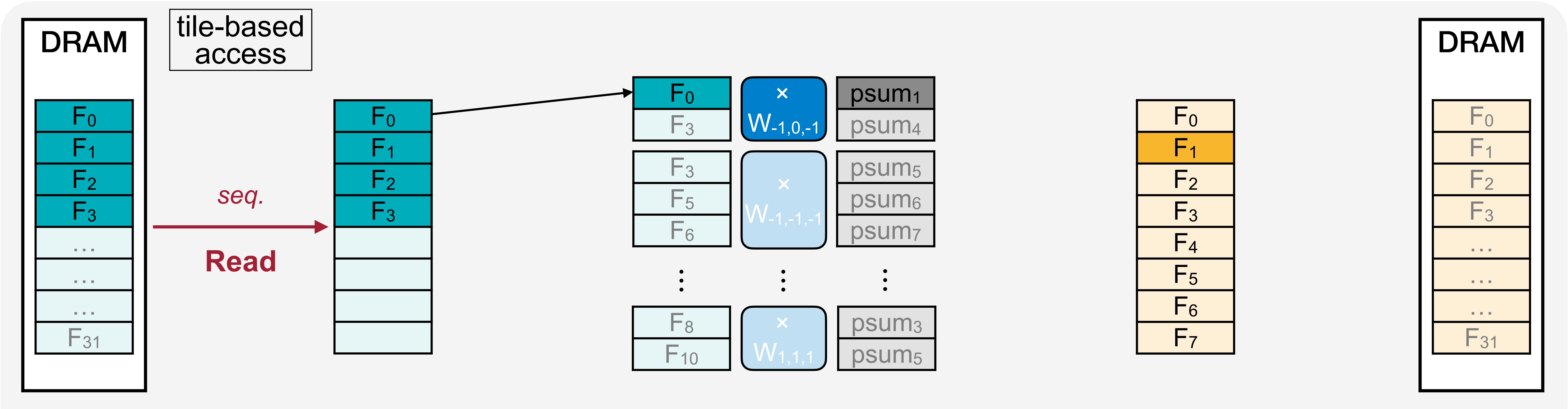
- Sequential fetch input features on demand in the granularity of tile (*i.e.*, "cache block")
  - No more random access for gathering features



PointAcc implementation

Streaming Sparse MatMul Operations

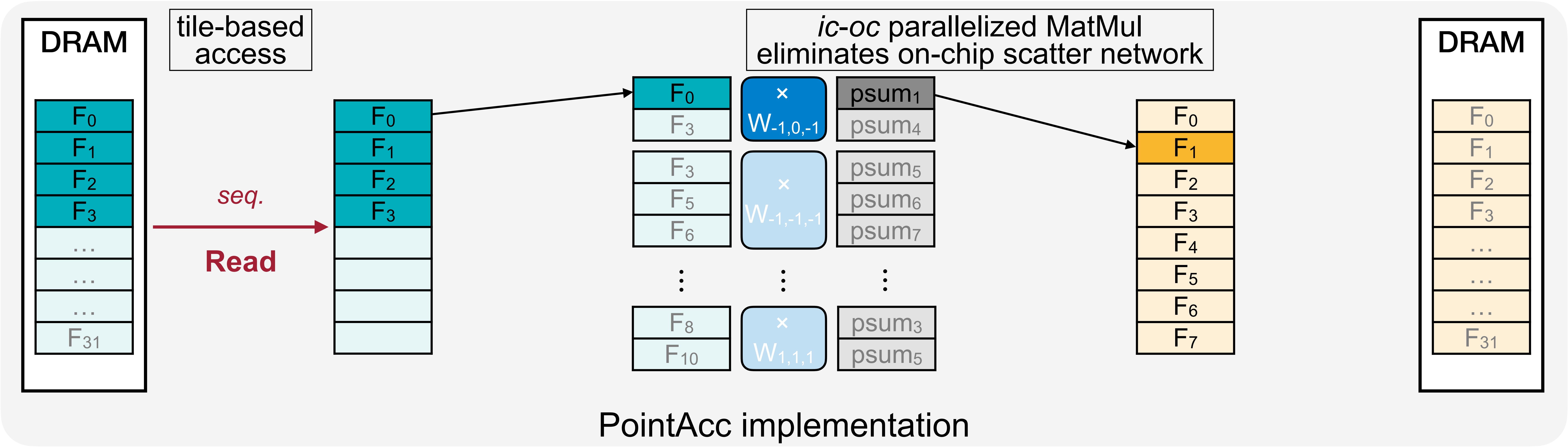
- Sequential fetch input features on demand in the granularity of tile (*i.e.*, "cache block")
  - No more random access for gathering features
- MatMul computing parallelizes input channel (*ic*) and output channel (*oc*) dimension



PointAcc implementation

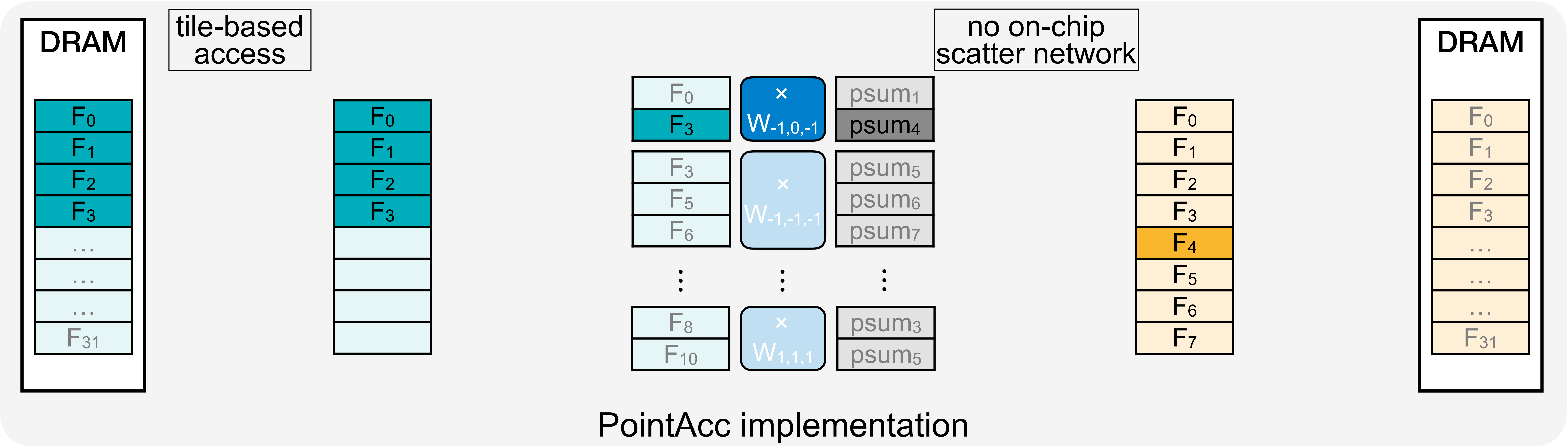
Streaming Sparse MatMul Operations

- Sequential fetch input features on demand in the granularity of tile (*i.e.*, "cache block")
  - No more random access for gathering features
- MatMul computing parallelizes input channel (*ic*) and output channel (*oc*) dimension
  - No need for on-chip scatter network for scattering psums of different points simultaneously



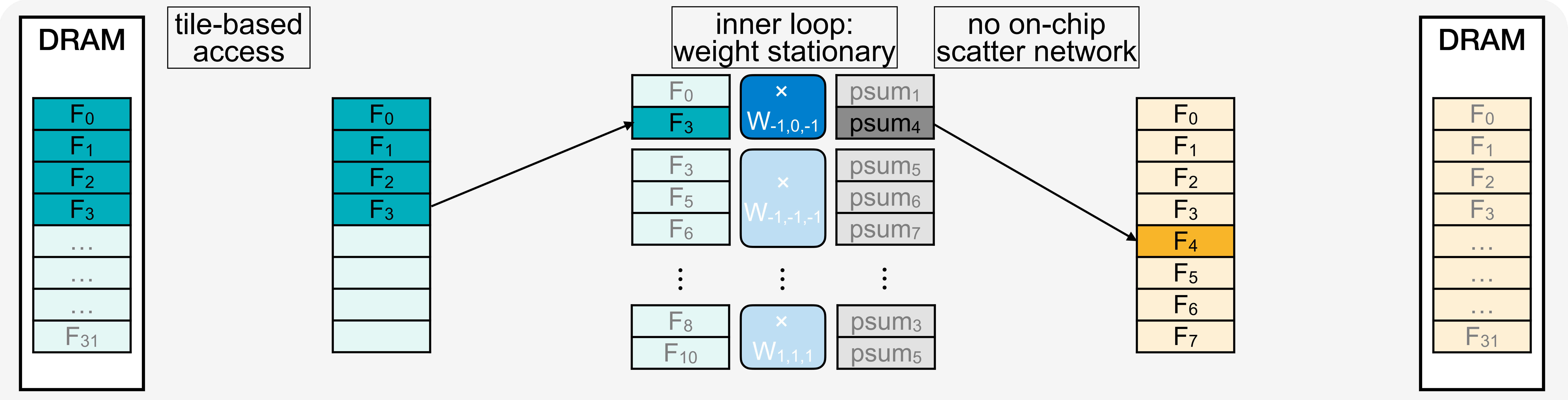
Streaming Sparse MatMul Operations

- Sequential fetch input features on demand in the granularity of tile (*i.e.*, "cache block")
- MatMul computing parallelizes *ic* and *oc* dimension → no need for on-chip scatter network



Streaming Sparse MatMul Operations

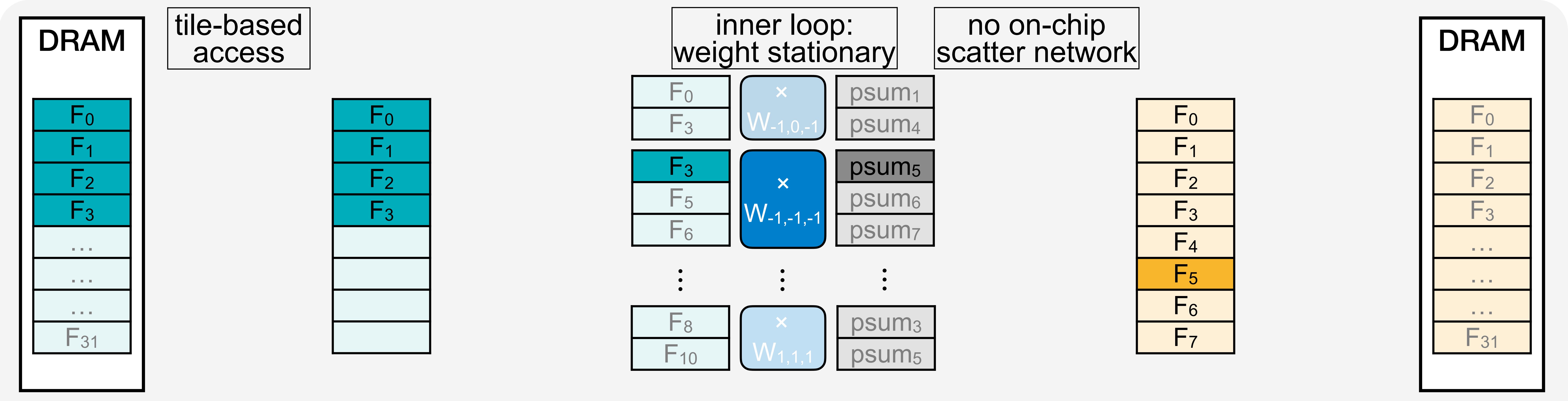
- Sequential fetch input features on demand in the granularity of tile (*i.e.*, "cache block")
- MatMul computing parallelizes *ic* and *oc* dimension → no need for on-chip scatter network
- Weight stationary saves the on-chip memory footprint
  - Key: #points ($10^3 \sim 10^5$) >> #channels ($10 \sim 10^3$)



PointAcc implementation

Streaming Sparse MatMul Operations

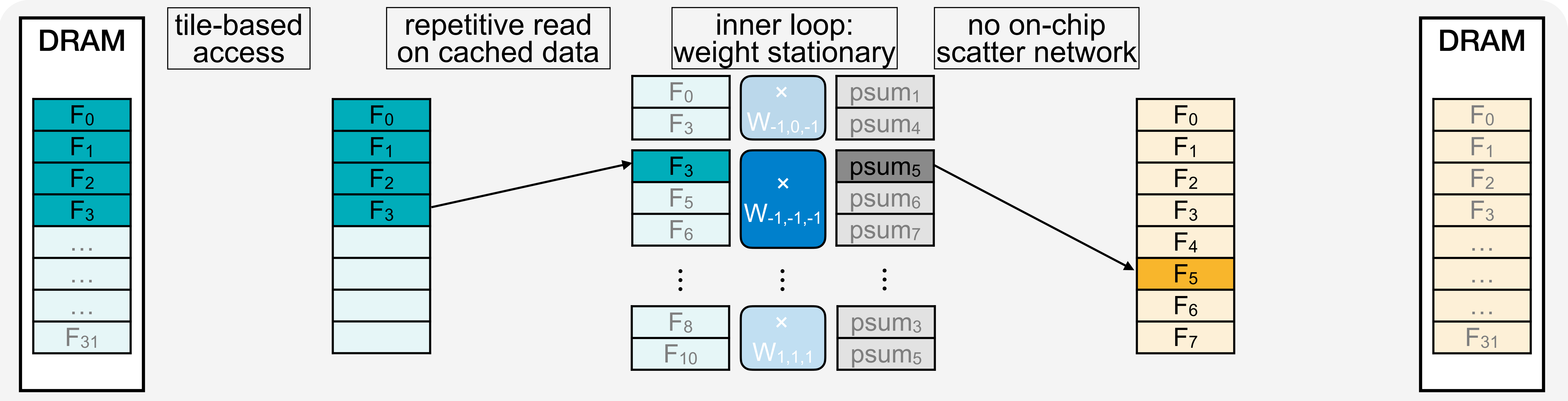
- Sequential fetch input features on demand in the granularity of tile (*i.e.*, “cache block”)
- MatMul computing parallelizes *ic* and *oc* dimension → no need for on-chip scatter network
- Weight stationary saves the on-chip memory footprint



PointAcc implementation

Streaming Sparse MatMul Operations

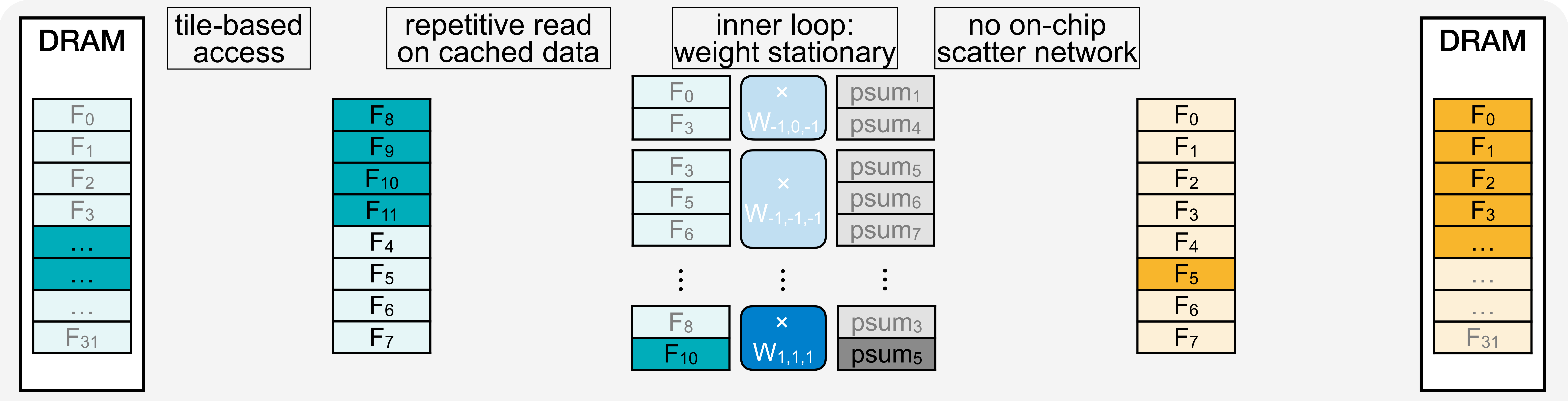
- Sequential fetch input features on demand in the granularity of tile (*i.e.*, "cache block")
- MatMul computing parallelizes *ic* and *oc* dimension → no need for on-chip scatter network
- Weight stationary saves the on-chip memory footprint
- Caching reduces the #off-chip read of reused data to nearly 1 access



PointAcc implementation

Streaming Sparse MatMul Operations

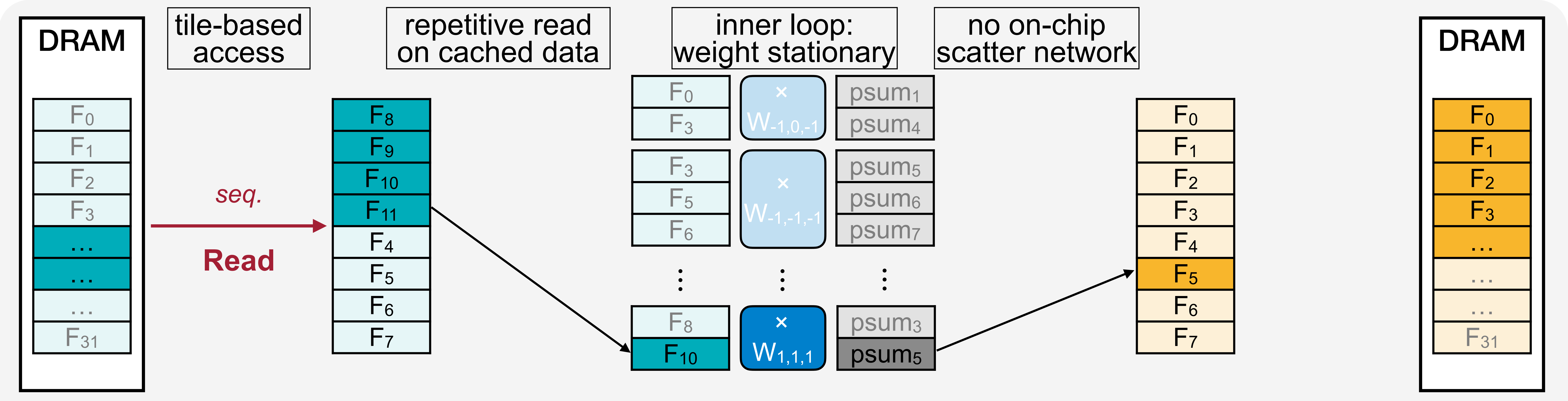
- Sequential fetch input features on demand in the granularity of tile (*i.e.*, “cache block”)
- MatMul computing parallelizes *ic* and *oc* dimension → no need for on-chip scatter network
- Weight stationary saves the on-chip memory footprint
- Caching reduces the #off-chip read of reused data to nearly 1 access



PointAcc implementation

Streaming Sparse MatMul Operations

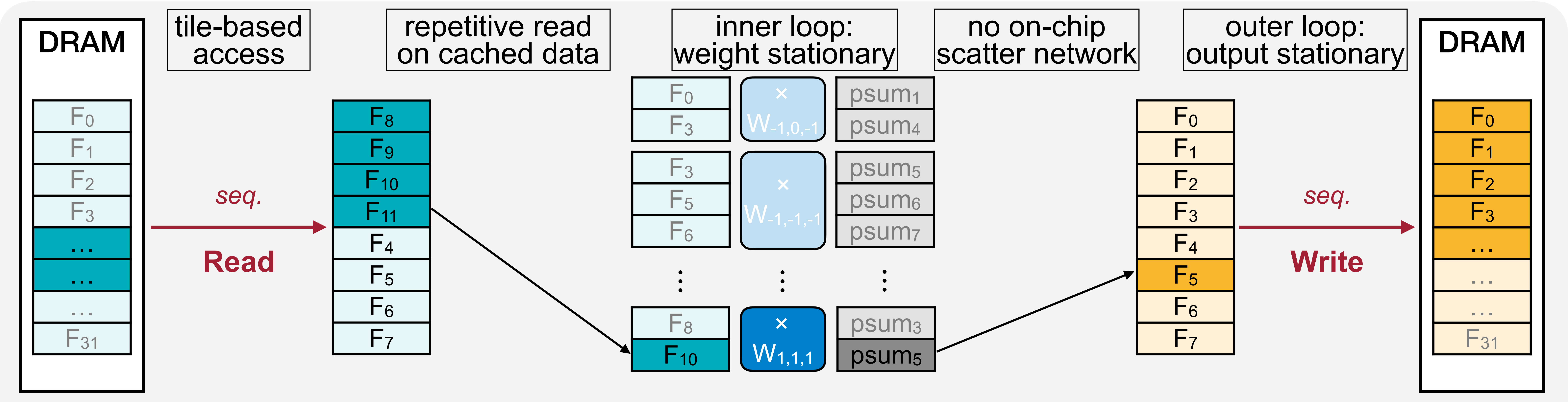
- Sequential fetch input features on demand in the granularity of tile (*i.e.*, "cache block")
- MatMul computing parallelizes *ic* and *oc* dimension → no need for on-chip scatter network
- Weight stationary saves the on-chip memory footprint
- Caching reduces the #off-chip read of reused data to nearly 1 access



PointAcc implementation

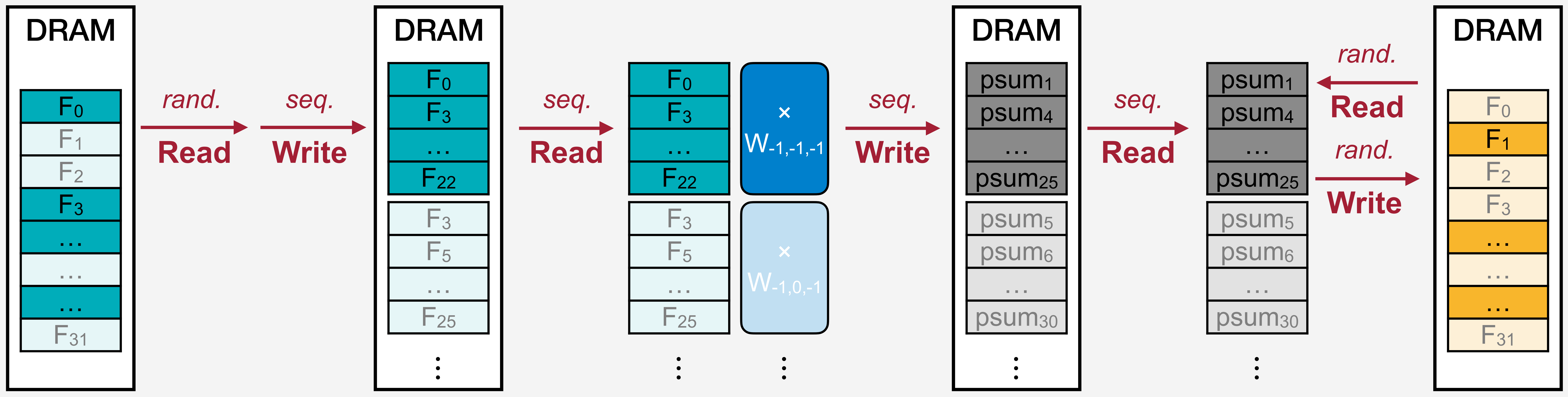
Streaming Sparse MatMul Operations

- Sequential fetch input features on demand in the granularity of tile (*i.e.*, "cache block")
- MatMul computing parallelizes *ic* and *oc* dimension → no need for on-chip scatter network
- Weight stationary saves the on-chip memory footprint
- Caching reduces the #off-chip read of reused data to nearly 1 access
- Output stationary eliminates the off-chip scattering of partial sums

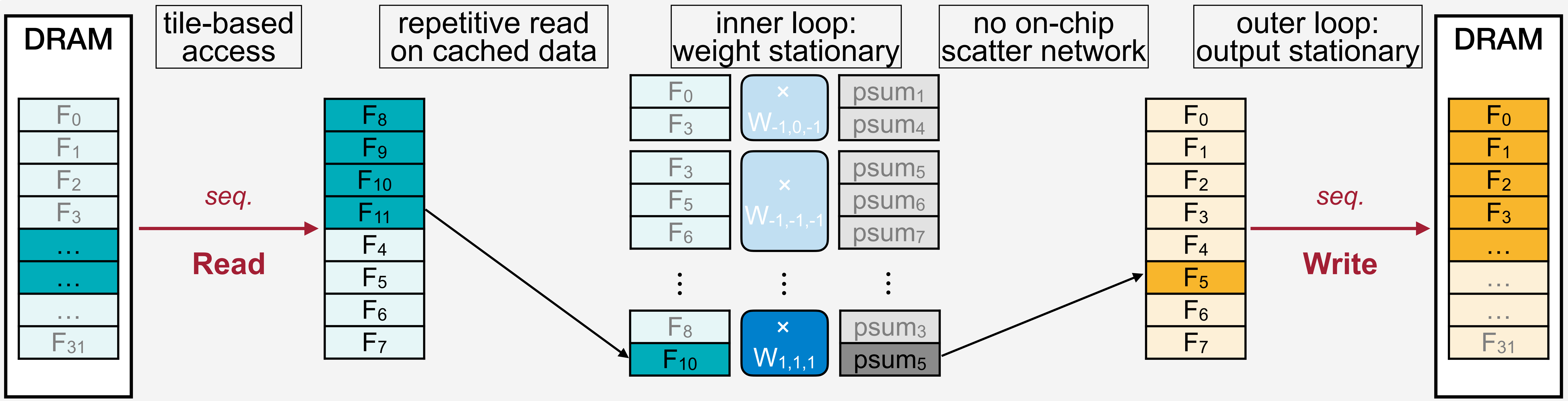


PointAcc implementation

Streaming Sparse MatMul Operations



the state-of-the-art GPU implementation



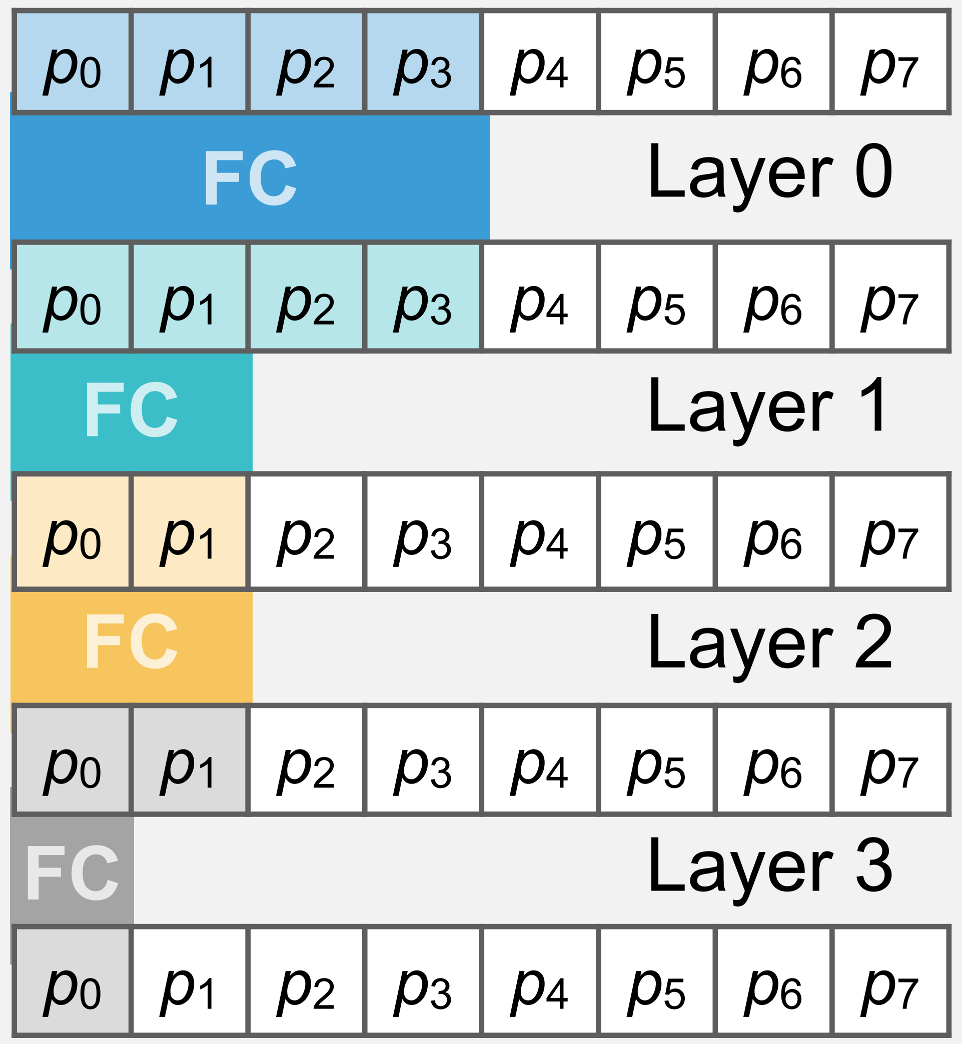
PointAcc implementation

Flexible Memory Management

Sparse Computation

Streaming Computation with Caching

Dense Computation

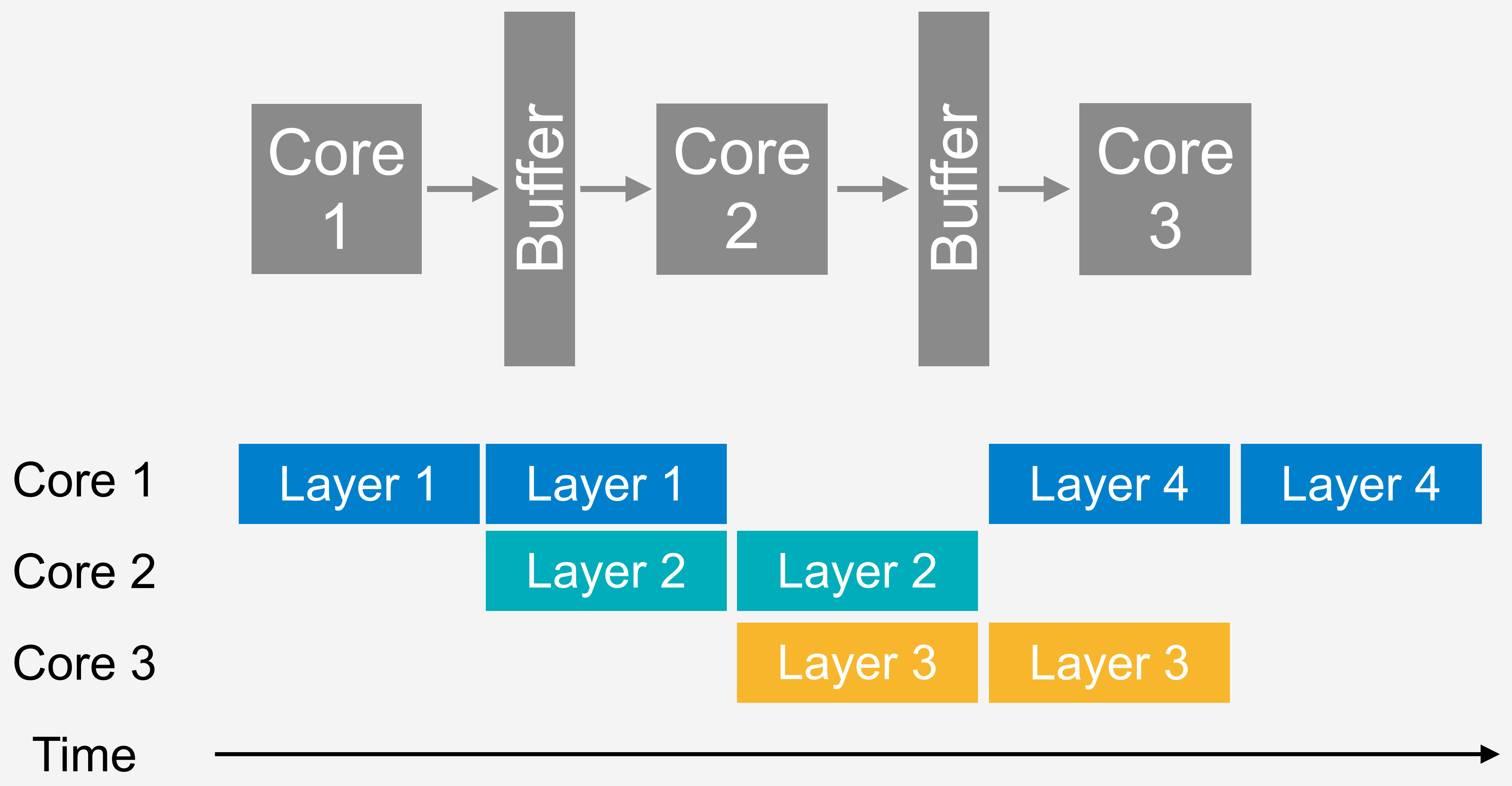


Temporal Layer Fusion

Step 4: Transform Input Features

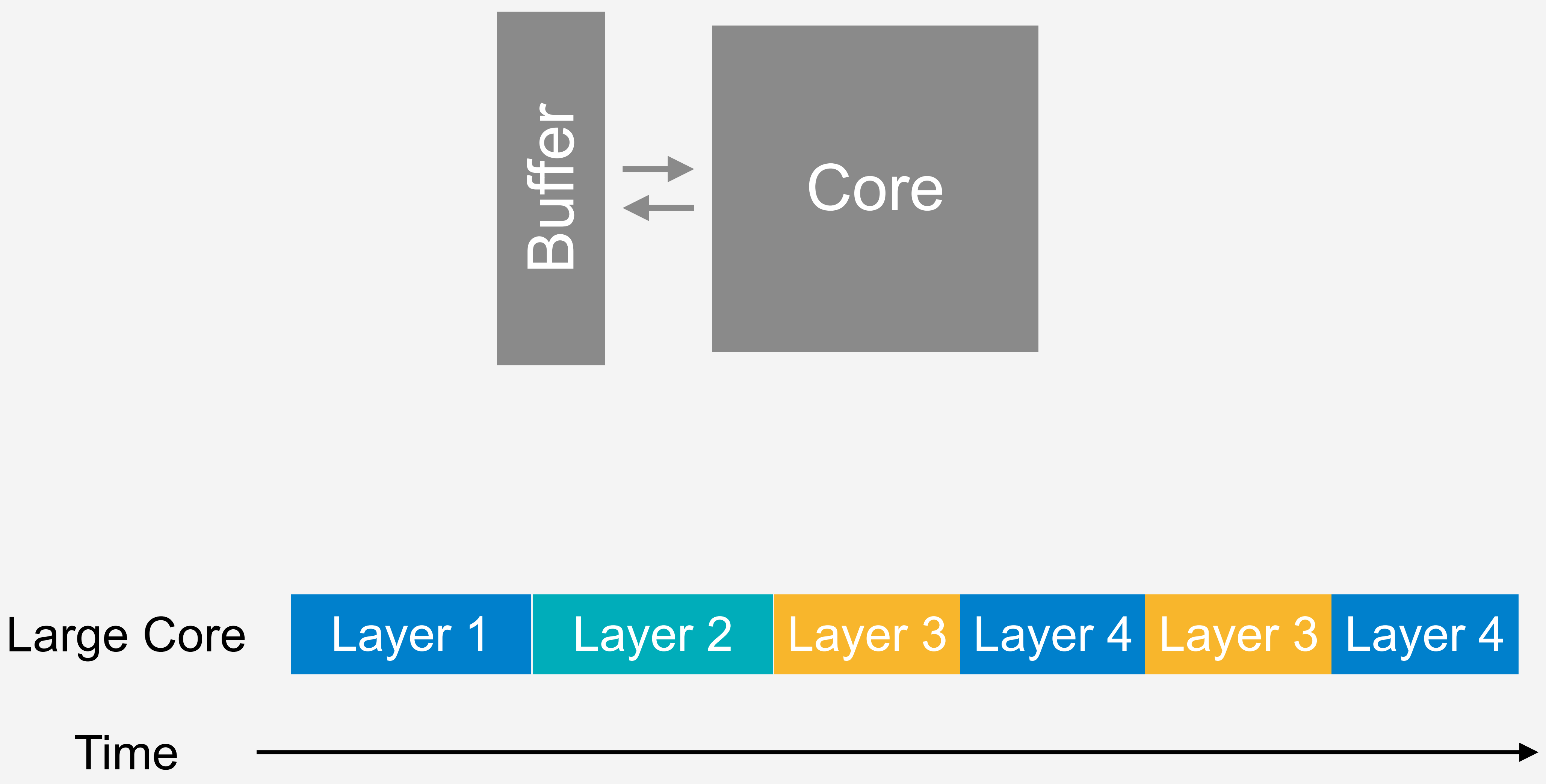
Layer Fusion

Spatial Layer Fusion



- ✓ Reduced off-chip memory access
- ✗ Complicated logic overhead
- ✗ Inflexible intermediate buffer use
- ✗ Fixed #layers for fusion

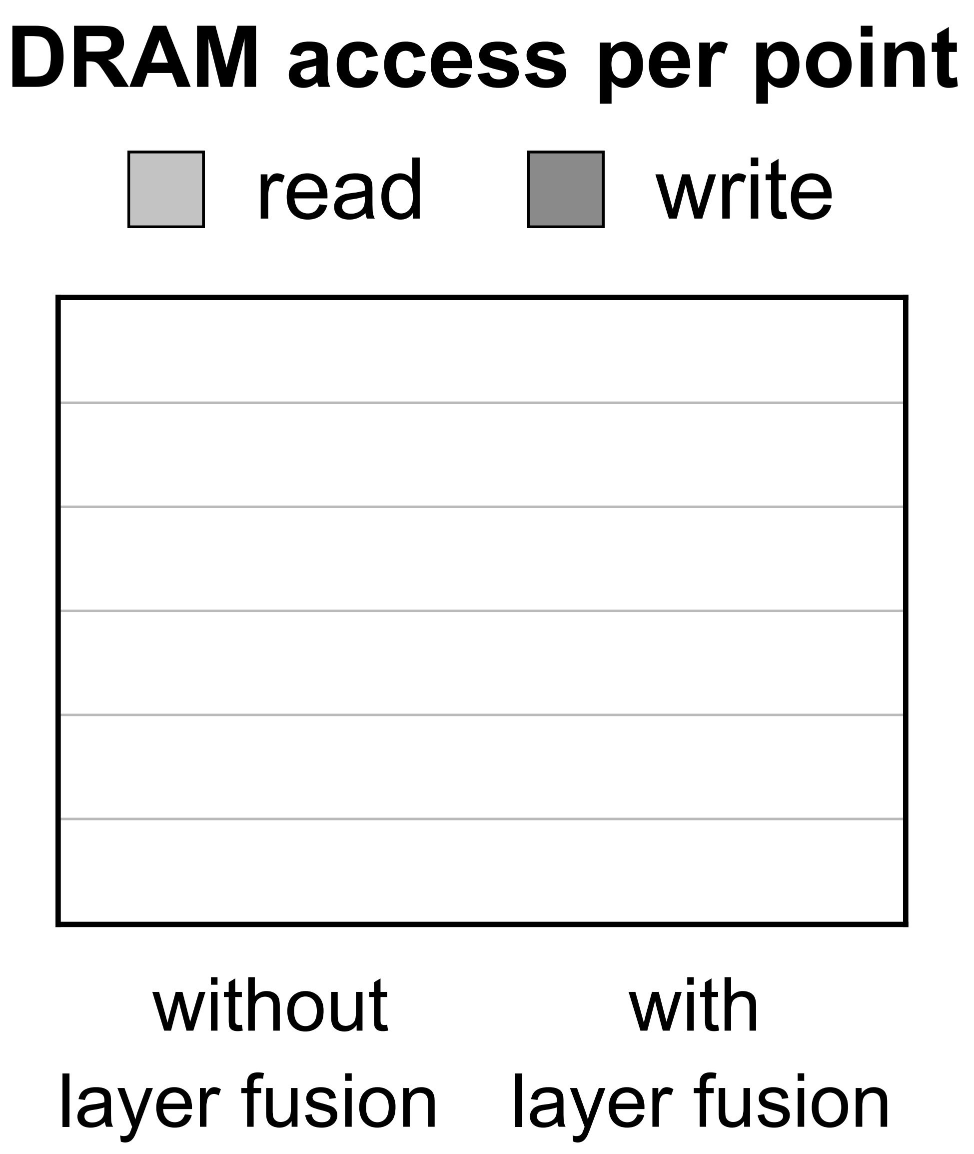
Temporal Layer Fusion

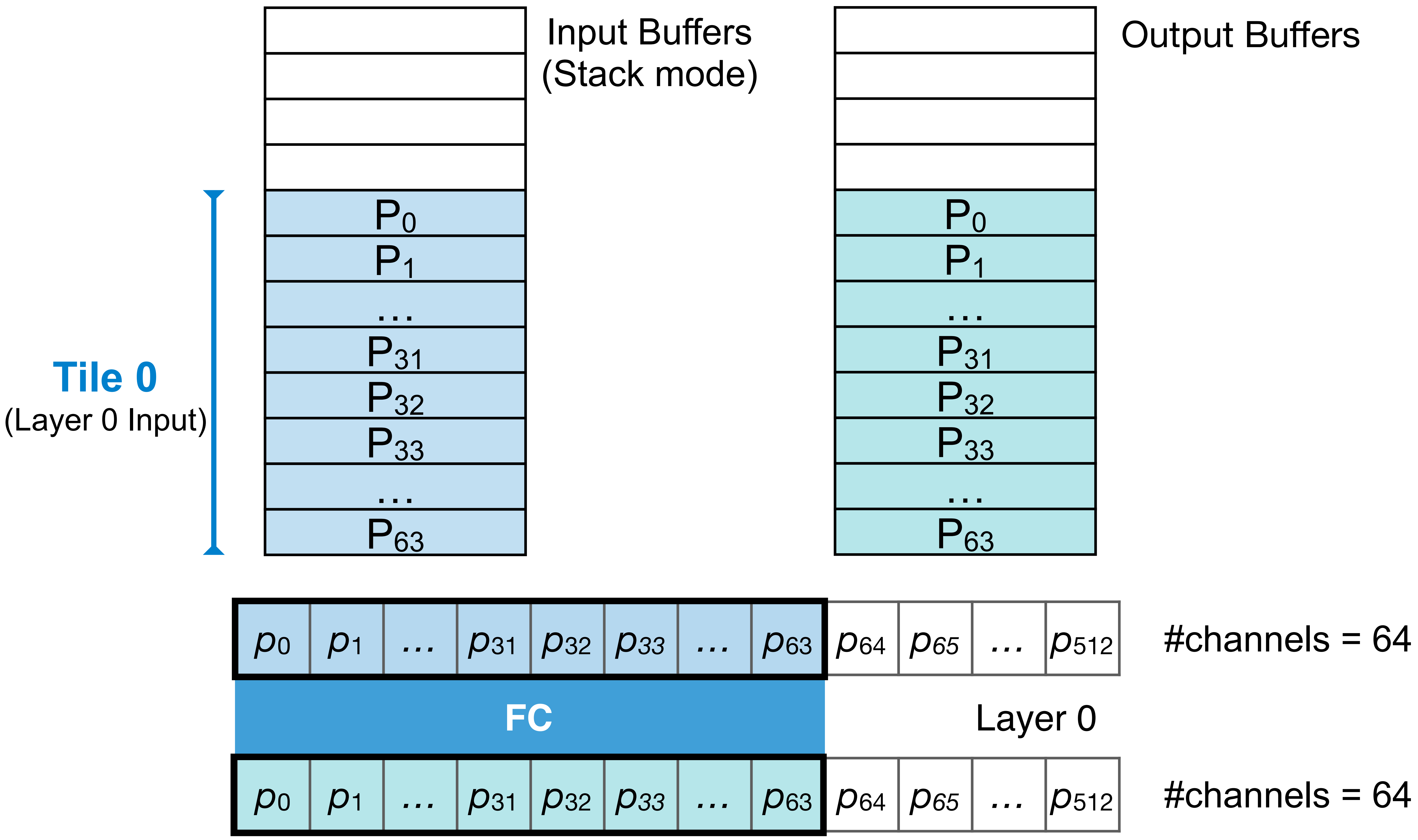


- ✓ Reduced off-chip memory access
- ✓ Simple logic modification
- ✓ Flexible buffer use for better layer tiling
- ✓ Flexible #layers for fusion

consecutive FCs varies among different point cloud NNs, and even among different blocks of the same model

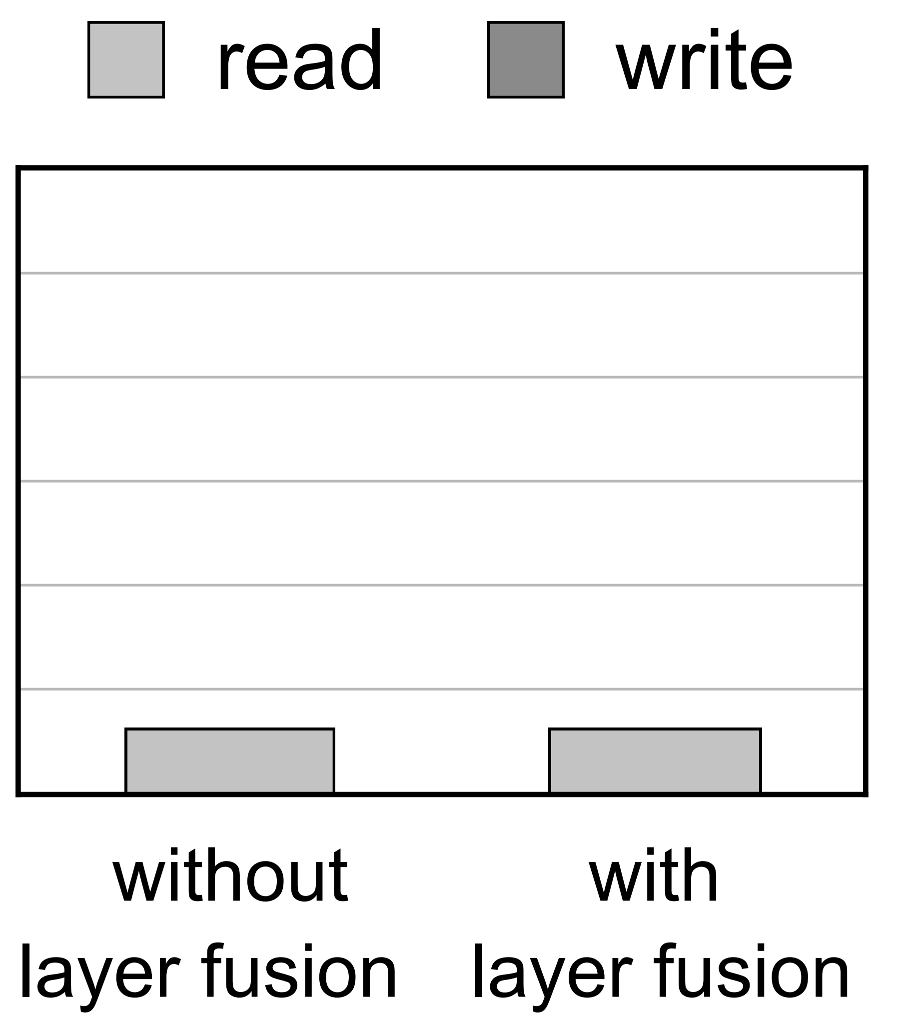
Temporal Layer Fusion on consecutive FCs

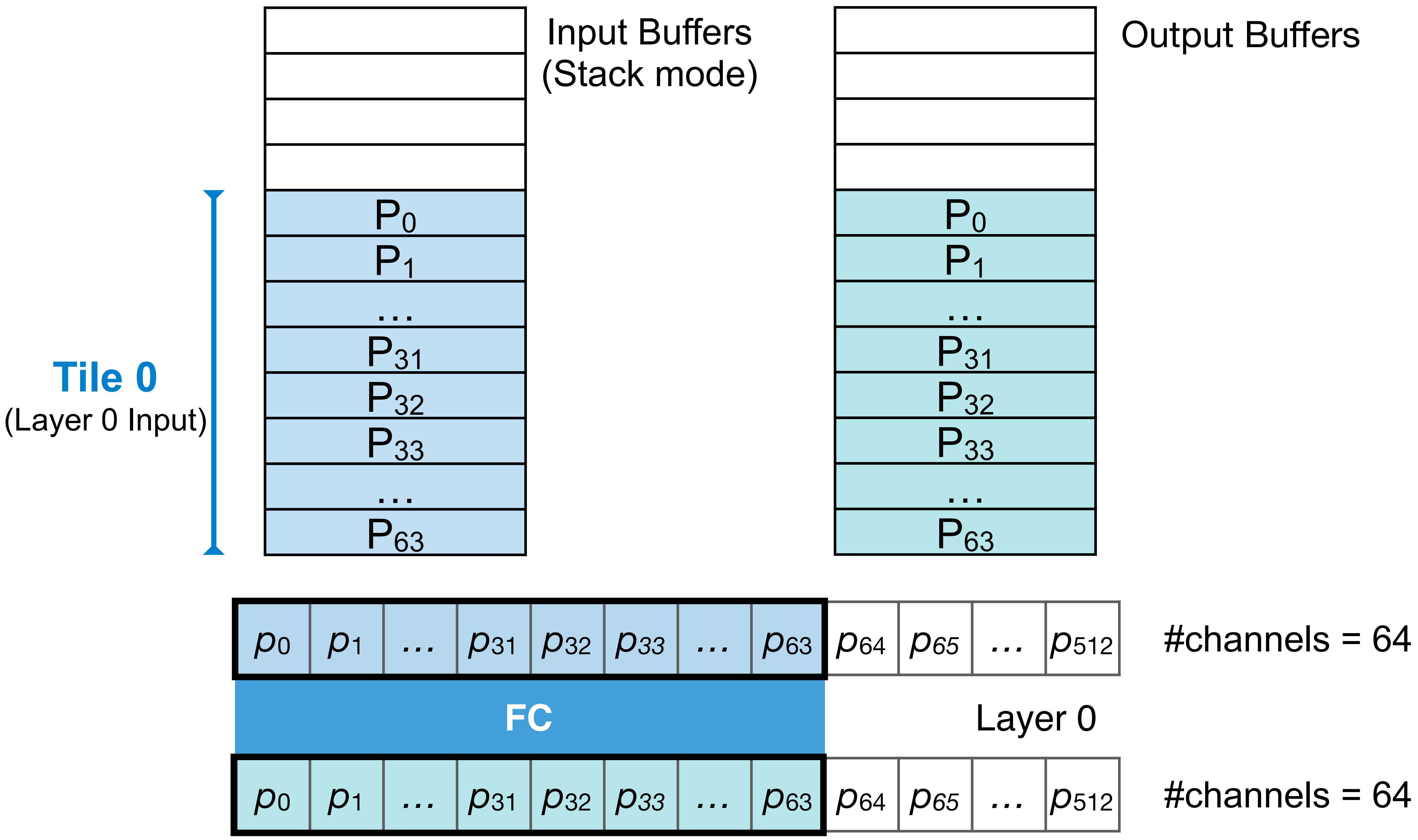




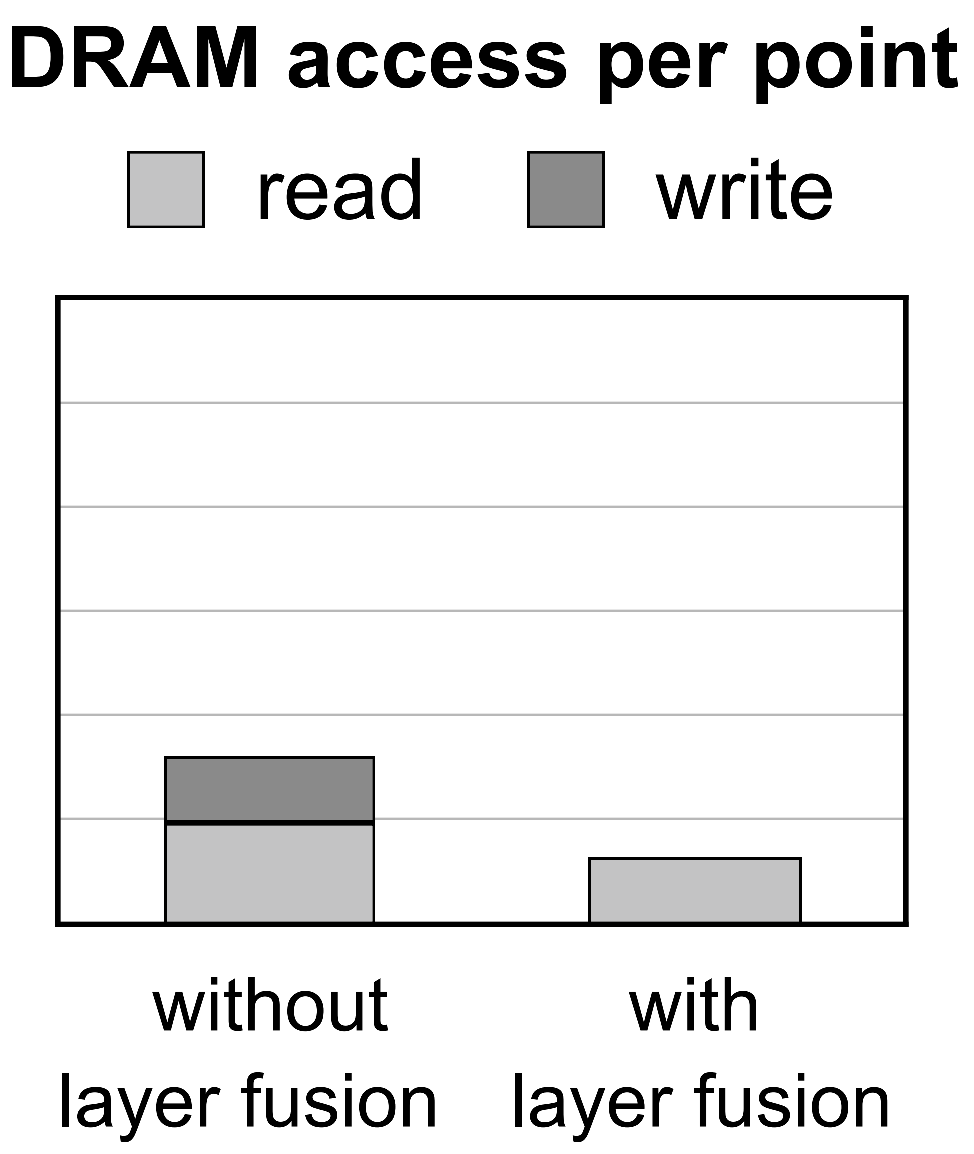
Temporal Layer Fusion on consecutive FCs

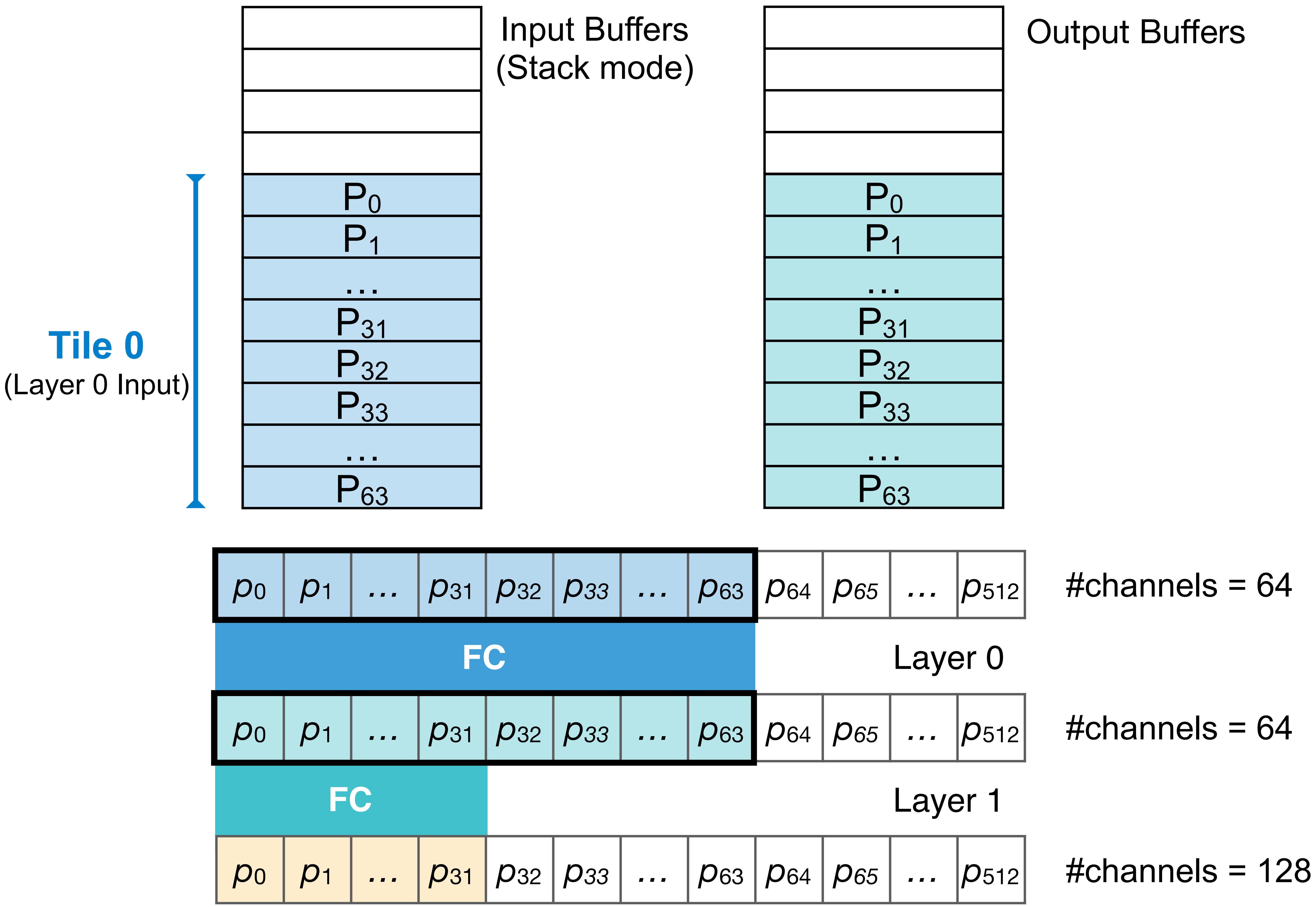
DRAM access per point





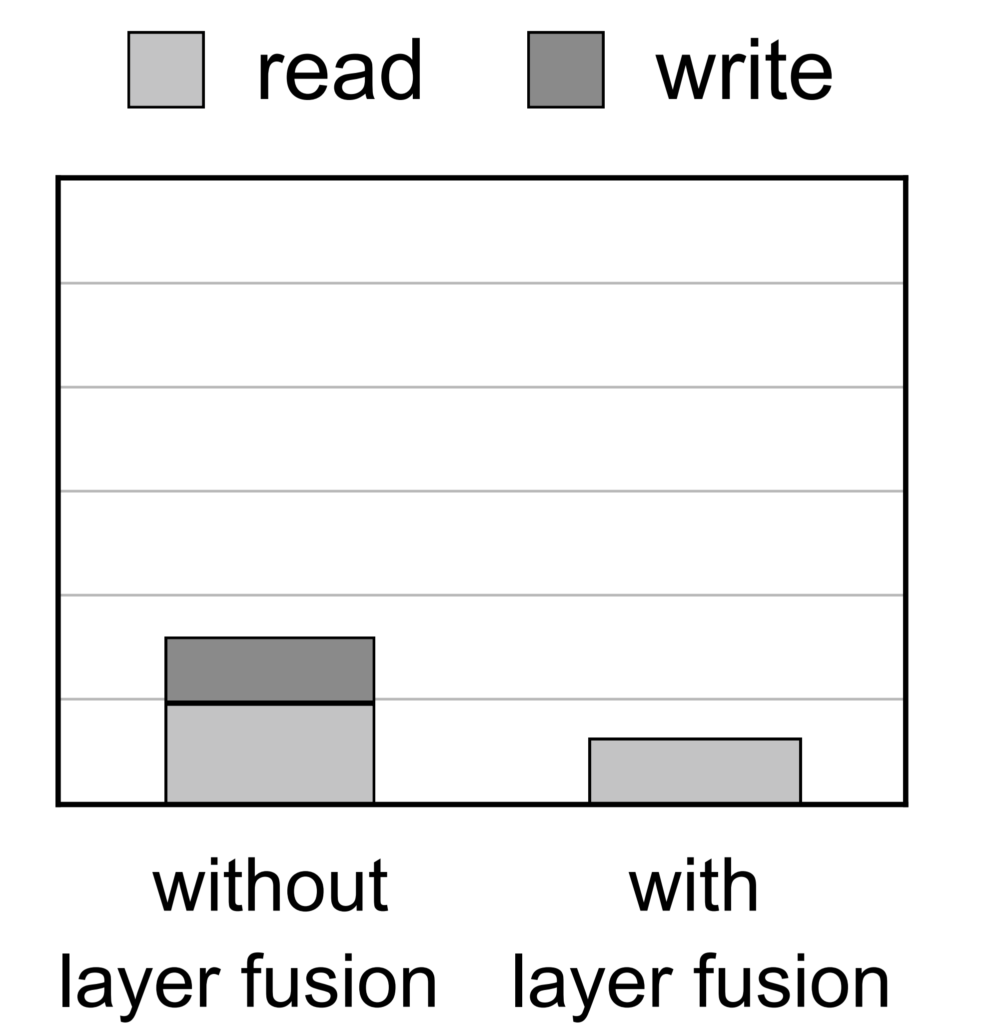
Temporal Layer Fusion on consecutive FCs

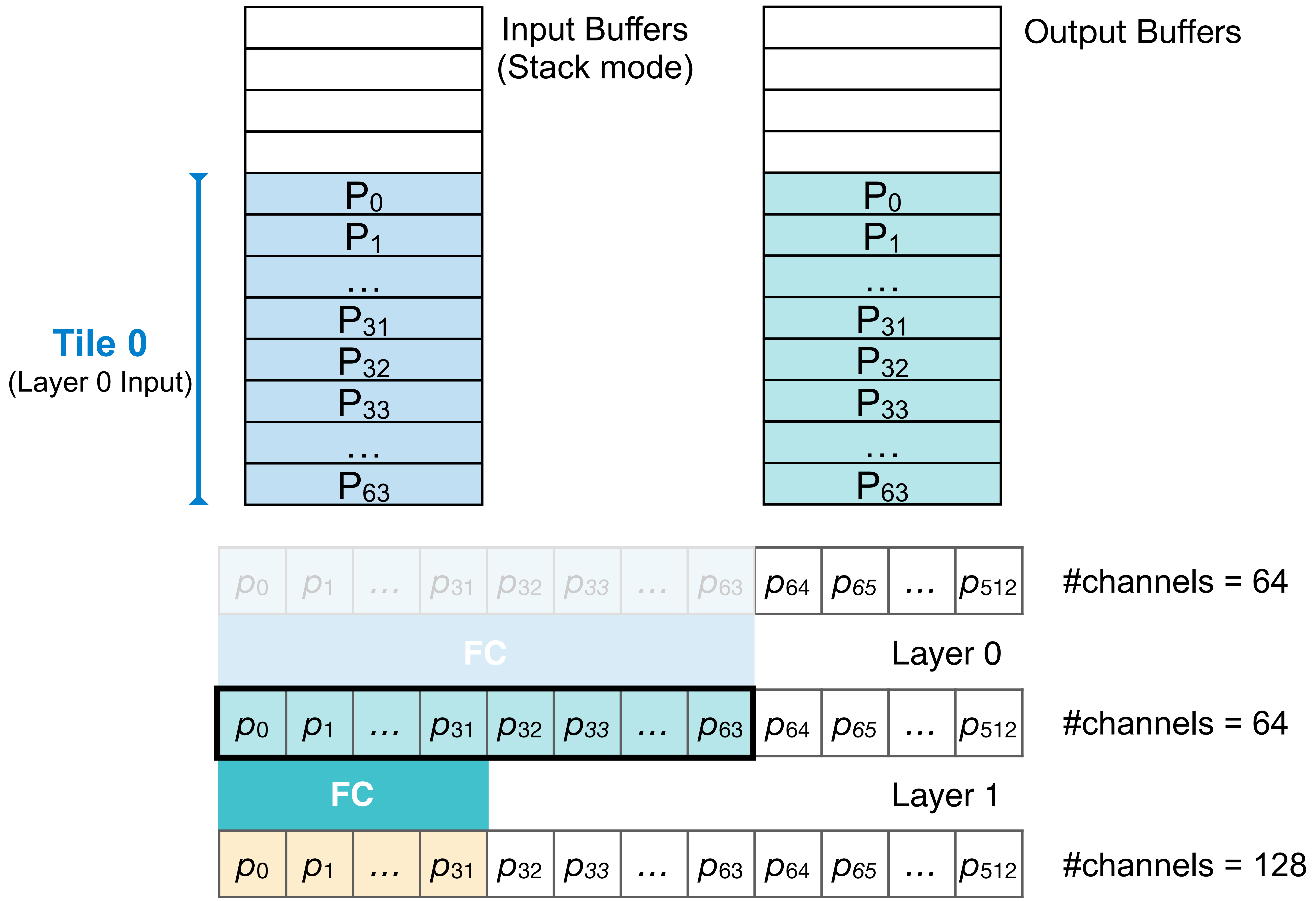




Temporal Layer Fusion on consecutive FCs

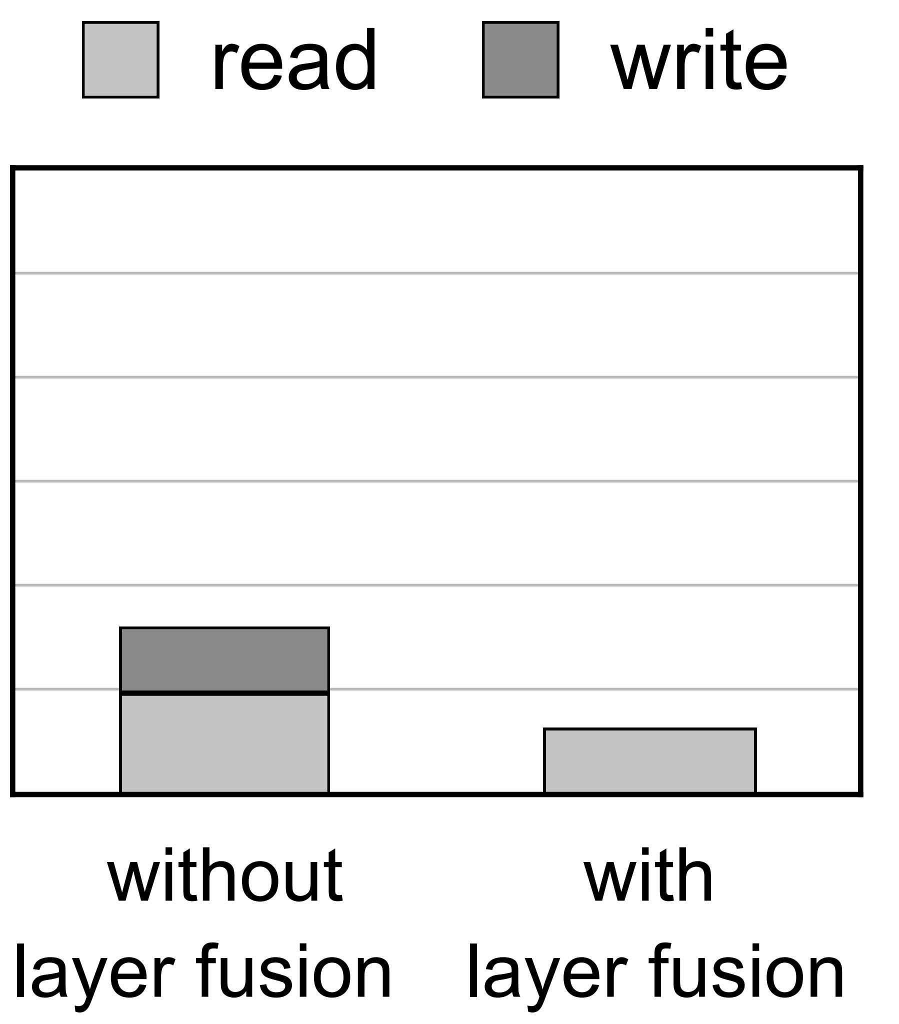
DRAM access per point

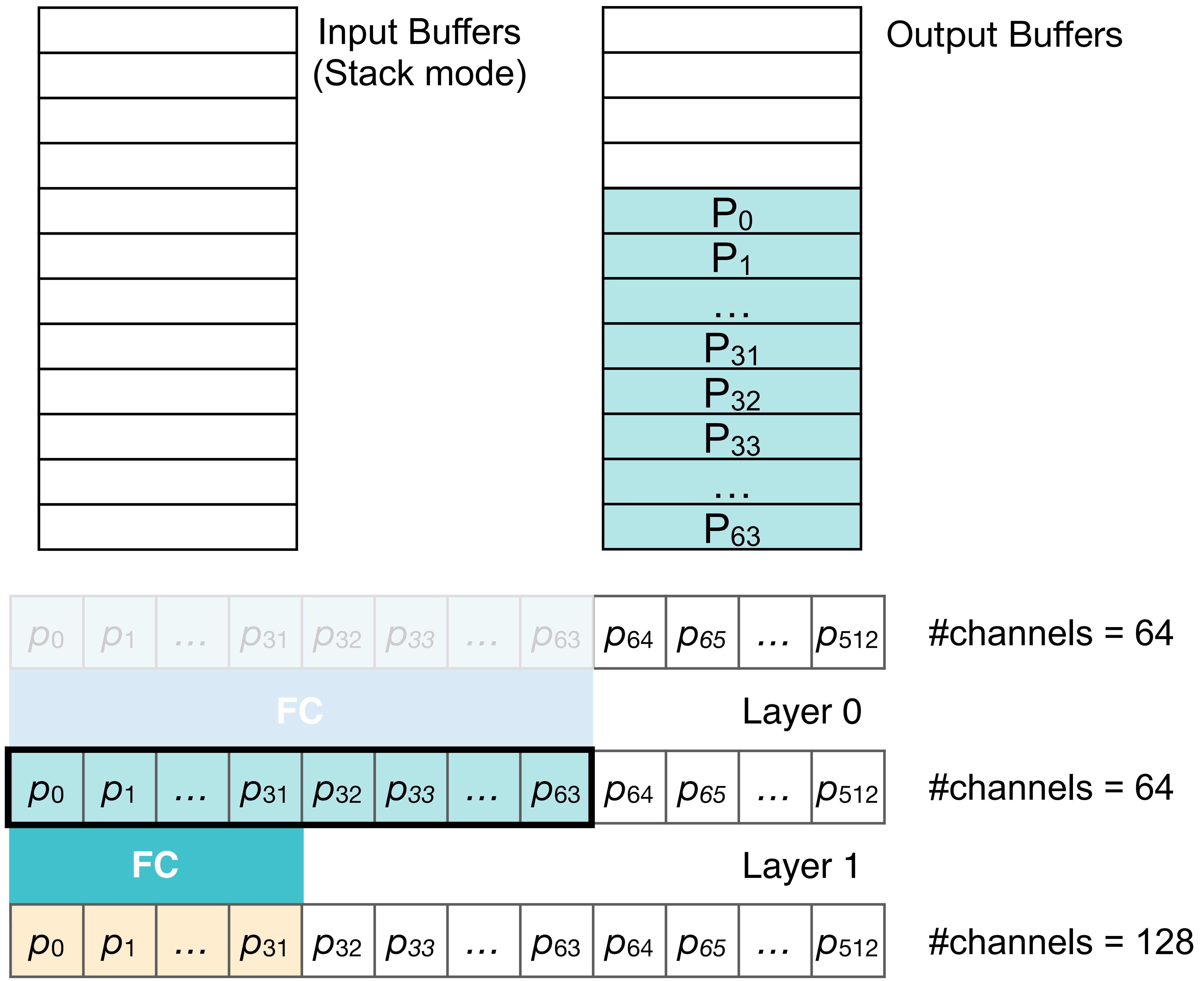




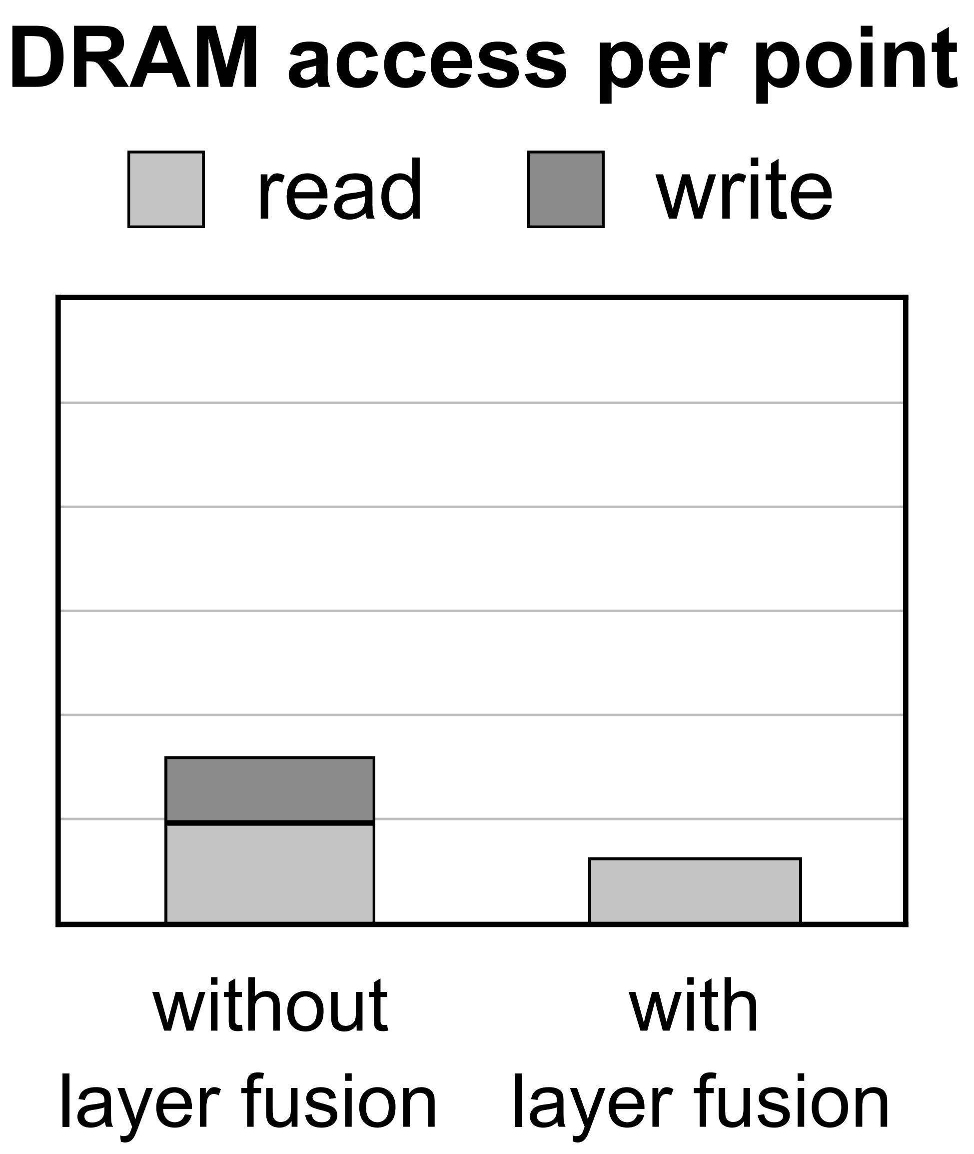
Temporal Layer Fusion on consecutive FCs

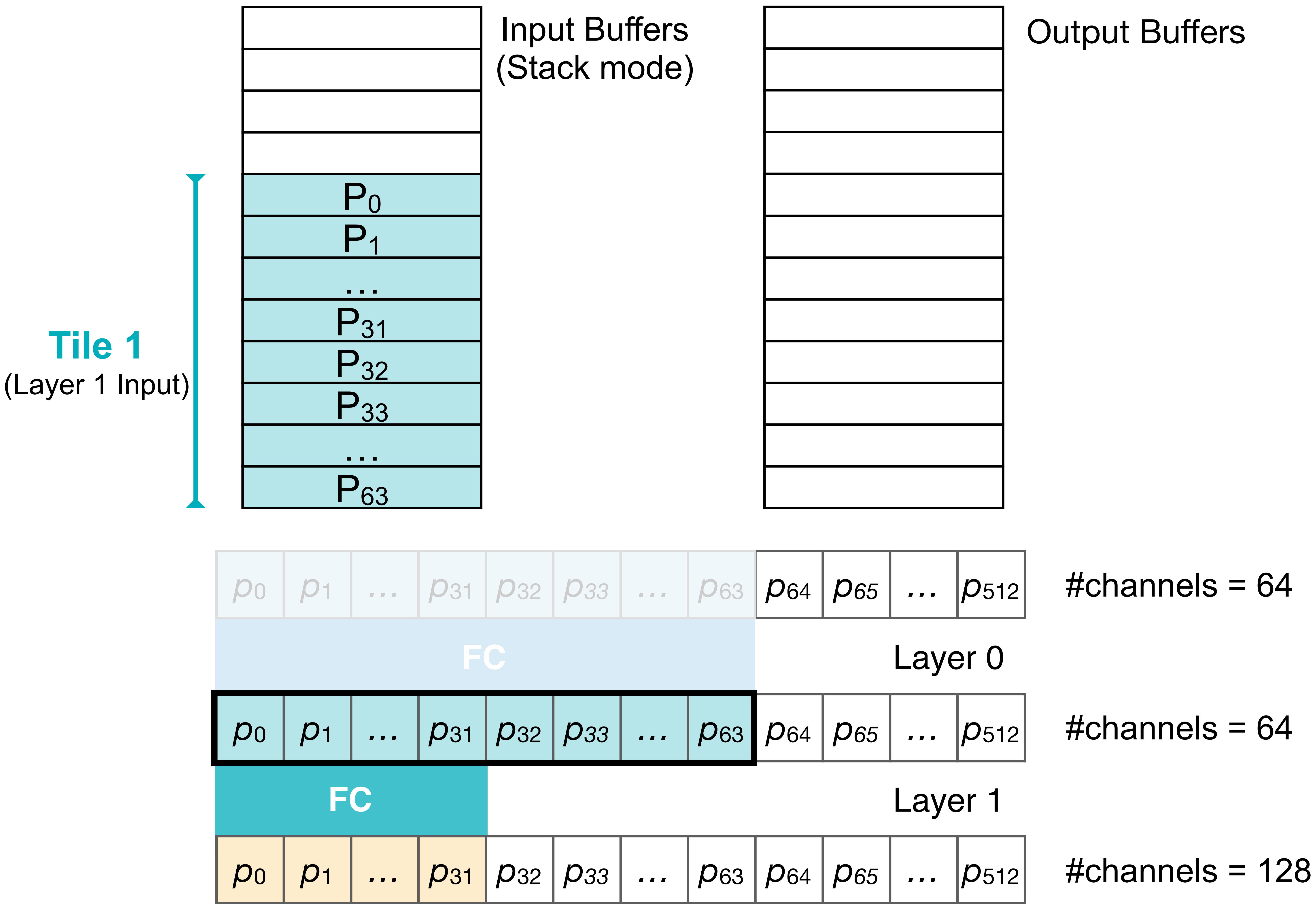
DRAM access per point



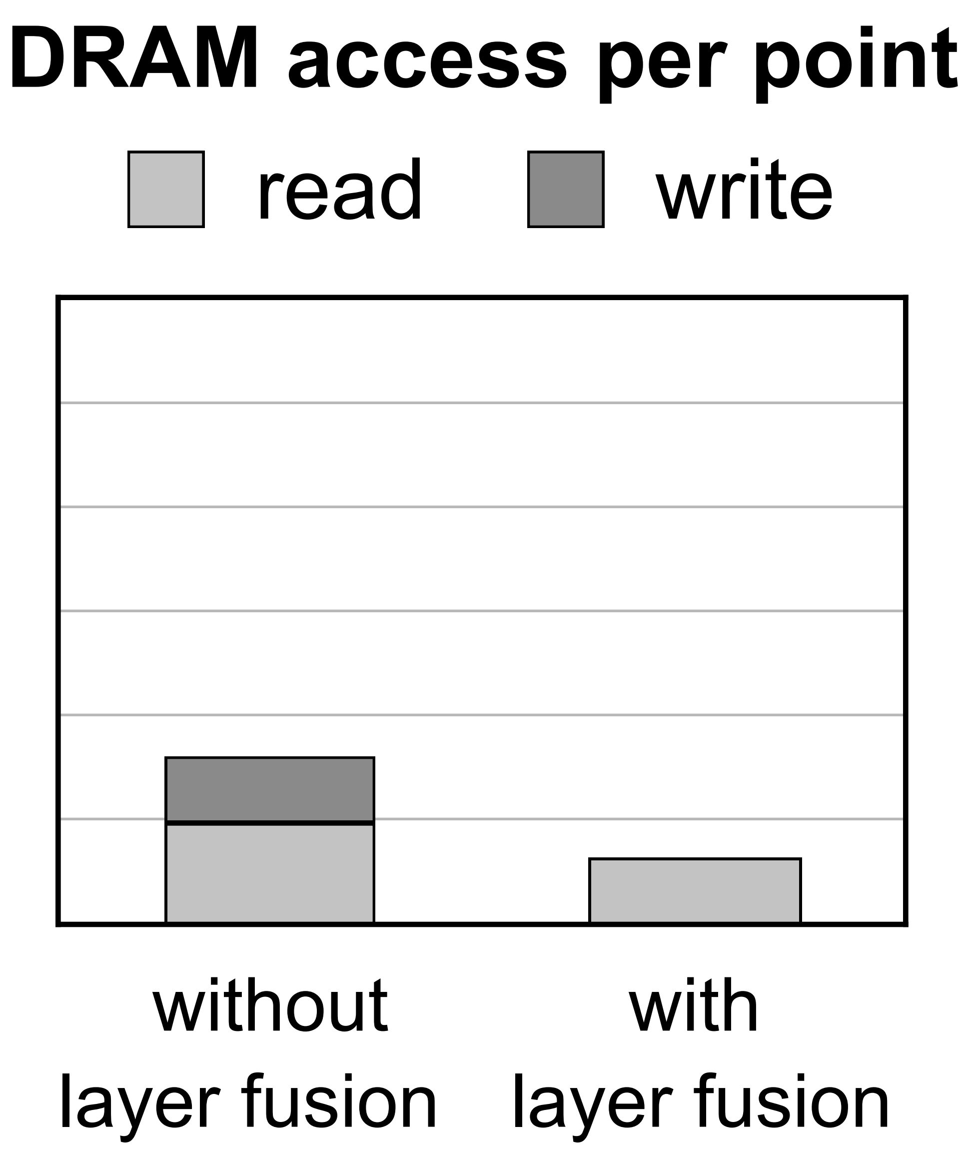


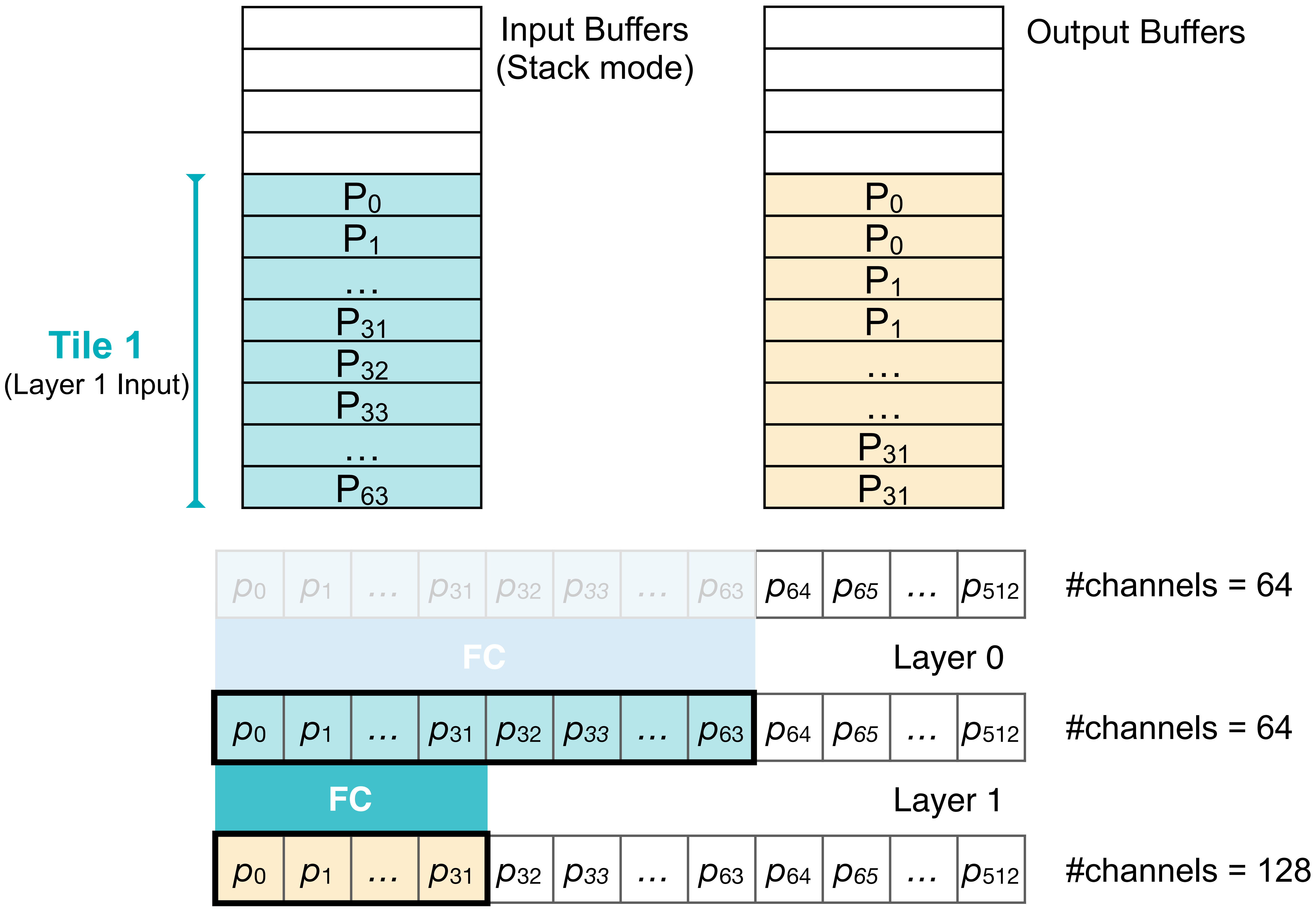
Temporal Layer Fusion on consecutive FCs



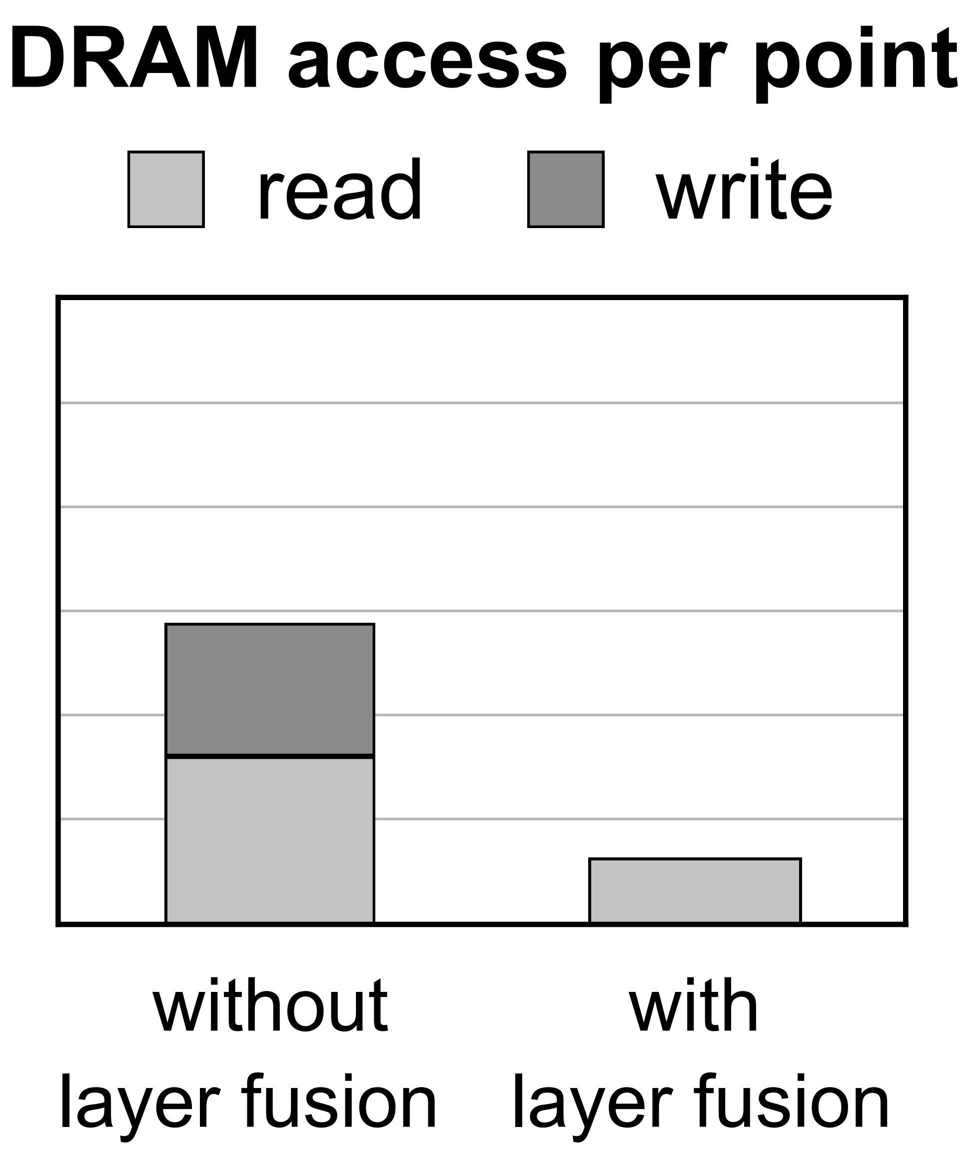


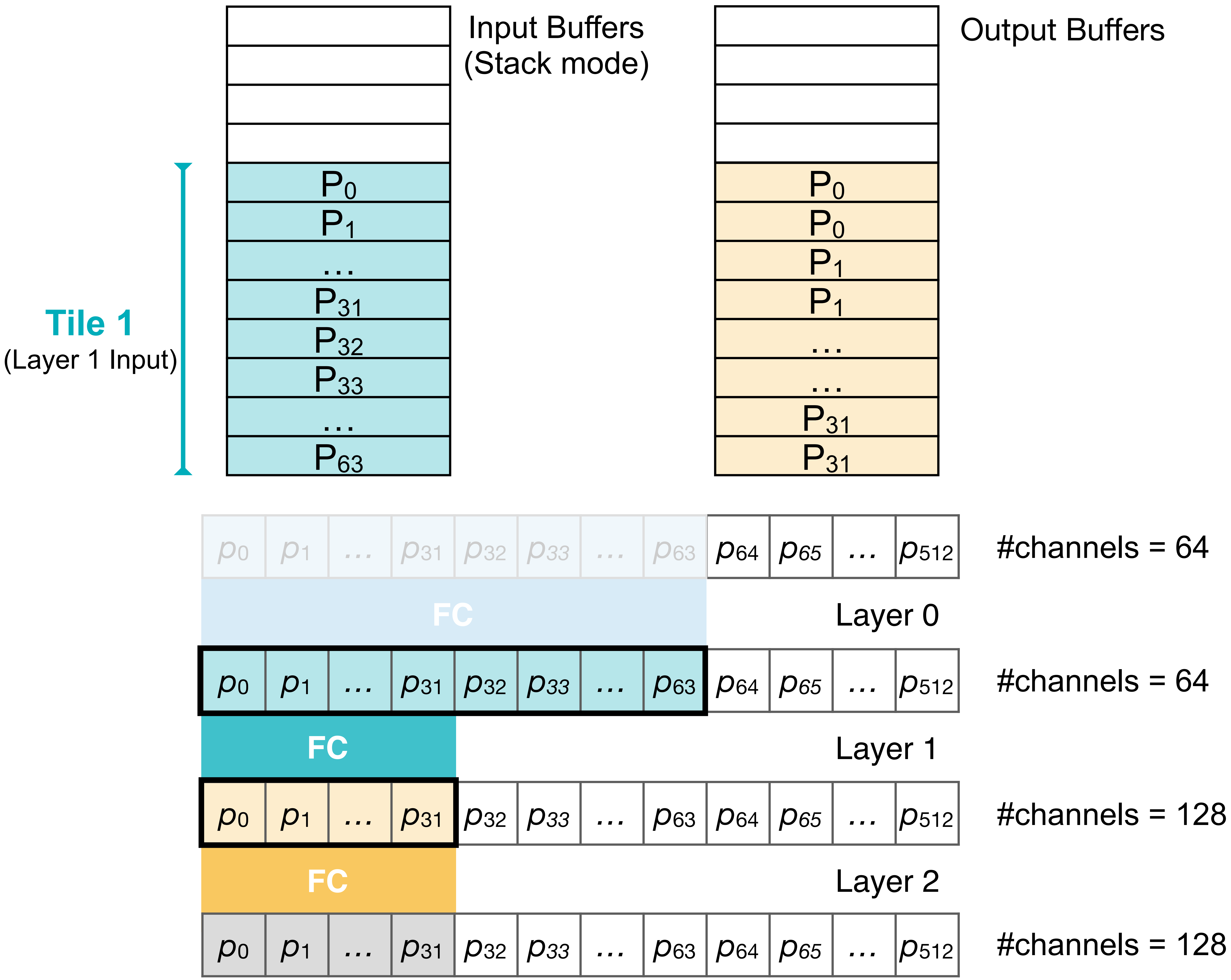
Temporal Layer Fusion on consecutive FCs



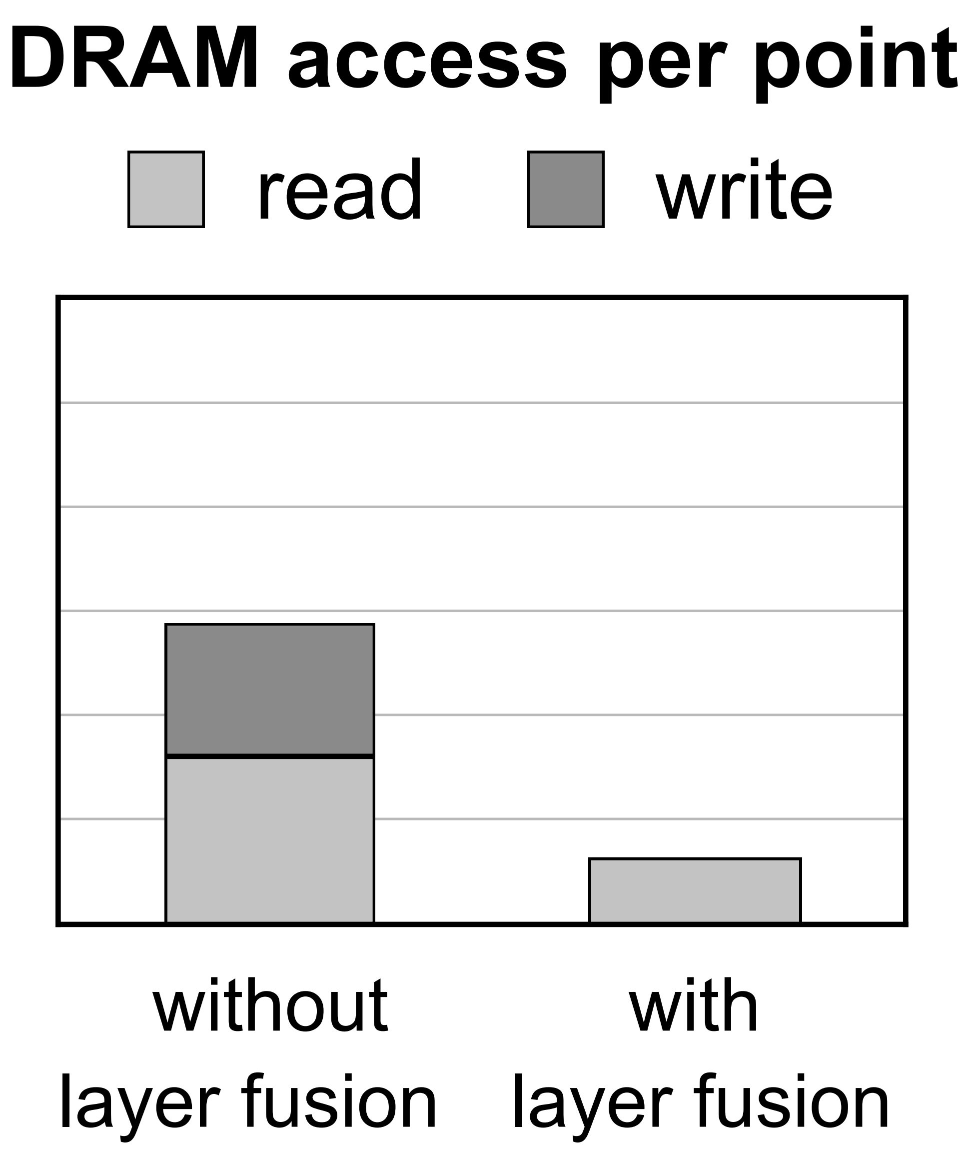


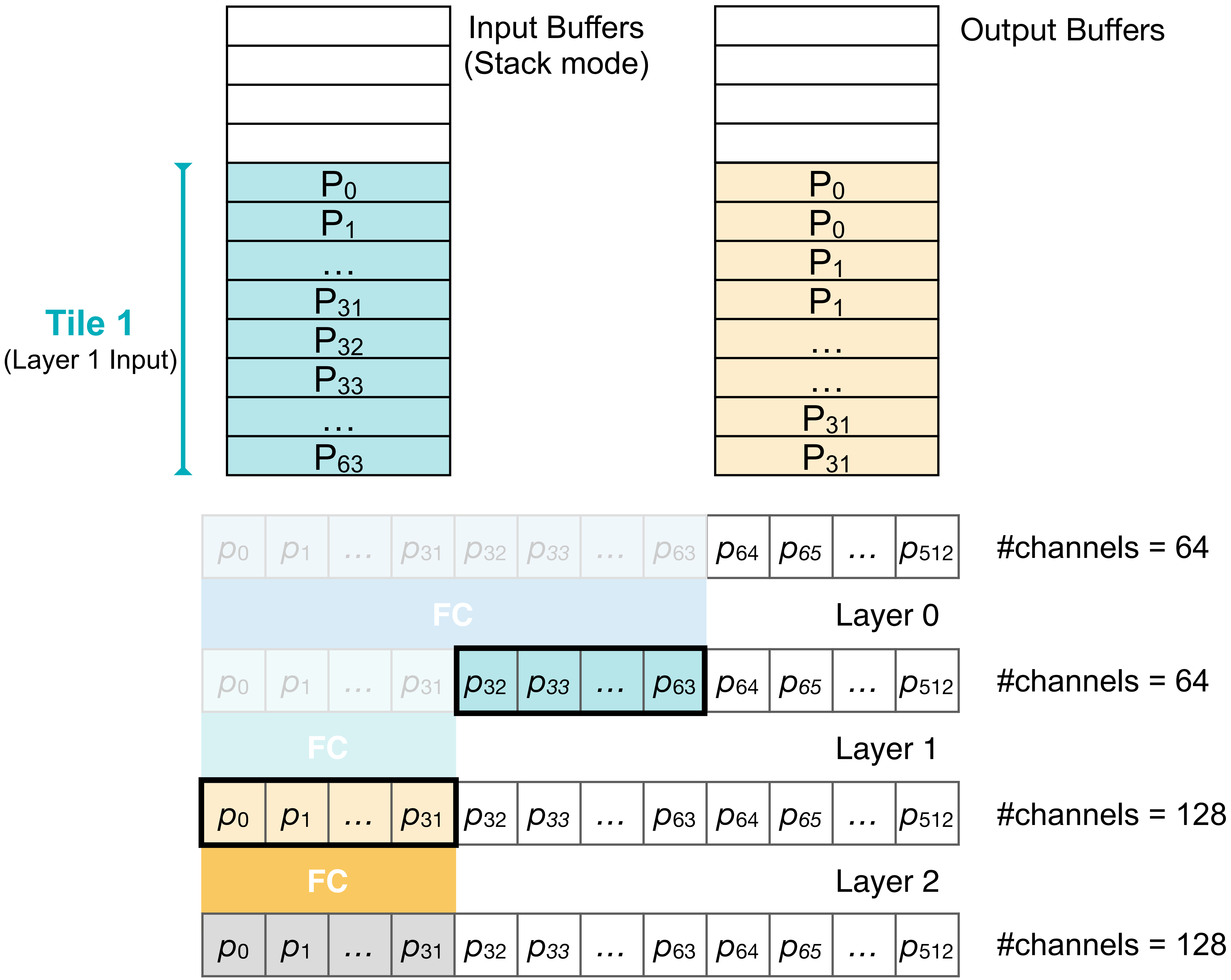
Temporal Layer Fusion on consecutive FCs





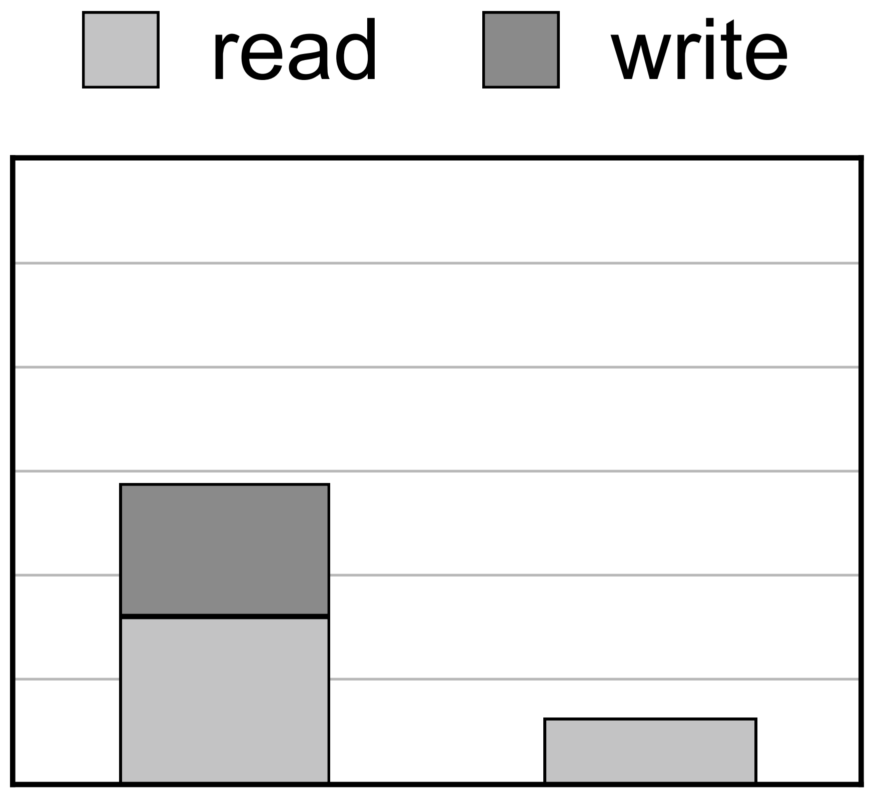
Temporal Layer Fusion on consecutive FCs





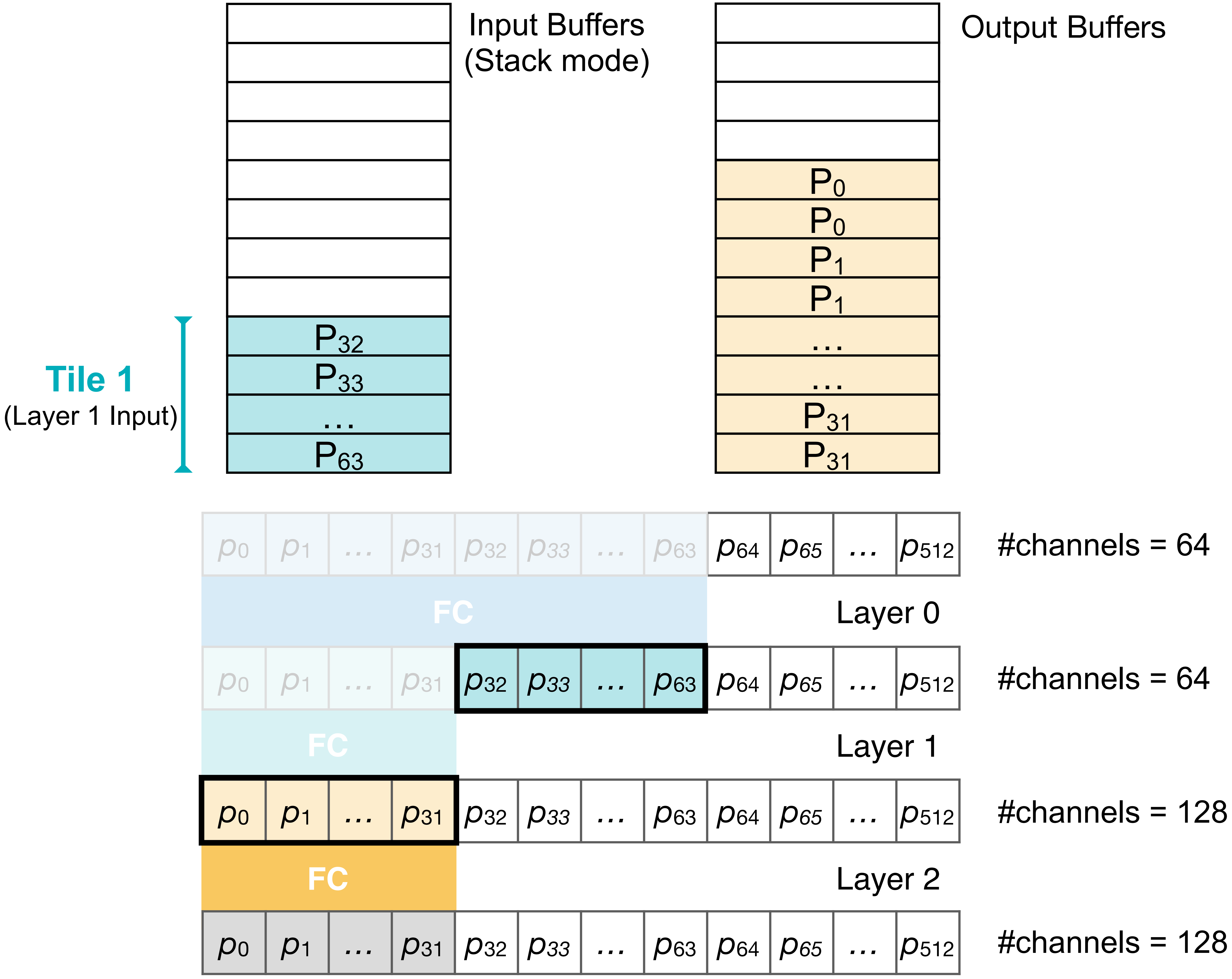
Temporal Layer Fusion on consecutive FCs

DRAM access per point

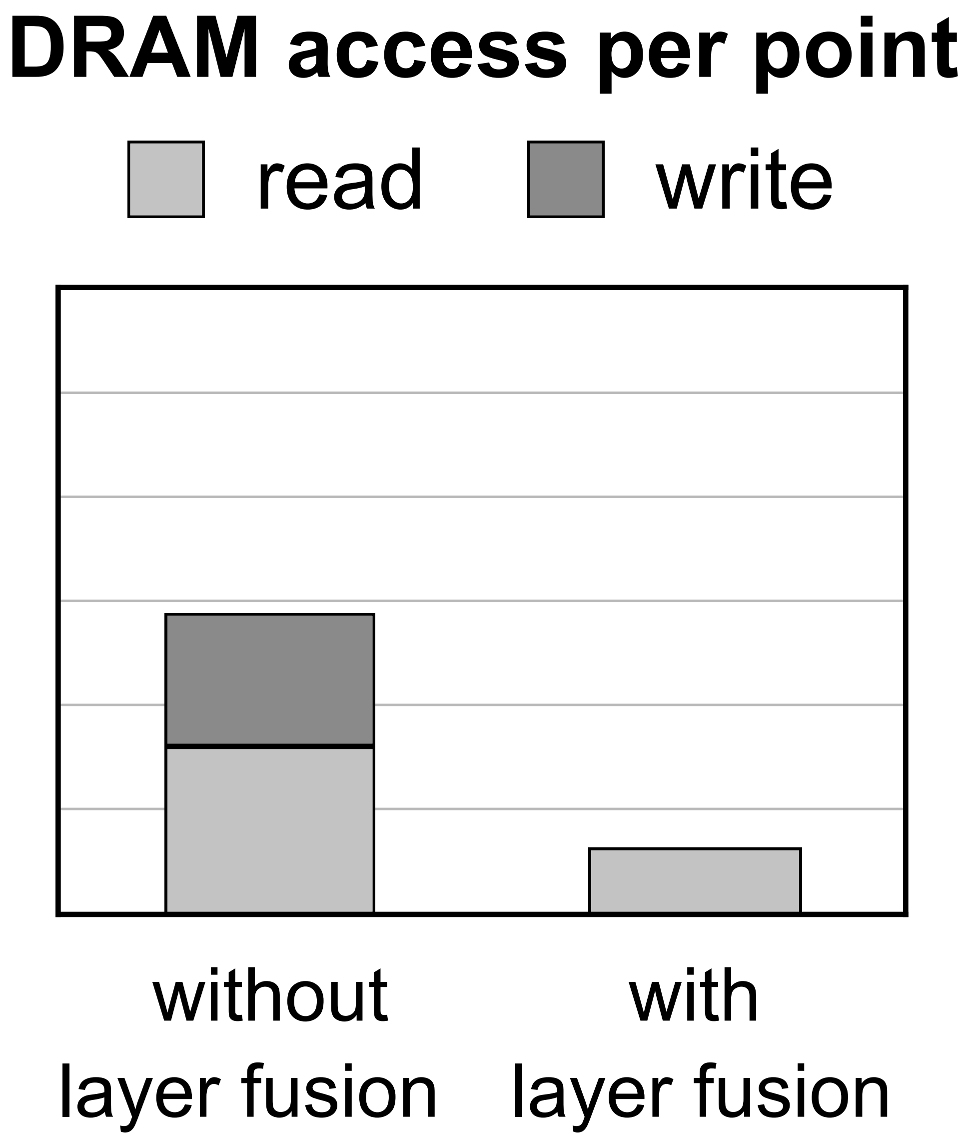


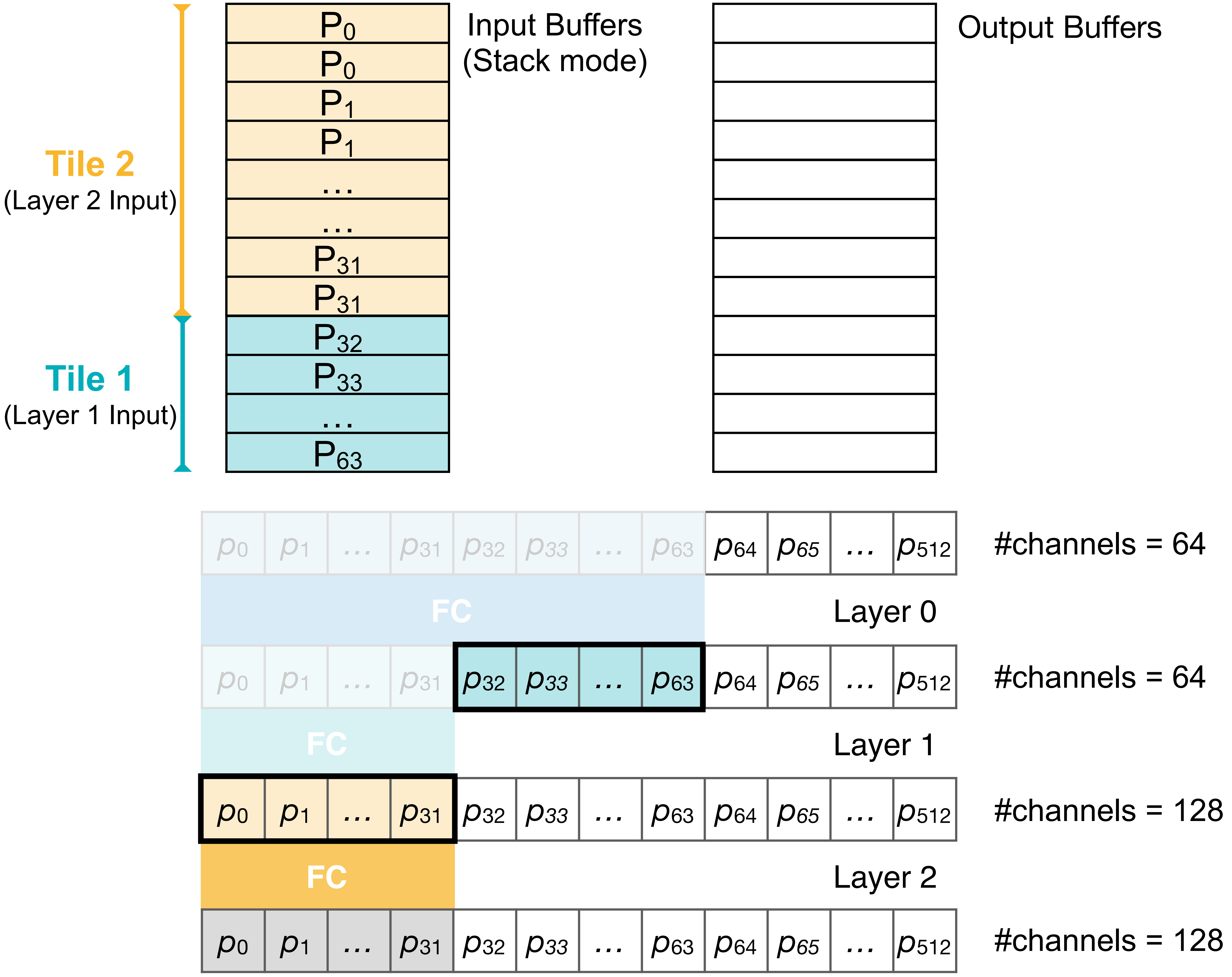
without layer fusion

with layer fusion

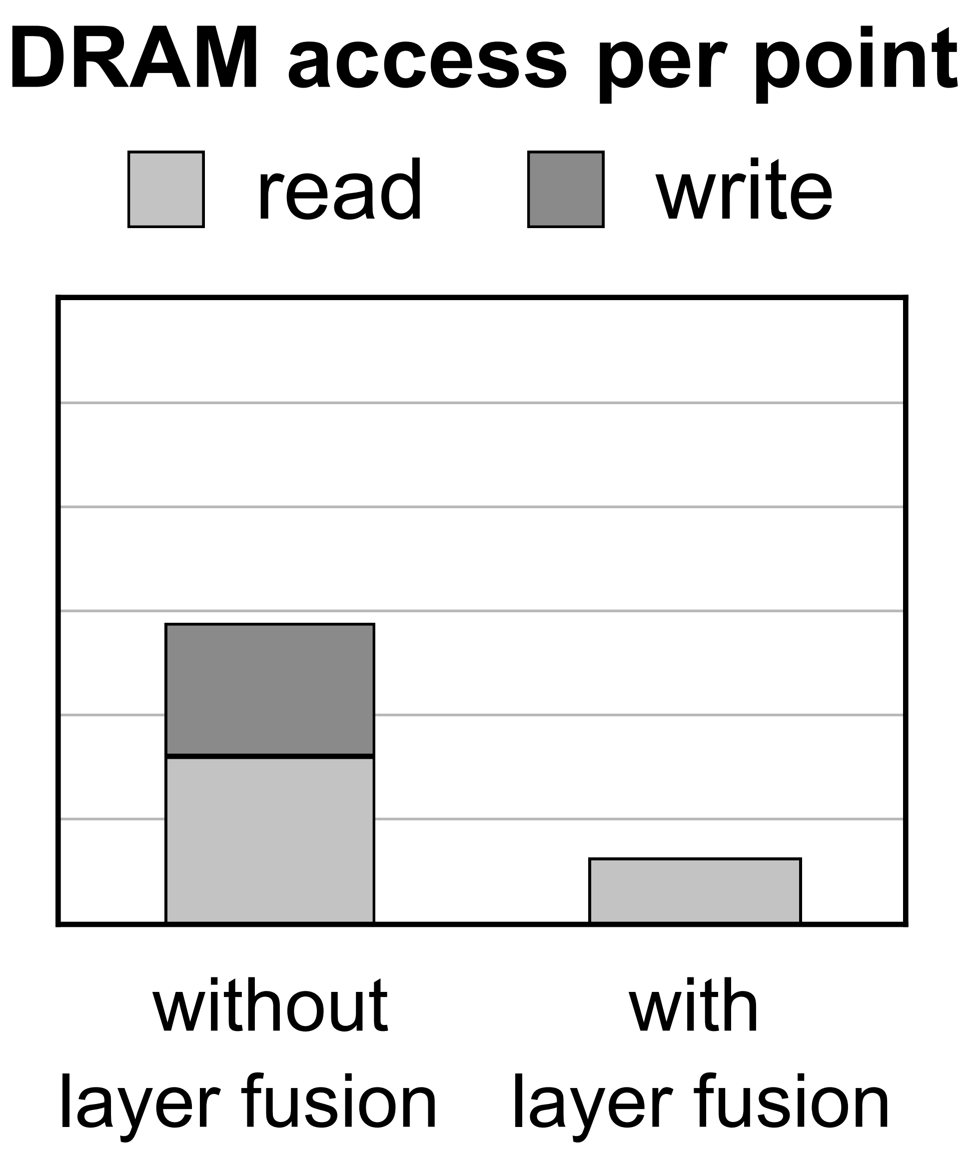


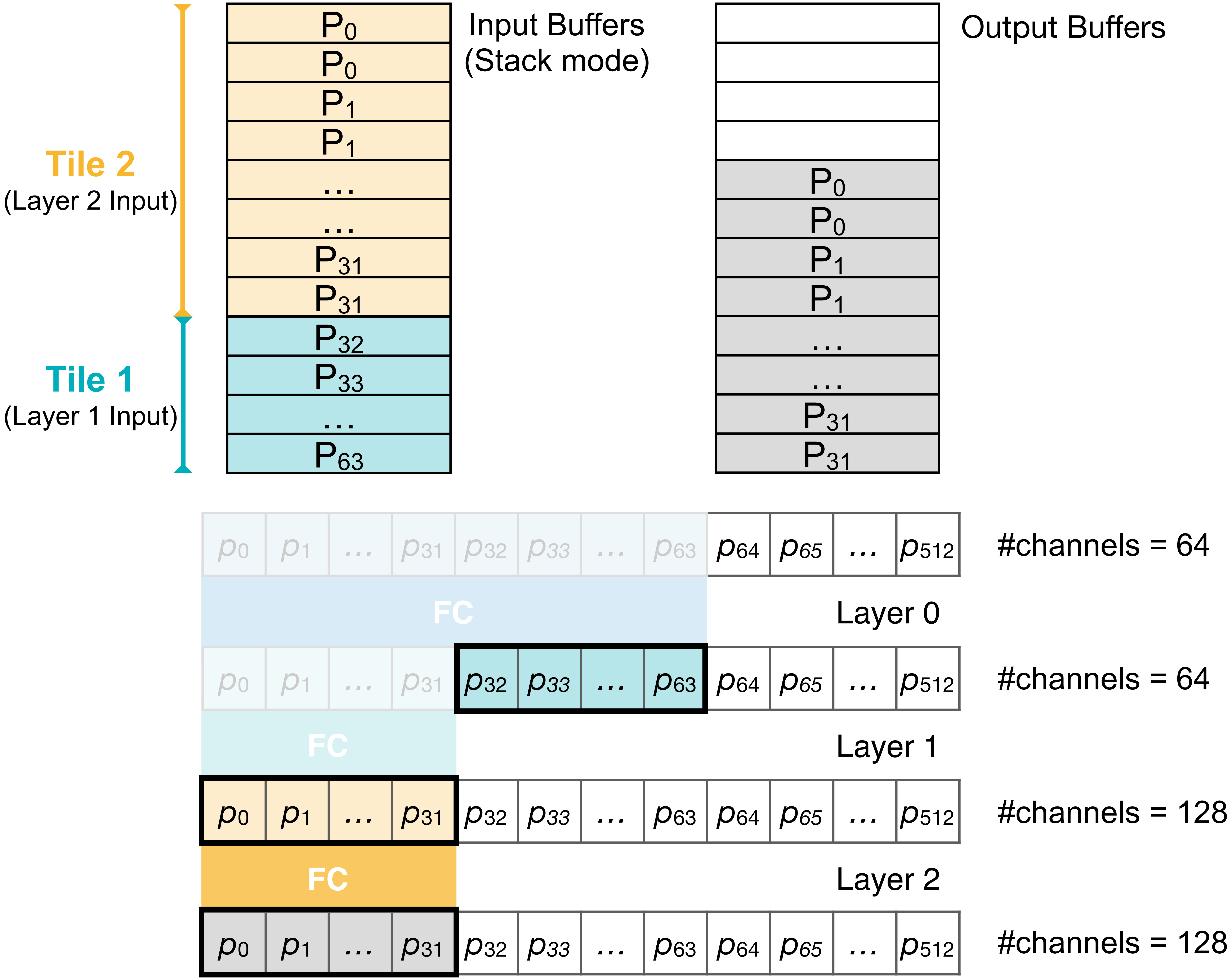
Temporal Layer Fusion on consecutive FCs



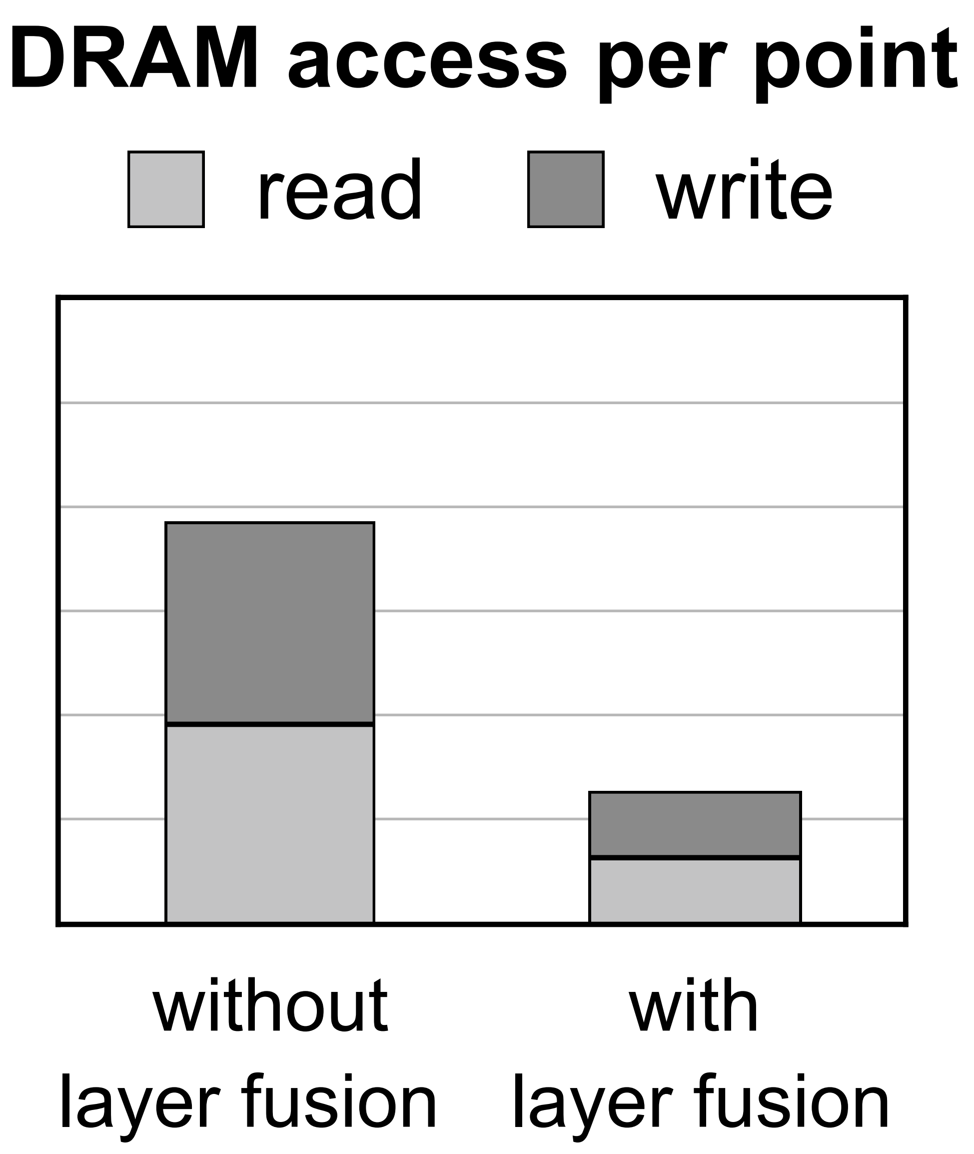


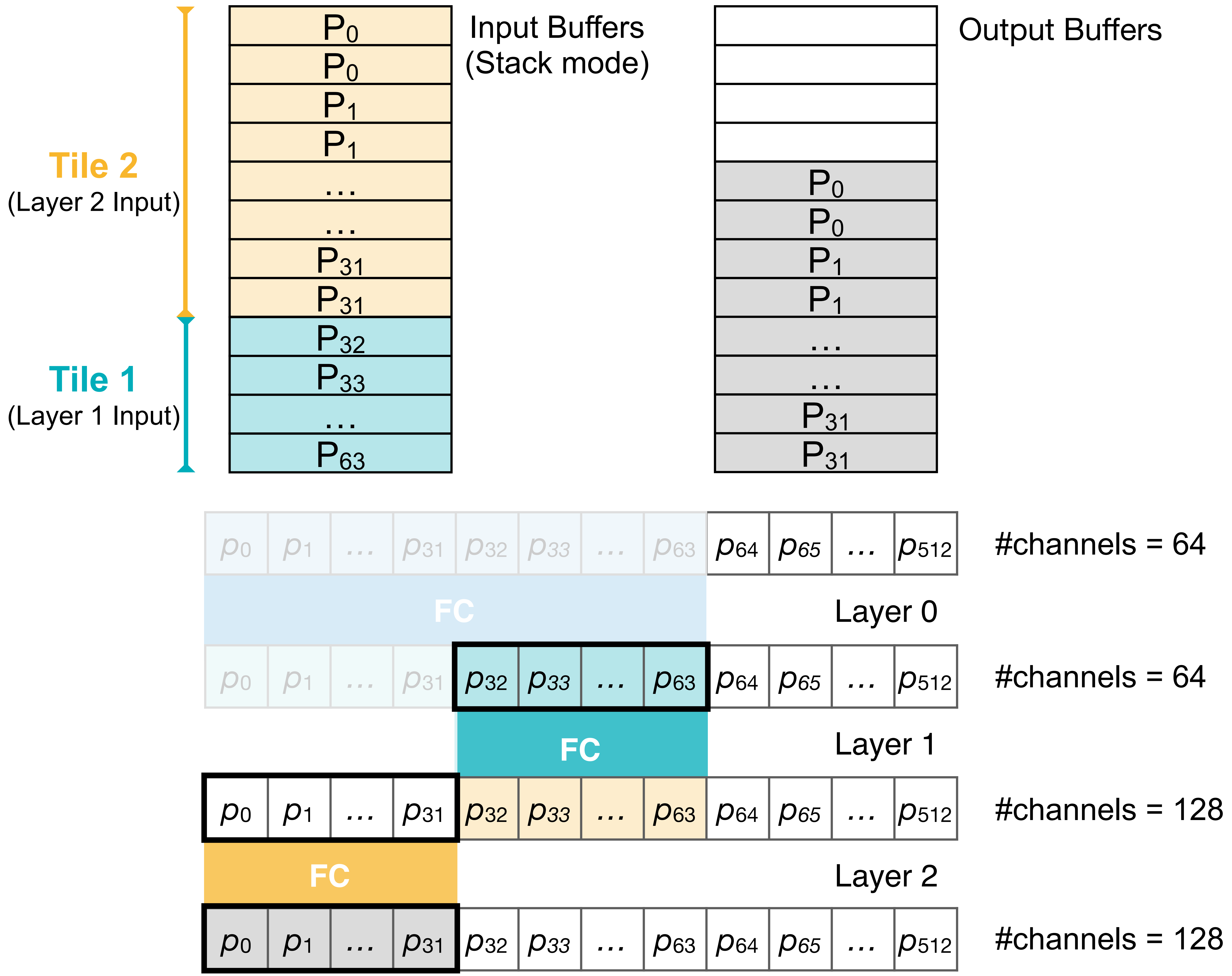
Temporal Layer Fusion on consecutive FCs





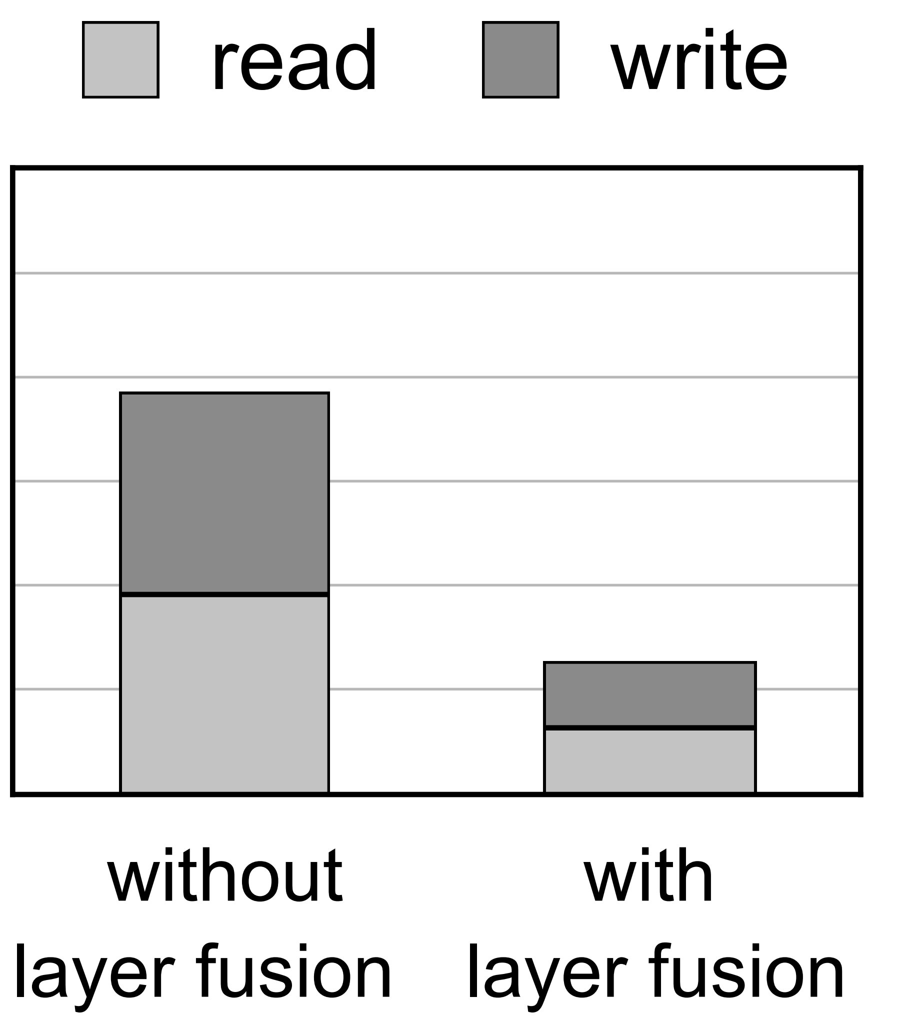
Temporal Layer Fusion on consecutive FCs

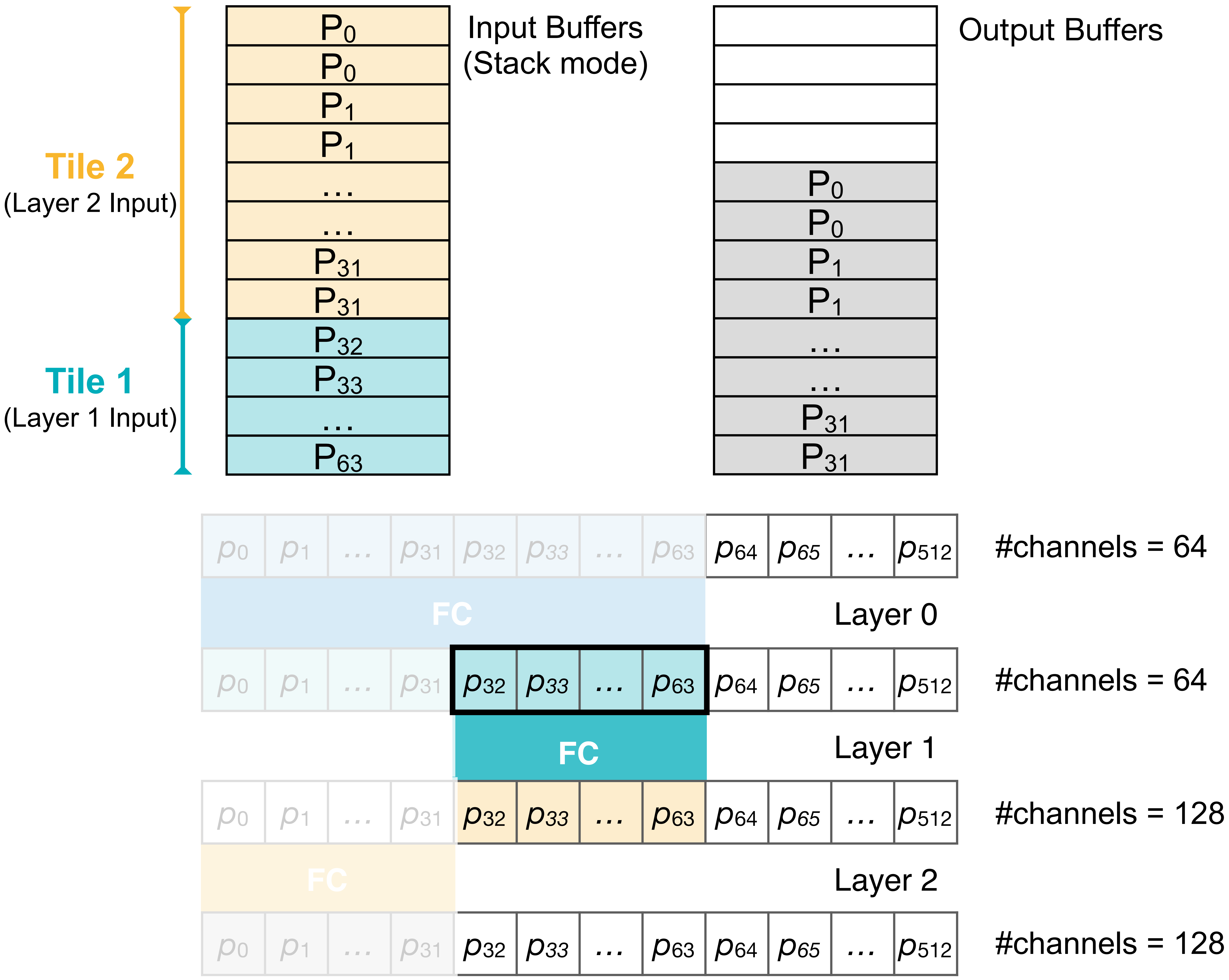




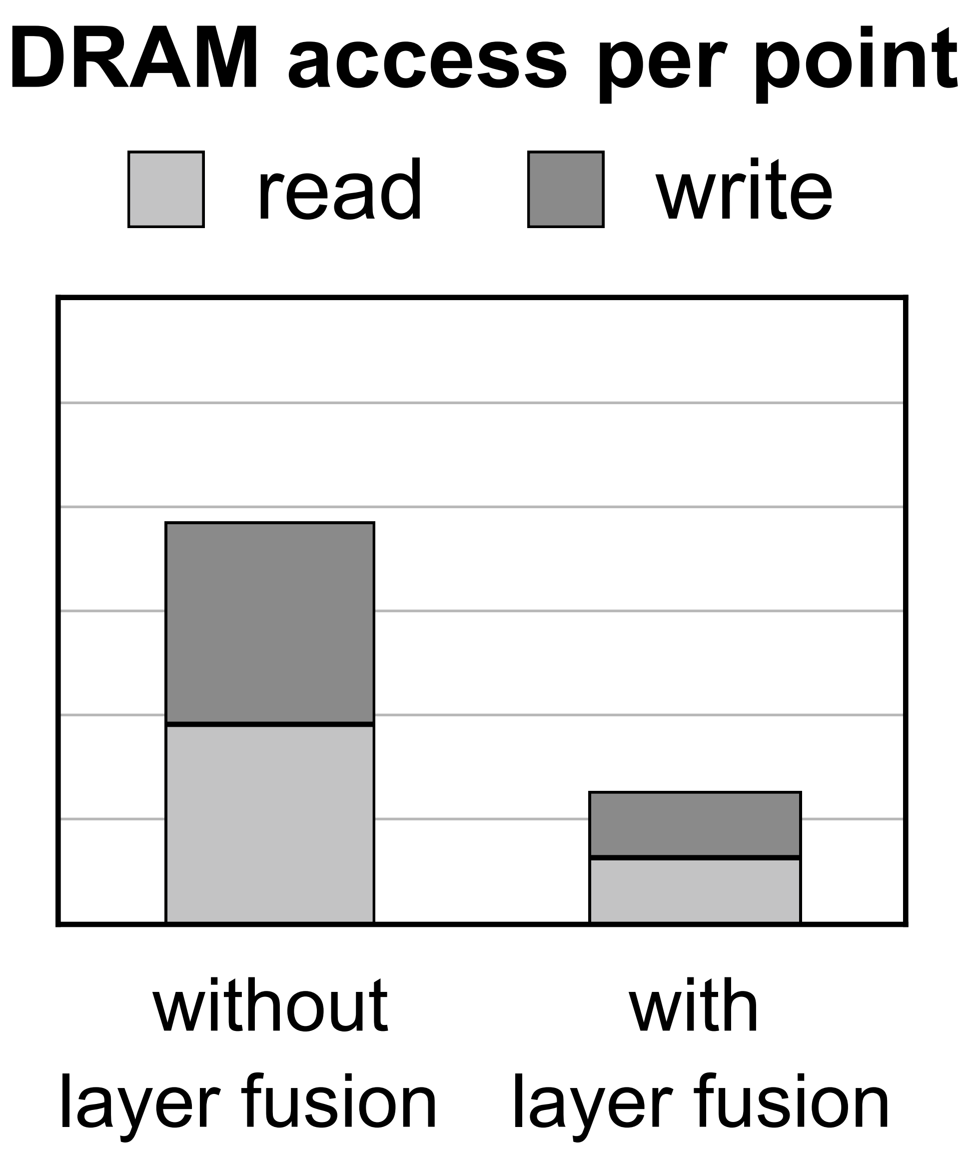
Temporal Layer Fusion on consecutive FCs

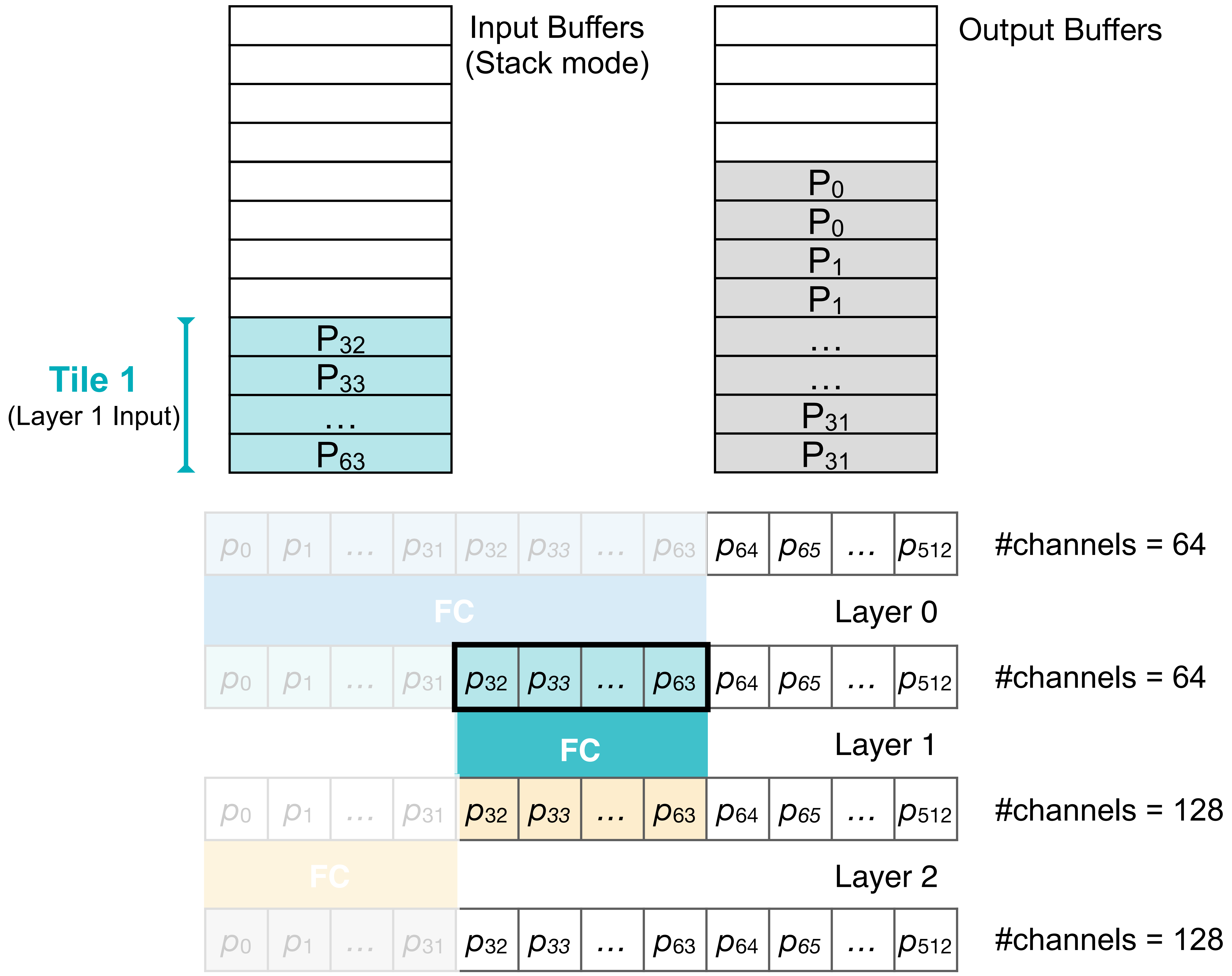
DRAM access per point



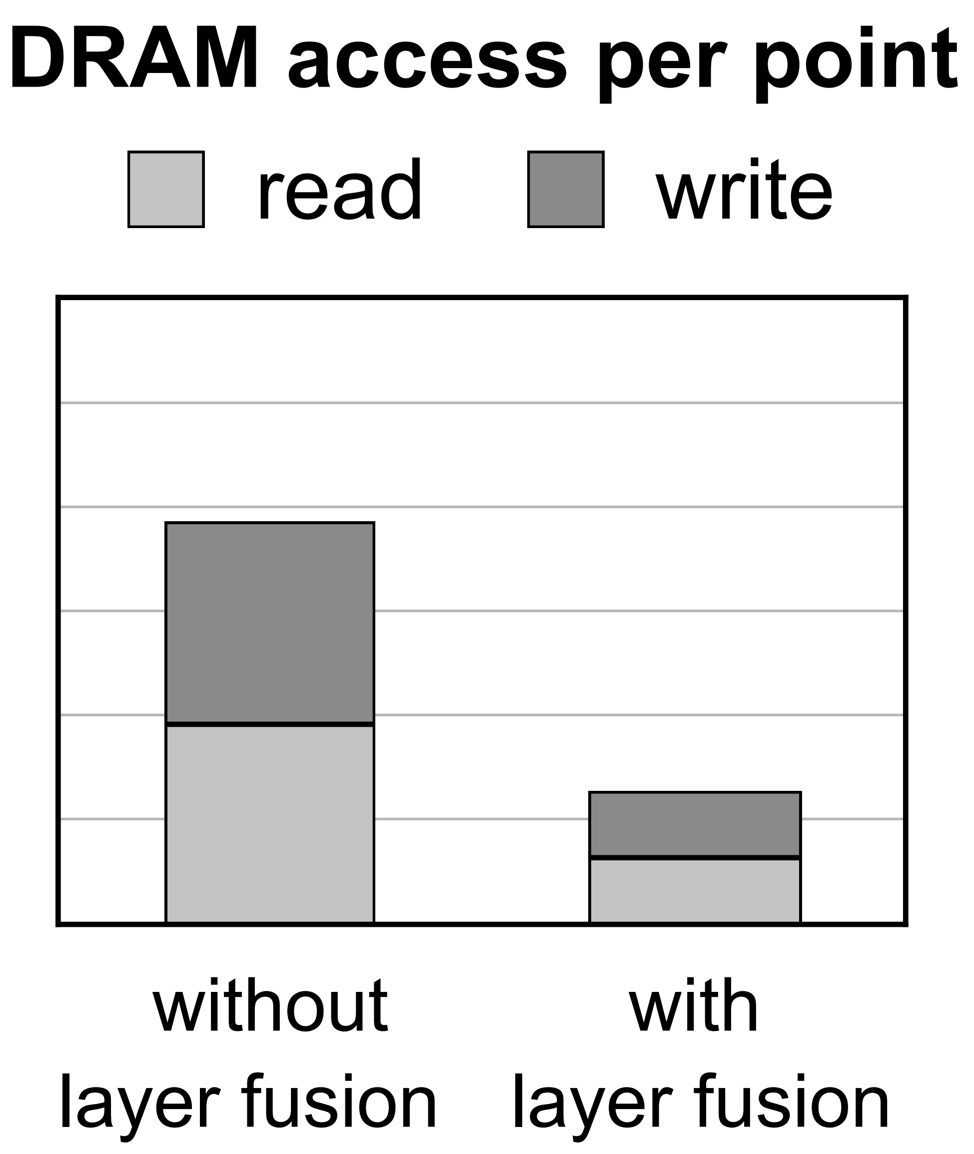


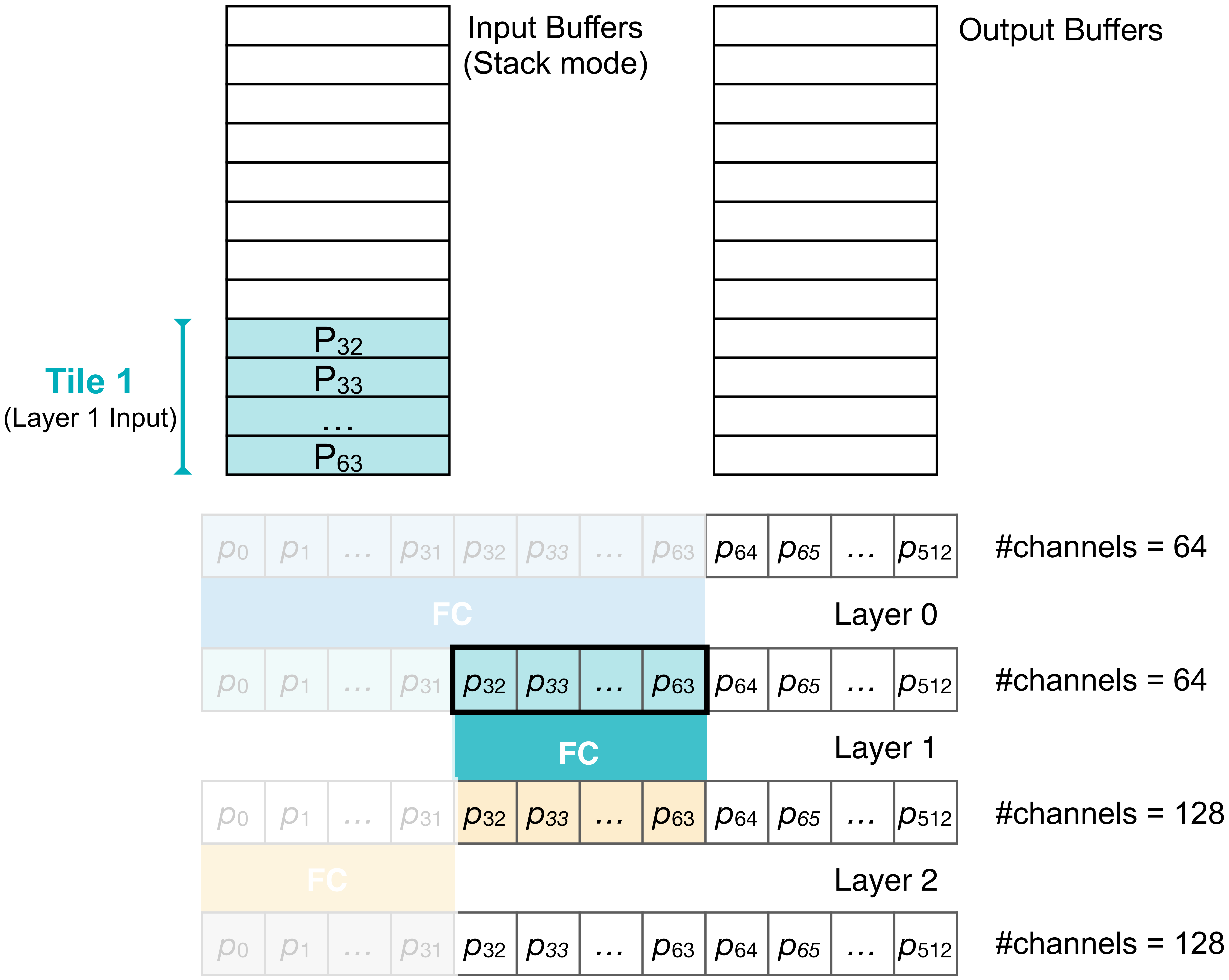
Temporal Layer Fusion on consecutive FCs





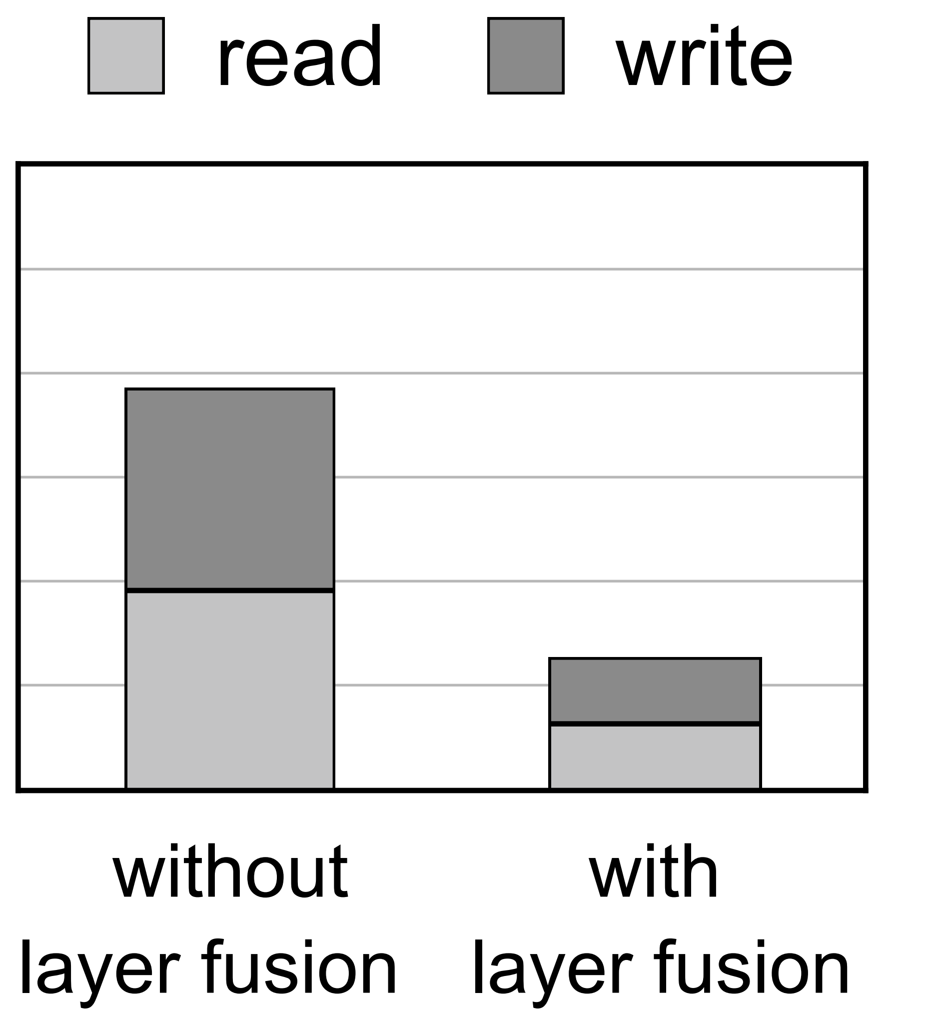
Temporal Layer Fusion on consecutive FCs

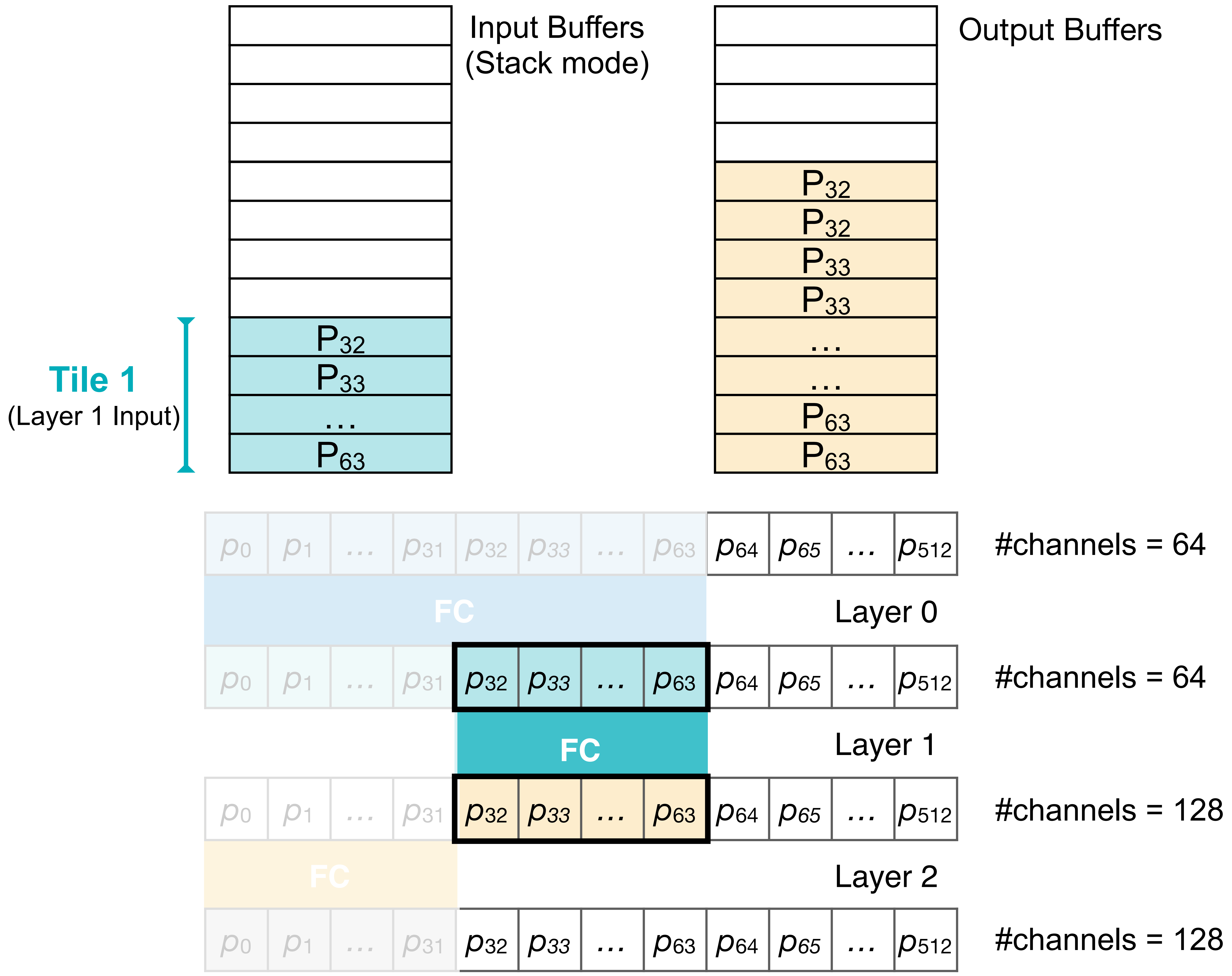




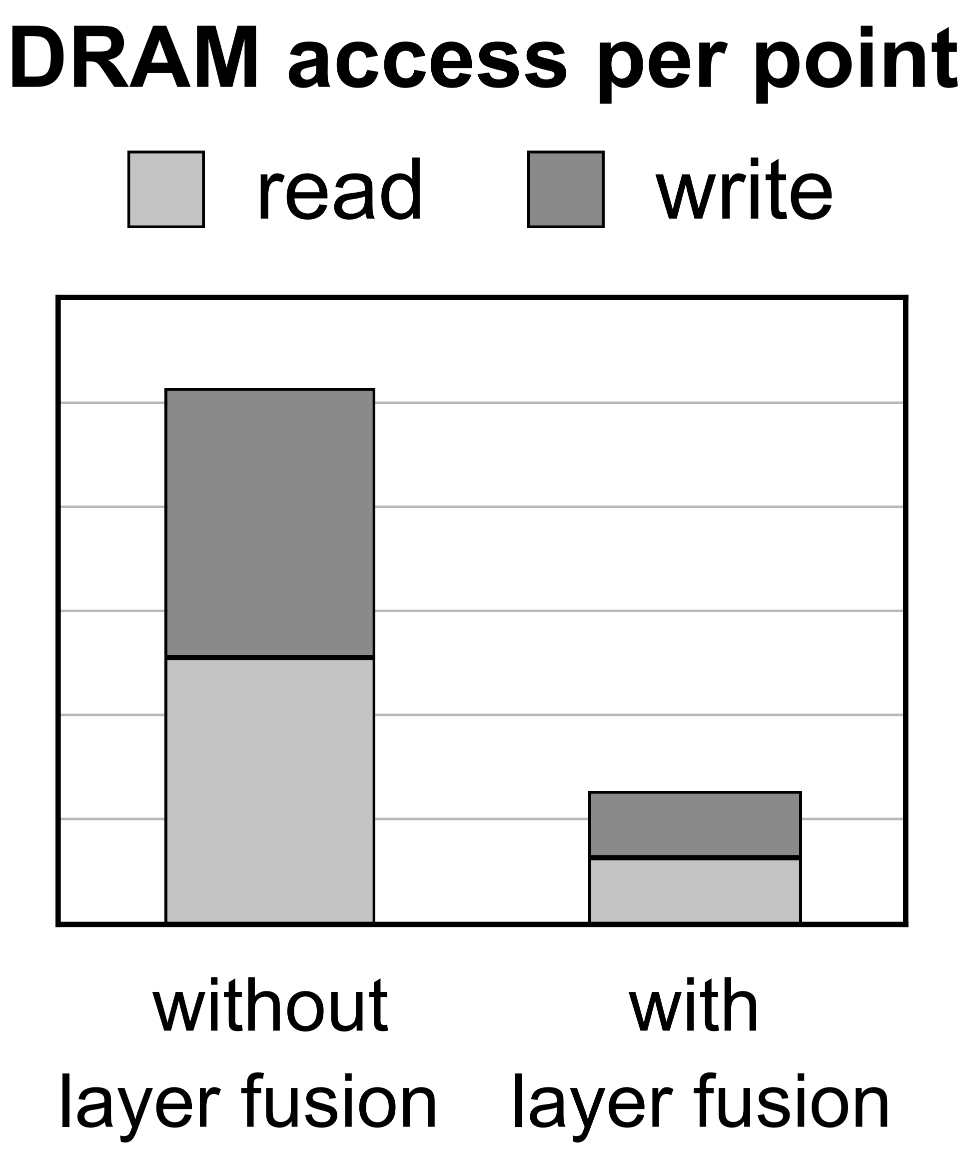
Temporal Layer Fusion on consecutive FCs

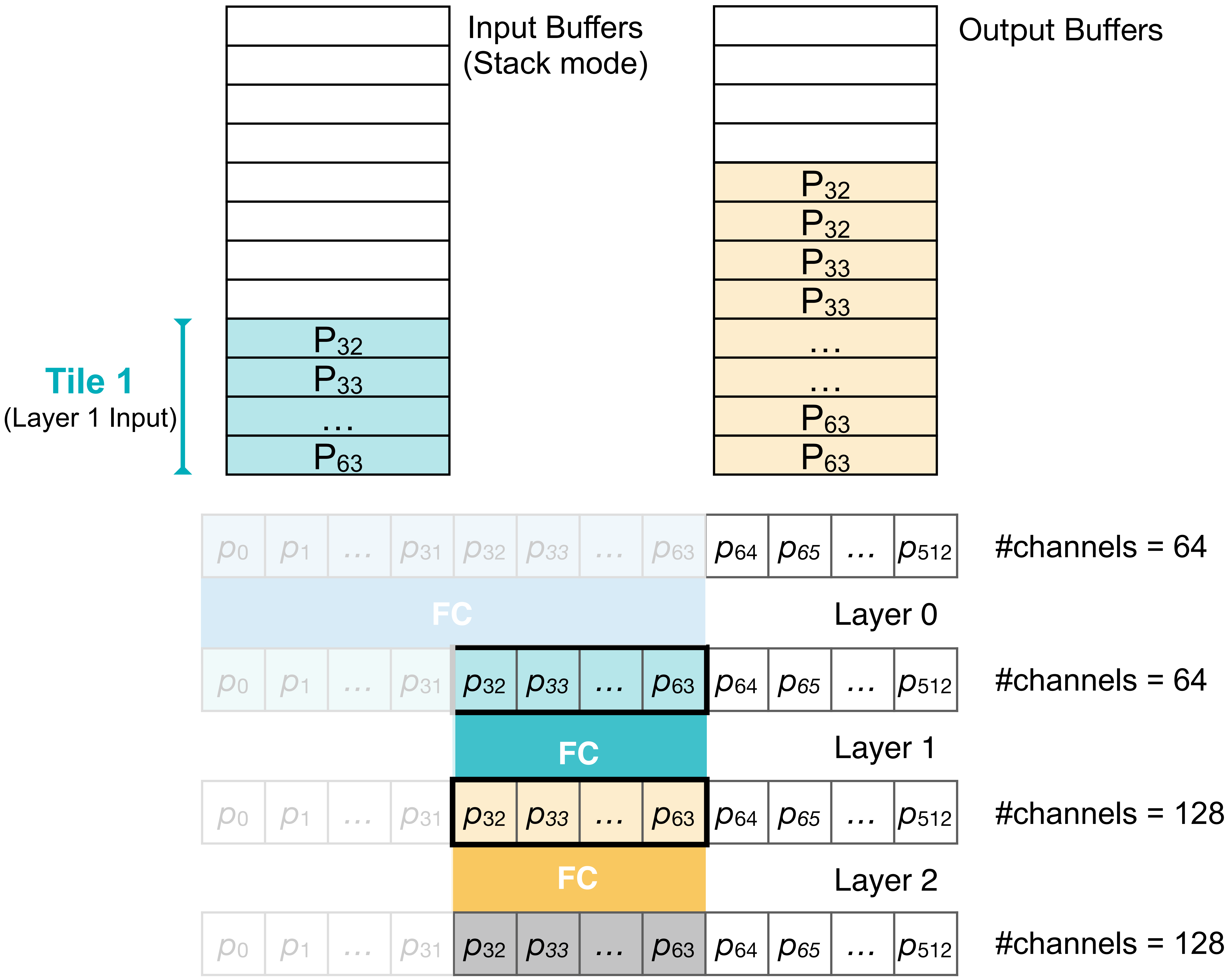
DRAM access per point





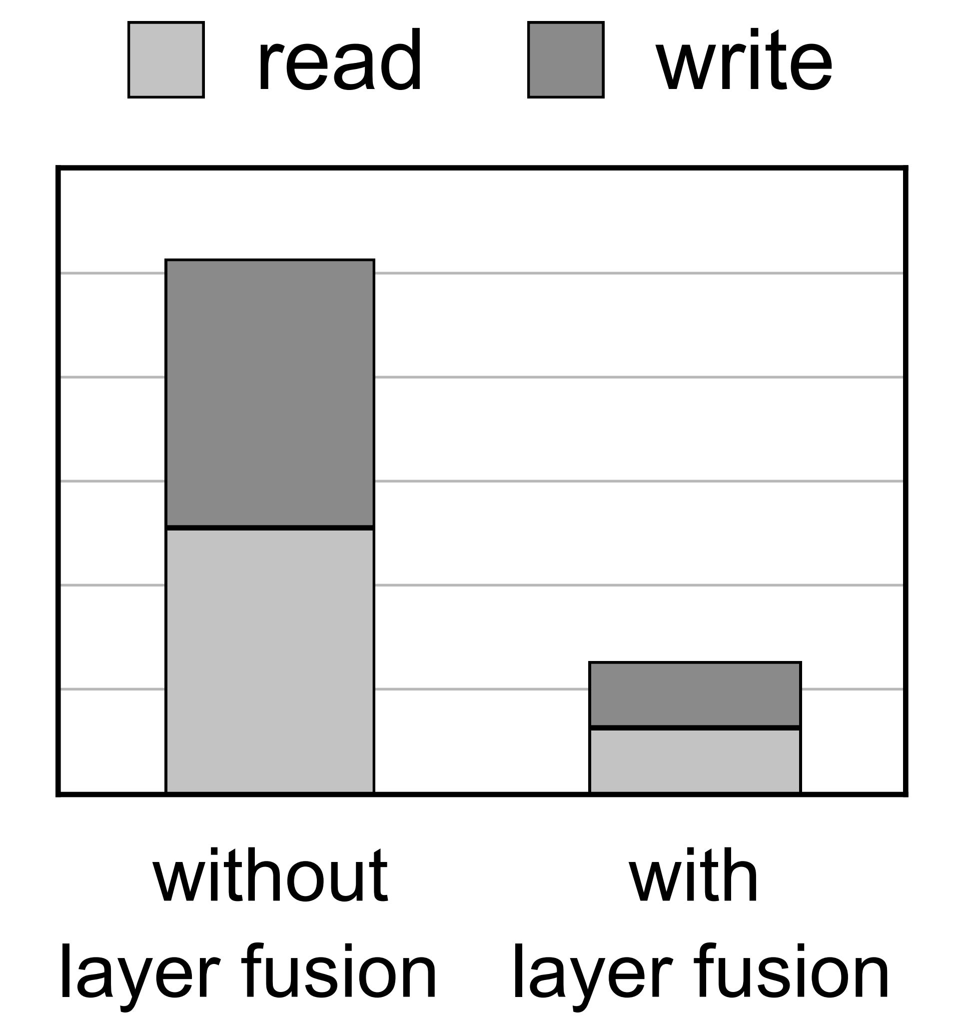
Temporal Layer Fusion on consecutive FCs

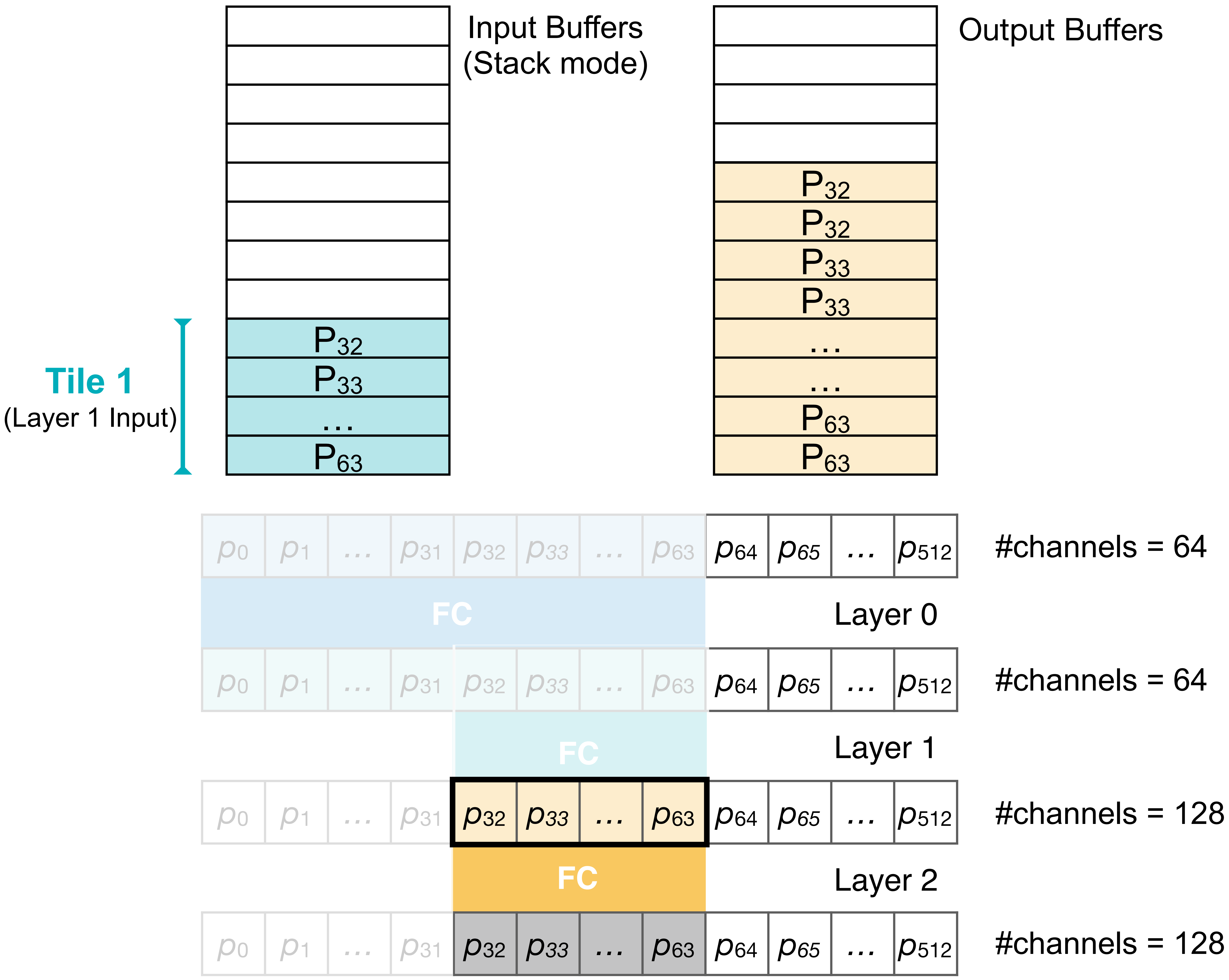




Temporal Layer Fusion on consecutive FCs

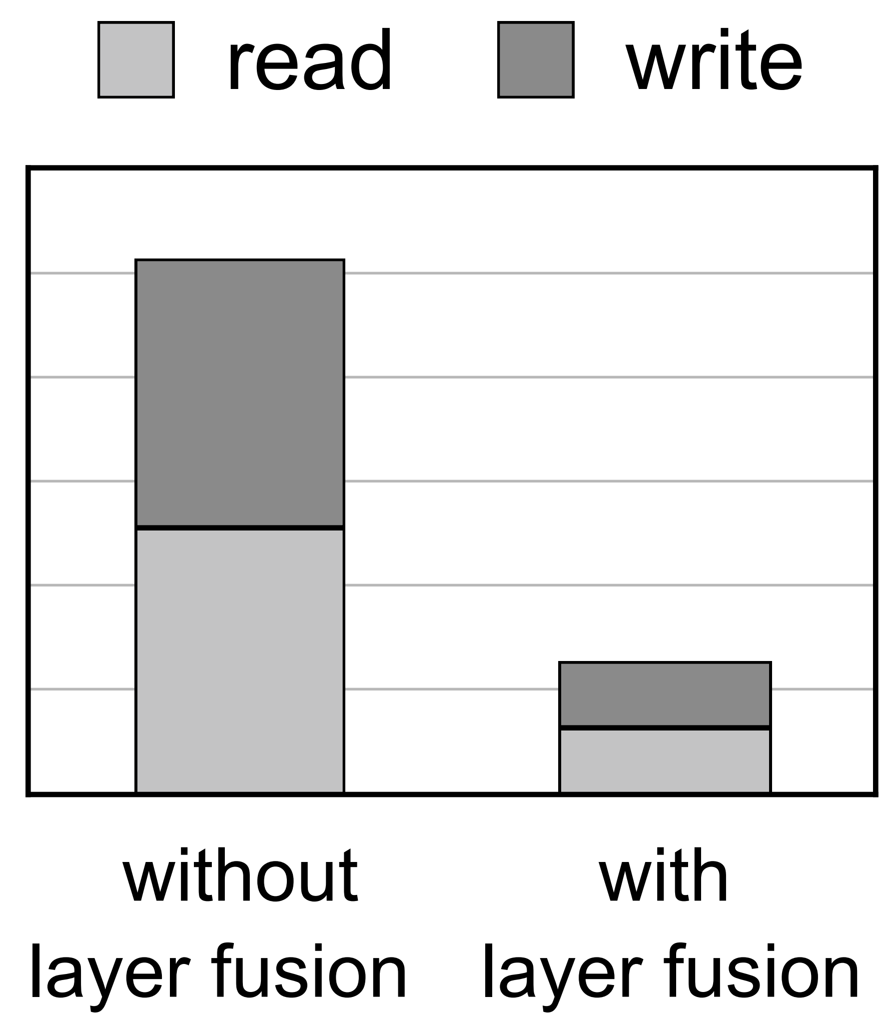
DRAM access per point

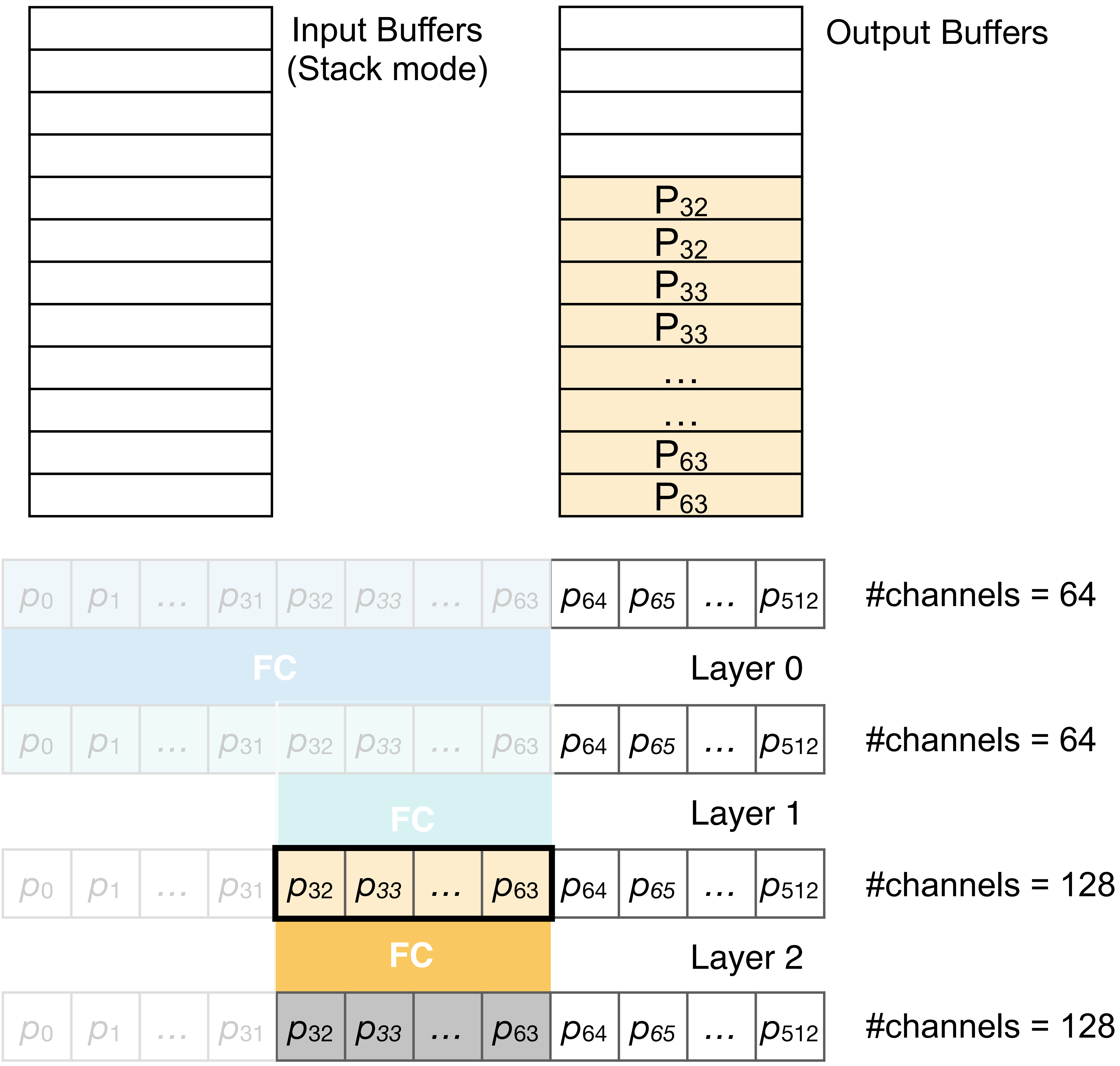




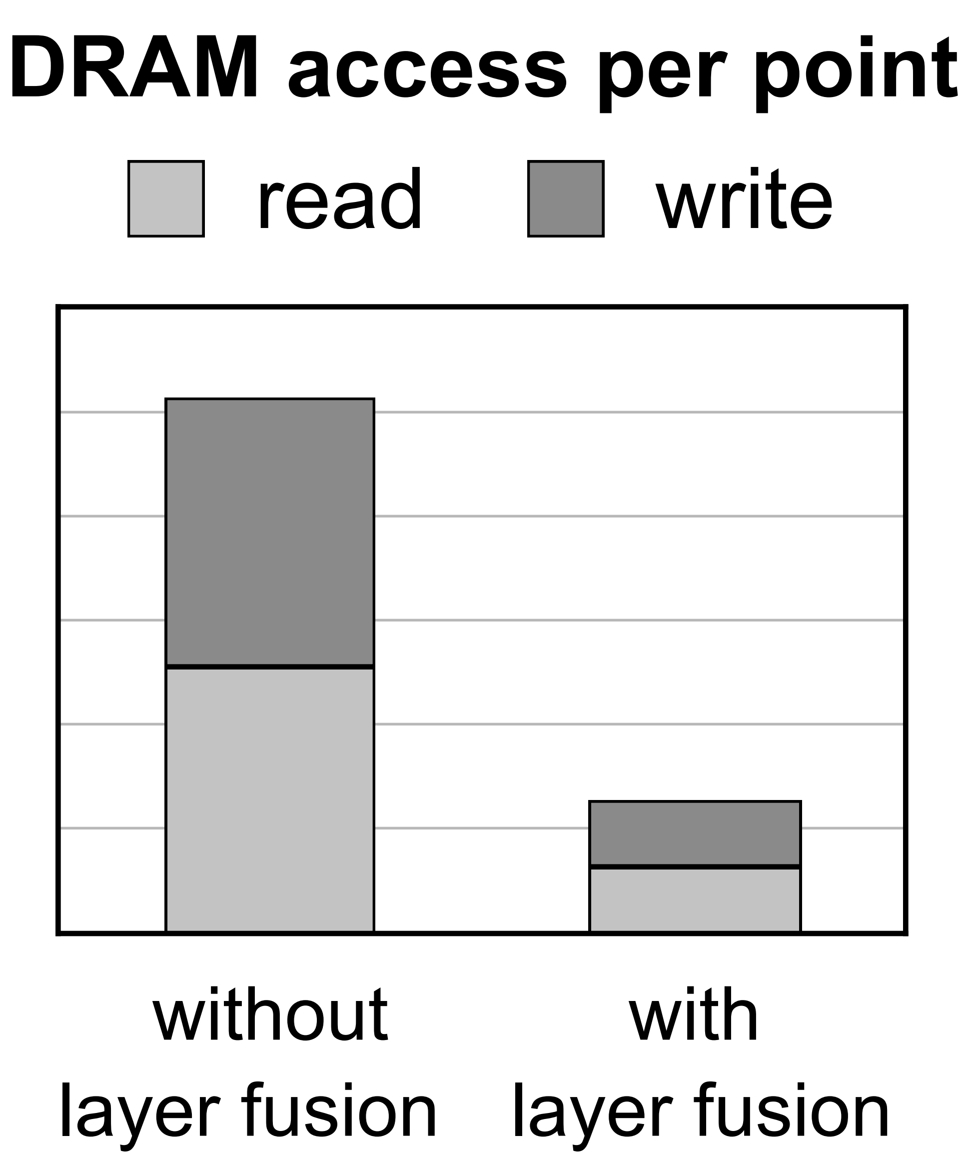
Temporal Layer Fusion on consecutive FCs

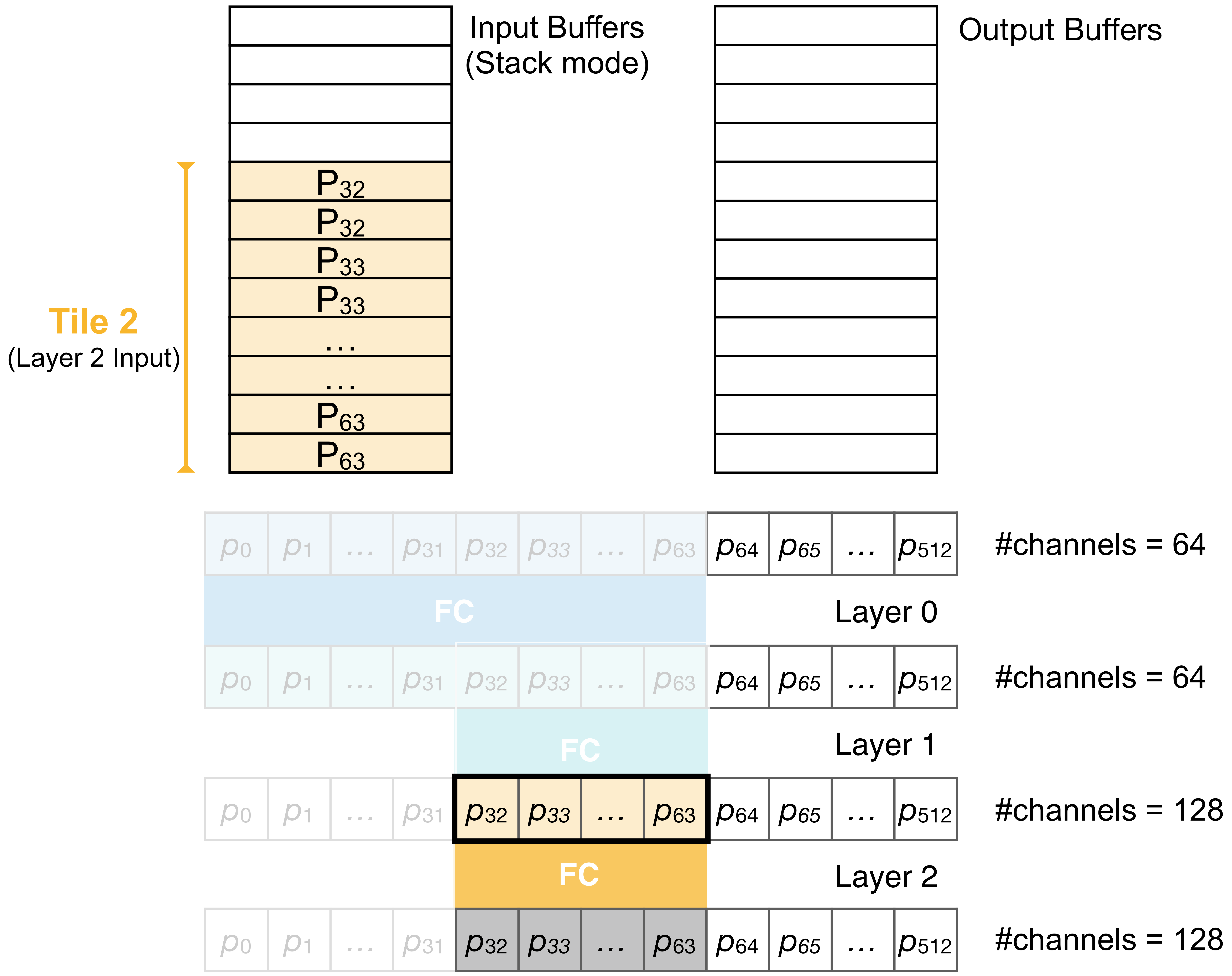
DRAM access per point



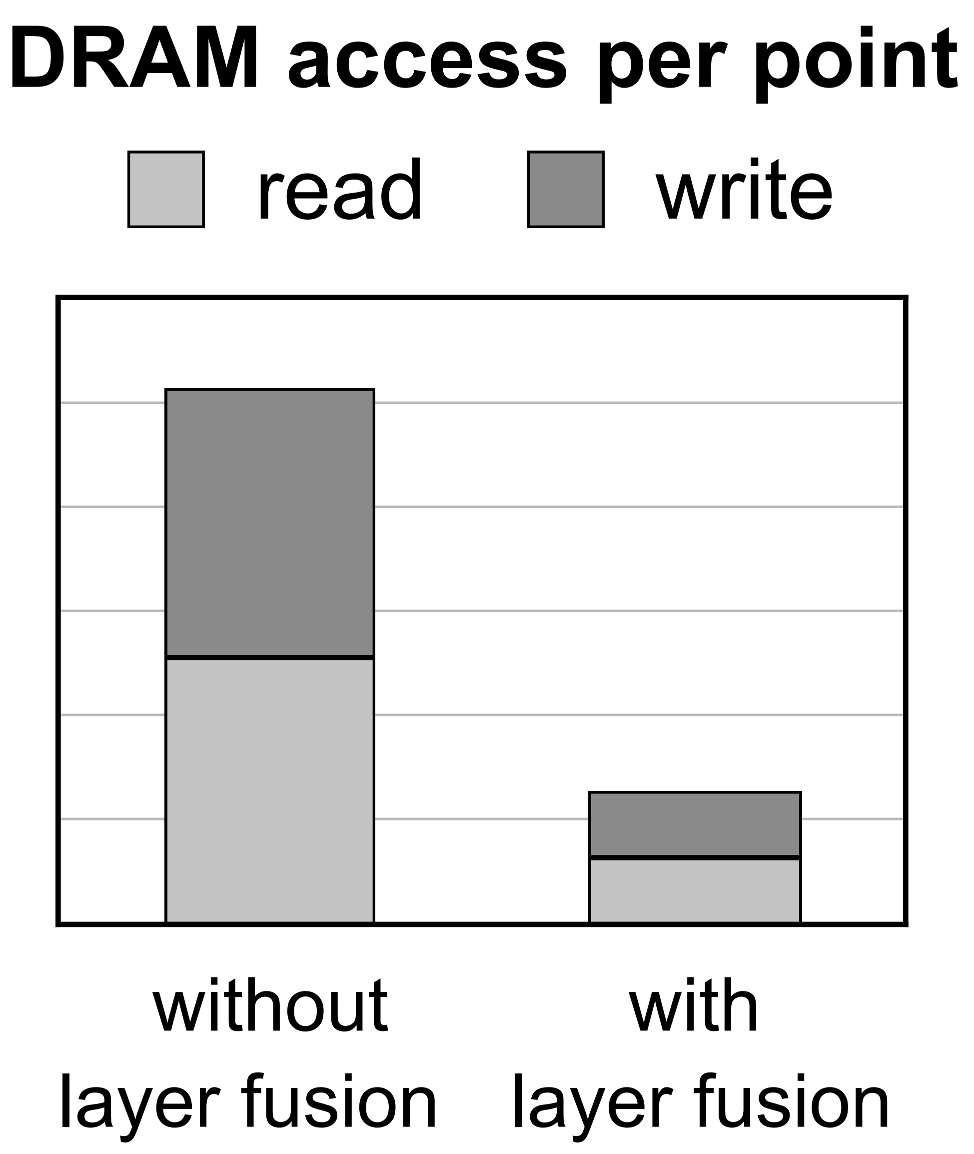


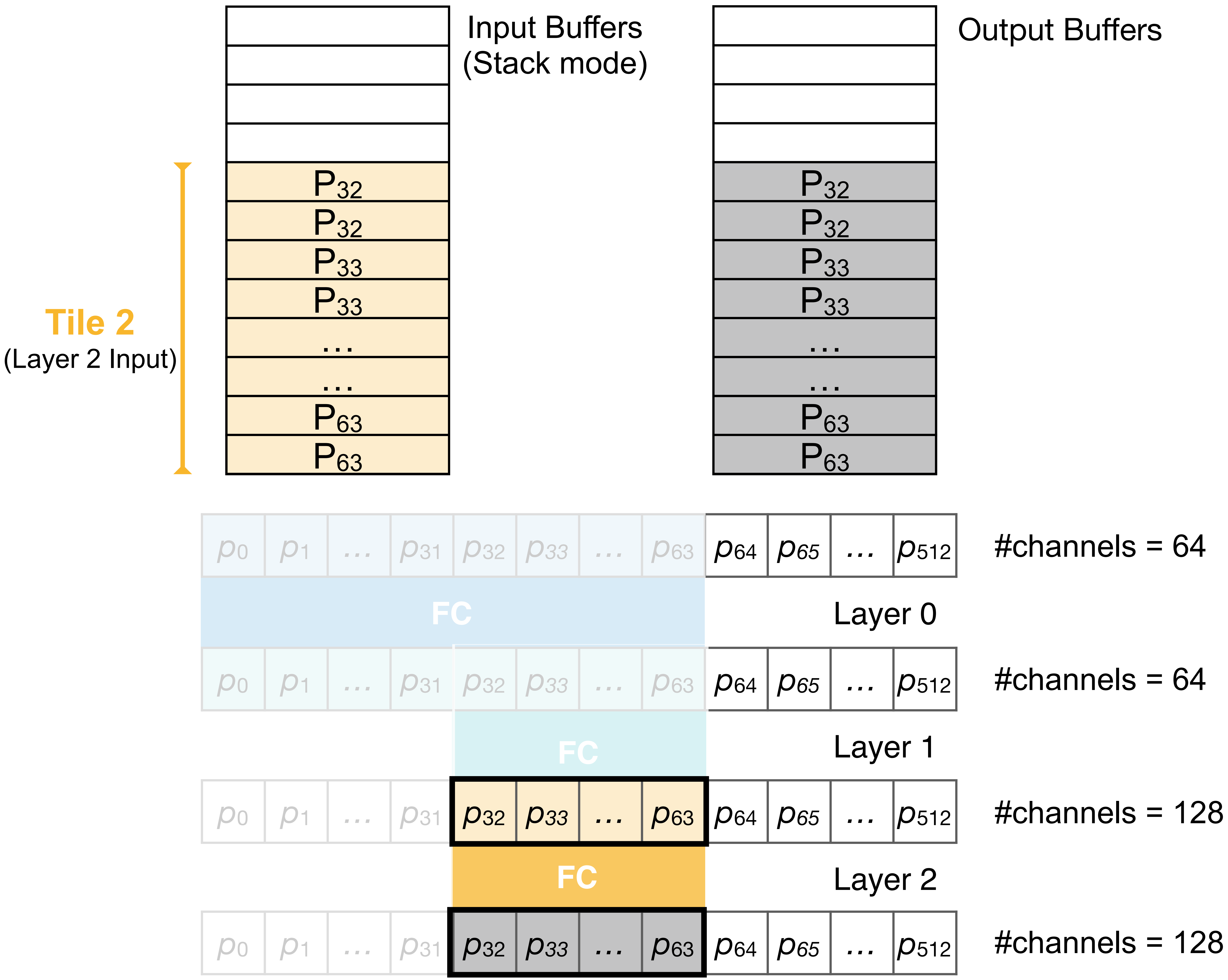
Temporal Layer Fusion on consecutive FCs



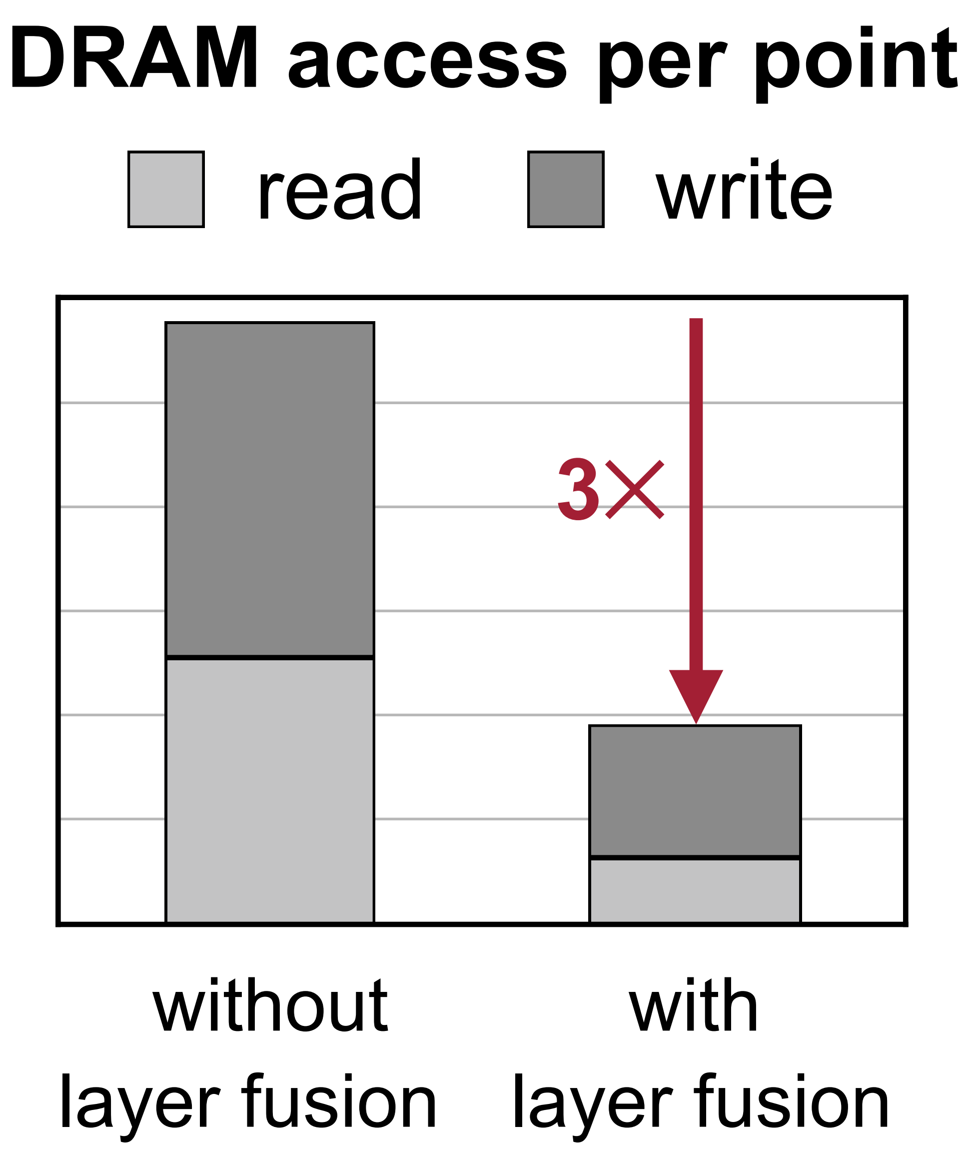


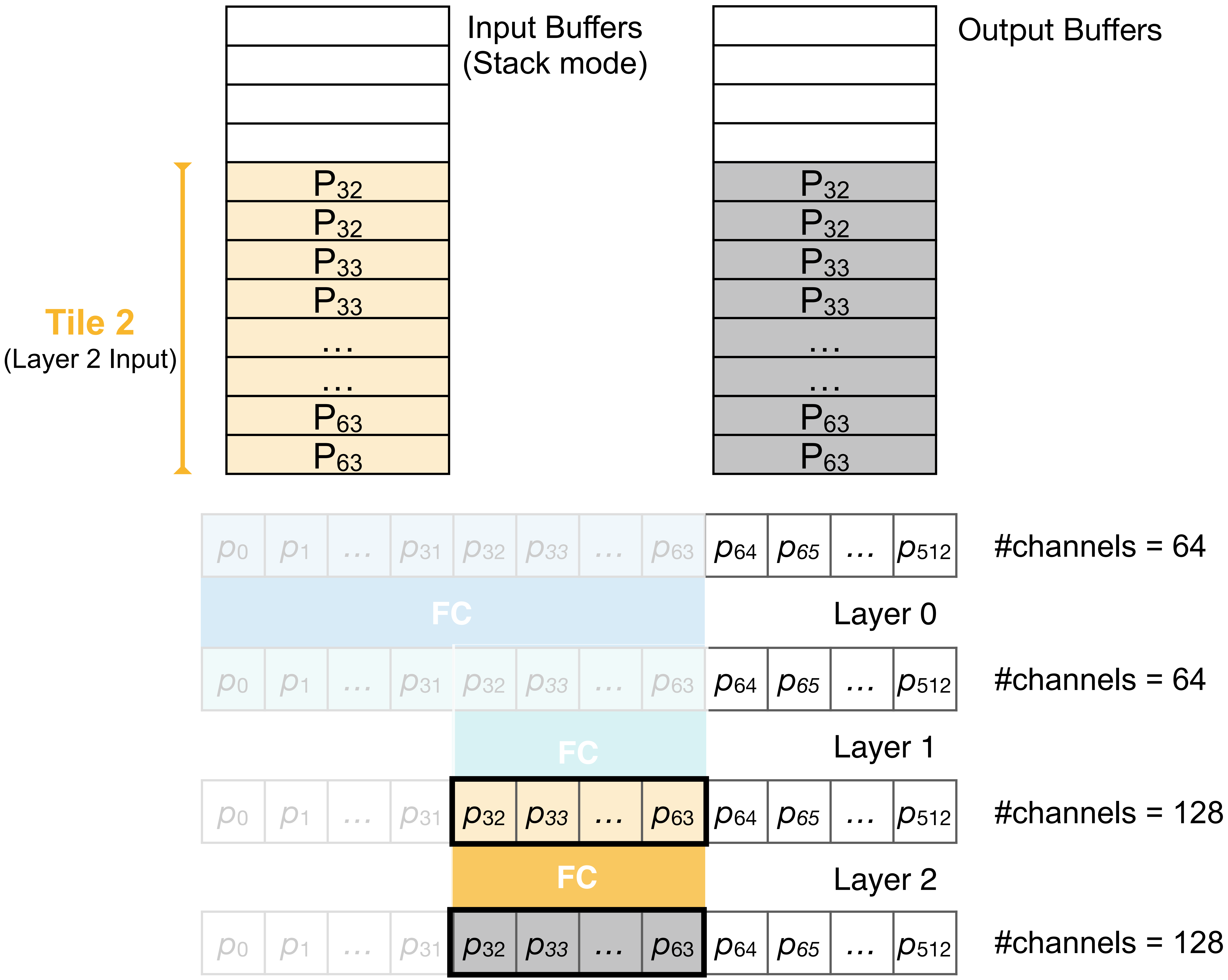
Temporal Layer Fusion on consecutive FCs





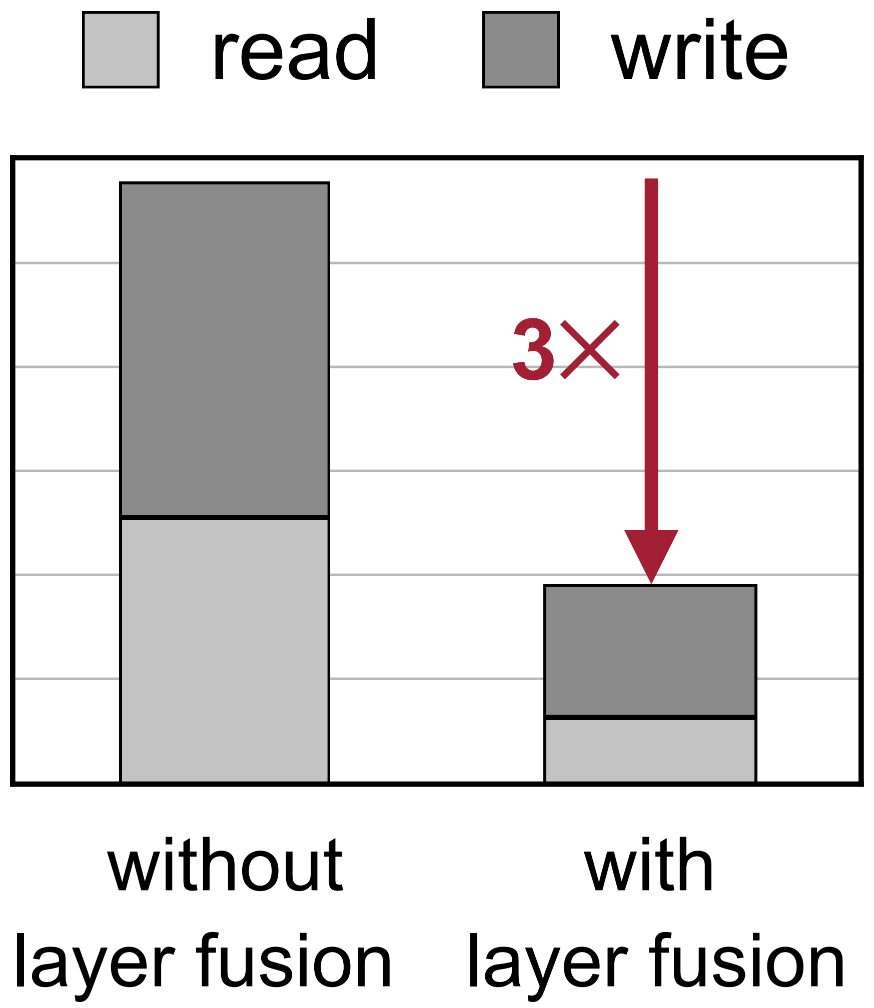
Temporal Layer Fusion on consecutive FCs

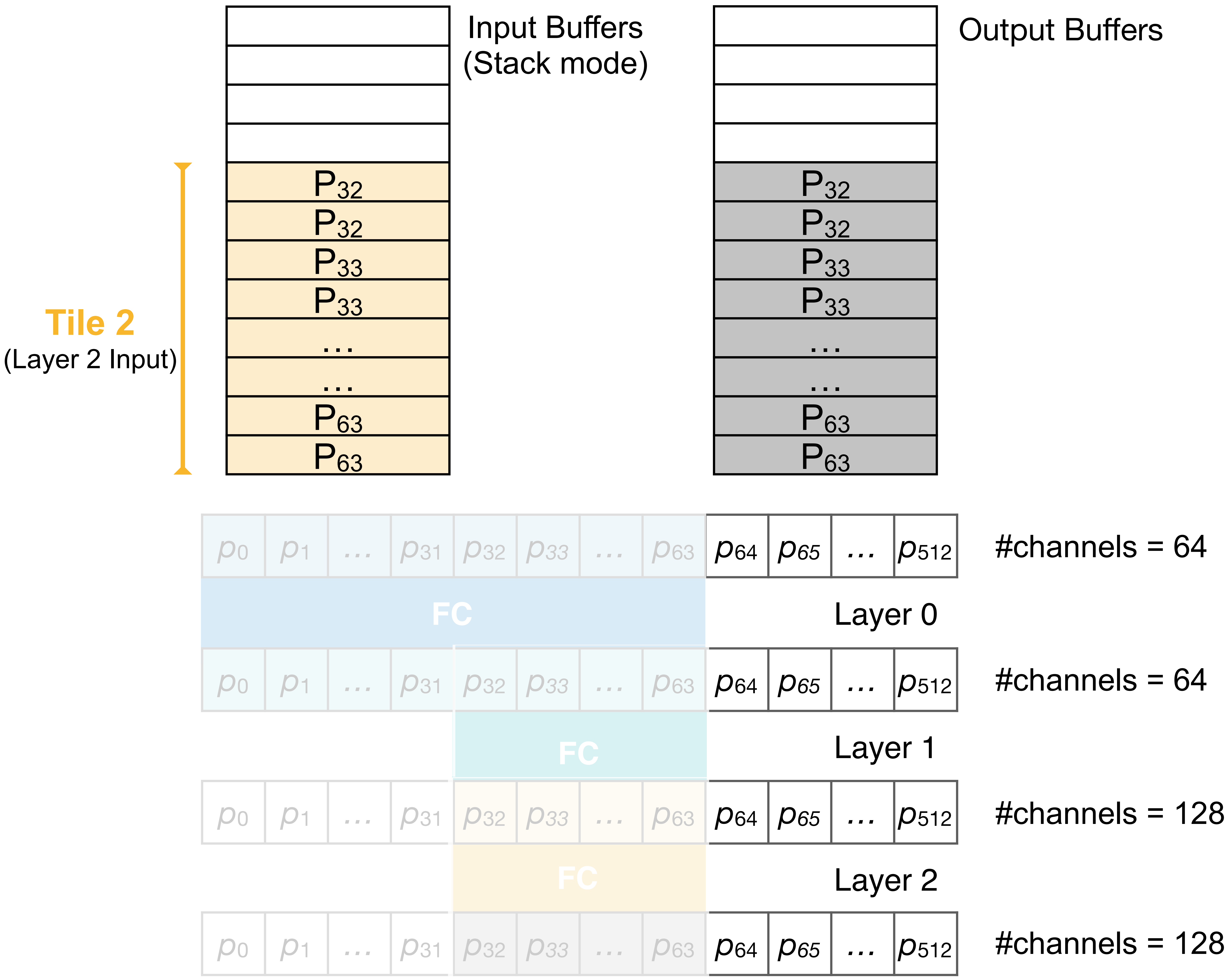




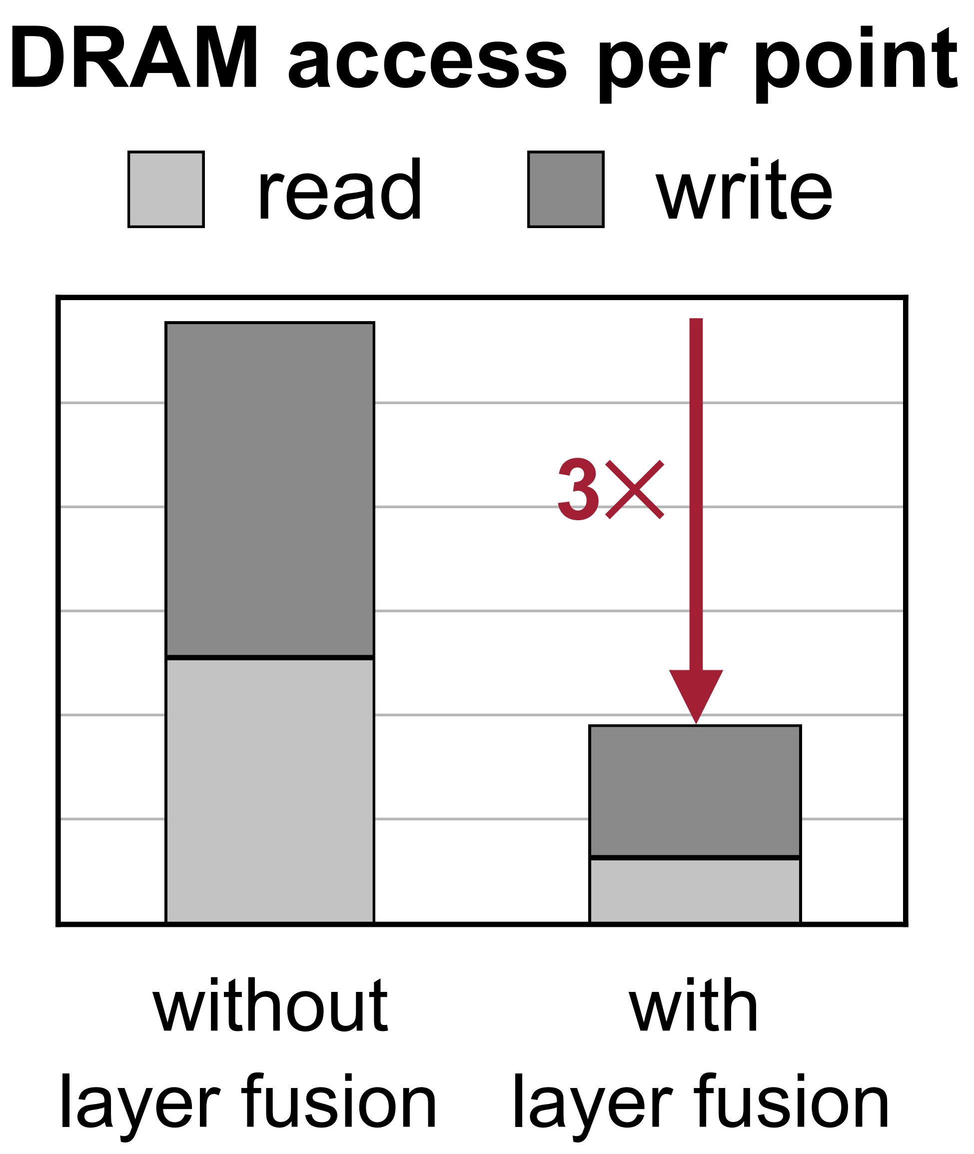
Temporal Layer Fusion on consecutive FCs

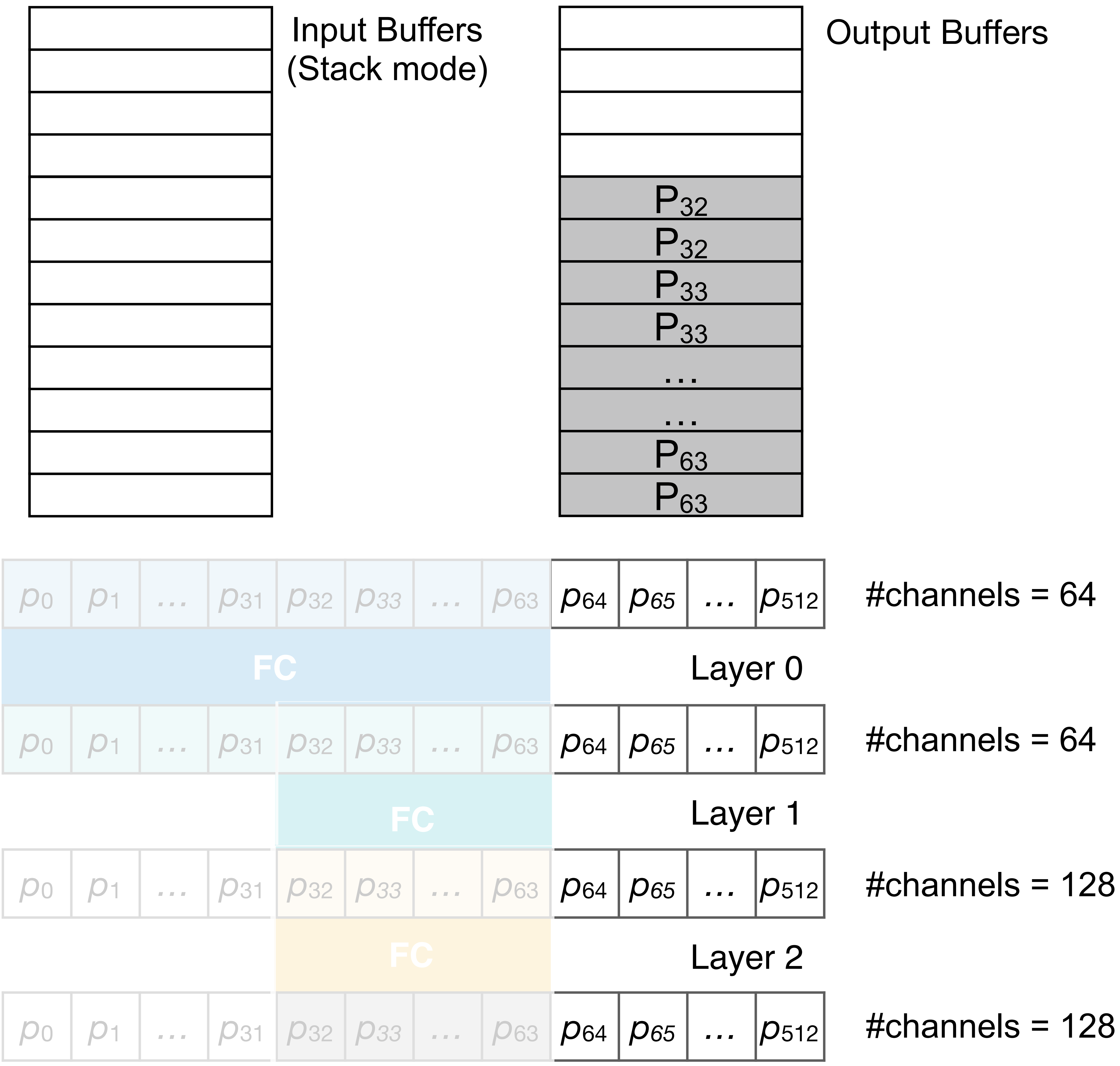
DRAM access per point



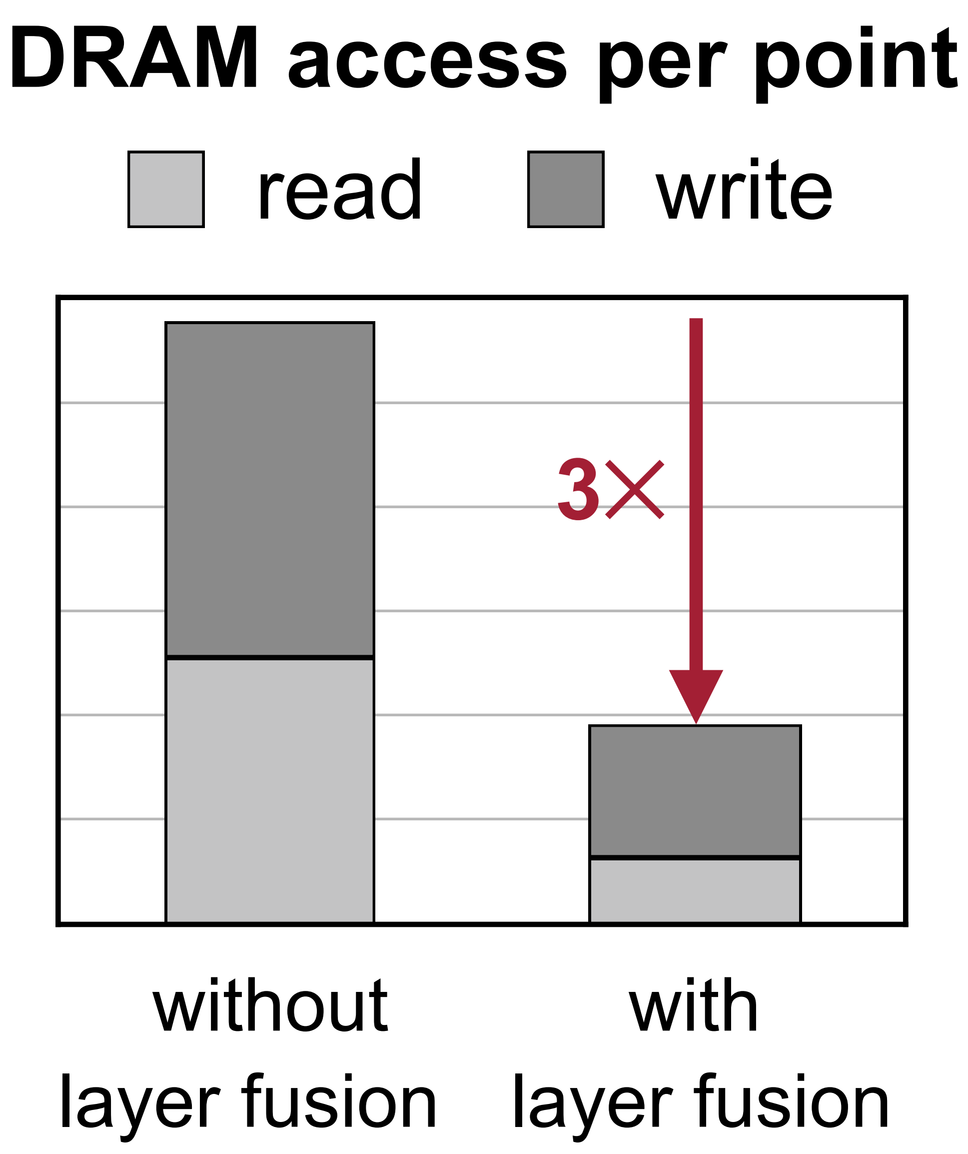


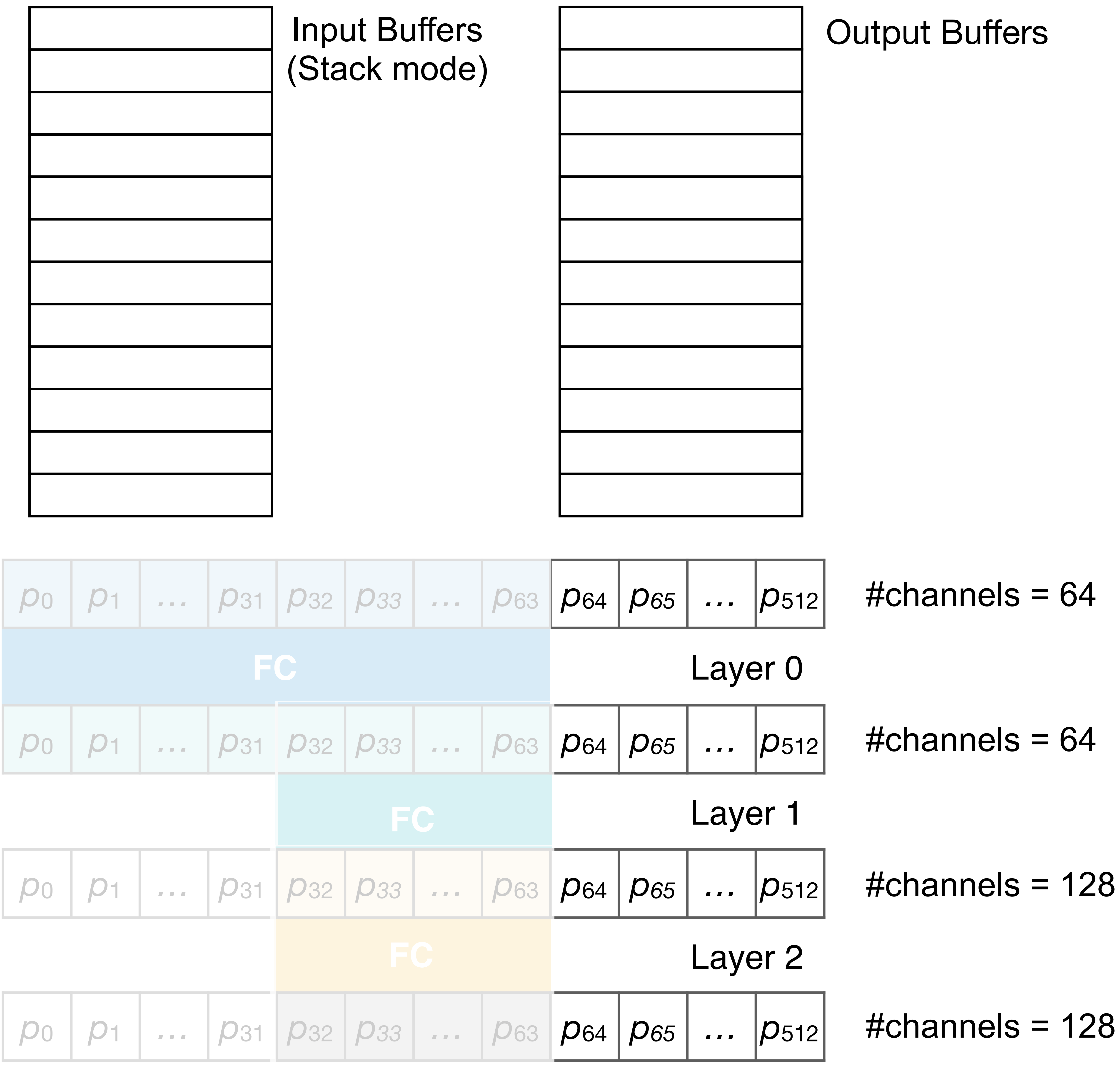
Temporal Layer Fusion on consecutive FCs



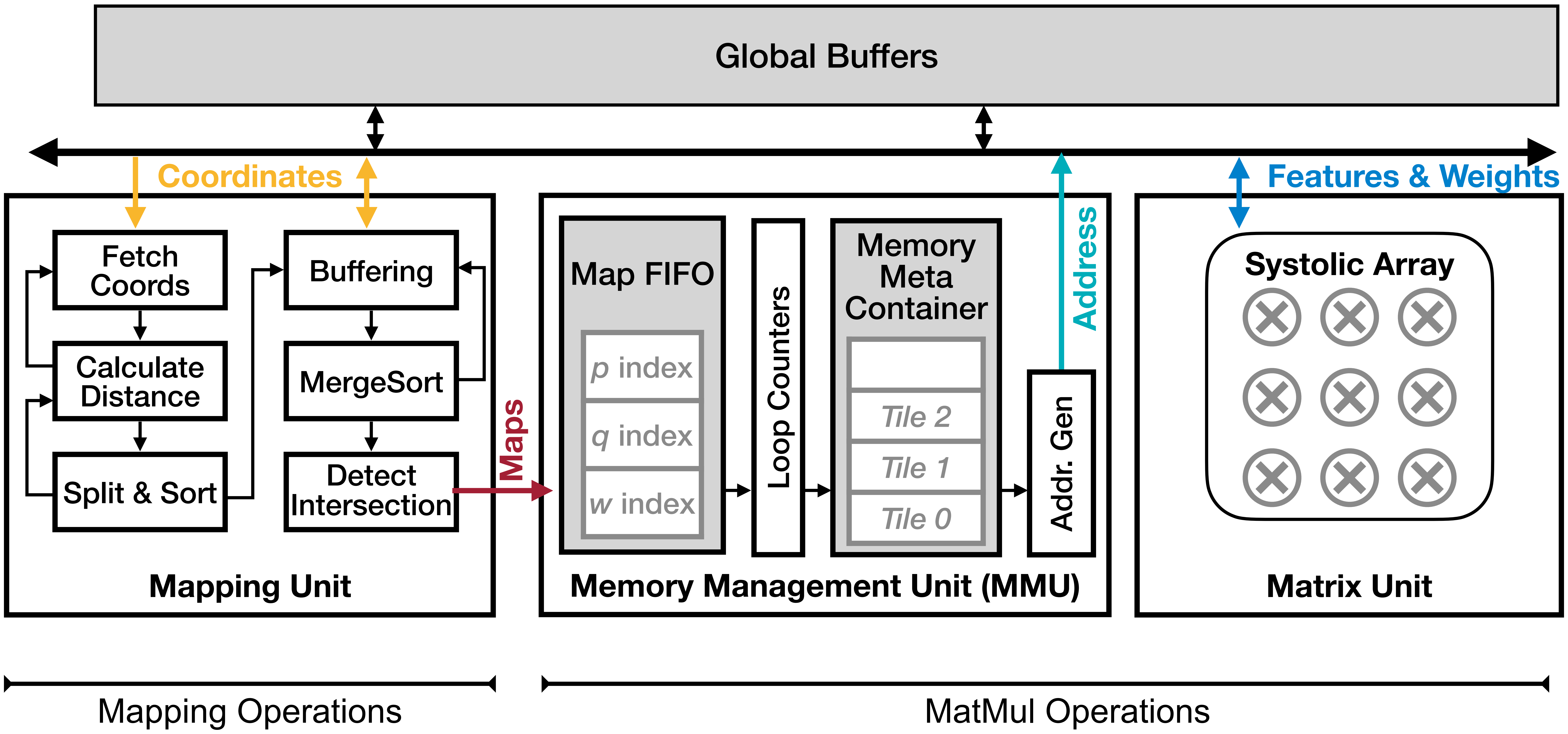


Temporal Layer Fusion on consecutive FCs





PointAcc: Efficient Point Cloud Accelerator



Evaluation Setup

Evaluation benchmarks

- 4 different applications
- 5 different datasets
- 8 different point cloud models

Application	Dataset	Scene	Model
Classification	ModelNet40	Object	PointNet
			PointNet++ (c)
Part Segementation	ShapeNet		PointNet++ (ps)
			DGCNN
Detection	KITTI	Outdoor	MinkNet(o)
Semantic Segmentation	S3DIS	Indoor	F-PointNet++
			PointNet++(s)
	SemanticKITTI	Outdoor	MinkNet(i)

Evaluation Setup

Evaluation benchmarks

- 4 different applications: classification, part segmentation, detection, semantic segmentation
- 5 different datasets, ranging from single object to indoor scenes to outdoor scenes
- 8 different point cloud models, including classical and the state-of-the-art ones

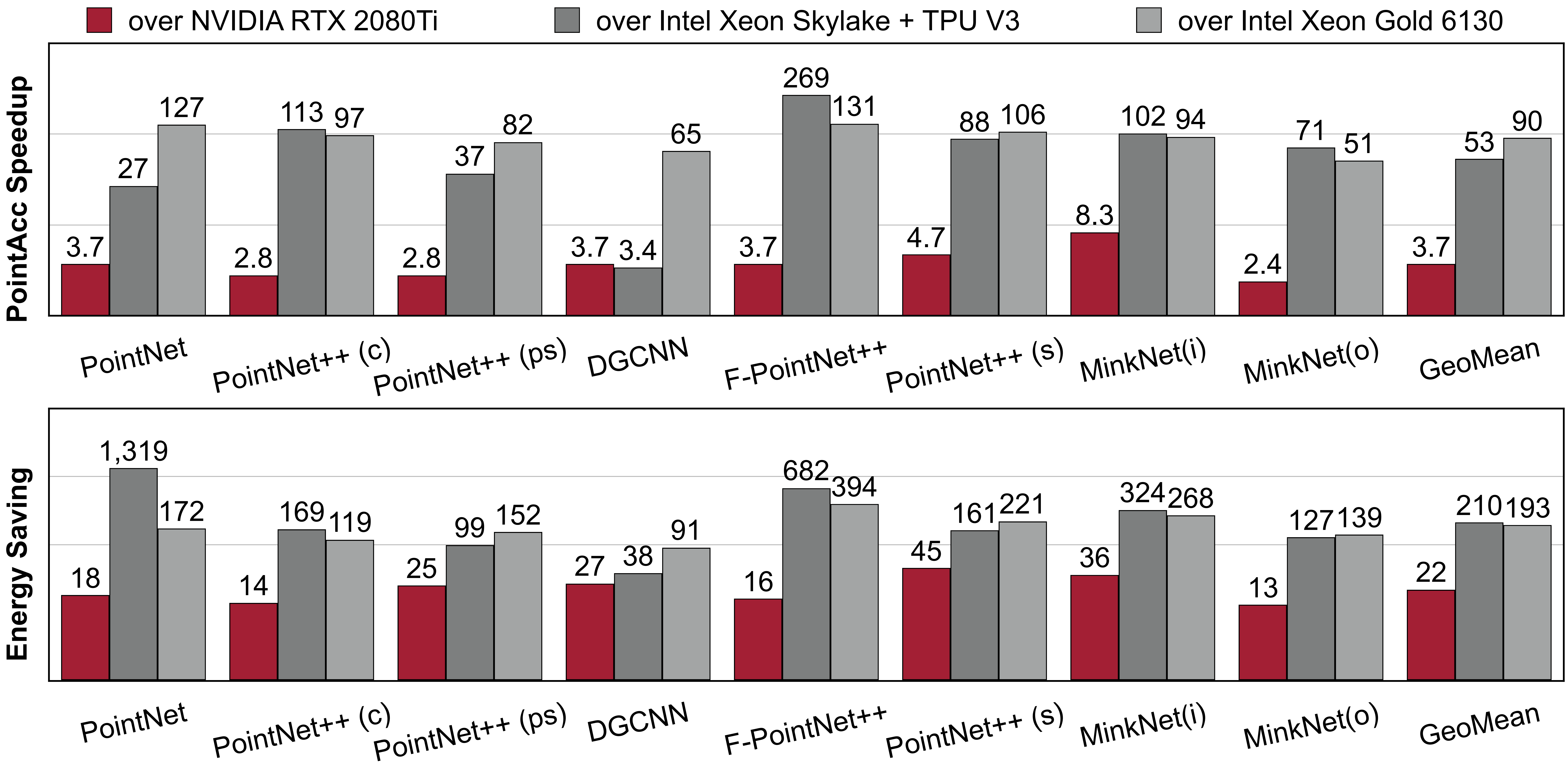
Hardware Baselines

- server-level products: Intel Xeon CPU, RTX 2080Ti GPU, TPU v3
- edge devices: Jetson Xavier NX, Jetson Nano, Raspberry Pi
- specialized point cloud NN ASIC: Mesorasi

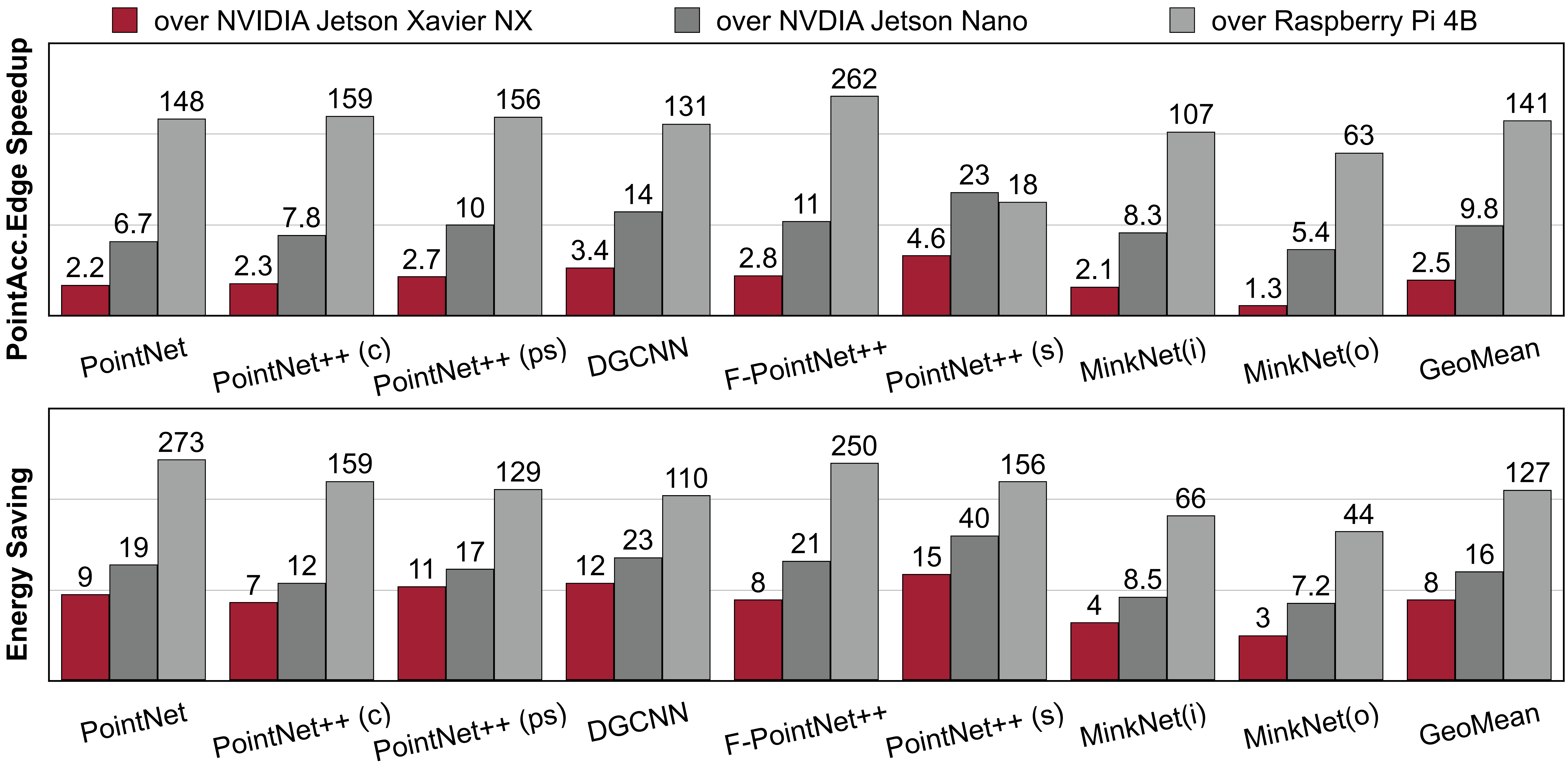
Variants

- PointAcc: 64 × 64 systolic array with 776 KB on-chip memory
- PointAcc.Edge: 16 × 16 systolic array with 274 KB on-chip memory

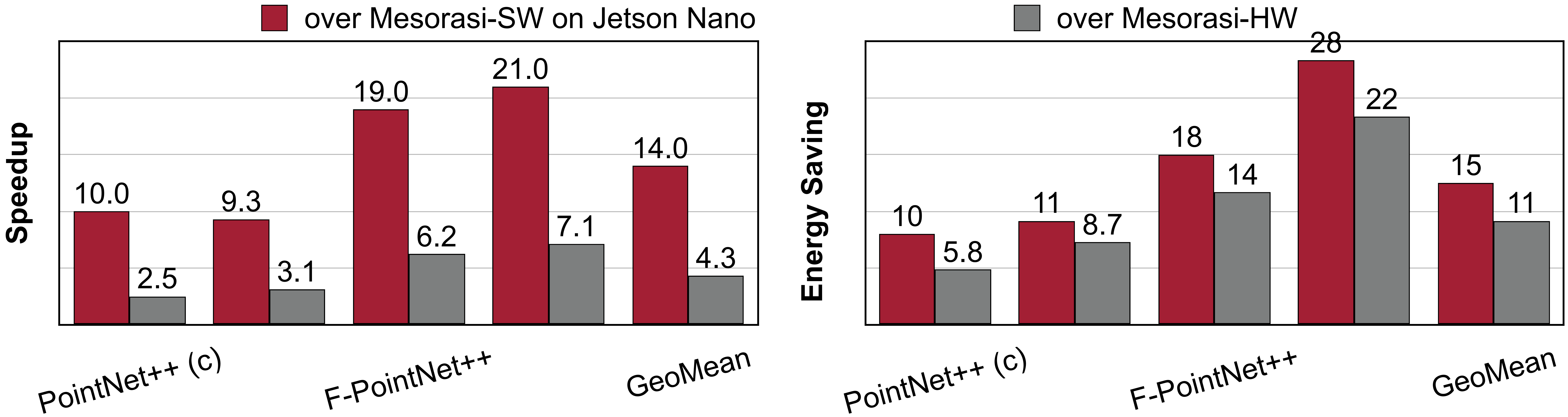
Performance Gain over the Server Products



Performance Gain over the Edge Devices

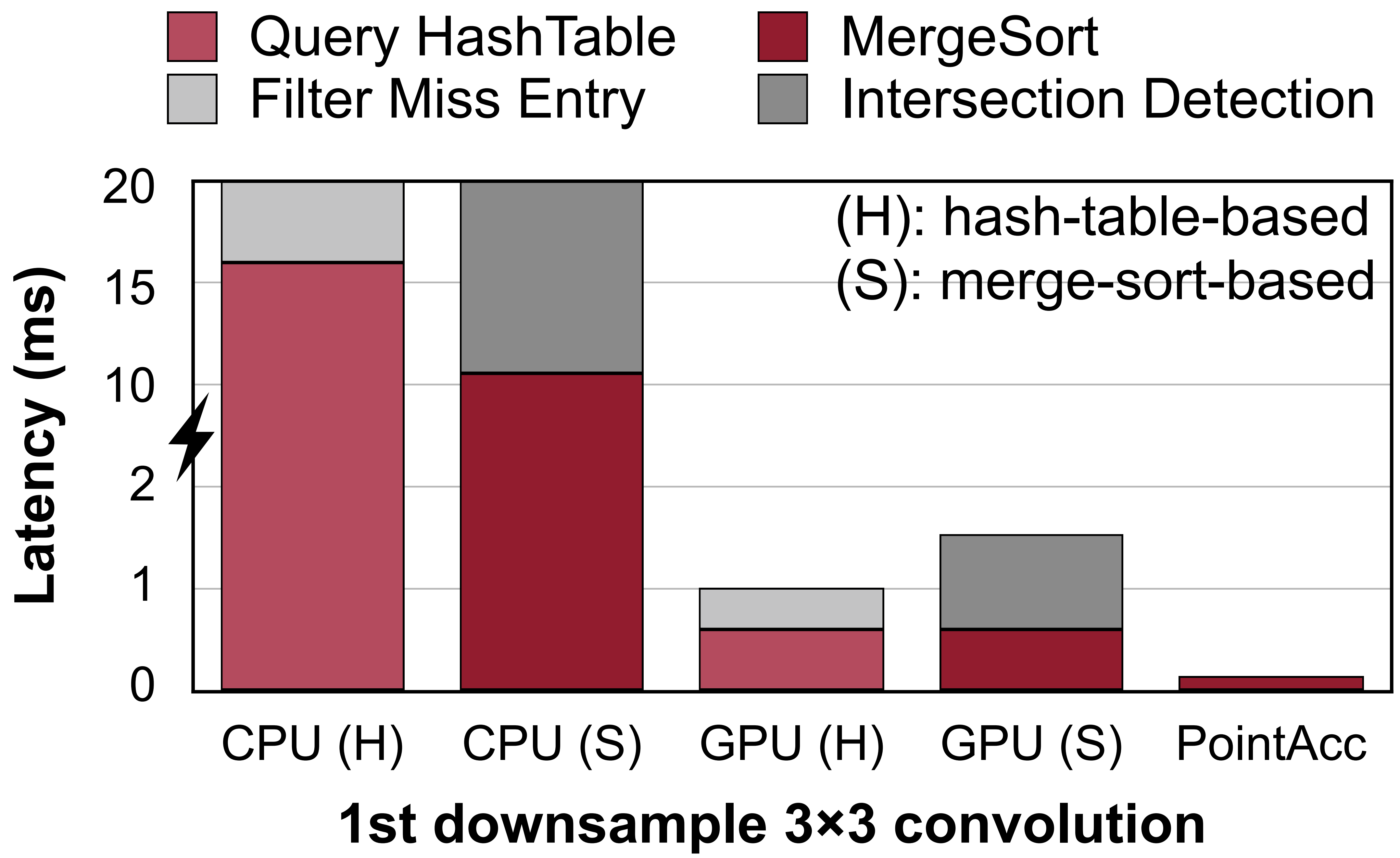


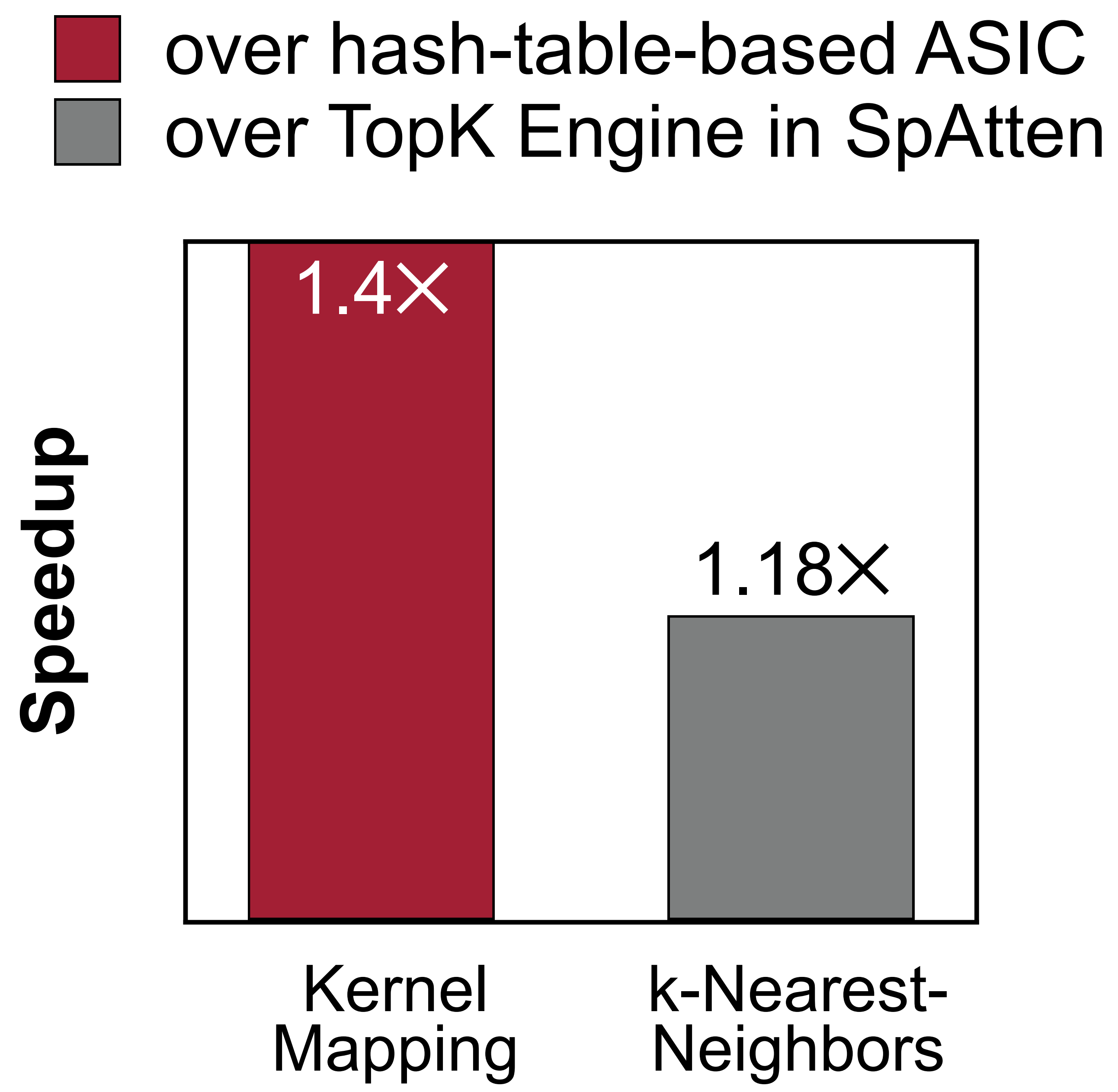
Performance Gain of PointAcc.Edge over Mesorasi



https://hanlab.mit.edu/projects/pointacc

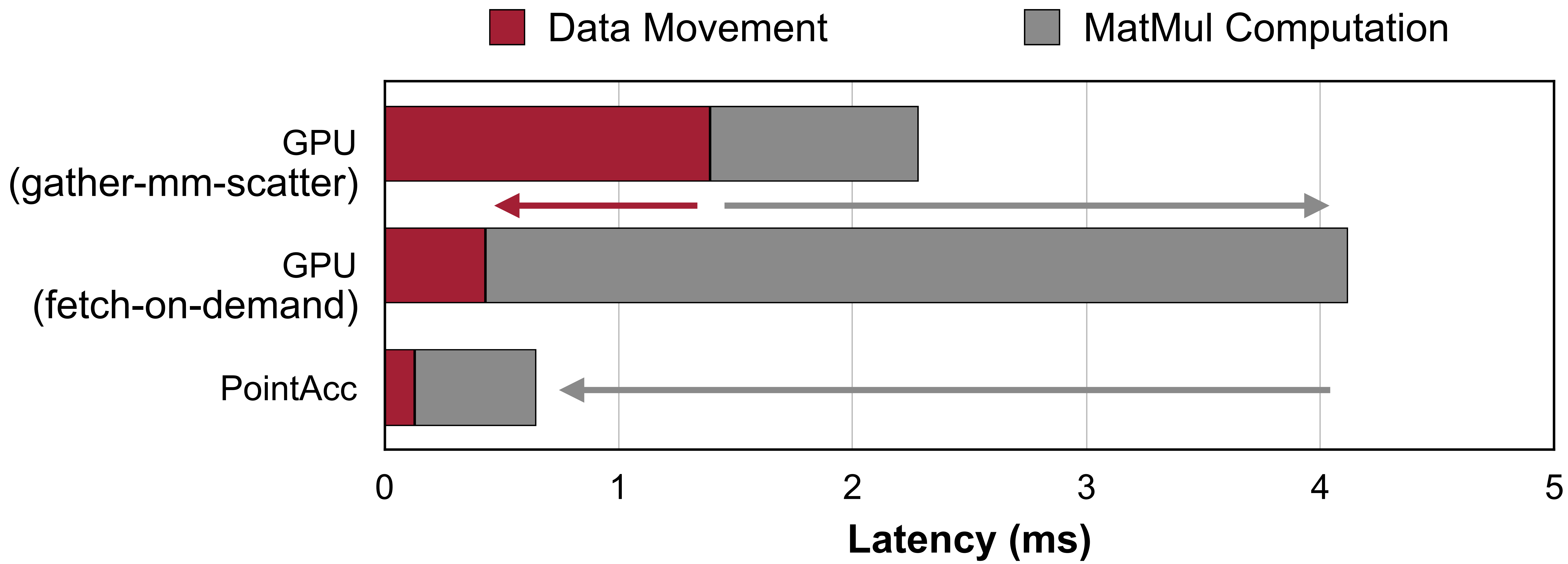
Source of Performance Gain





- Merge-sort-based implementation worsens the performance on CPU/GPU:
  - Excessive off-chip memory access between each stage of merge-sort.
  - Doubled #points in post-processing (2× #points after merge sort).
- PointAcc spatially pipelines the stages of merge sort and intersection detection.
- Using one versatile architecture for different mapping ops does not hinder the performances.

Source of Performance Gain



1st 3×3 convolution

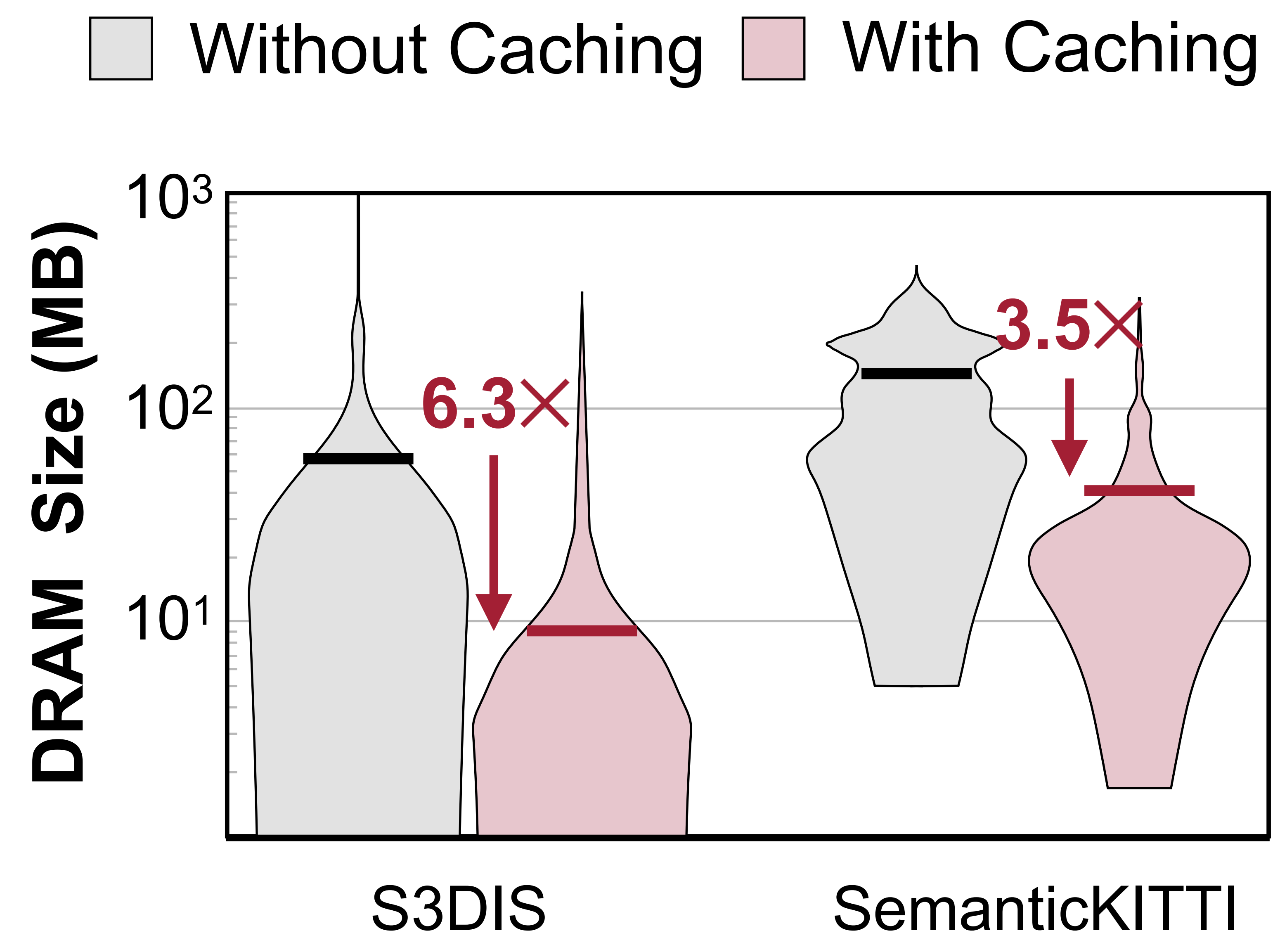
On GPU

- Fetch-on-demand flow saves the data movement cost by 3✕.
- Decomposing the MM into MV multiplication significantly increases the computation overhead.

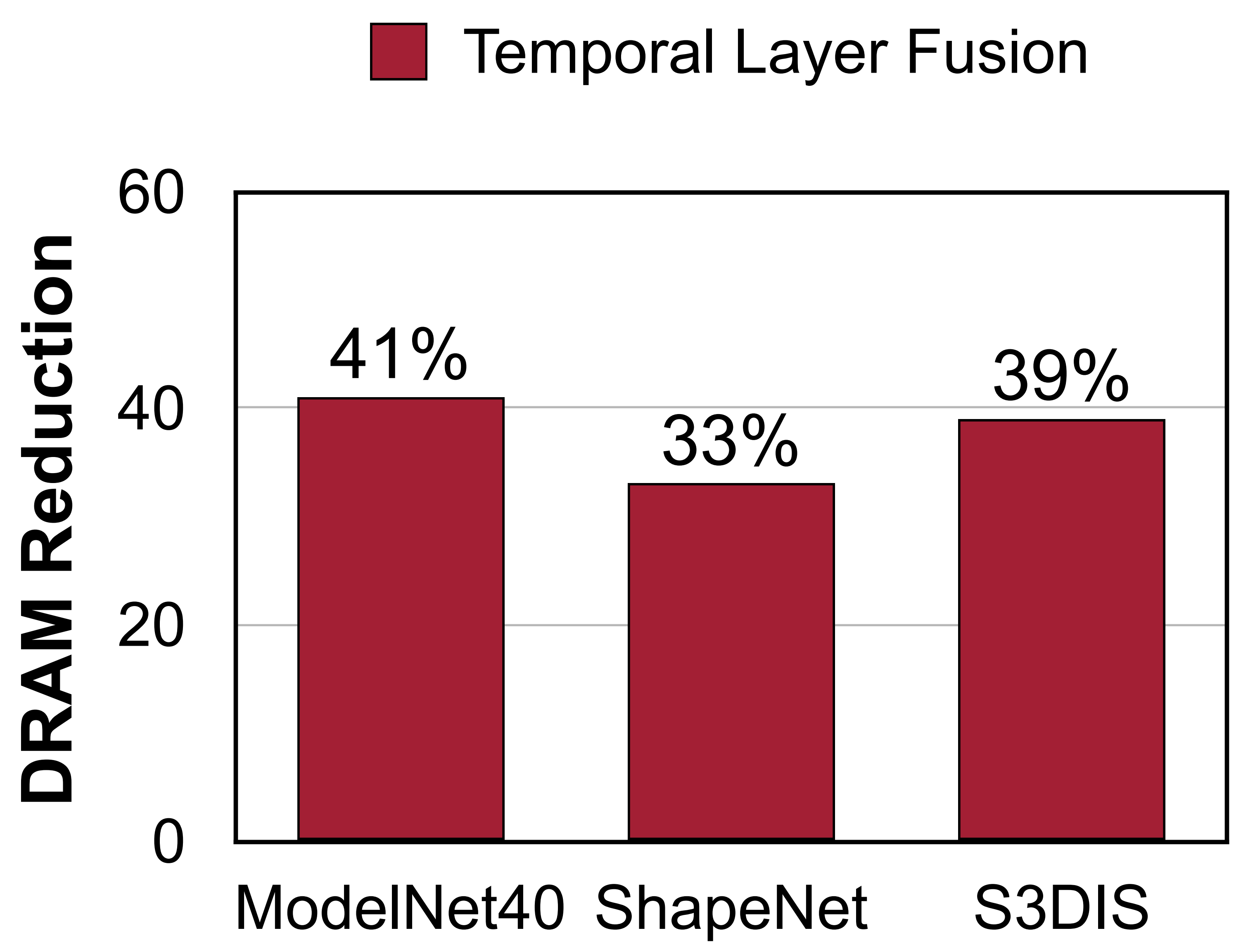
PointAcc

- Decoupled data orchestration and powerful systolic array cancel the MV overhead.

Source of Performance Gain



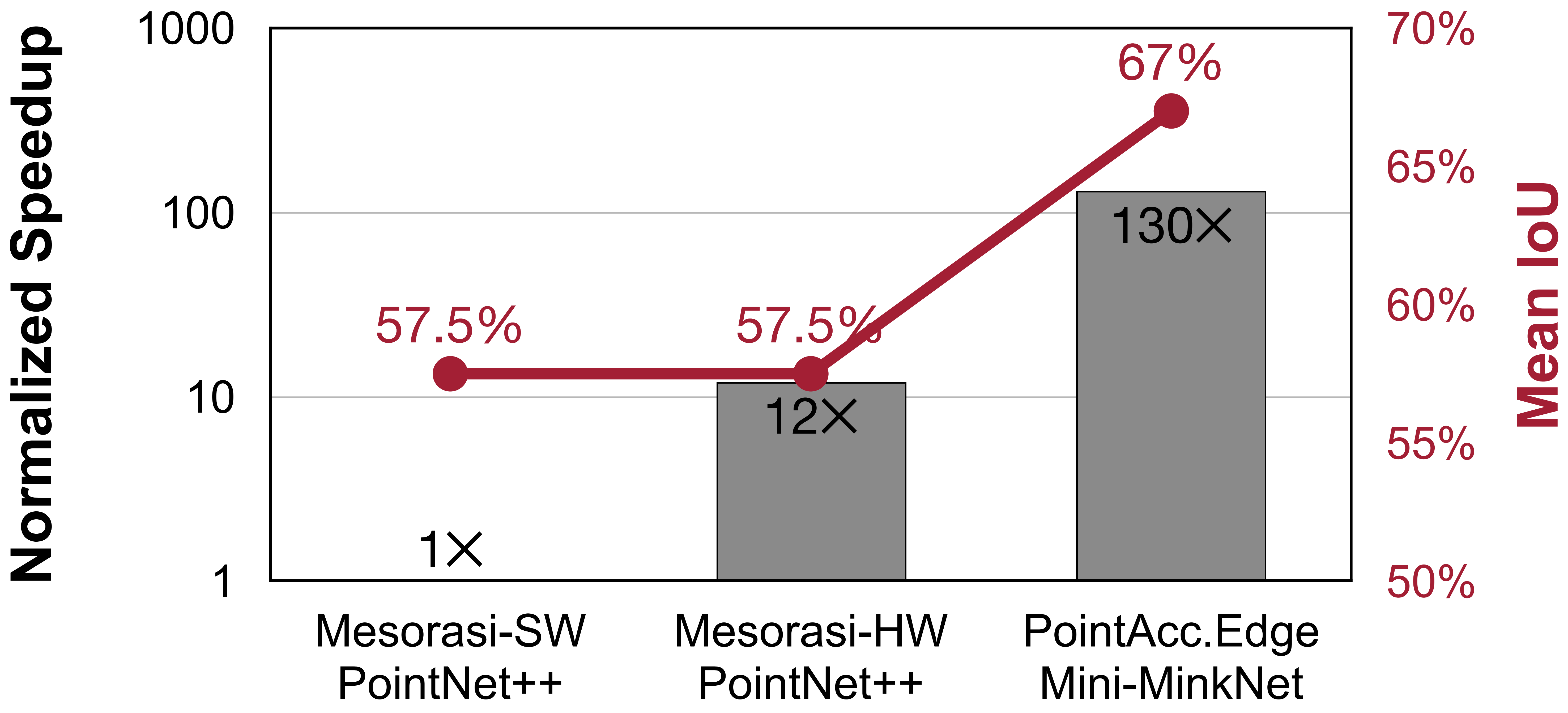
running sparse-conv-based NNs



running pointnet++-based NNs

- The shape of distribution of the off-chip access data size per layer is nearly the same with and without caching → caching works consistently on different layers and different datasets.
- Caching reduces the off-chip memory footprint by 3.5× to 6.3×.
- Temporal layer fusion cuts the off-chip memory footprint from 33% to 41%.

PointAcc.Edge v.s. Mesorasi



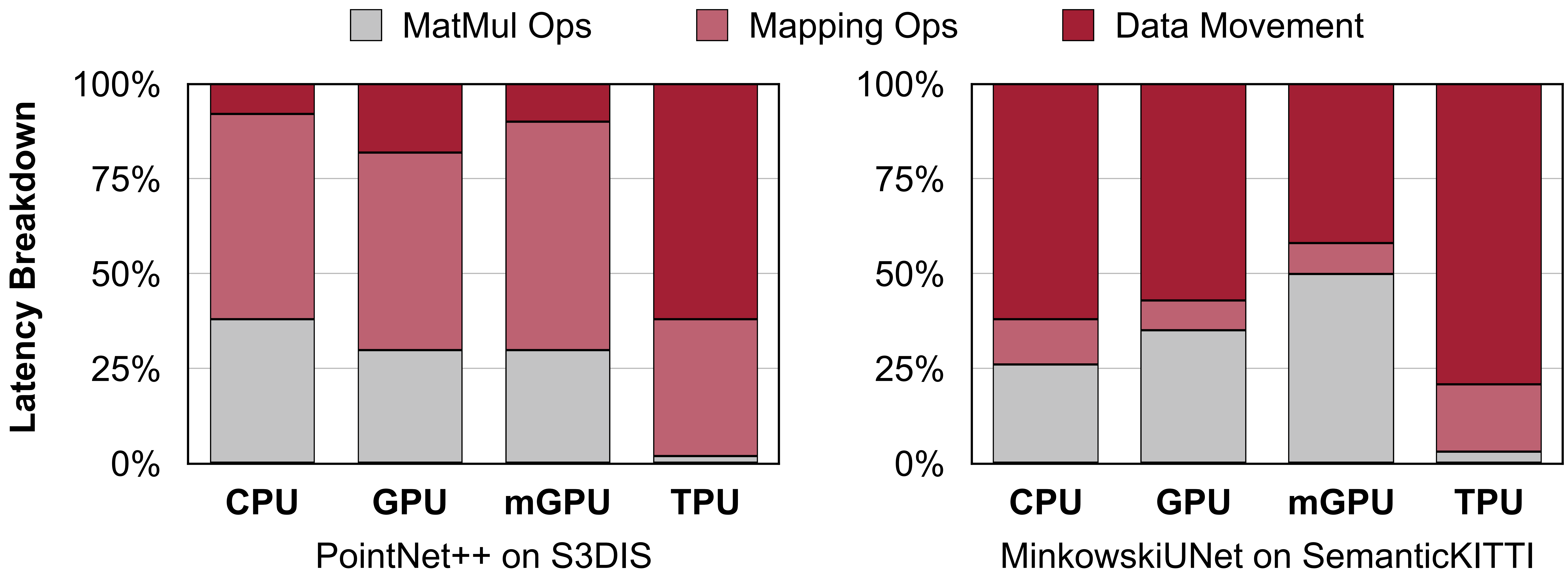
- Mesorasi does not support independent weights for different neighbors
- The state-of-the-art models tend to use independent weights for different neighbors
- Running the same segmentation task on S3DIS dataset, PointAcc.Edge is **9.1% higher accuracy with 130× lower latency**.

Conclusion

- With the rise of point cloud modality, the rapid development of point cloud deep learning brings new challenges and exciting opportunities for intelligent hardware design.

Conclusion

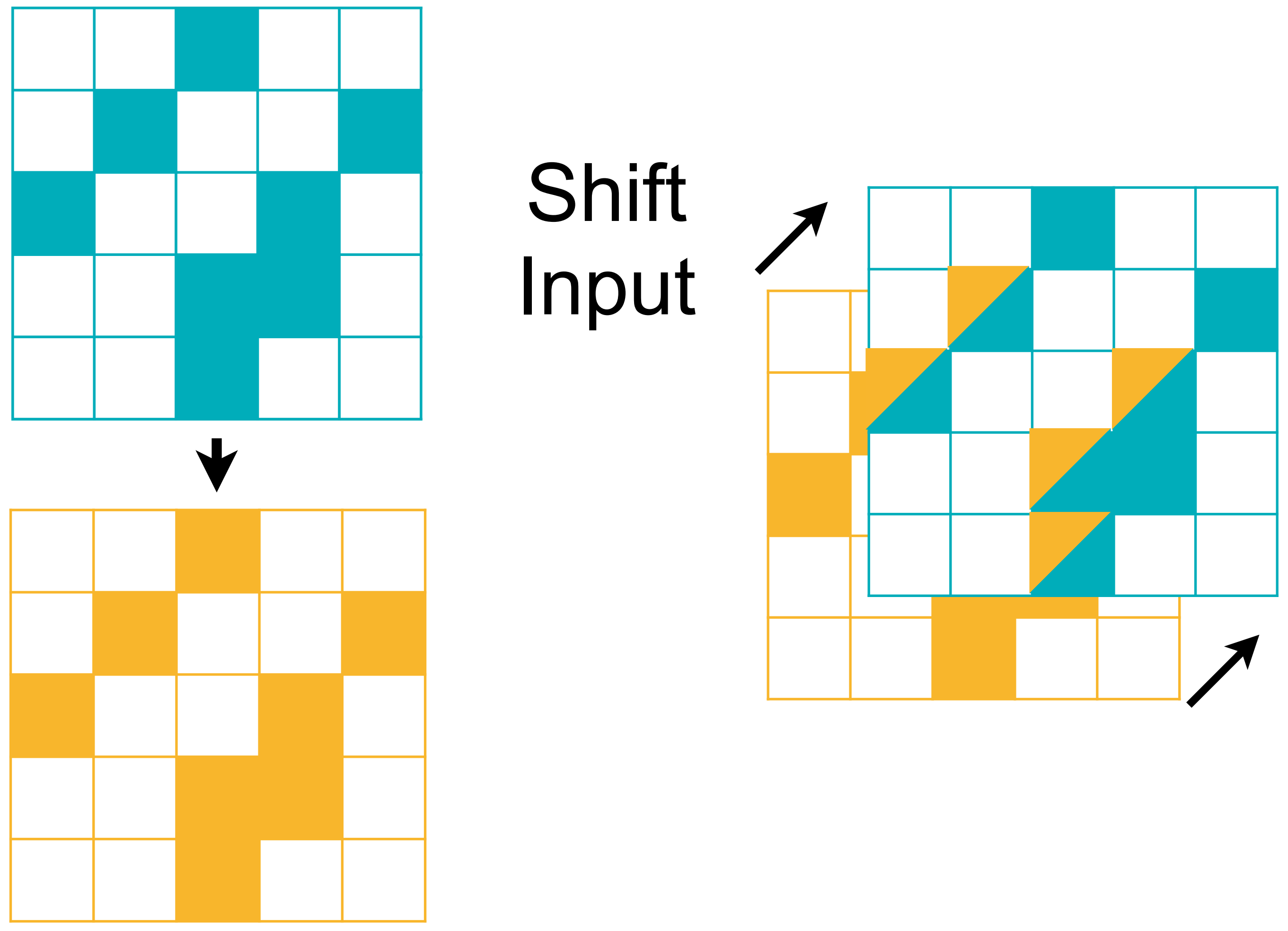
- With the rise of LiDAR sensors, the rapid development of point cloud deep learning brings new challenges and exciting opportunities for intelligent hardware design.
- The sparsity of point clouds in the physical space leads to two execution bottlenecks:
  - Newly introduced mapping operations for searching (input, output, weight) maps
  - Data movement overhead from gather and scatter the sparsely distributed features



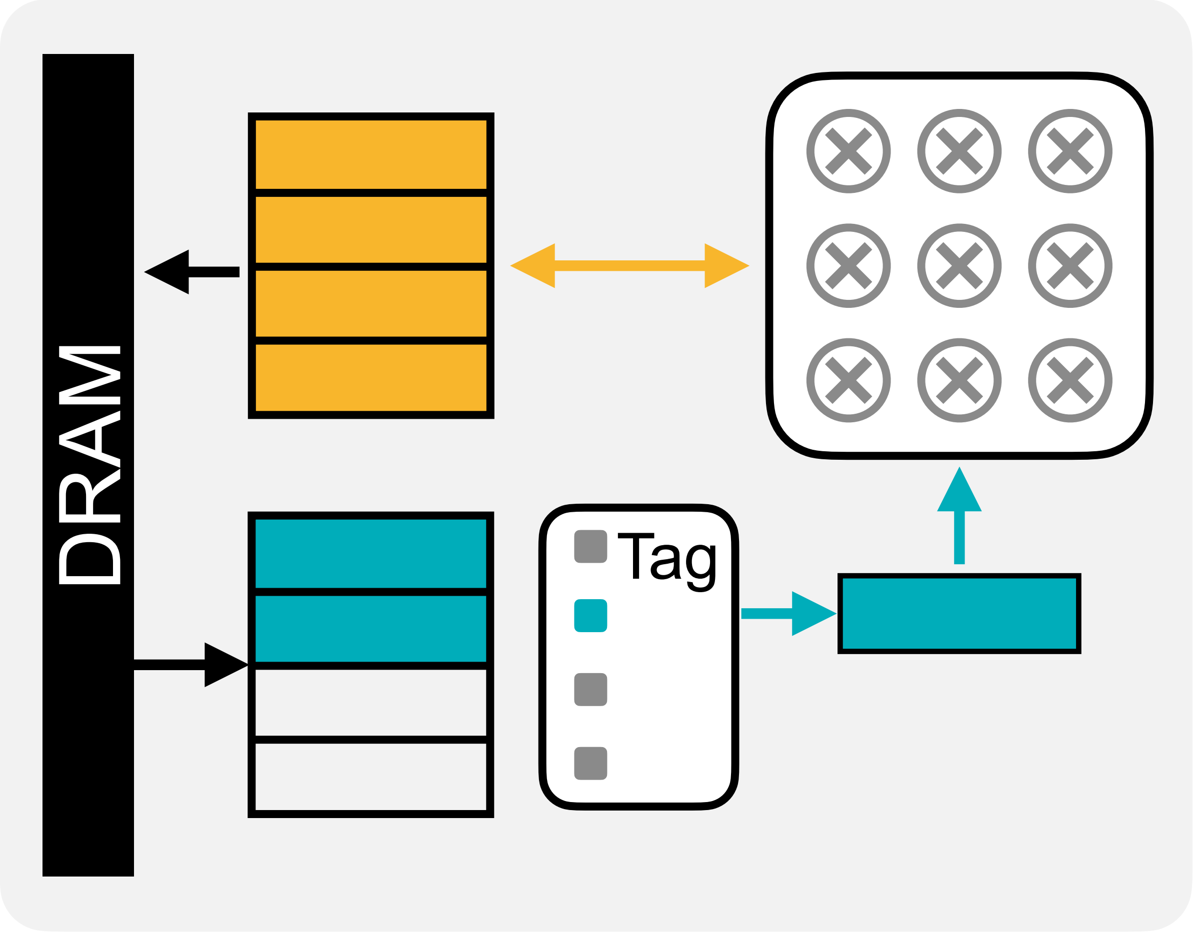
https://hanlab.mit.edu/projects/pointacc

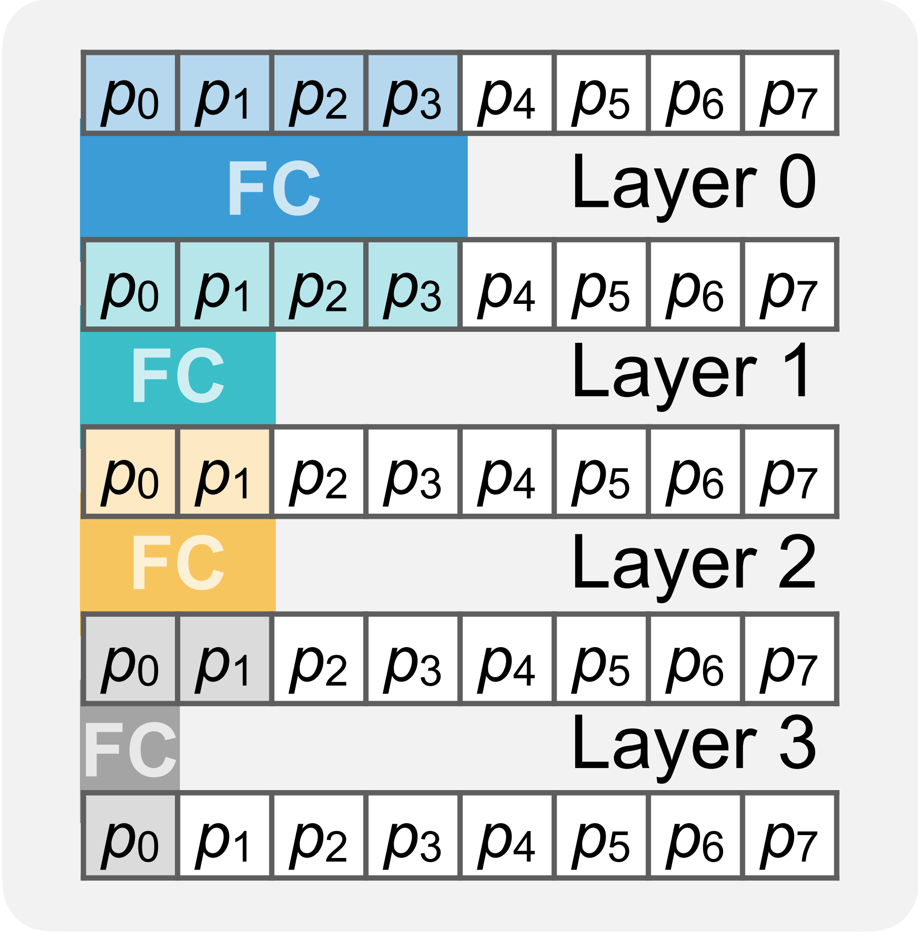
Conclusion

- With the rise of LiDAR sensors, the rapid development of point cloud deep learning brings new challenges and exciting opportunities for intelligent hardware design.
- The sparsity of point clouds in the physical space leads to two execution bottlenecks:
  - Newly introduced mapping operations for searching (input, output, weight) maps
  - Data movement overhead from gather and scatter the sparsely distributed features
- PointAcc maps diverse mapping ops into sort-based computation with one versatile architecture.



- PointAcc reduces off-chip memory access and minimize the overhead of gather and scatter by flexible caching and layer fusion.





https://hanlab.mit.edu/projects/pointacc

Thank You

{yujunlin, songhan}@mit.edu

https://hanlab.mit.edu/projects/pointacc