ASIA SOUTH PACIFIC DESIGN AUTOMATION CONFERENCE

北京航空航天大學 BEIHANG UNIVERSITY

FuseFPS: Accelerating Farthest Point Sampling with Fusing KD-tree Construction for Point Clouds

Meng Han[1,2], Liang Wang[1,2], Limin Xiao[1,2], Hao Zhang[1,2],

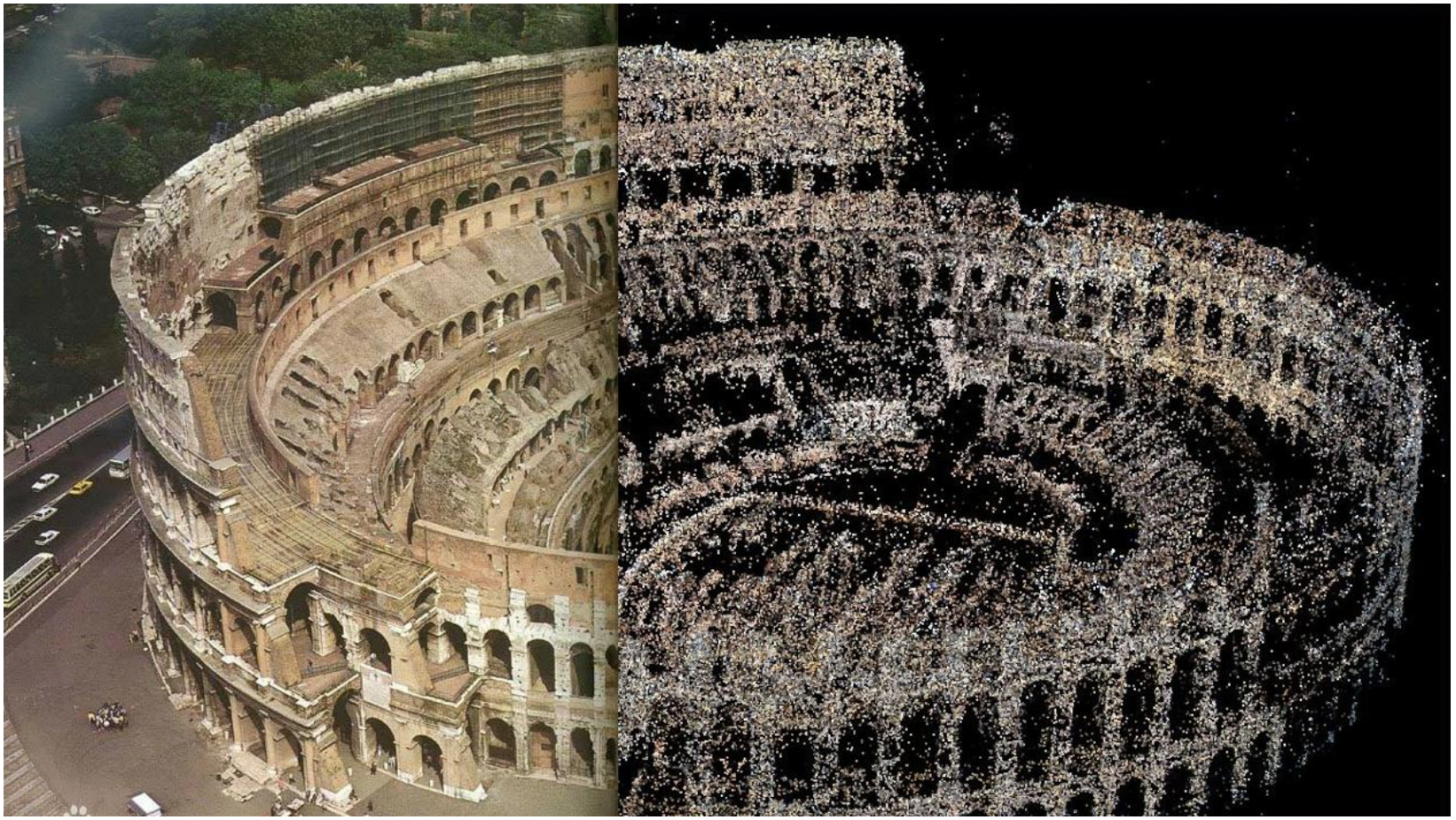
Chenhao Zhang[1,2], Xilong Xie[1,2], Shuai Zheng[1,2], Jin Dong[3]

[1]State Key Laboratory of Software Development Environment, Beijing, China

[2]School of Computer Science and Engineering, Beihang University, Beijing, China

[3]Beijing Academy of Blockchain and Edge Computing, Beijing, China

A Point cloud is a collection of points that directly represent 3D scenes or objects



$$P = \{< x, y, z > | x, y, z \in R\}$$

Point cloud-based machine perception has become increasingly prevalent in embedded and mobile platforms

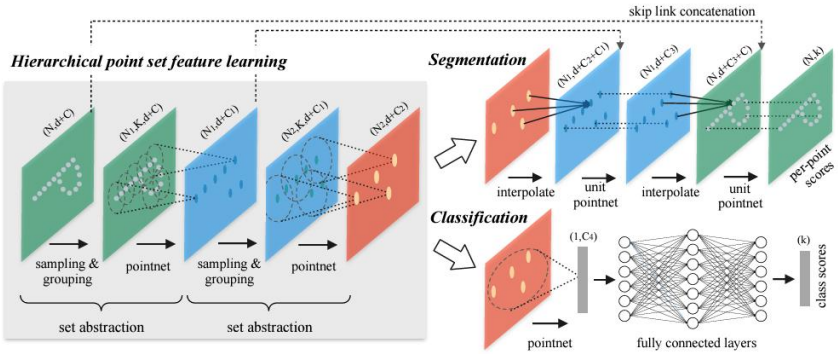
Autonomous Driving

Robotics

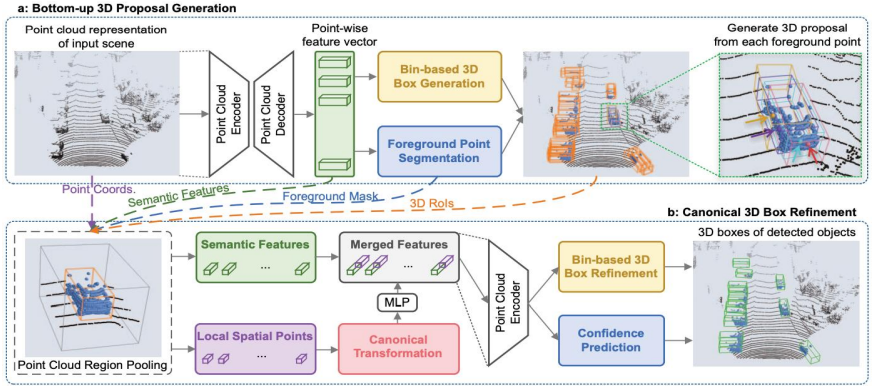
VR/AR

3D Modeling

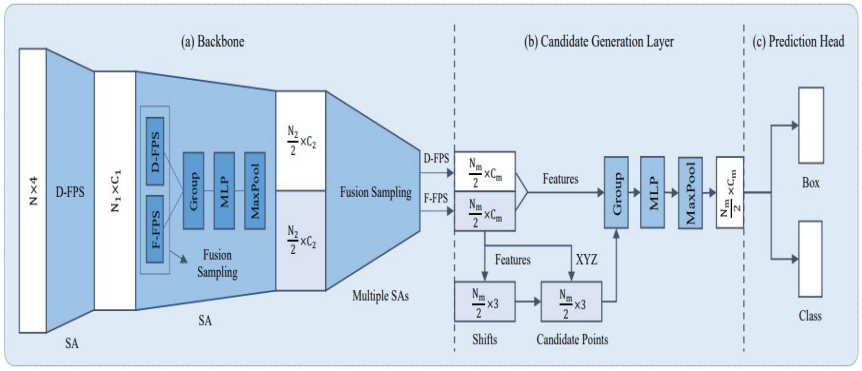
Farthest point sampling (FPS) kernel is commonly in point cloud neural networks



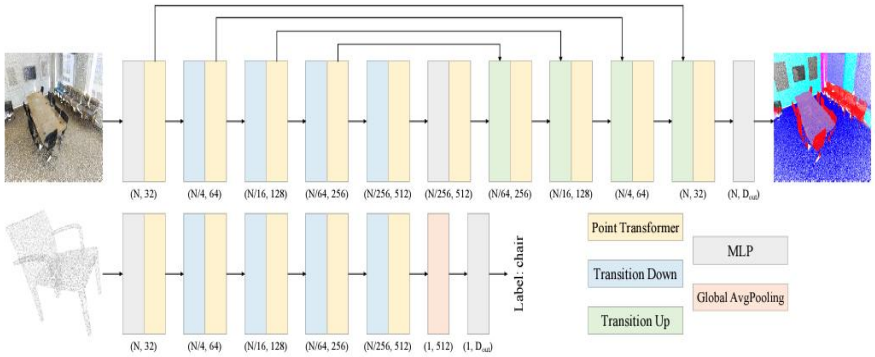
PointNet++[Qi et al., NeurIPS, 2017]



PointRCNN[Shi et al., CVPR, 2019]



3DSSD[Yang et al., CVPR, 2020]



PointTransformer[Zhao et al., ICCV, 2021]

Farthest point sampling (FPS) kernel is commonly in point cloud neural networks

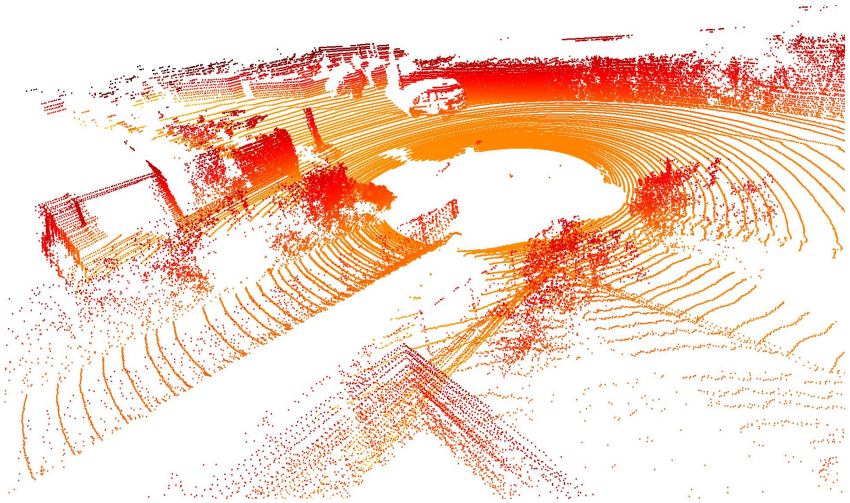
Original

Point cloud

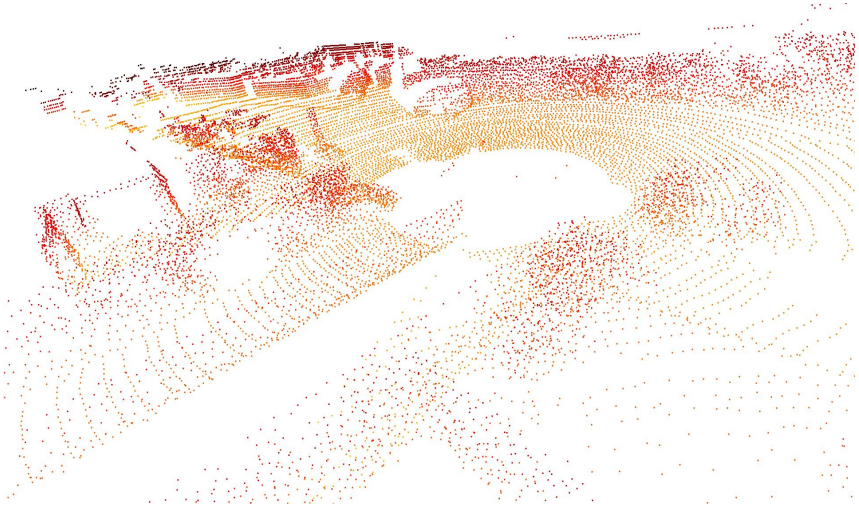
FPS | 25%

Downsampled

Point cloud

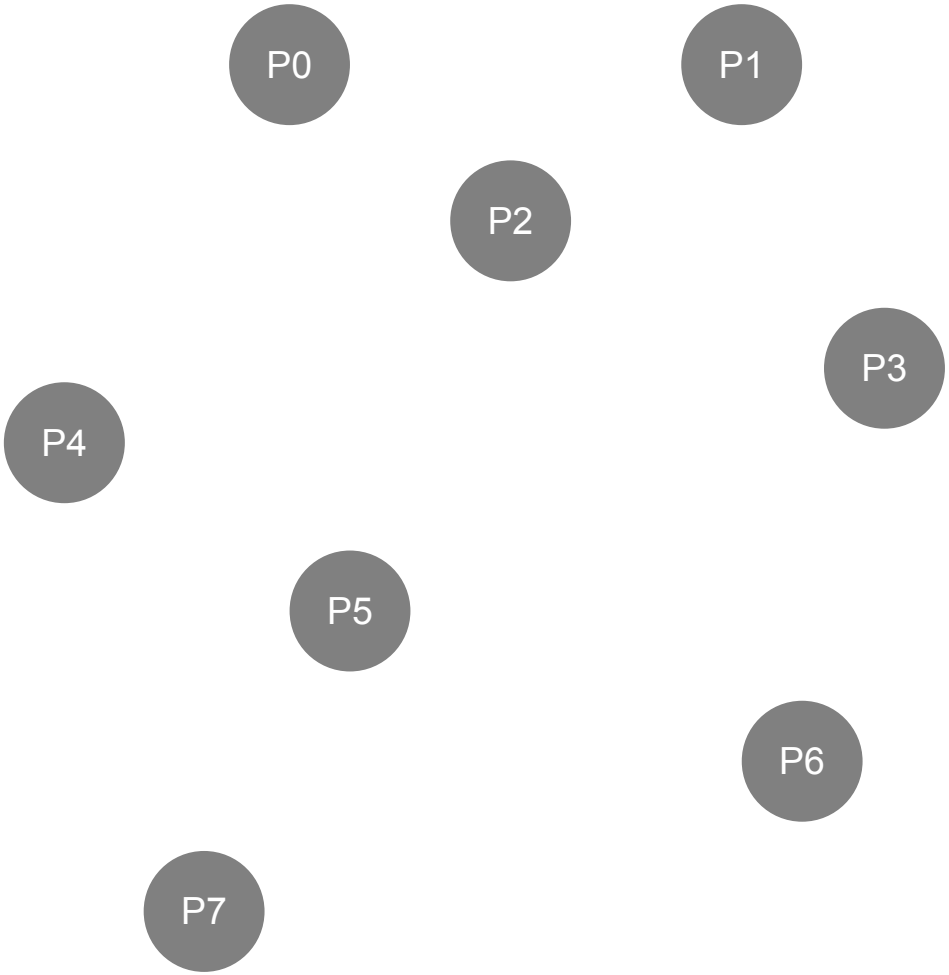


120,000 Points



30,000 Points

FPS iteratively generates a subset of the point cloud

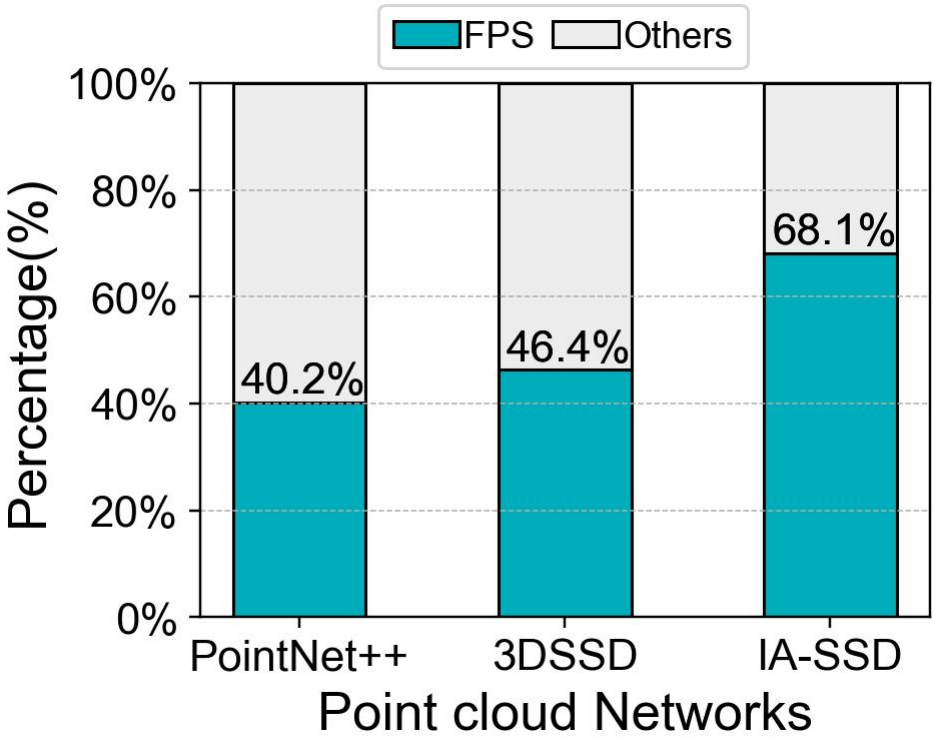


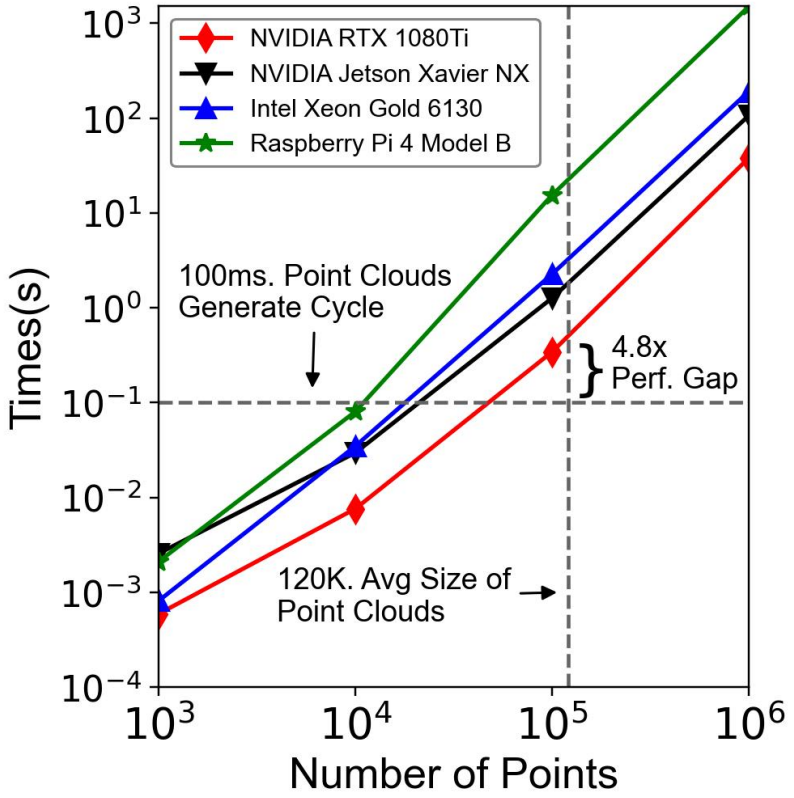
Unsampled point

Sampled point

Point	Distance
P0	0
P1	6
P2	3.5
P3	5
P4	6
P5	6
P6	0
P7	8

The execution of FPS incurs heavy memory access and high computational costs



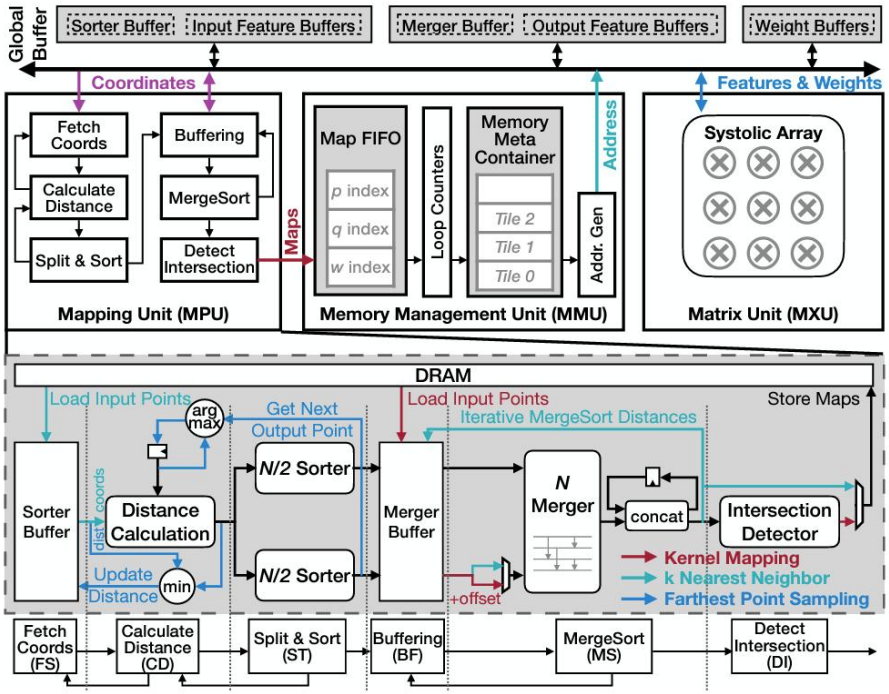


Take up to 200s to sample 10% points from a large-scale point cloud (N~10^6)

QuickFPS[Han et al. TCAD, 2023]

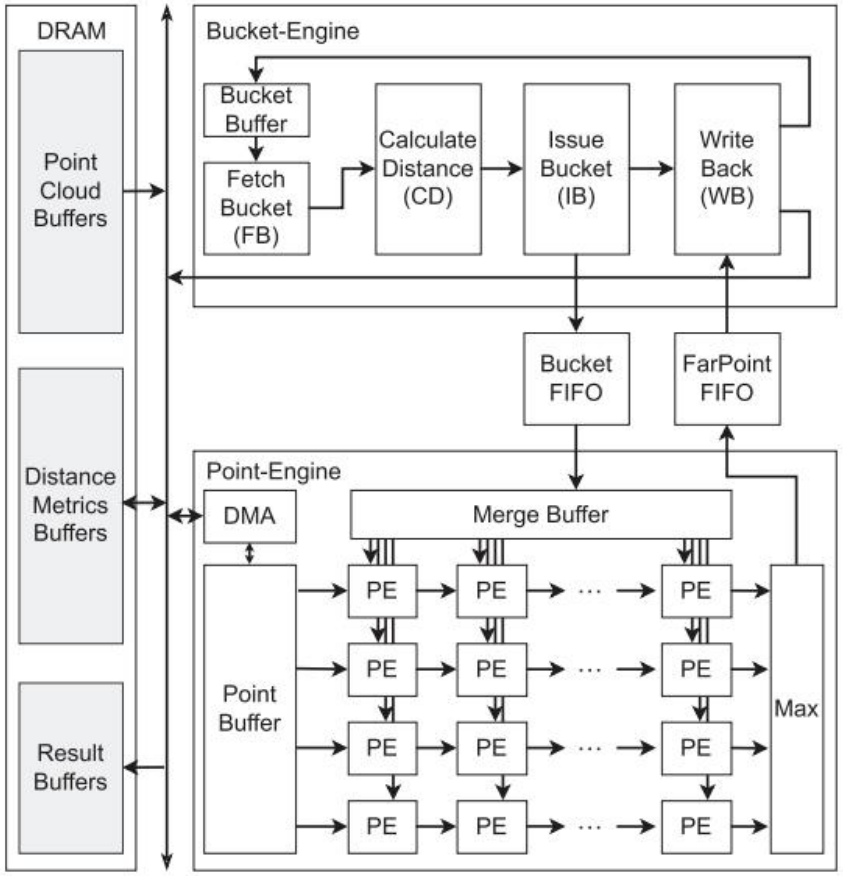
Prior works

- PointAcc [Lin et al., MICRO 2021]
  - 6-stage pipeline MPU
  - Processing points in parallelism
  - Inefficiency at large-scale point cloud

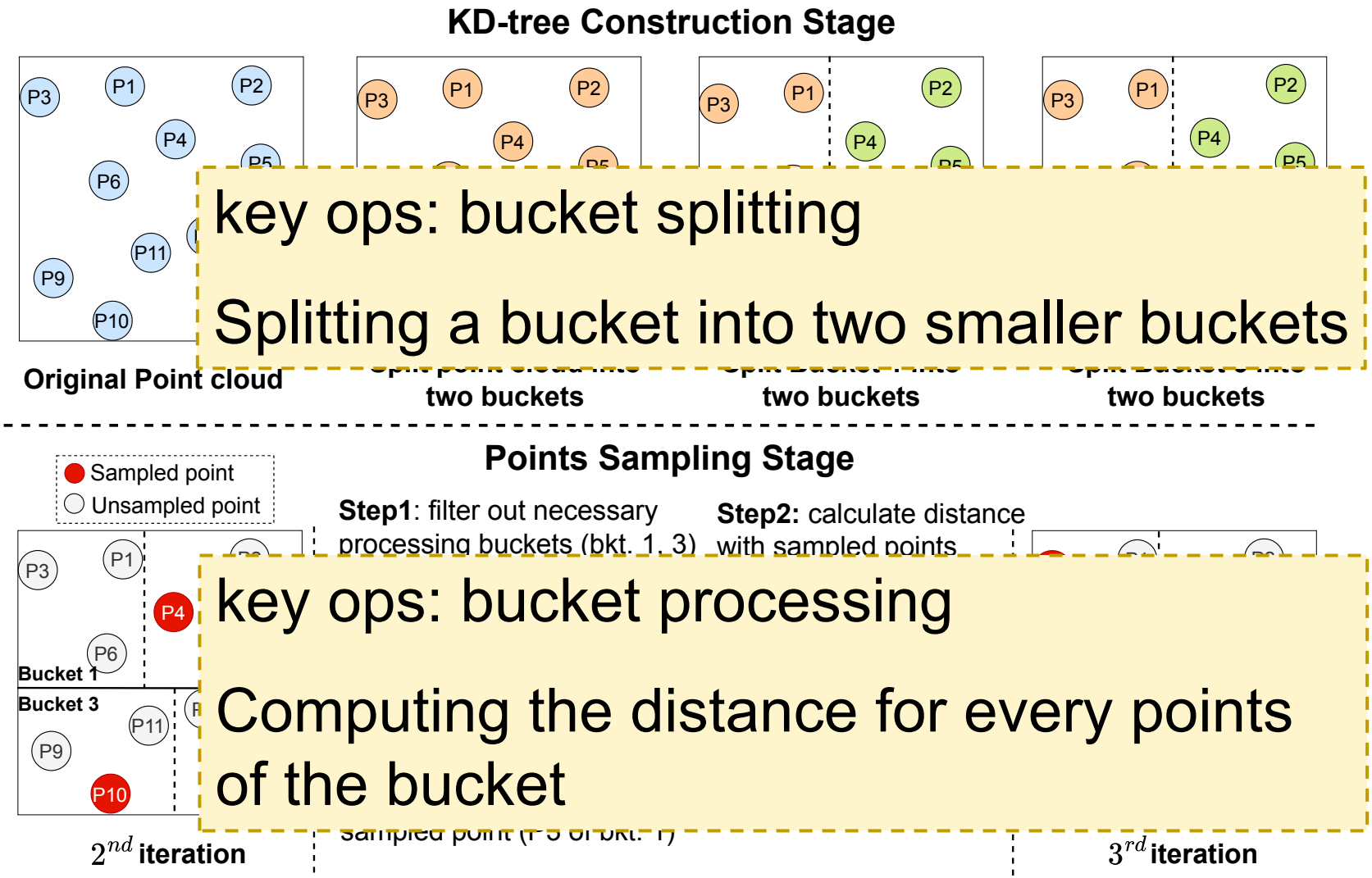


Prior works (our baseline)

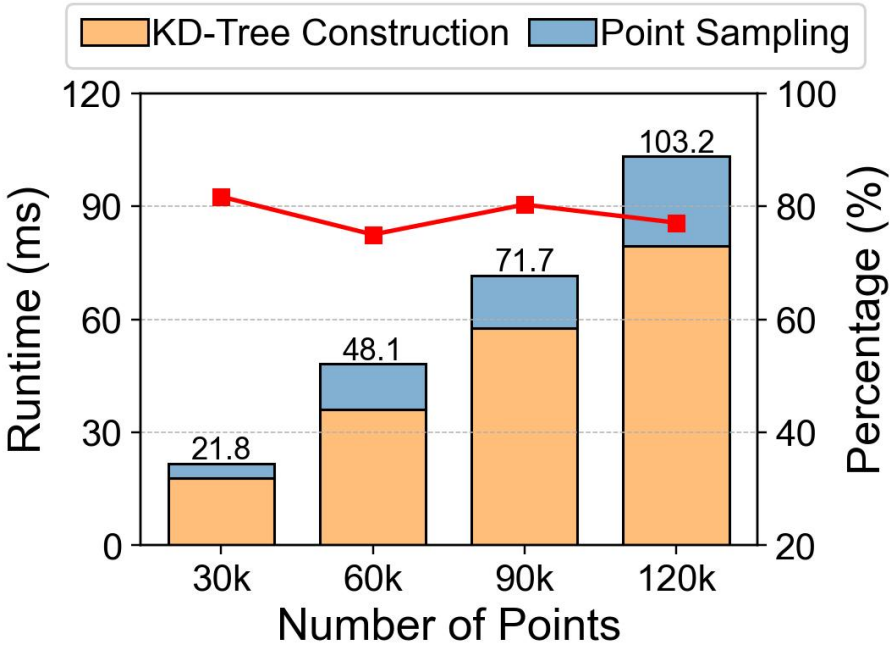
- QuickFPS [Meng et al., TCAD 2023]
  - Bucket-based FPS algorithm (BFPS), reducing unnecessary computing
  - 4-stage pipeline engine
  - PE array and Max tree



Bucket-based FPS algorithm (BFPS)



Profiling BFPS on QuickFPS

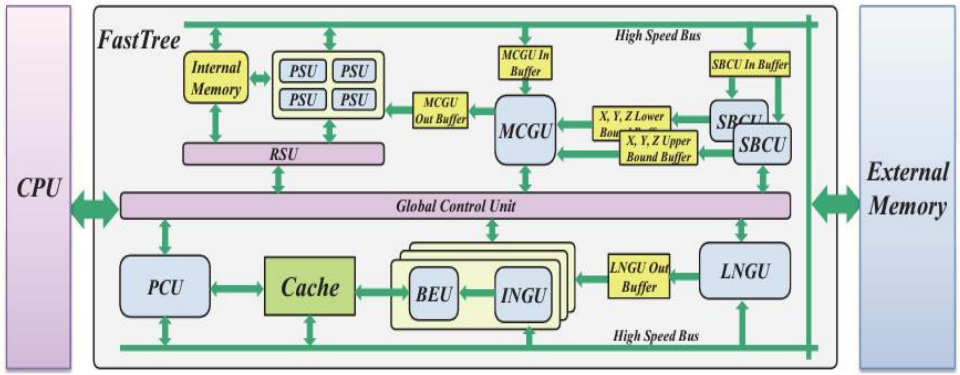


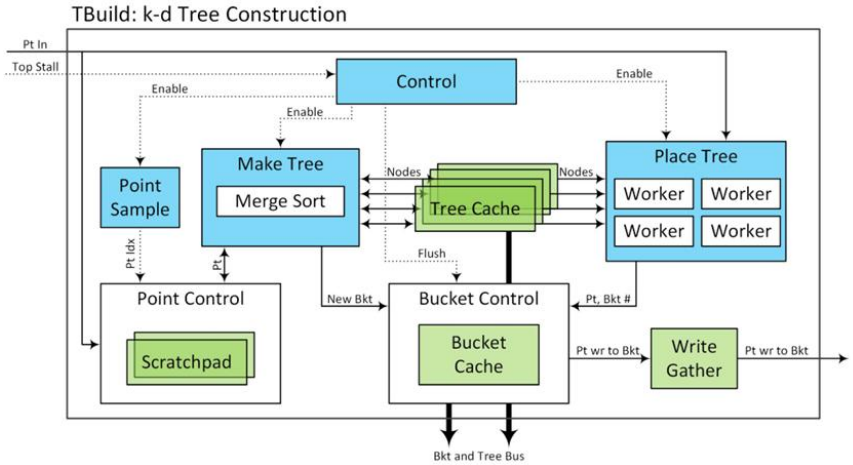
Take ≈80% runtime on

KD tree construction

The need for further optimizations in the KD-tree construction stage

Related works on KD-tree construction





FastTree [Liu et al., DATE 2015]

QuickNN [Pinkham et al., HPCA 2020]

- Sort-based KD-tree construction
  - Highly hardware cost (1.4mm^2@28nm for FastTree)
- Dedicated Unit for KD-tree construction
  - Hardware resources underutilization

FuseFPS's contribution:

1. A hardware-friendly KD-tree construction method
2. Fusing the KD-tree construction stage into the point sampling stage

Key Idea1: Hardware-friendly bucket splitting method

- Using arithmetic mean for splitting the point cloud
  - Do not affect the sampling results
  - Necessitate a minimal set of hardware components
    - accumulators, a divider
    - summation buffers

Key Idea1: Hardware-friendly bucket splitting method

- Using arithmetic mean for splitting the point cloud
  - Do not affect the sampling results
  - Necessitate a minimal set of hardware components
    - accumulators, a divider
    - summation buffers

X	3	14	24	2	6	9	1	11
Y	1	5	2	0	16	20	4	6

Sum of X dim $\downarrow$ $O(n)$

$\sum p.coord[x] = 70$

Arithmetic mean=8.75

splitting bucket by the value of 8.75

sub bucket 0

X	3	2	6	1
Y	1	0	16	4

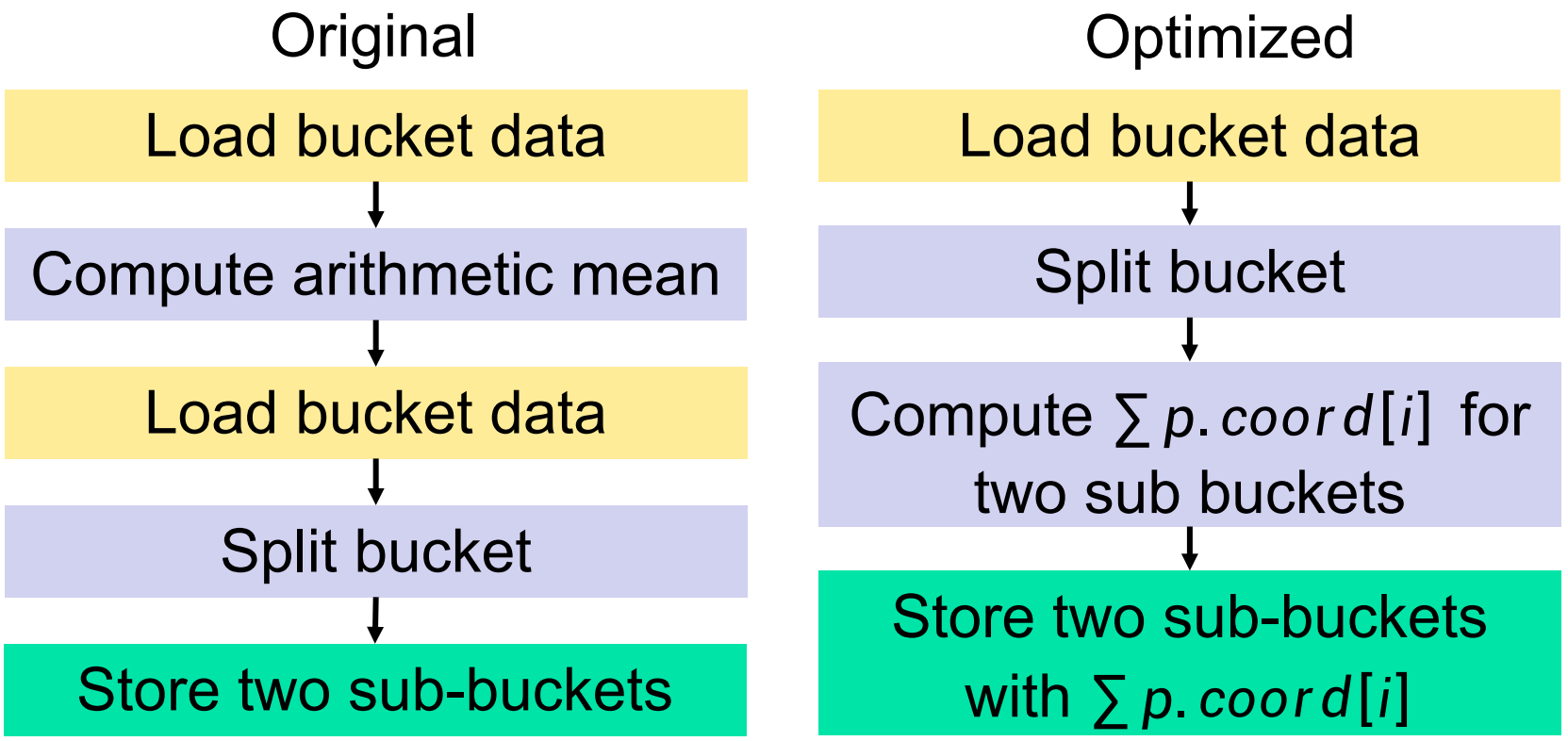
sub bucket 1

X	14	24	9	11
Y	5	2	20	6

Arithmetic mean-based method

Key Idea1: Hardware-friendly bucket splitting method

- Optimizing arithmetic mean-based splitting
  - Computing $\sum p.coord[i]$ of two sub-buckets during splitting
  - Reducing bucket data access from 2 times to 1 time



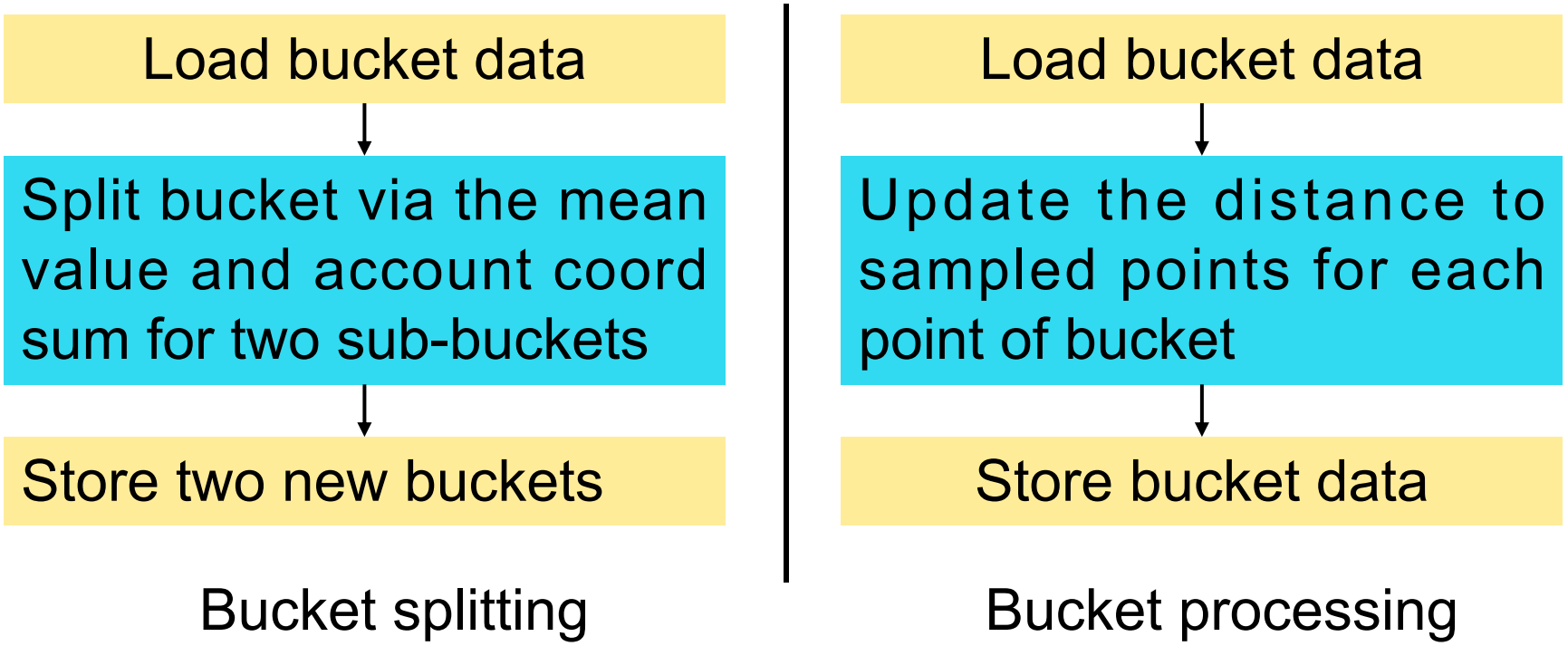
Key Idea1: Hardware-friendly bucket splitting method

- Modification of bucket structure
  - *float coordSum[3]* to store the coordinate summations
  - *height* identifies whether this bucket is required to be split

```
struct Bucket{
   // Original members of BFPS
   float bound[6]; // Axis-aligned bounding box, including up and down value
   int32 pointPtr; // Point array address
   int32 pointSize; // Point array size
   float farPoint[3]; // The point with maximum distance
   float farPointDist; // The distance of farPoint
   float referenceBuffer[4][3];//Points that require processing in post iterations
   // Additional members for FuseFPS
   float coordSum[3]; // The coordinate summations of each dimension
   int8 height; // The height of the node
};
```

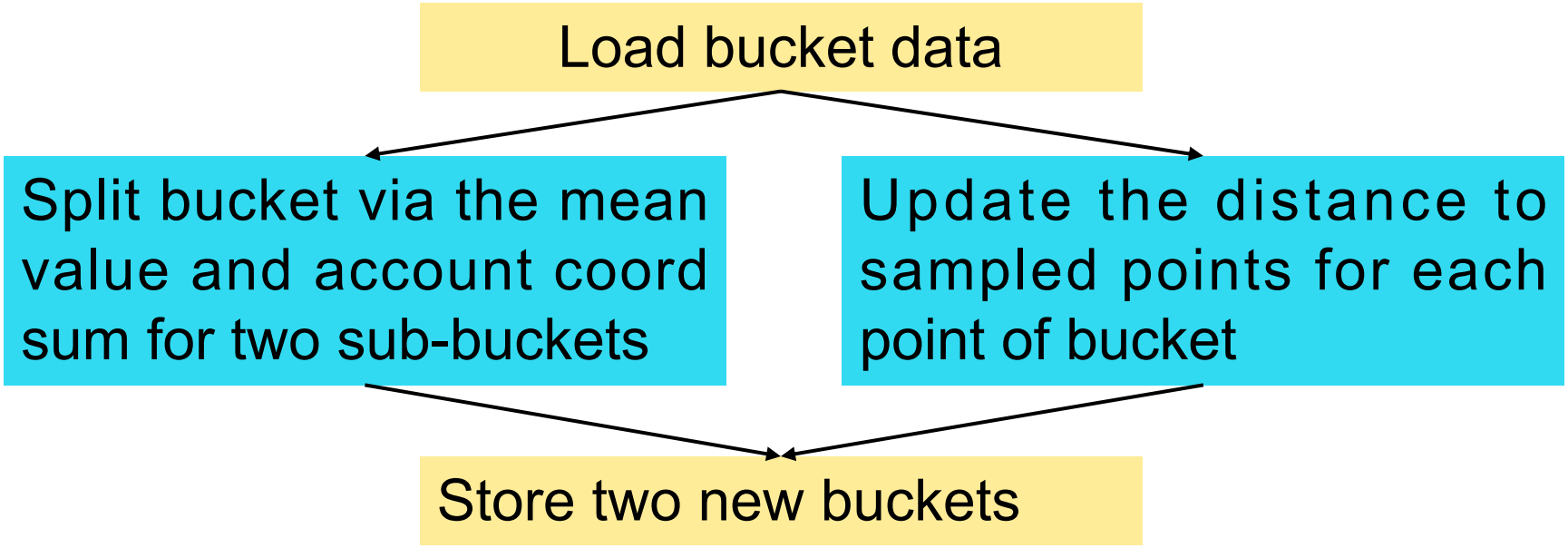
Key Idea2: Sampled-driven KD-tree construction

- Fusing the KD-tree construction stage into the point sampling stage
  - Similarity between bucket splitting in the KD-tree construction stage and bucket processing in the point sampling stage



Key Idea2: Sampled-driven KD-tree construction

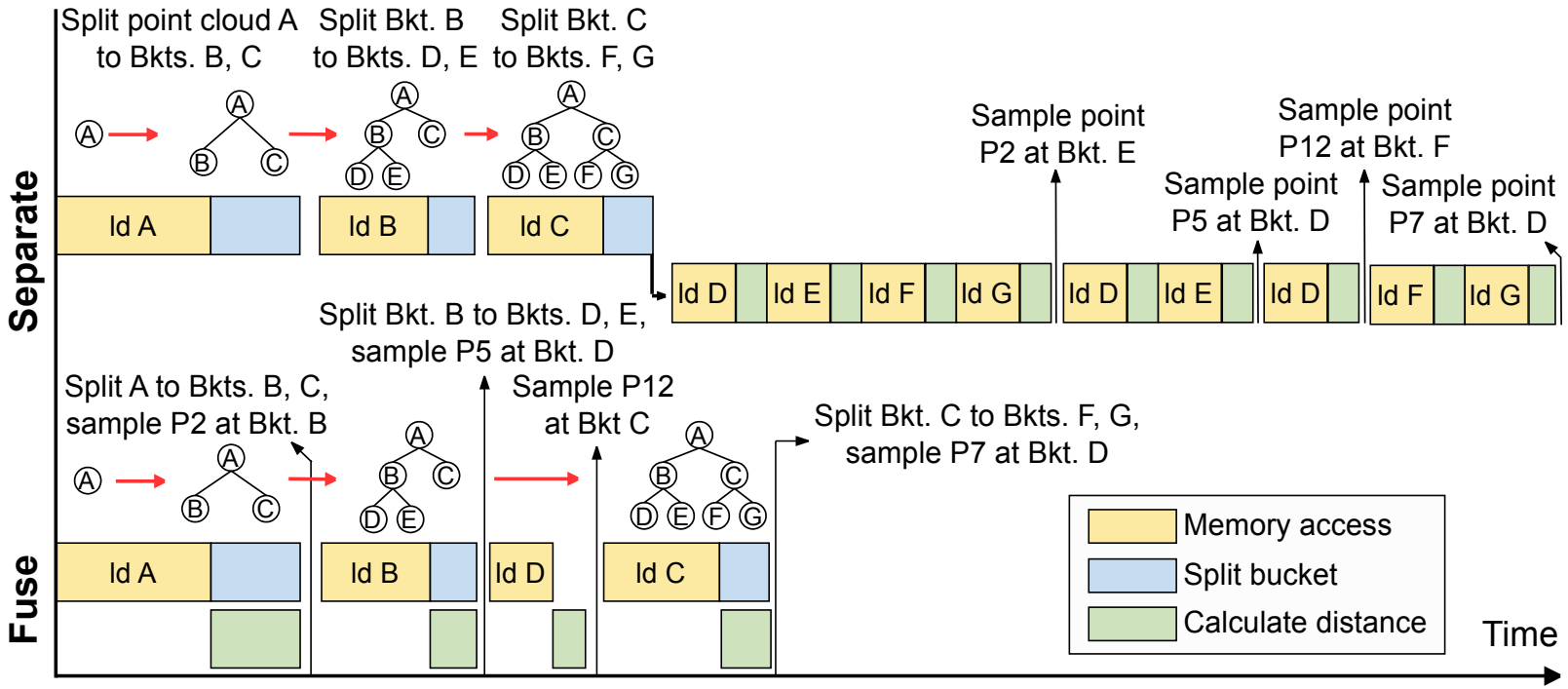
- Fusing the KD-tree construction stage into the point sampling stage
  - Benefit: reducing memory access



Fusing bucket splitting and bucket processing

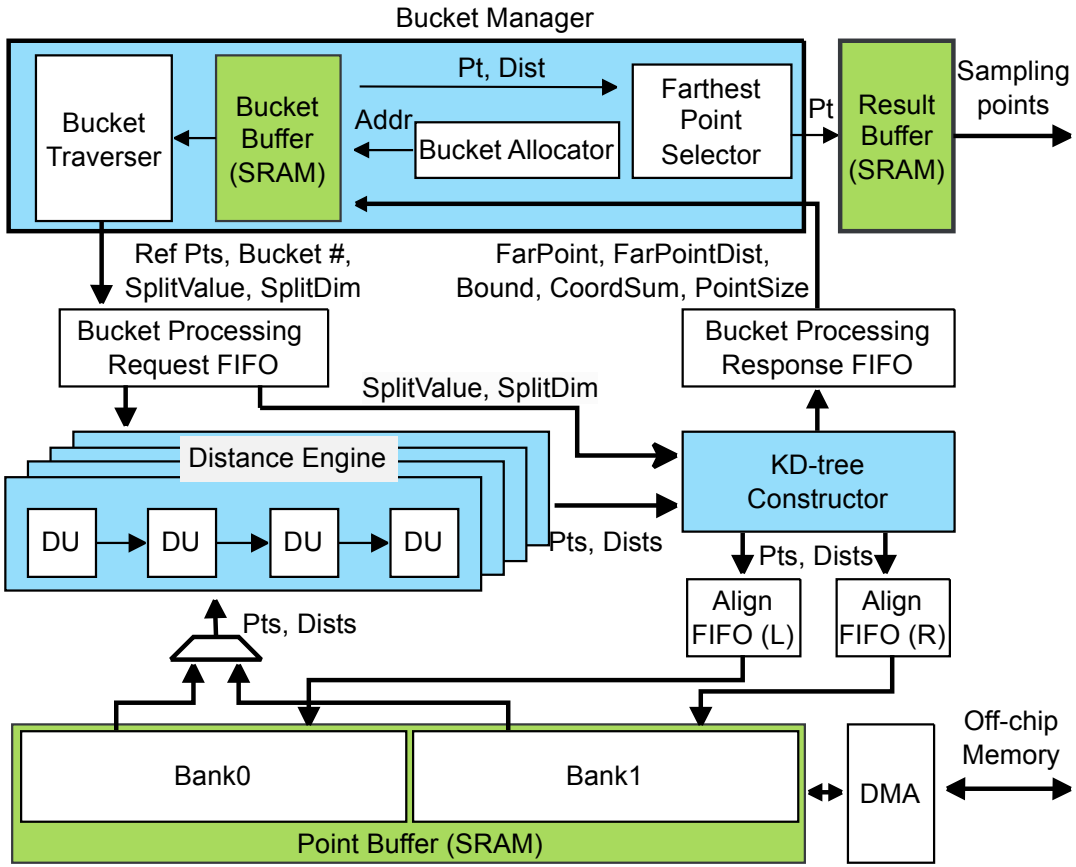
Key Idea2: Sampled-driven KD-tree construction

- Sampled-driven KD-tree construction
  - Splitting bucket when it executes bucket processing
  - Reduce latency



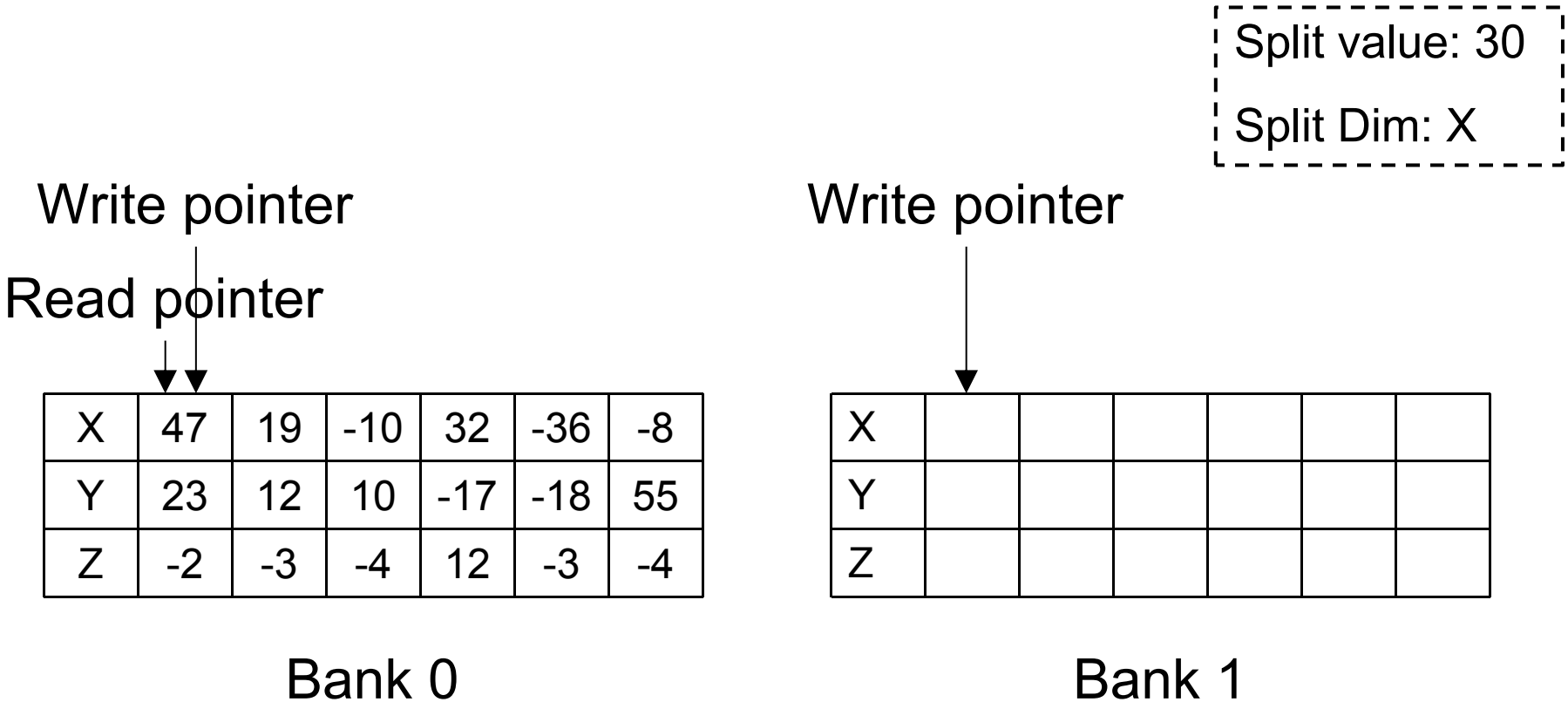
FuseFPS: Accelerator

- Bucket Manager
  - FSM-based controller
  - 4-stage pipeline
- Distance Engine
  - 16 distance units
- KD-tree Constructor
- Point Buffer
  - 2 SRAM banks
- DMA



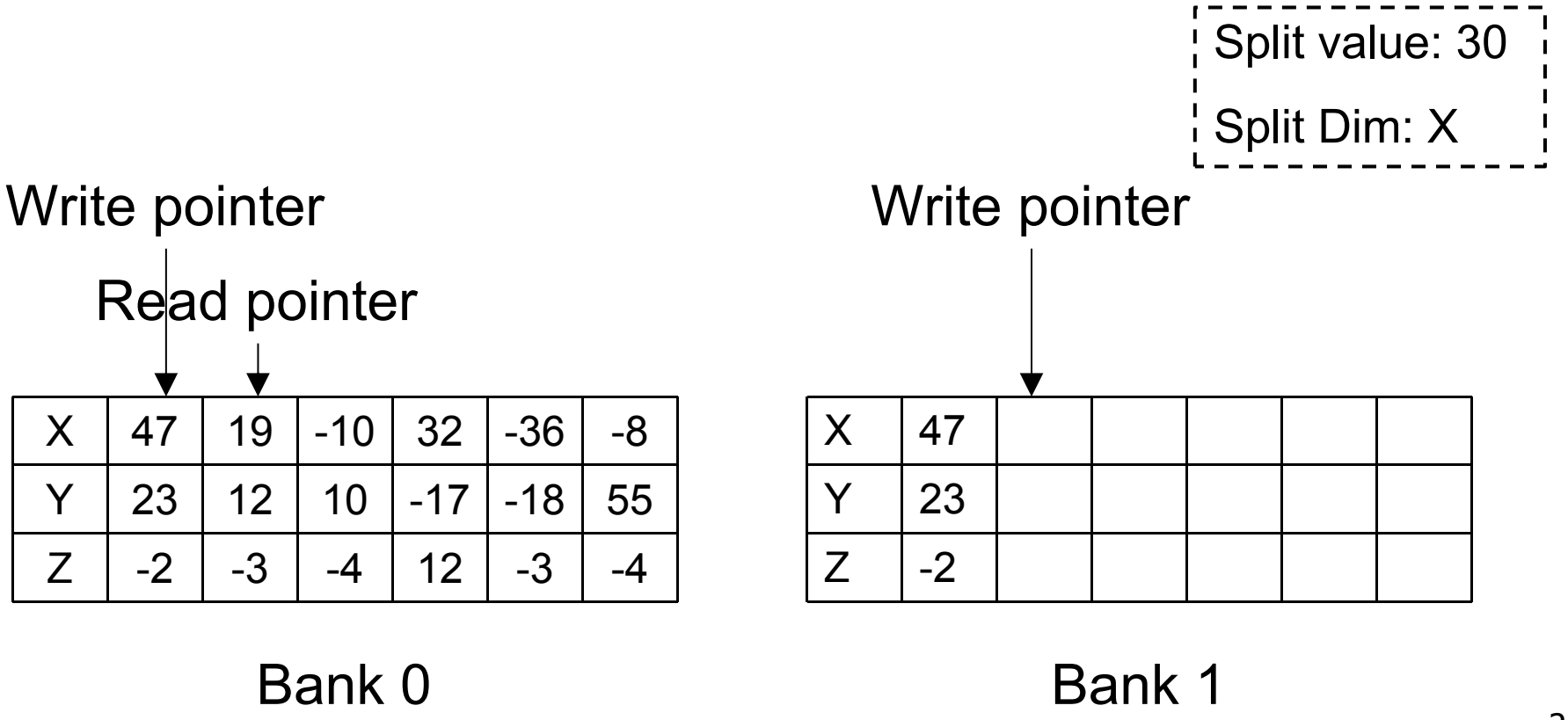
Bucket Splitting in FuseFPS

- Load Bucket data into Point buffer's Bank 0
- Splitting with one read pointer and two write pointers



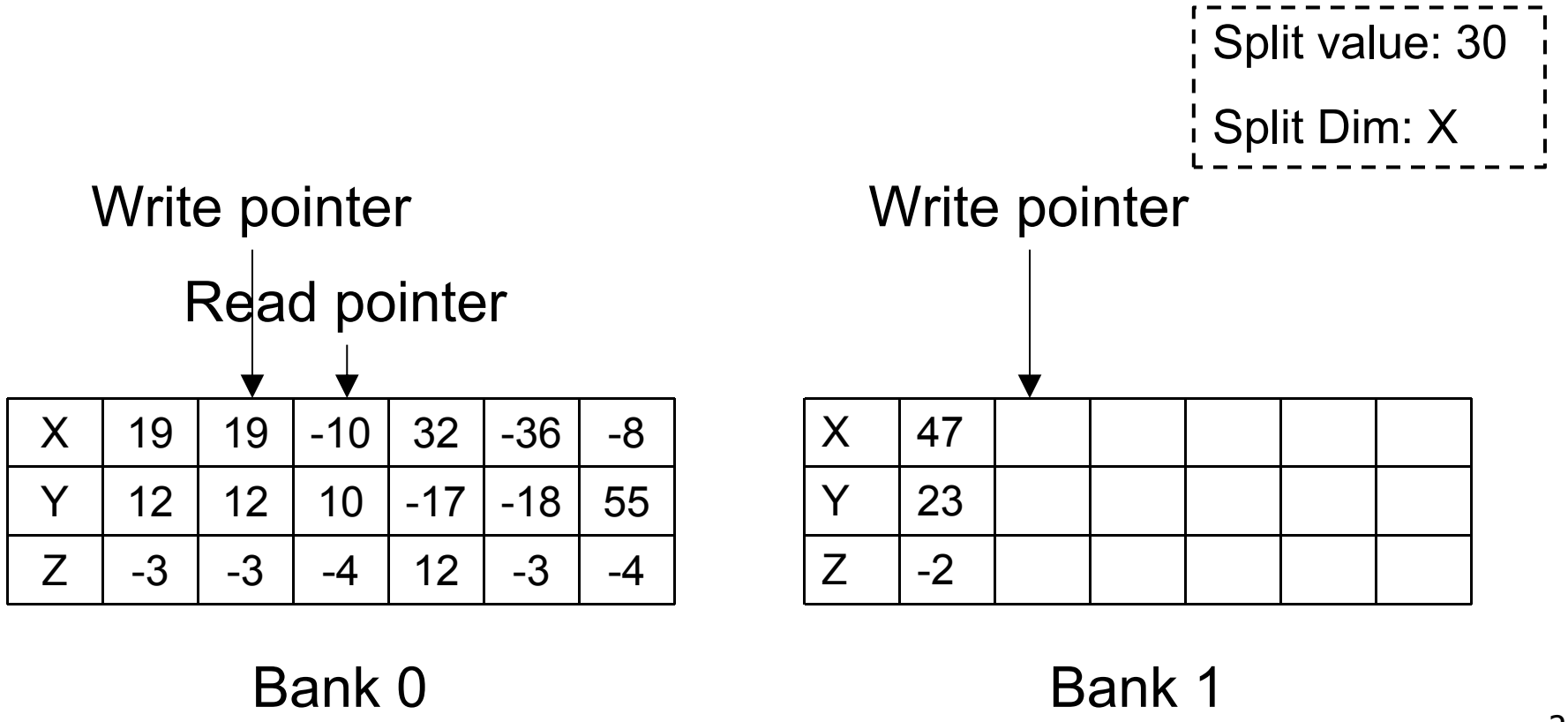
Bucket Splitting in FuseFPS

- Load Bucket data into Point buffer's Bank 0
- Splitting with one read pointer and two write pointers



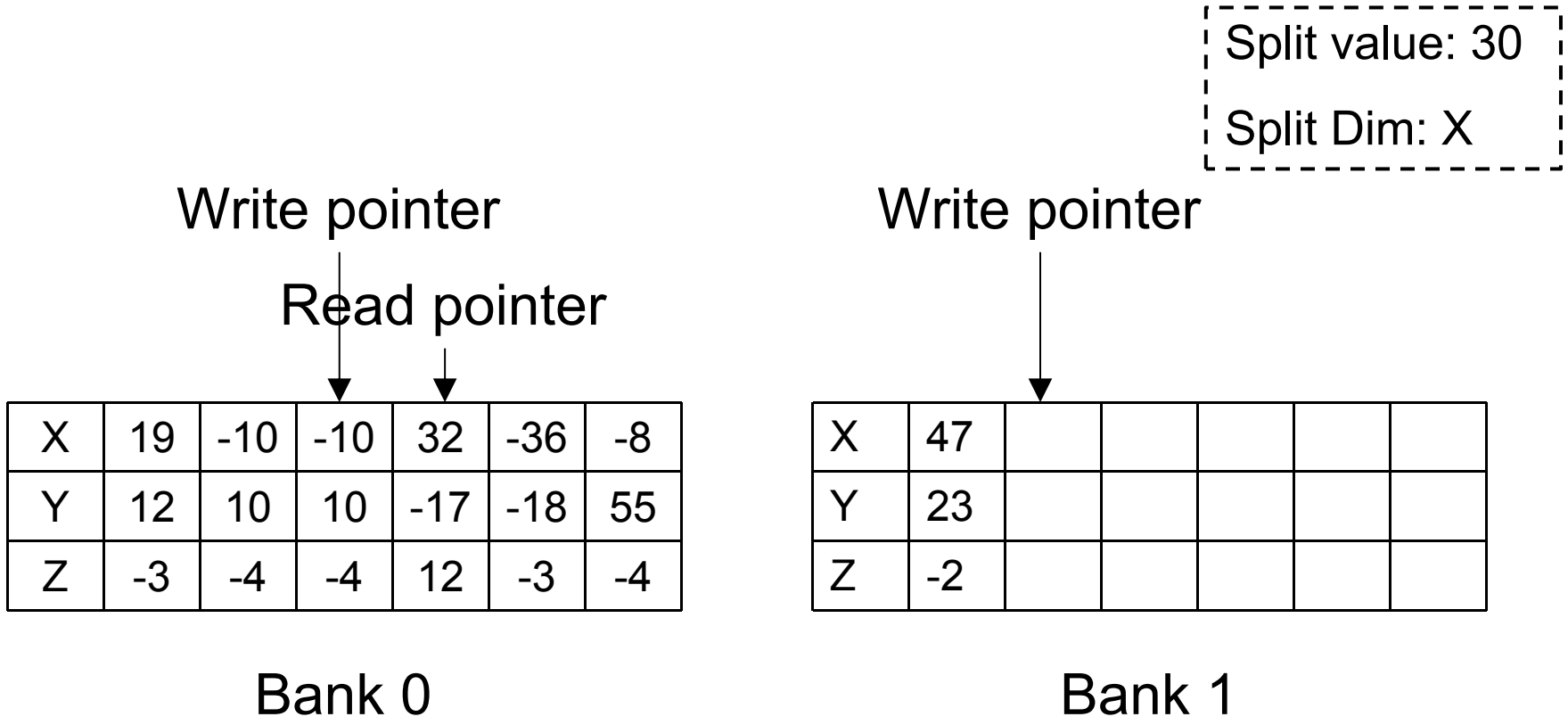
Bucket Splitting in FuseFPS

- Load Bucket data into Point buffer's Bank 0
- Splitting with one read pointer and two write pointers



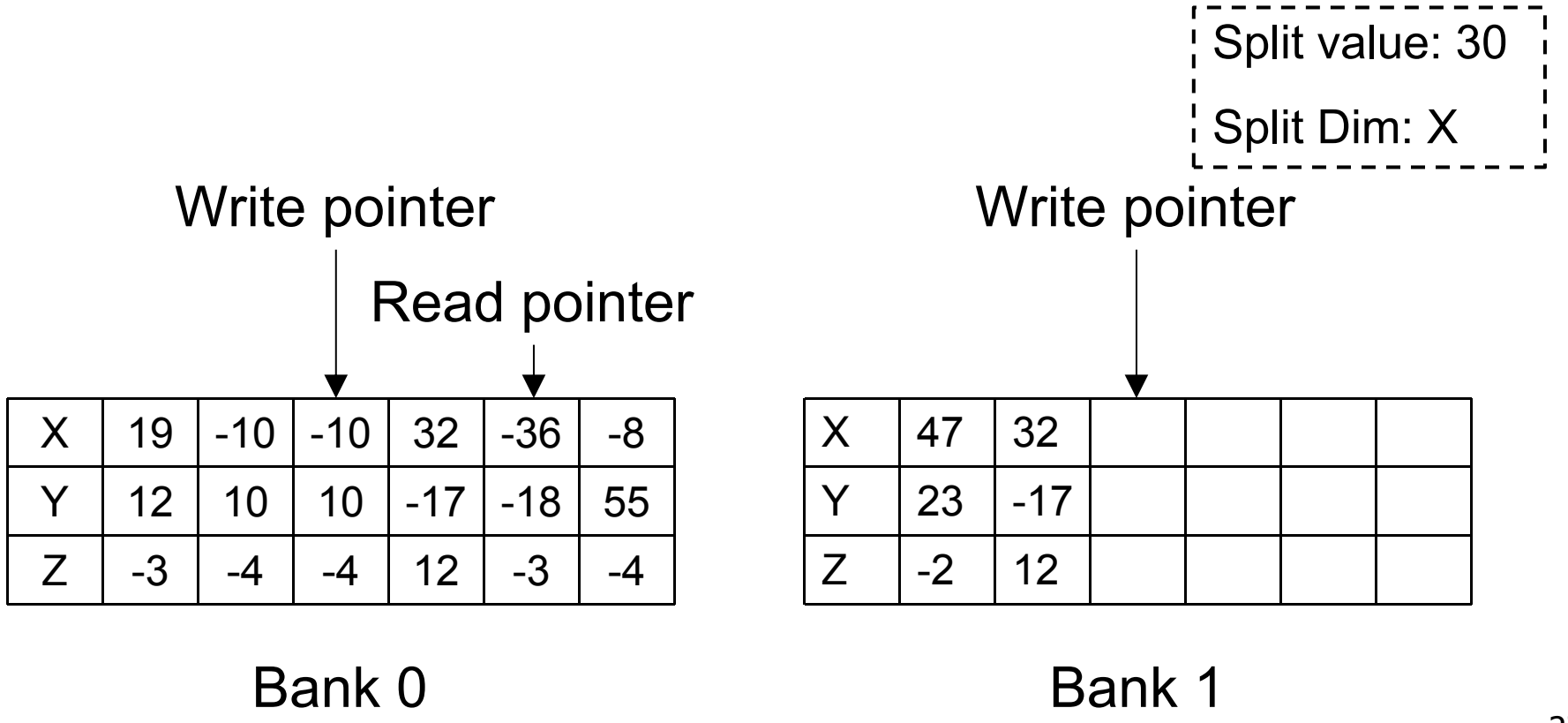
Bucket Splitting in FuseFPS

- Load Bucket data into Point buffer's Bank 0
- Splitting with one read pointer and two write pointers



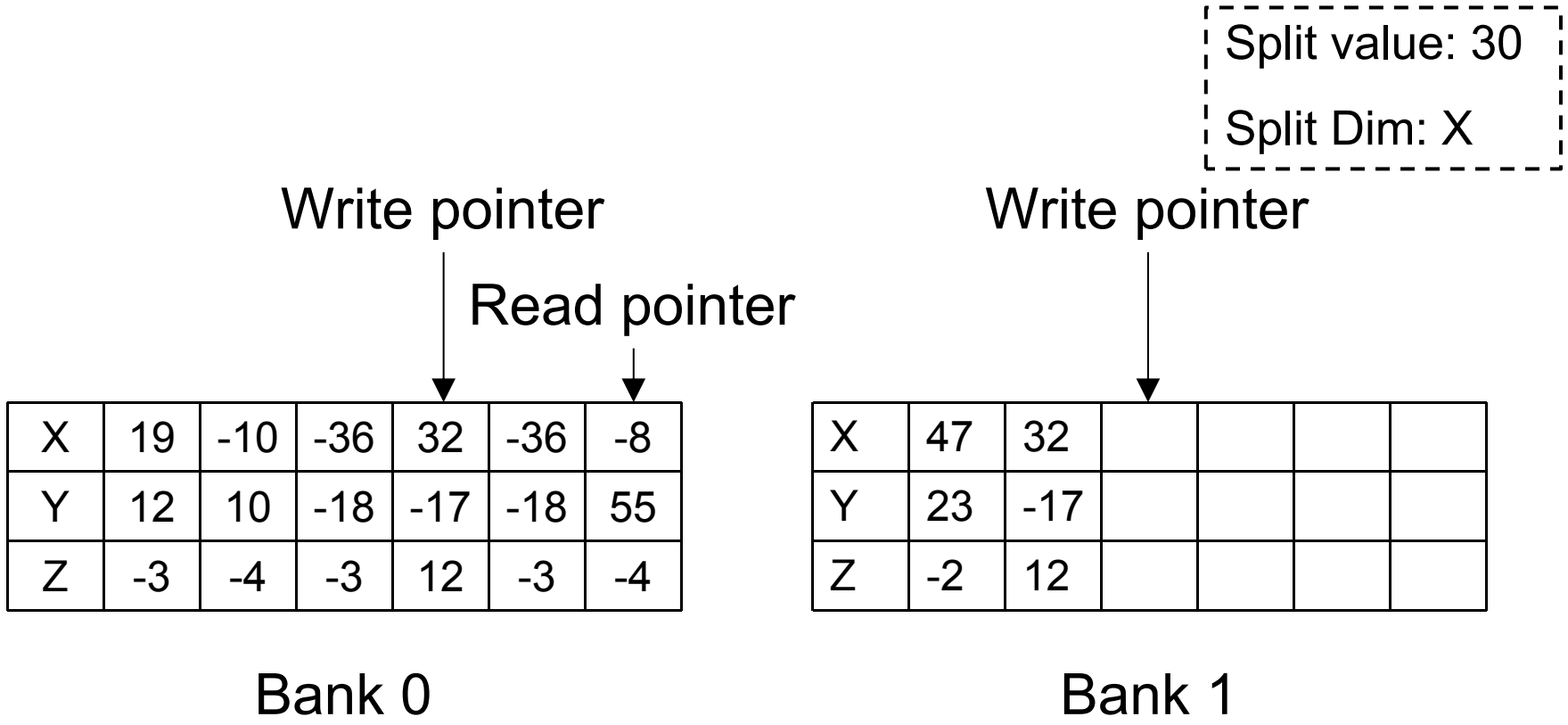
Bucket Splitting in FuseFPS

- Load Bucket data into Point buffer's Bank 0
- Splitting with one read pointer and two write pointers



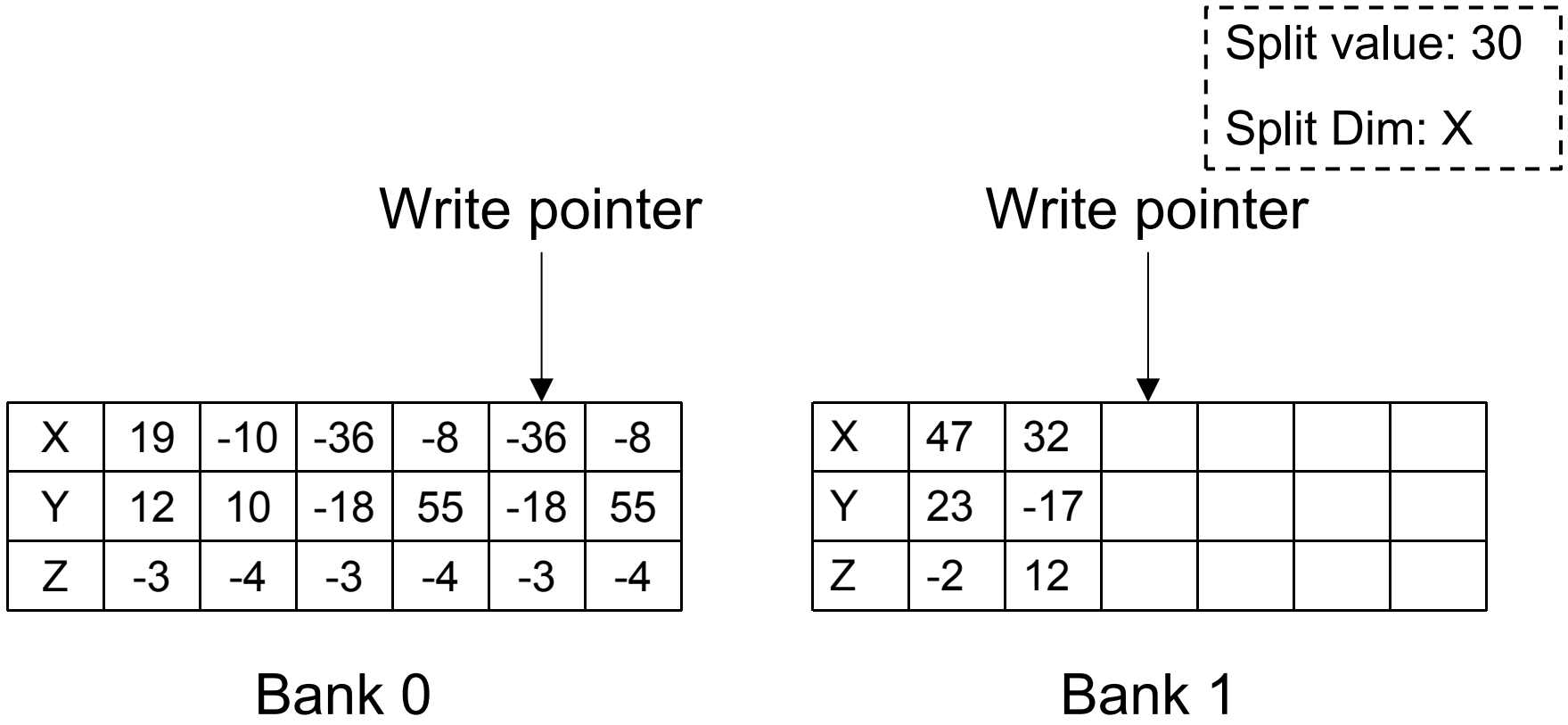
Bucket Splitting in FuseFPS

- Load Bucket data into Point buffer's Bank 0
- Splitting with one read pointer and two write pointers



Bucket Splitting in FuseFPS

- Load Bucket data into Point buffer's Bank 0
- Splitting with one read pointer and two write pointers



Evaluation

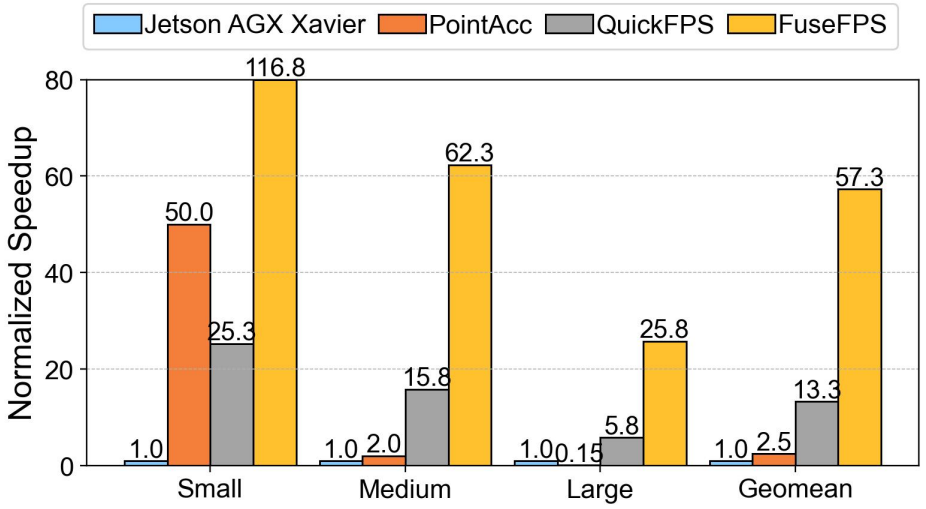
Methodology

- Performance evaluation
  - FuseFPS
  - GPU version of bucket-based FPS for Jetson AGX Xavier
  - PointAcc [Lin et al., MICRO 2021]

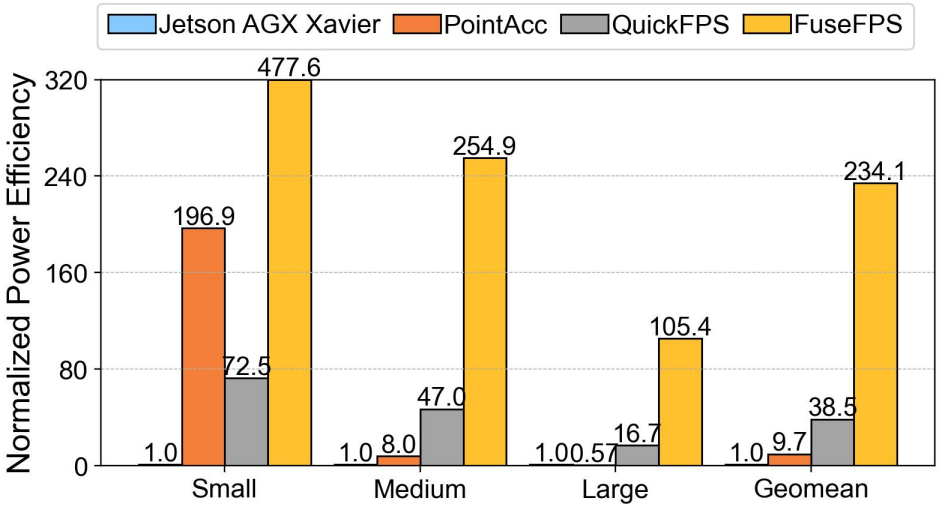
Workload	Point size	Sample rate	Dataset
Small	4.0×10^3	25%	S3DIS
Medium	1.6×10^4	25%	KITTI
Large	1.2×10^5	25%	SemanticKITTI

  - Design Compiler and DRAMsim3 for FuseFPS, PointACC, QuickFPS
  - PowerEstimator for Jetson AGX Xavier

Speedup and Power Efficiency



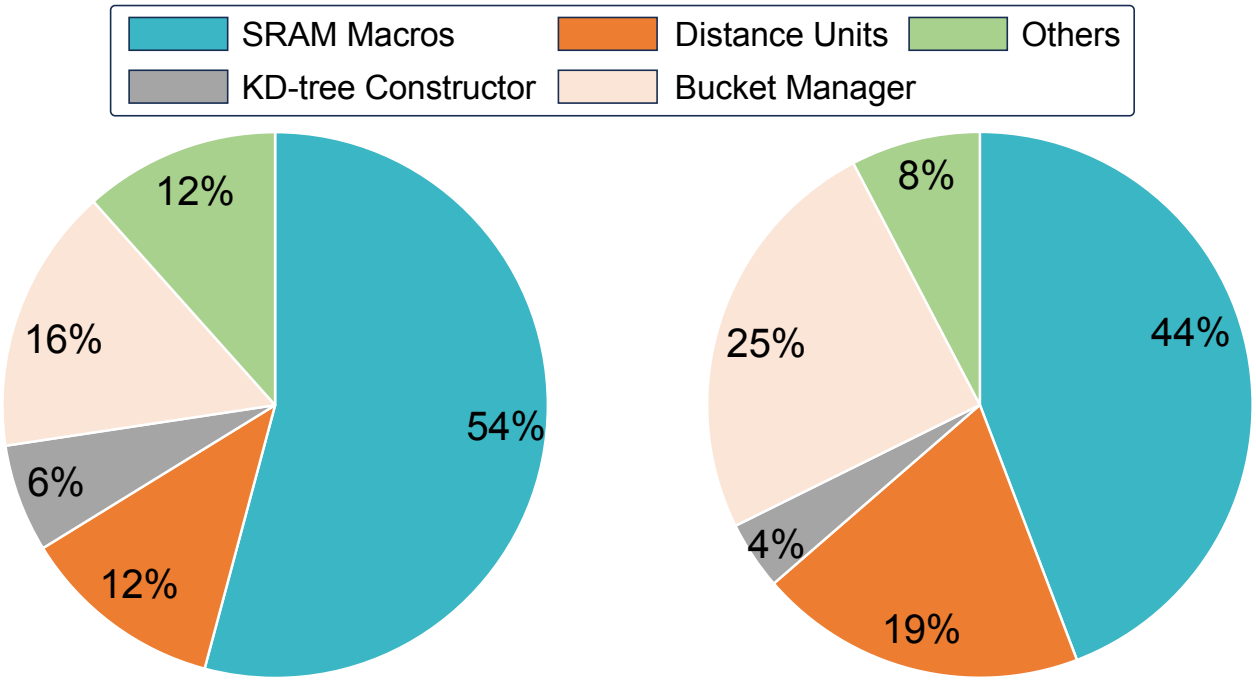
Speedup



Power Efficiency

FuseFPS, on average, achieves 6.08X power efficiency improvement, compared to QuickFPS and 24.13X, compared to PointAcc

Area and Energy Breakdown

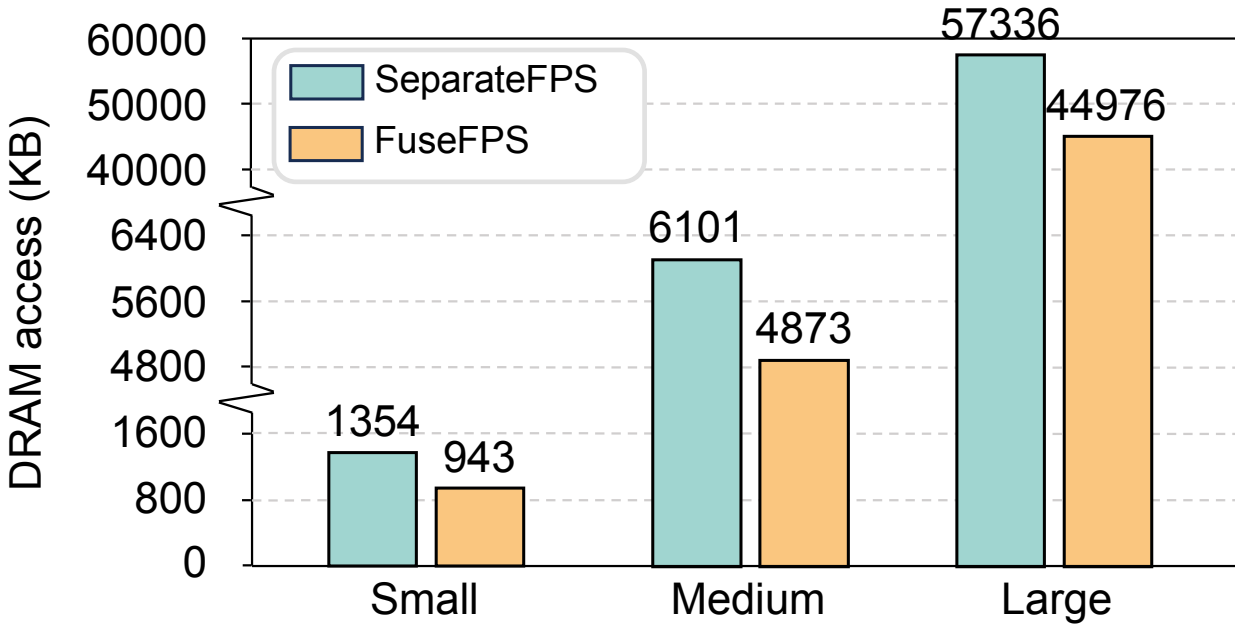


(a) Area breakdown.

(b) On-chip energy breakdown.

The overhead of the KD-tree constructor is low, accounting for 6% of the total area and 4% of on-chip energy

Sensitivity on Fusion



SeparateFPS

A hardware accelerator that separately executes the KD-tree construction and point sampling

Compared to SeparateFPS, FuseFPS, on average, achieves 16.9% DRAM access reduction

Conclusion

- FuseFPS's contribution
  - A hardware-friendly KD-tree construction method
  - Fusing the KD-tree construction stage into the point sampling stage
  - An efficient accelerator
- Presenting Architecture and algorithm co-design for FPS:
  - 1. Sampled-driven KD-tree construction algorithm
    - Reducing memory access
  - 2. Using arithmetic mean value to split the point cloud
    - More hardware-friendly
  - 3. Designing an efficient accelerator
    - can offload the entire FPS kernel at a low hardware cost

Thank You

Questions and Comments Are Welcome