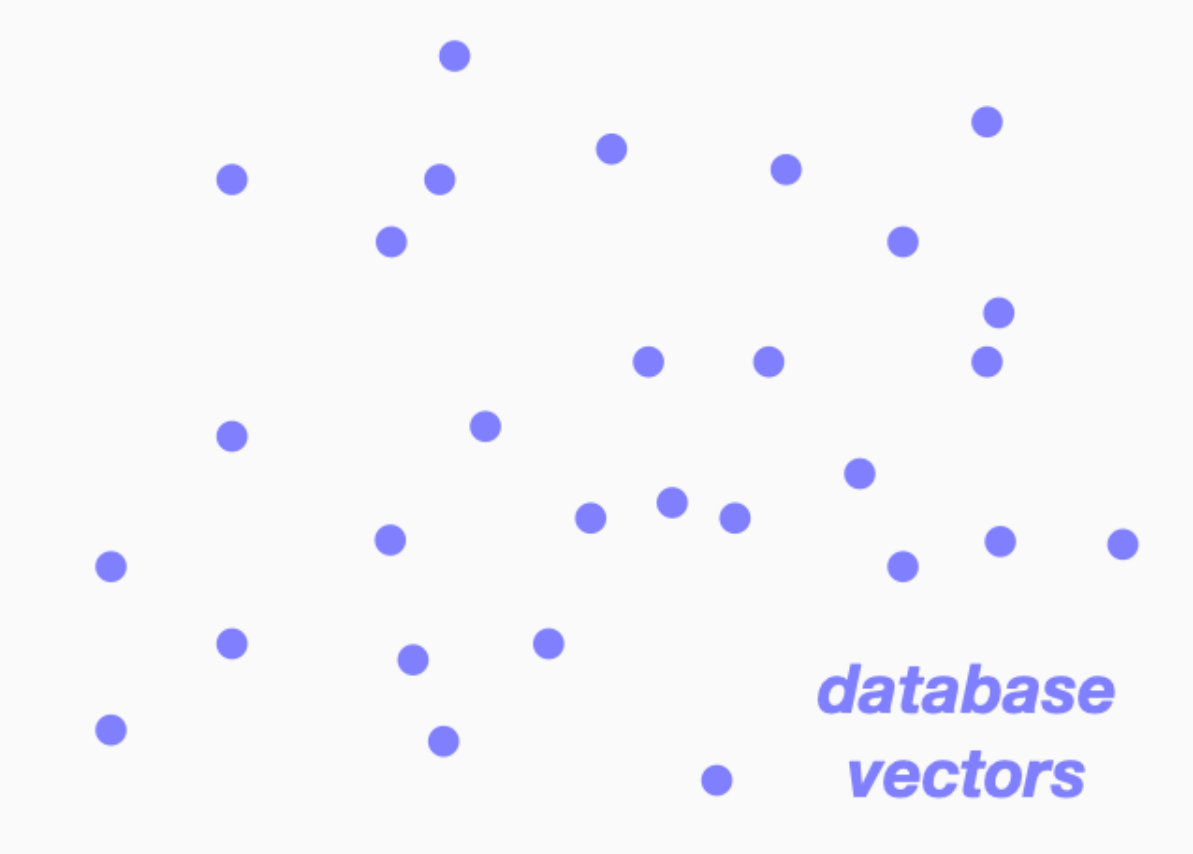
# Co-design Hardware and Algorithm for Vector Search

Wenqi Jiang, Shigang Li, Yu Zhu, Johannes de Fine Licht, Zhenhao He, Runbin Shi, Cedric Renggli,
Shuai Zhang, Theodoros Rekatsina, Torsten Hoefler, and Gustavo Alonso
Department of Computer Science, ETH Zürich

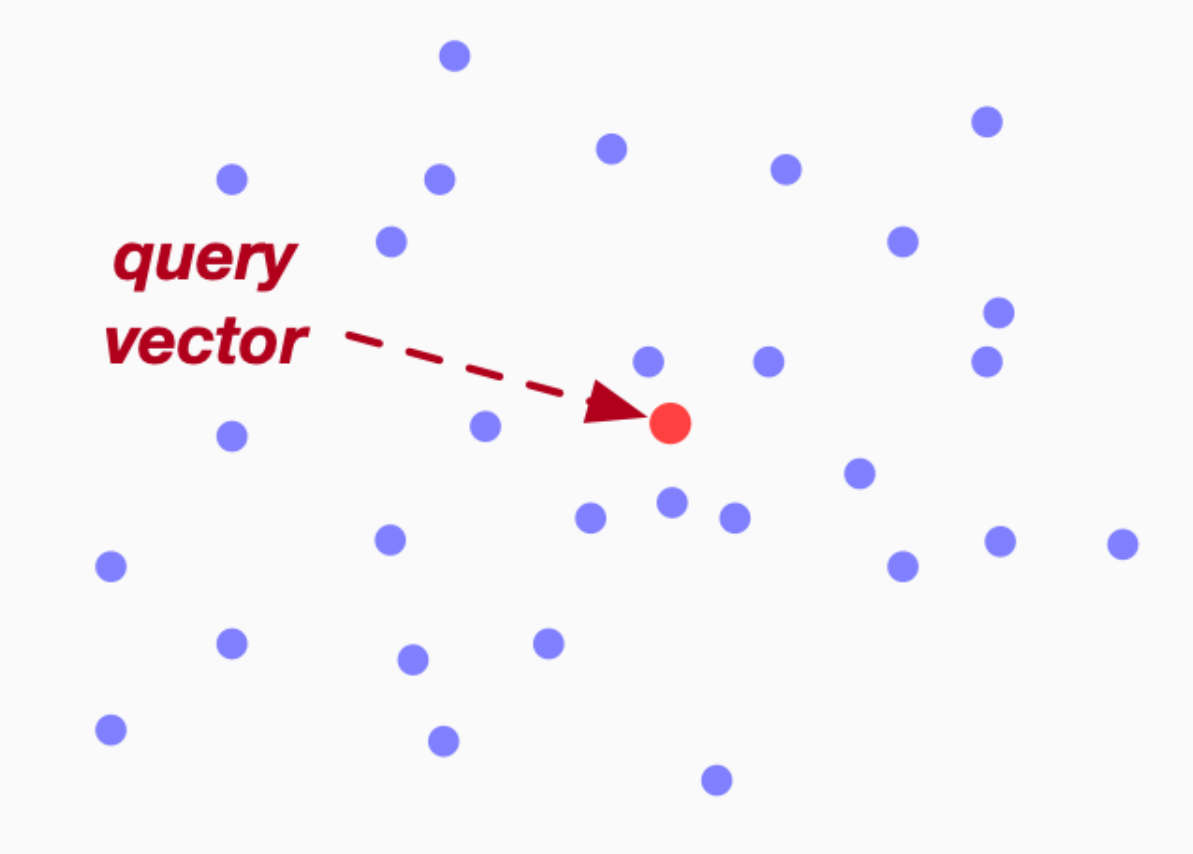
April 9, 2024

Systems@ETH zürich

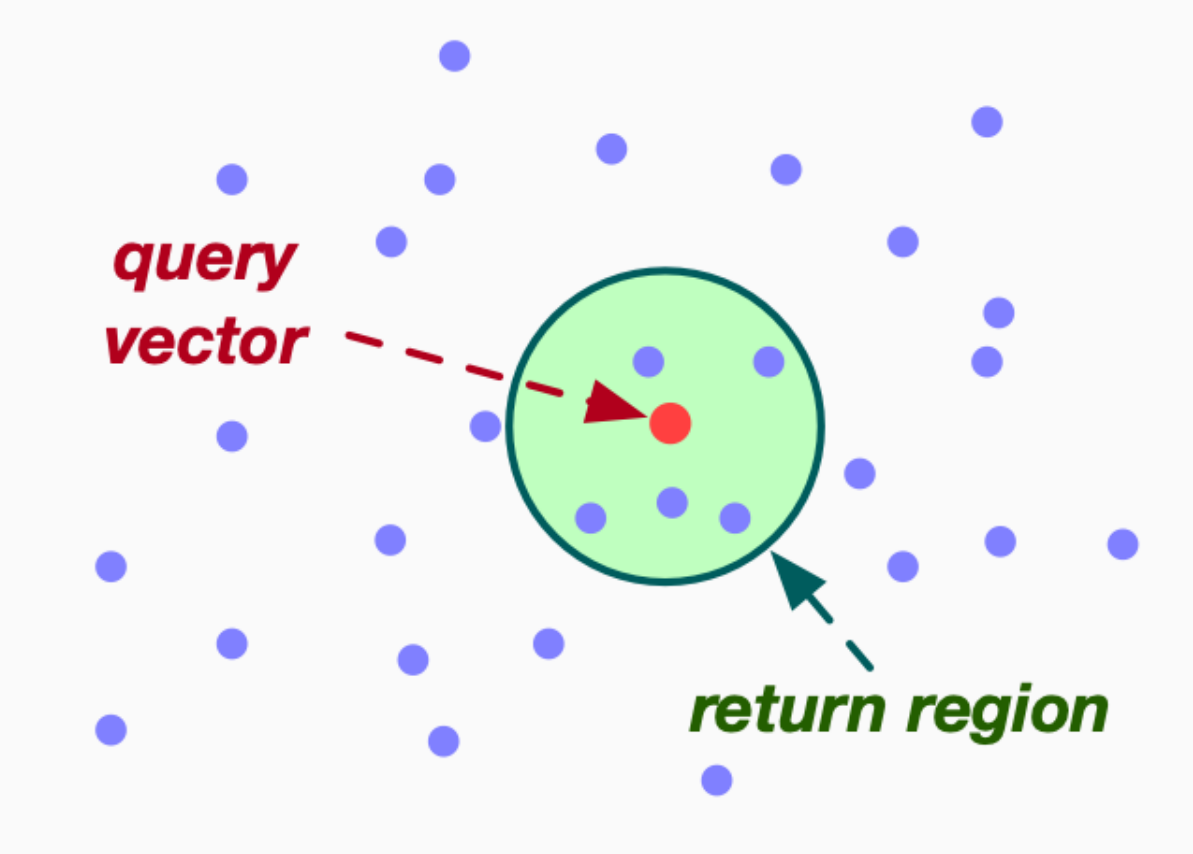
# Vector search: problem definition



# Vector search: problem definition



# Vector search: problem definition



# Vector search in modern AI applications

Search engines 

Recommender systems 


Generative AI 

# Approximate nearest neighbor search (ANNS)

Exact nearest neighbor search is hard to afford

- linearly scan the whole dataset
- full-precision dataset vectors

People use approximate search in practice

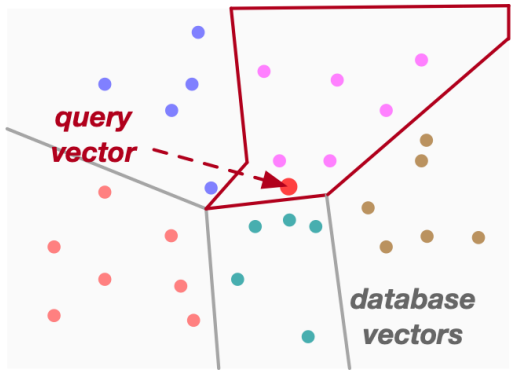
Quality metric: recall

- the fraction of the true nearest neighbors retrieved by the ANNS

# IVF-PQ for large-scale ANNS

Inverted-file (IVF) index

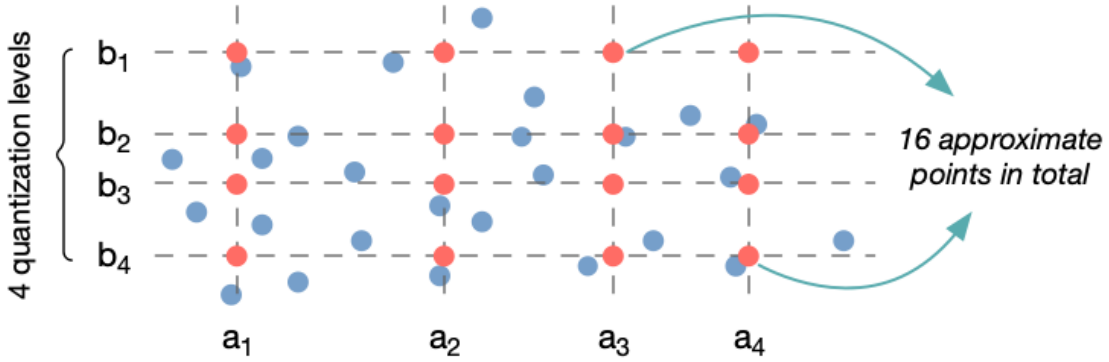
prune the search space



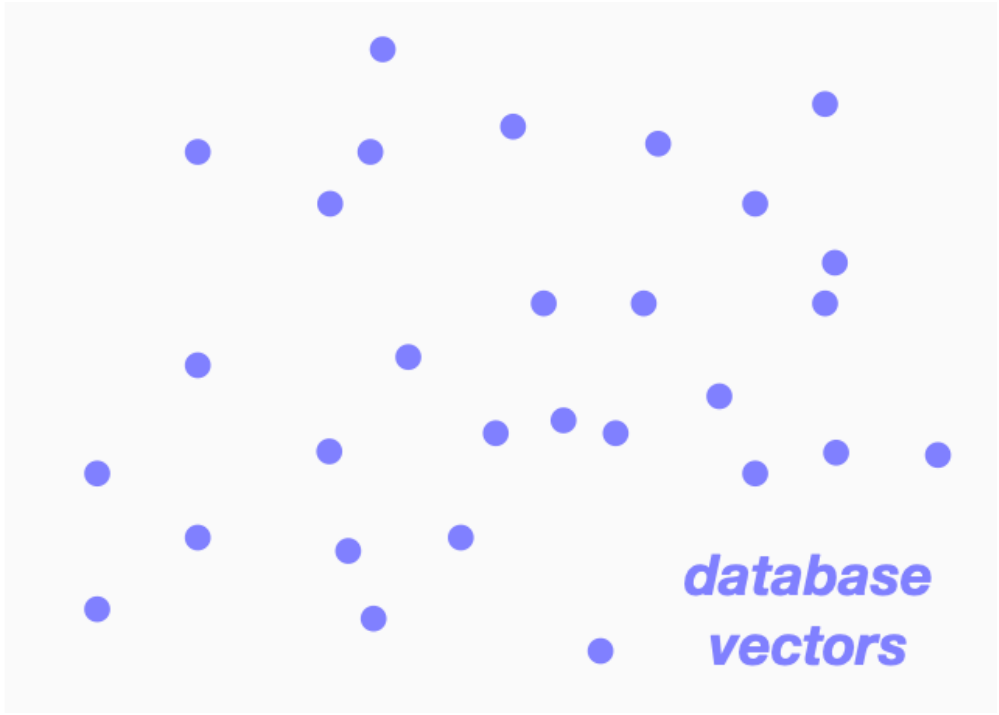

Product quantization (PQ)

quantize database vectors

speedup distance computation

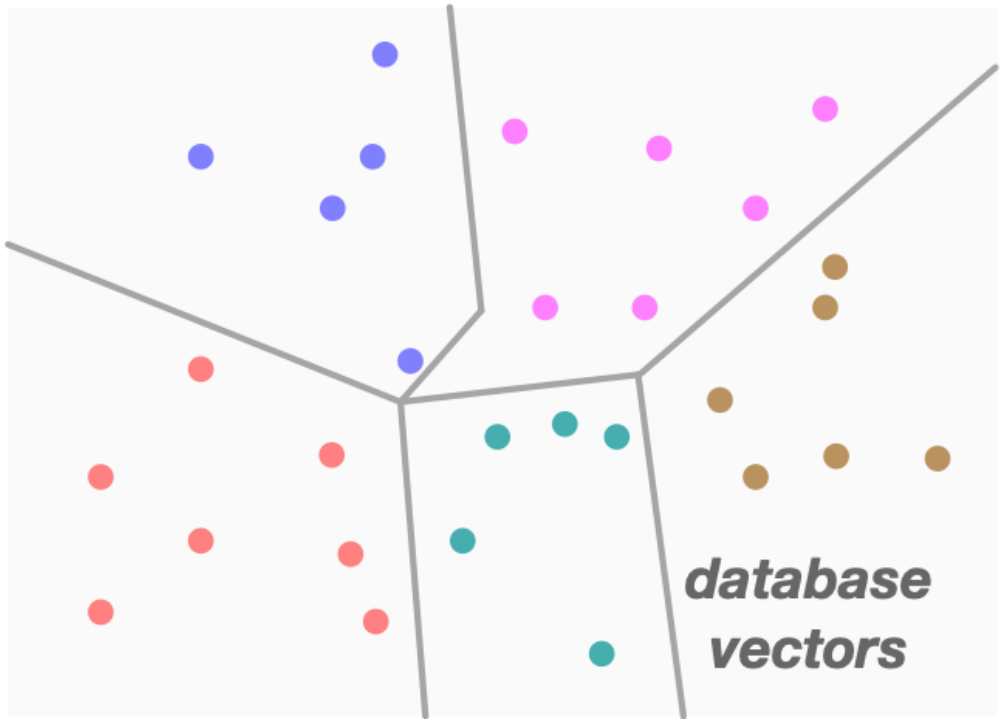


# Inverted-file (IVF) index



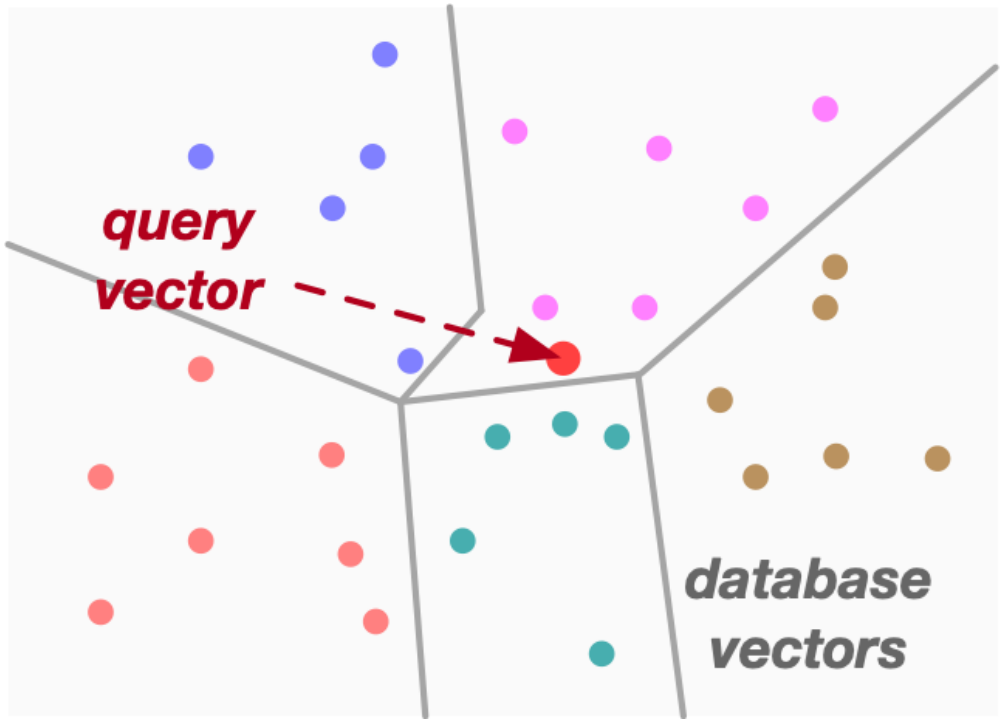

Goal: prune the search space to avoid brute-force scan

# Inverted-file (IVF) index



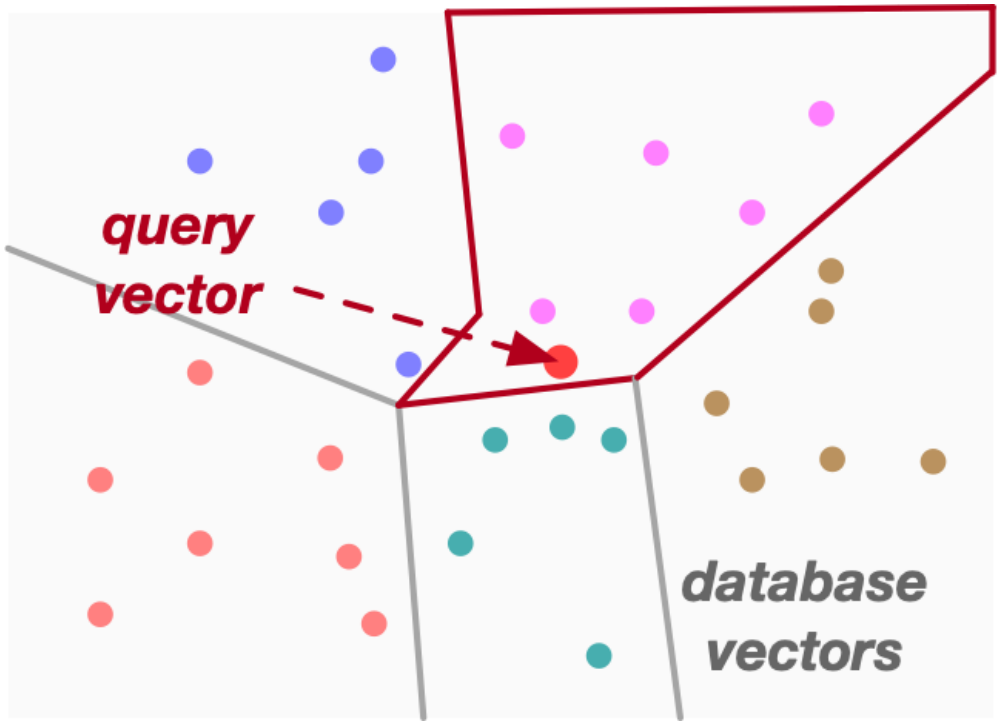

Training: cluster database vectors into IVF lists

# Inverted-file (IVF) index



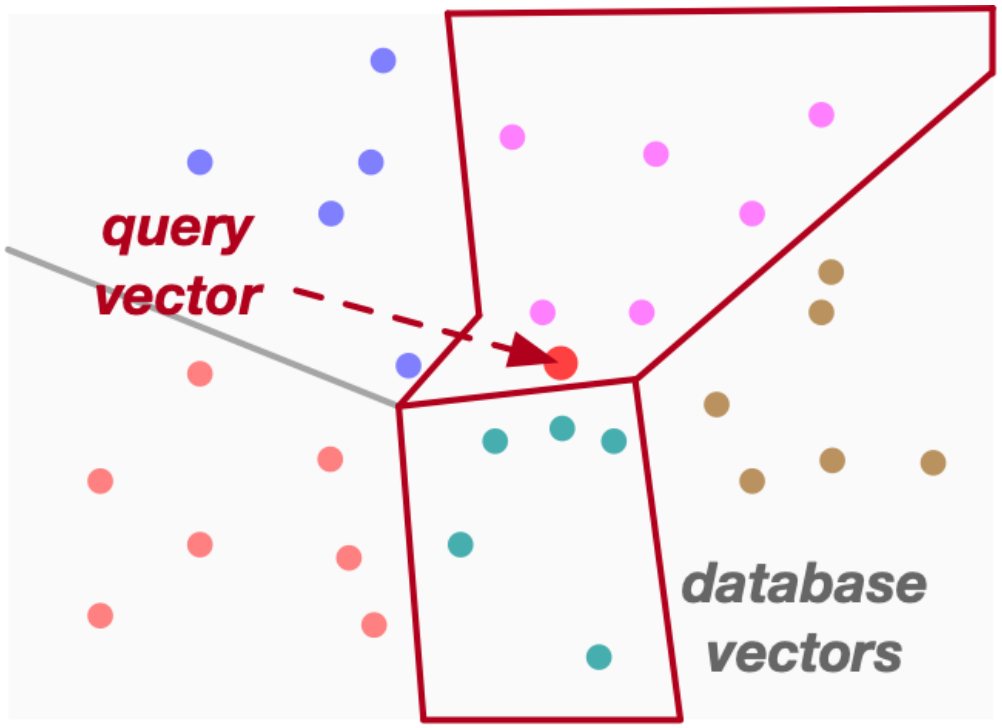

Searching: scan only a subset of IVF lists

# Inverted-file (IVF) index



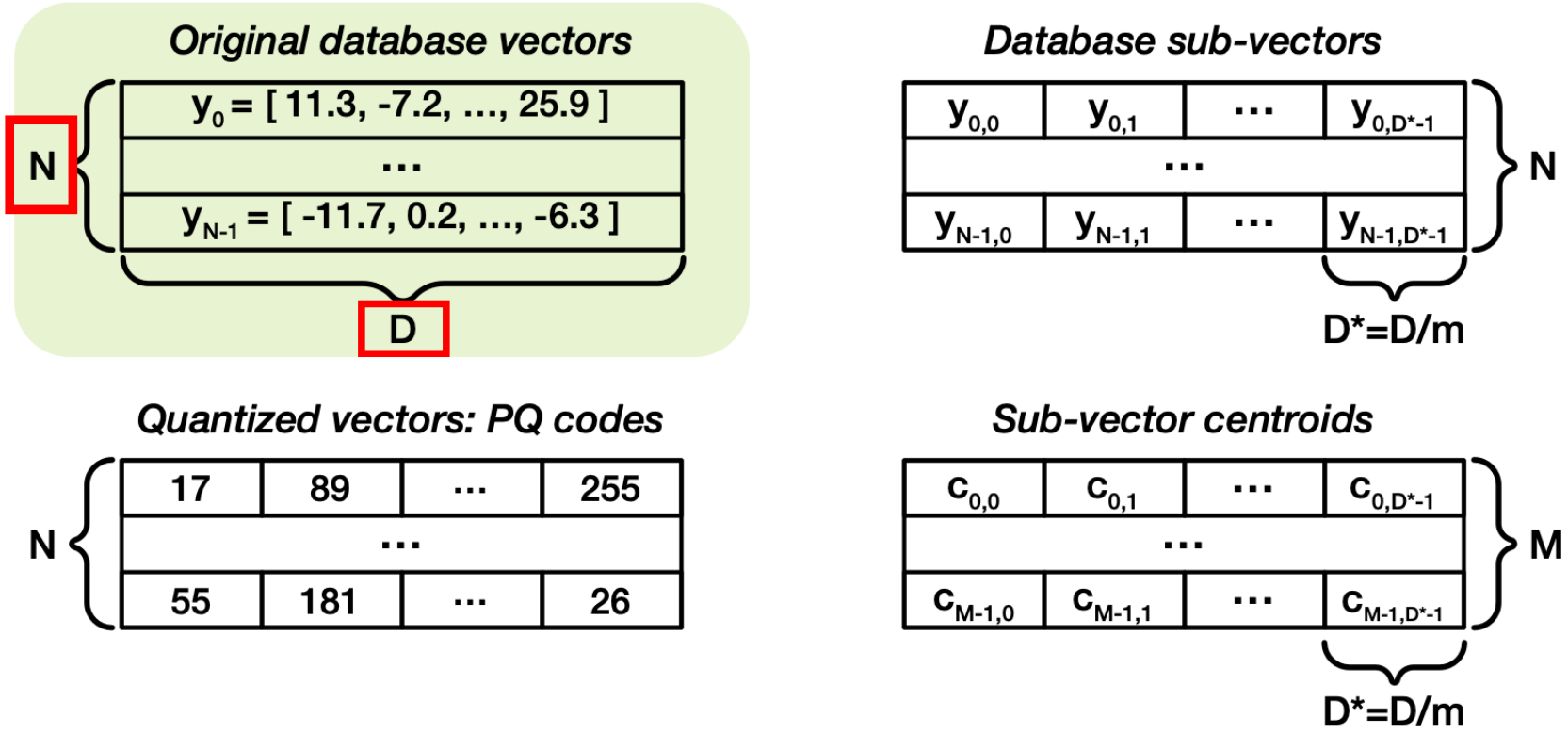

Searching: scan only a subset of IVF lists

# Inverted-file (IVF) index



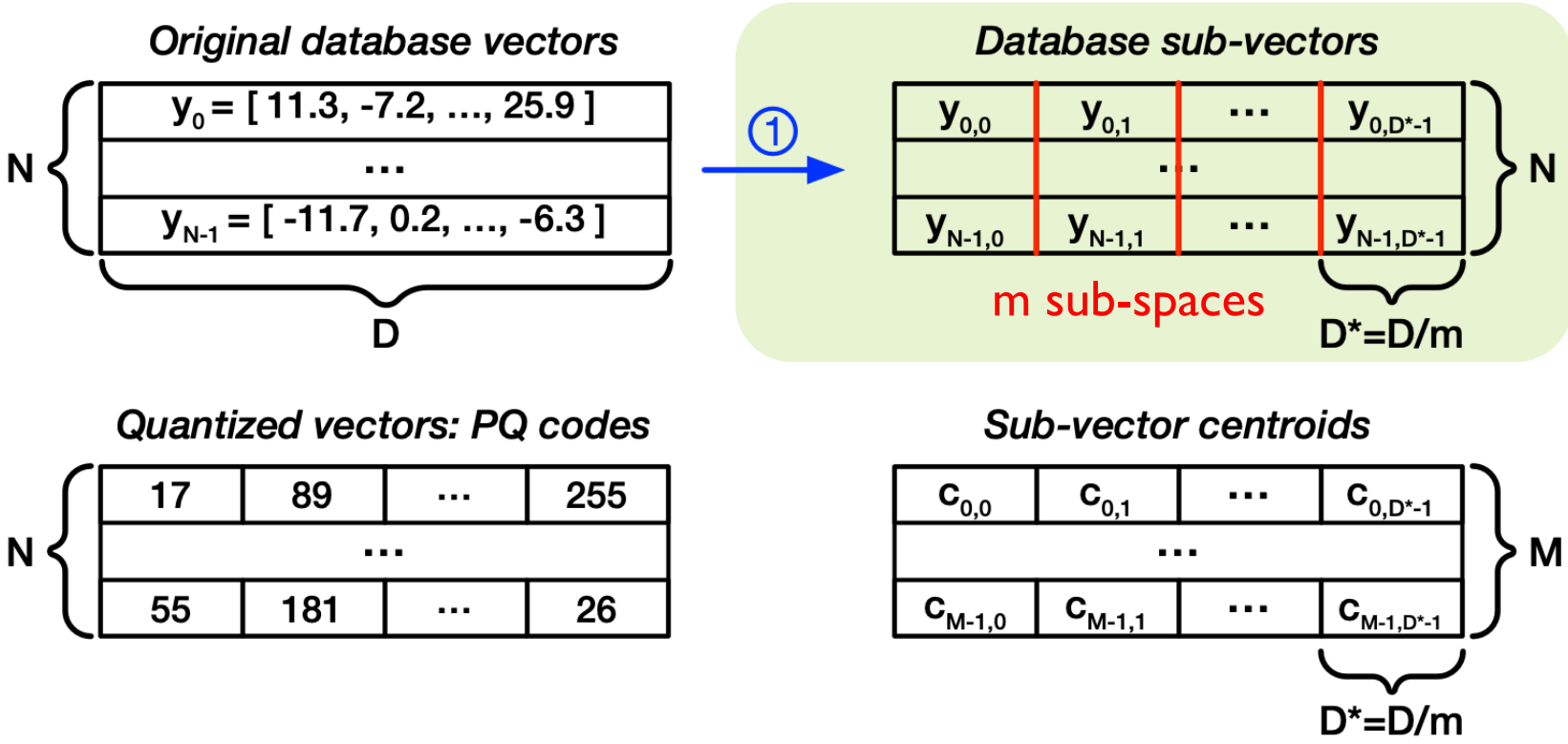

Searching: scan only a subset of IVF lists

# Product quantization (PQ): training



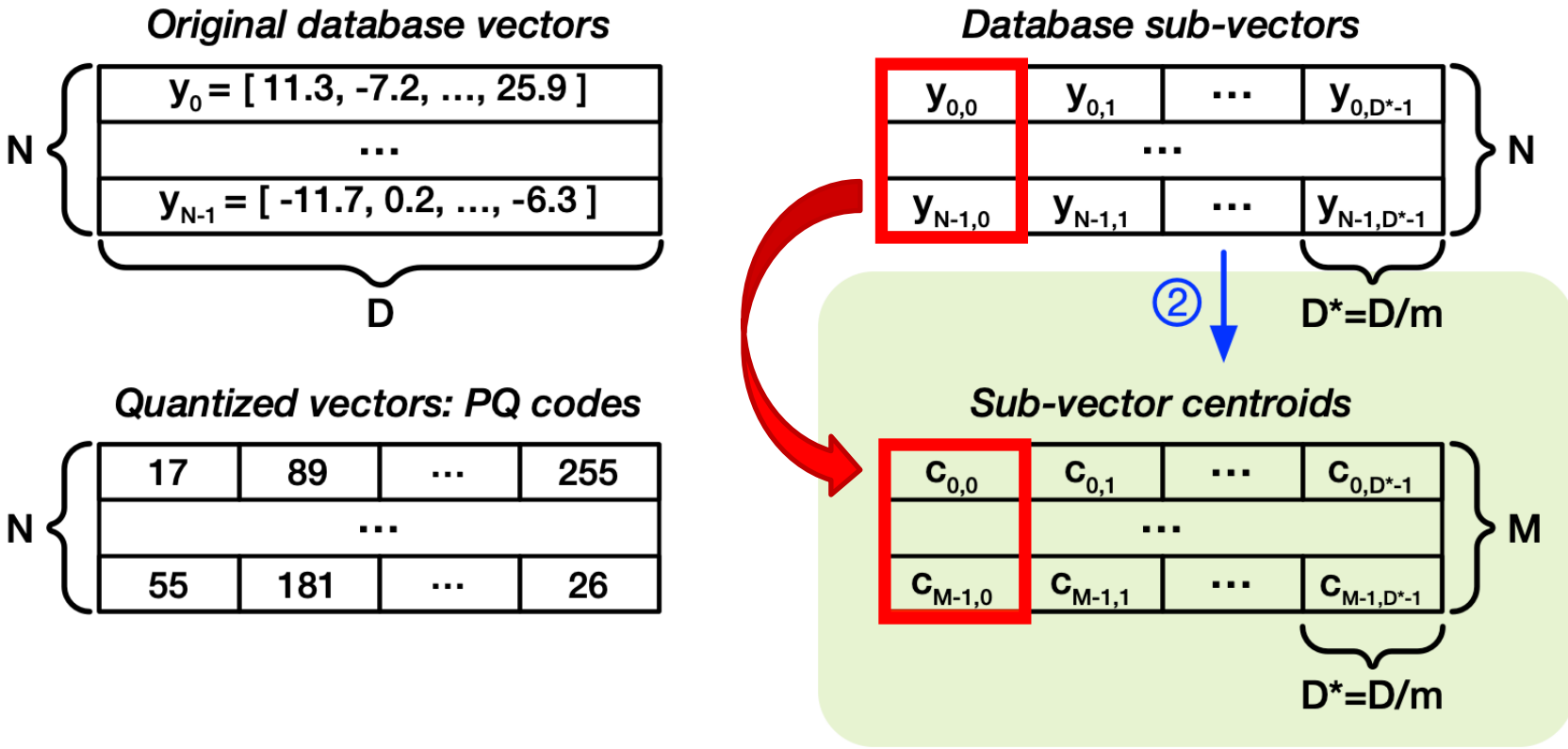

Goal: quantize the vectors to a few bytes of PQ-codes

# Product quantization (PQ): training



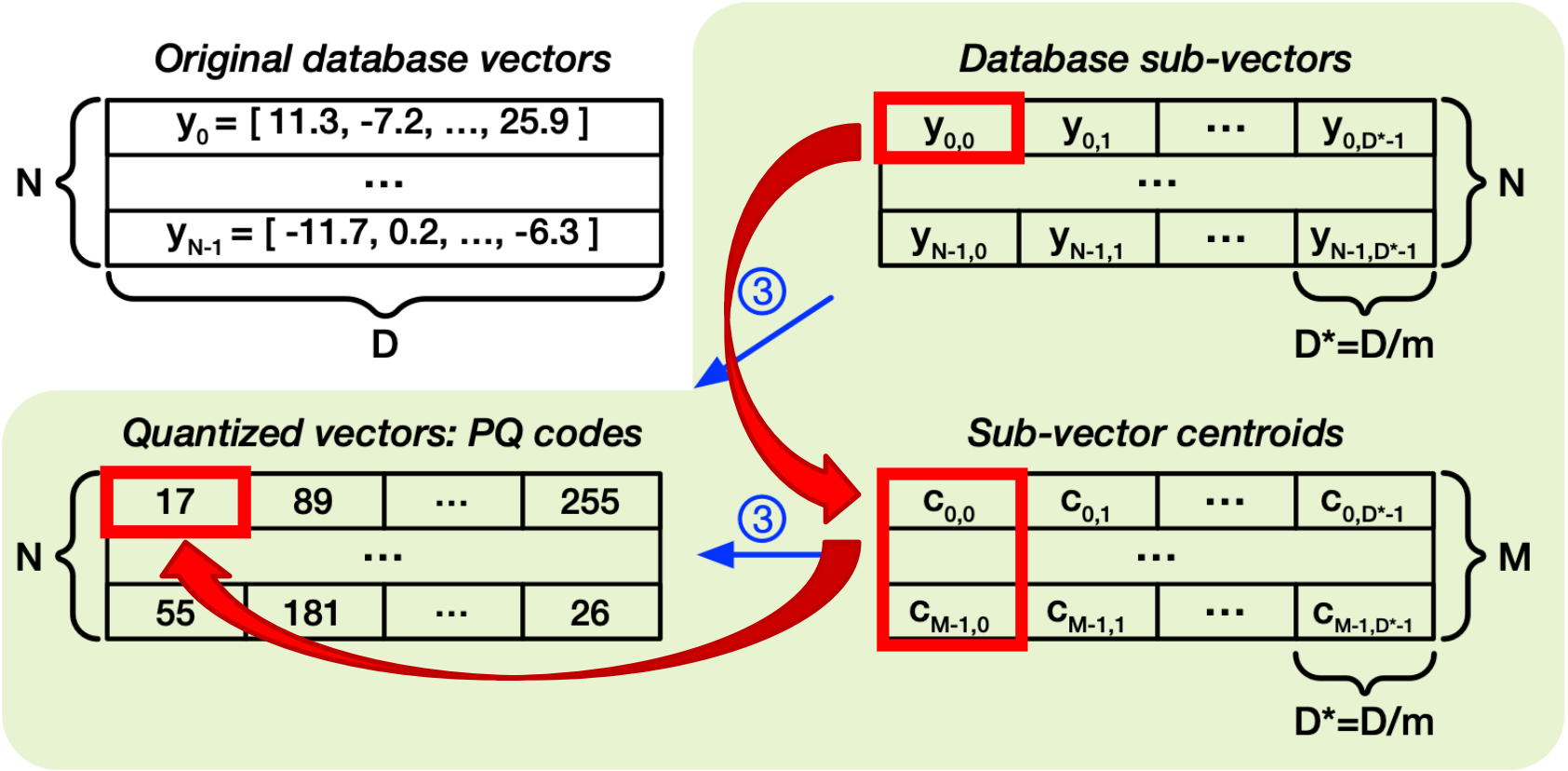

Step 1: divide database vectors into m sub-vectors

# Product quantization (PQ): training



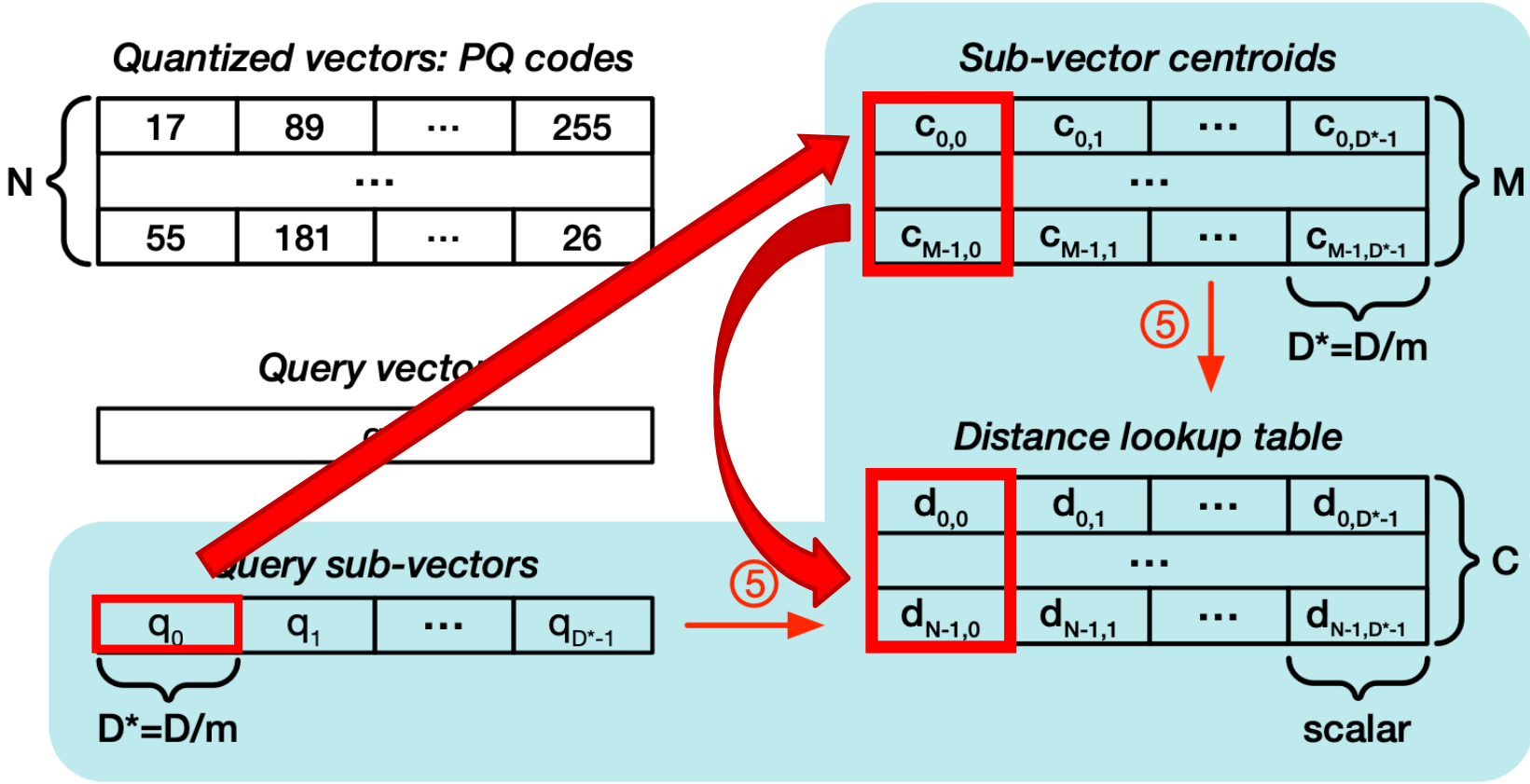

Step 2: run clustering on each sub-vector space, each with M=256 centroids

# Product quantization (PQ): training



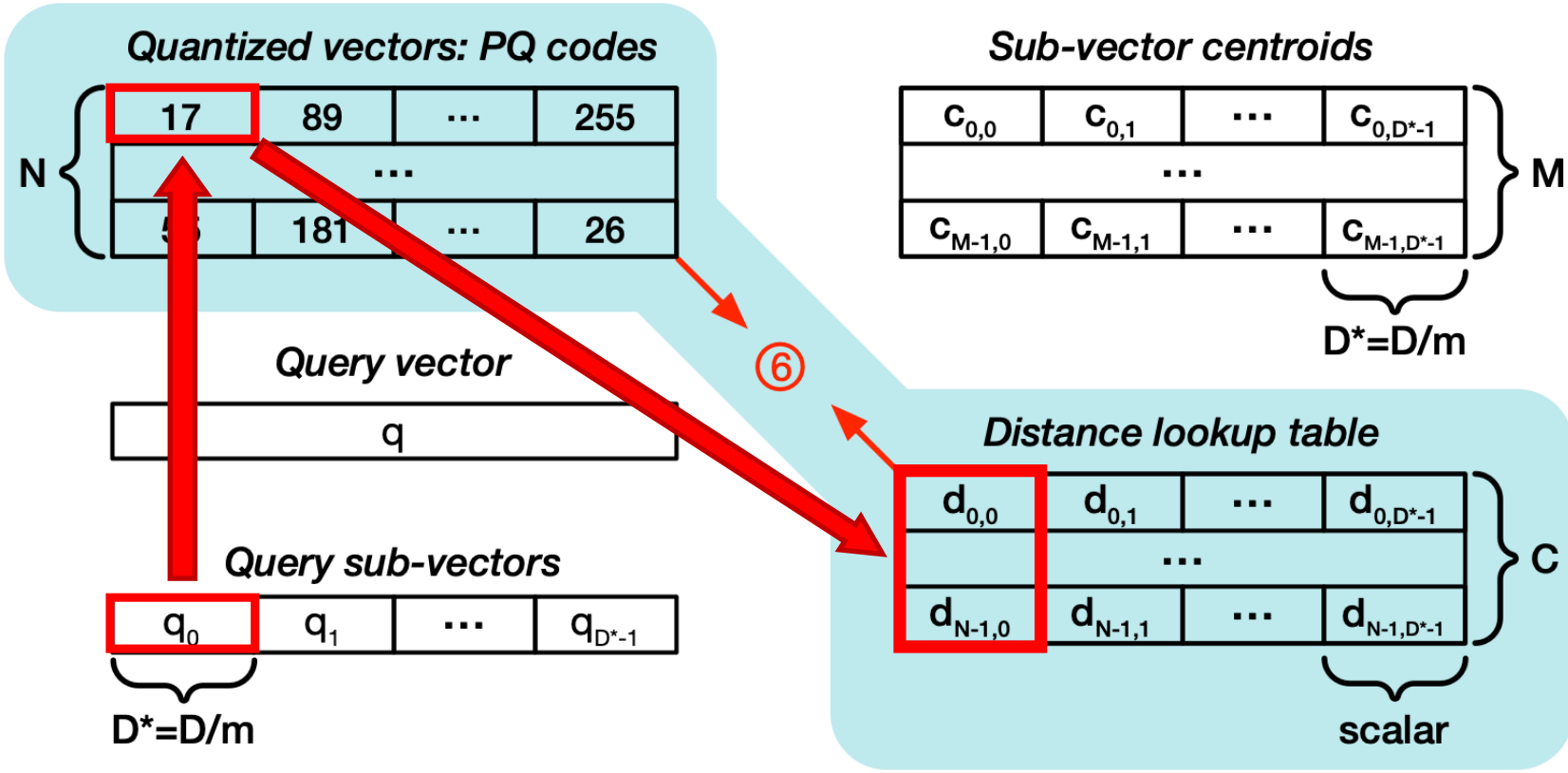

Step 3: approximate each sub-vector by closest centroid ID

# Product quantization (PQ): searching



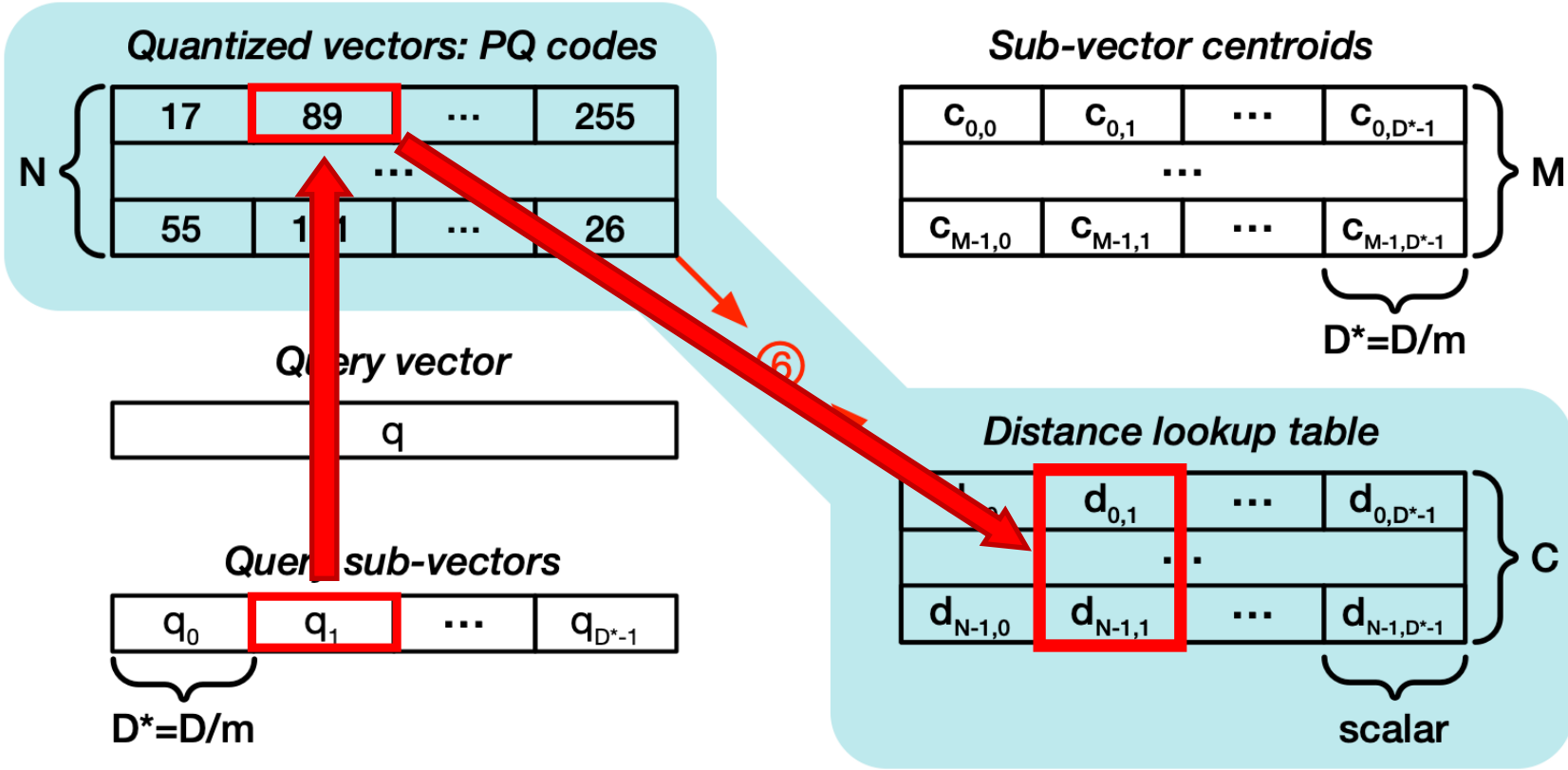

Step 1: construct a distance lookup table

# Product quantization (PQ): searching



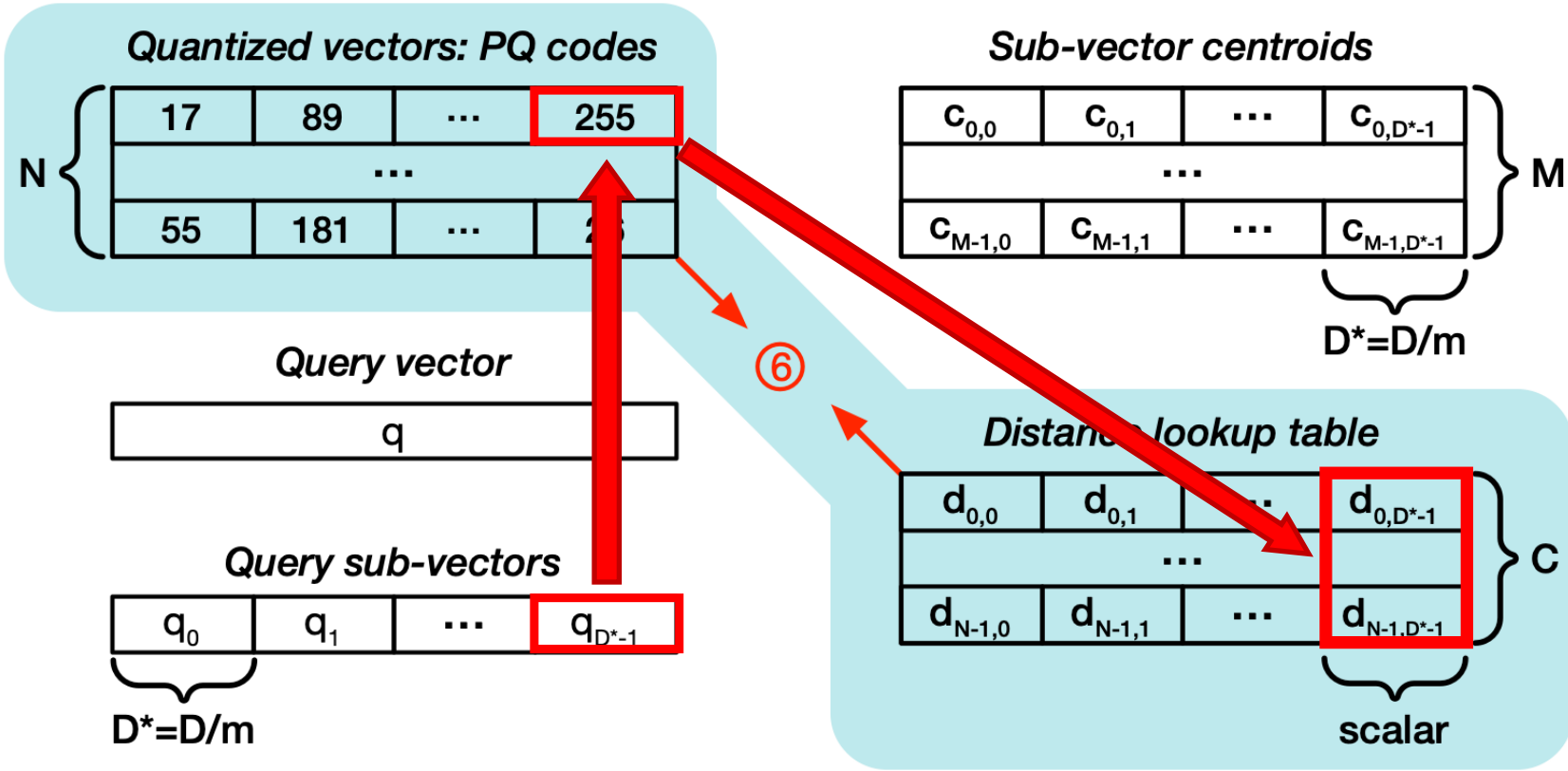

Step 2: compute distances to database vectors by distance lookups

# Product quantization (PQ): searching



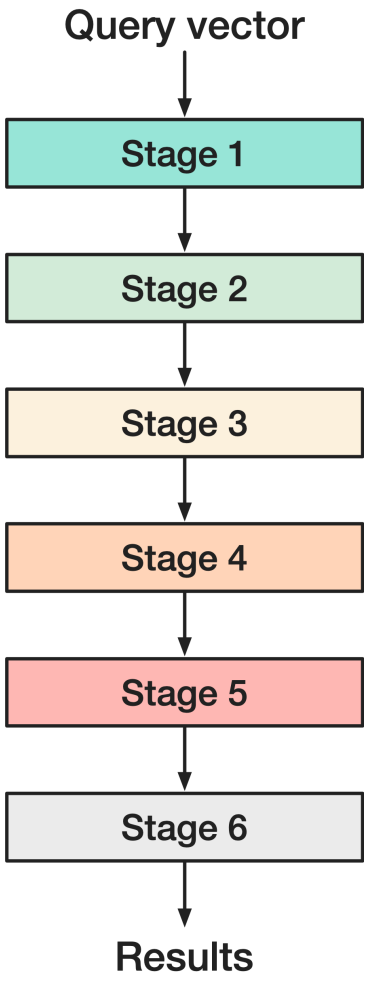

Step 2: compute distances to database vectors by distance lookups

# Product quantization (PQ): searching

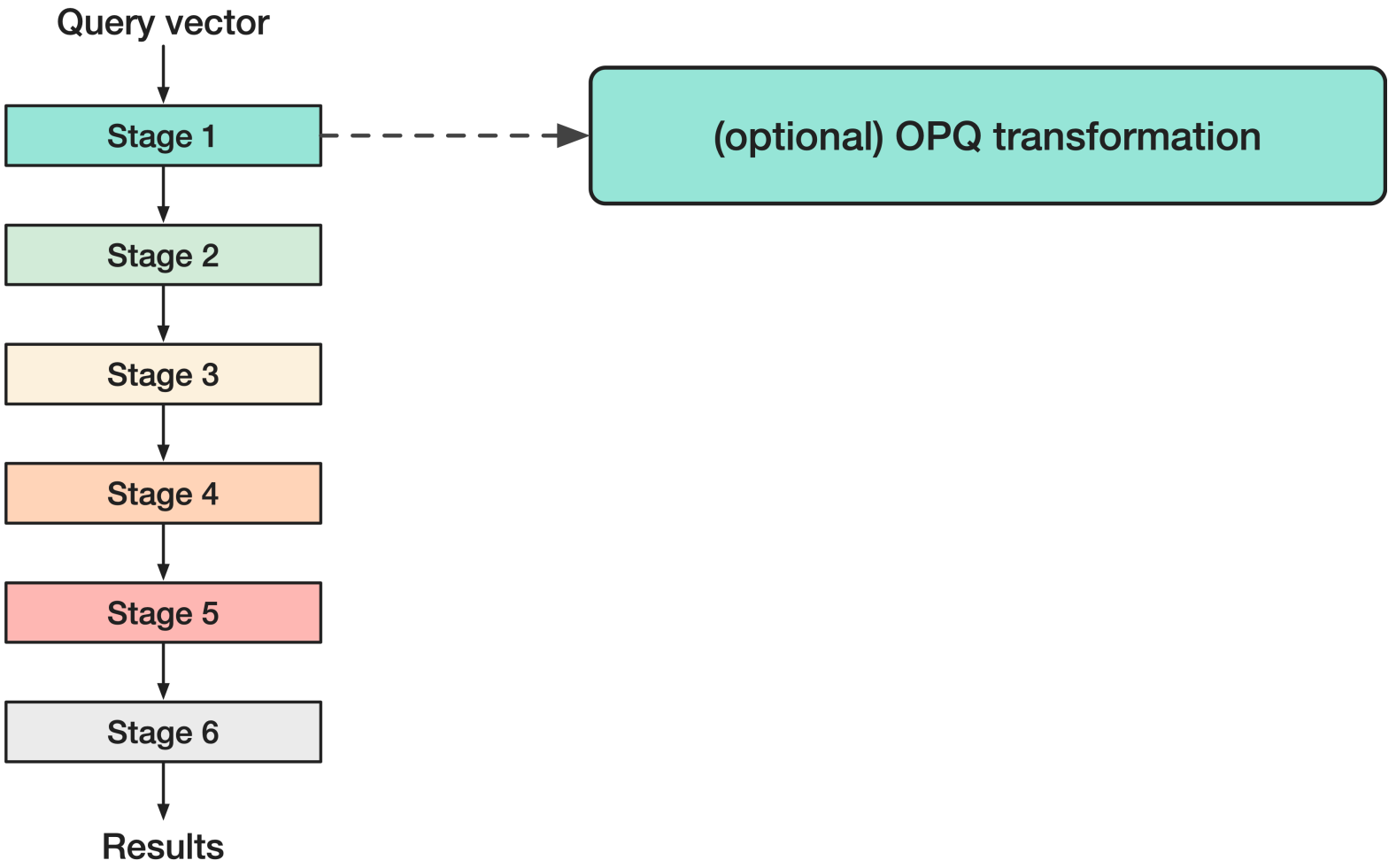



Step 2: compute distances to database vectors by distance lookups

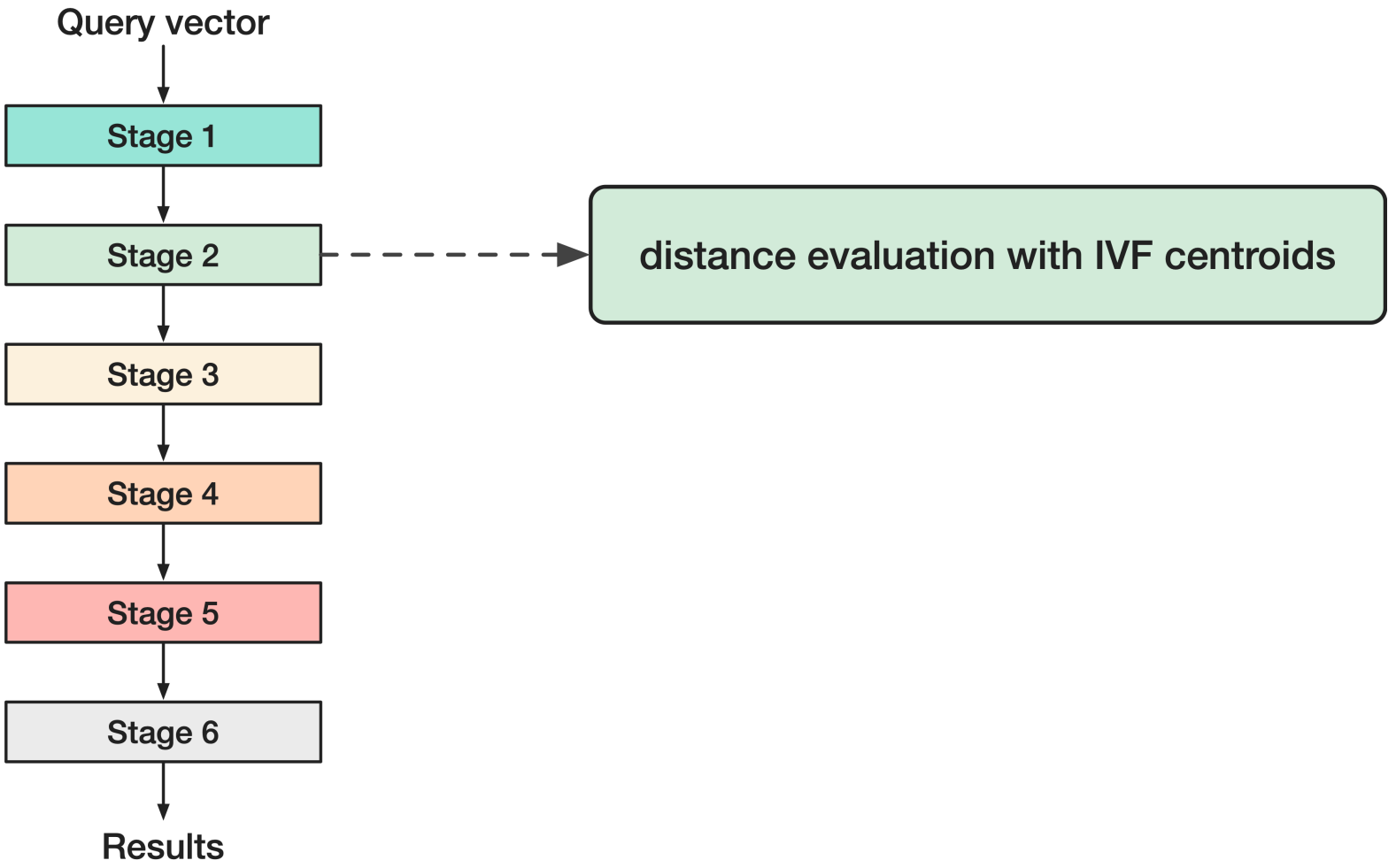
# Putting them together: IVF-PQ



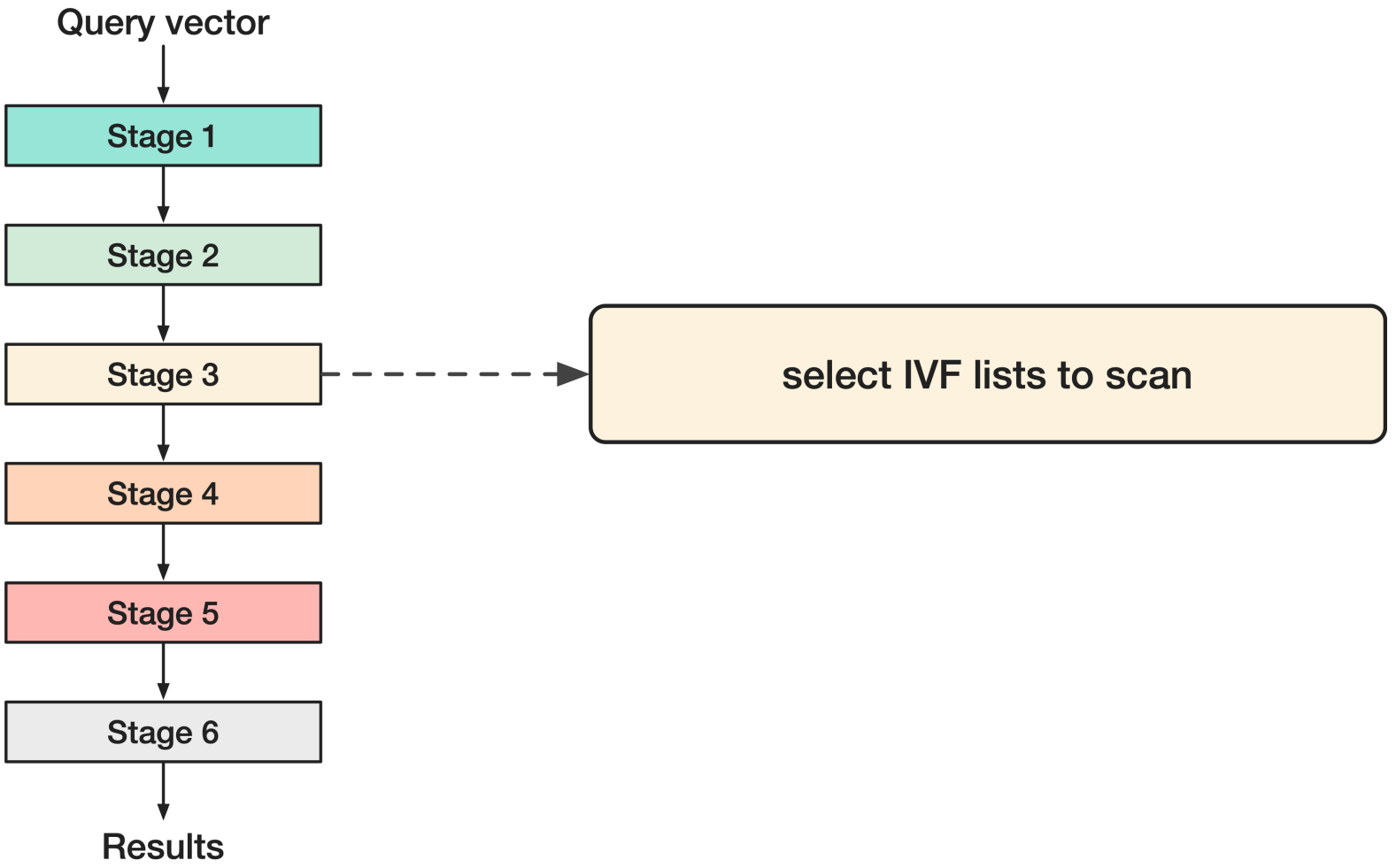
# Putting them together: IVF-PQ



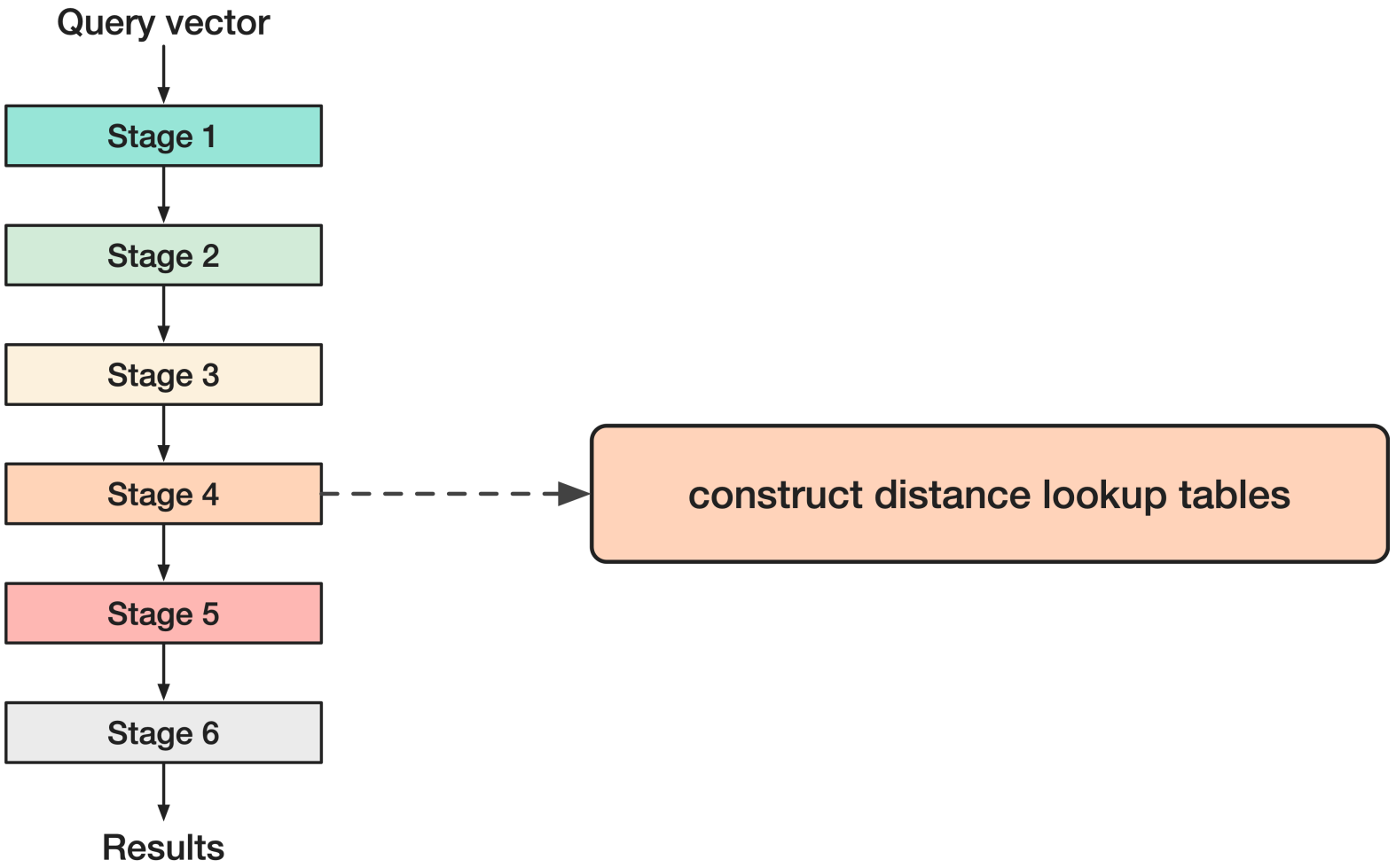
# Putting them together: IVF-PQ



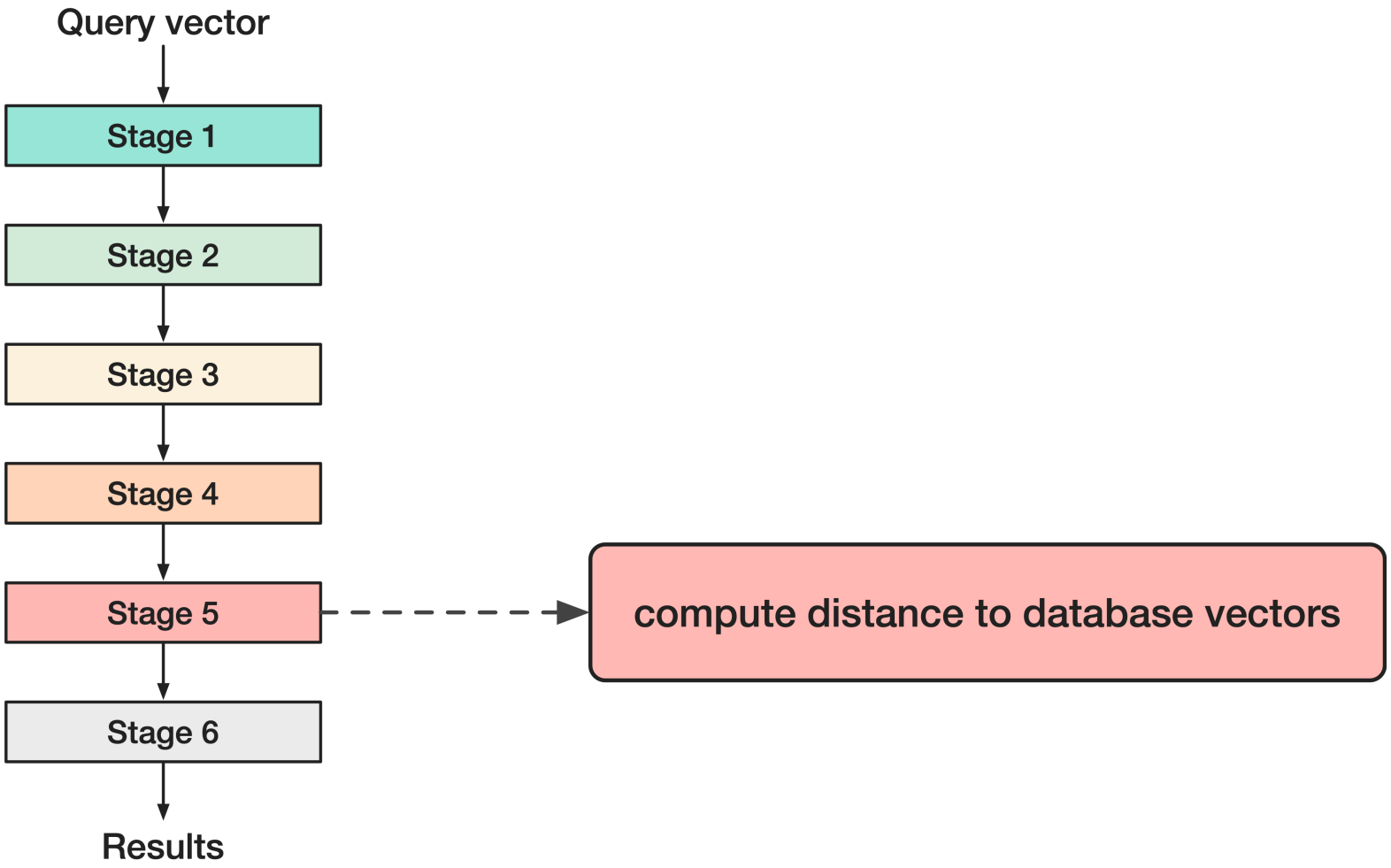
# Putting them together: IVF-PQ



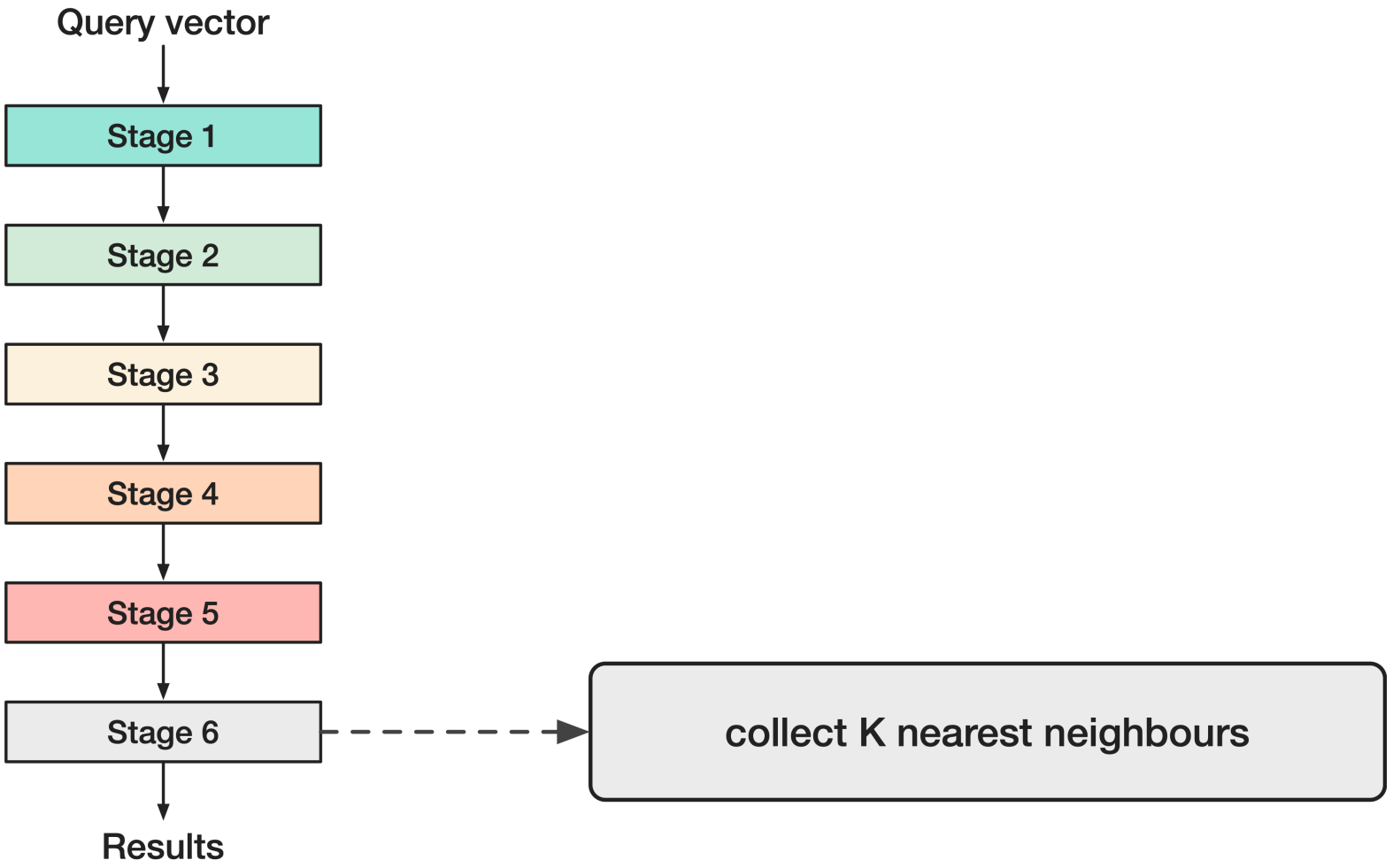
# Putting them together: IVF-PQ



# Putting them together: IVF-PQ

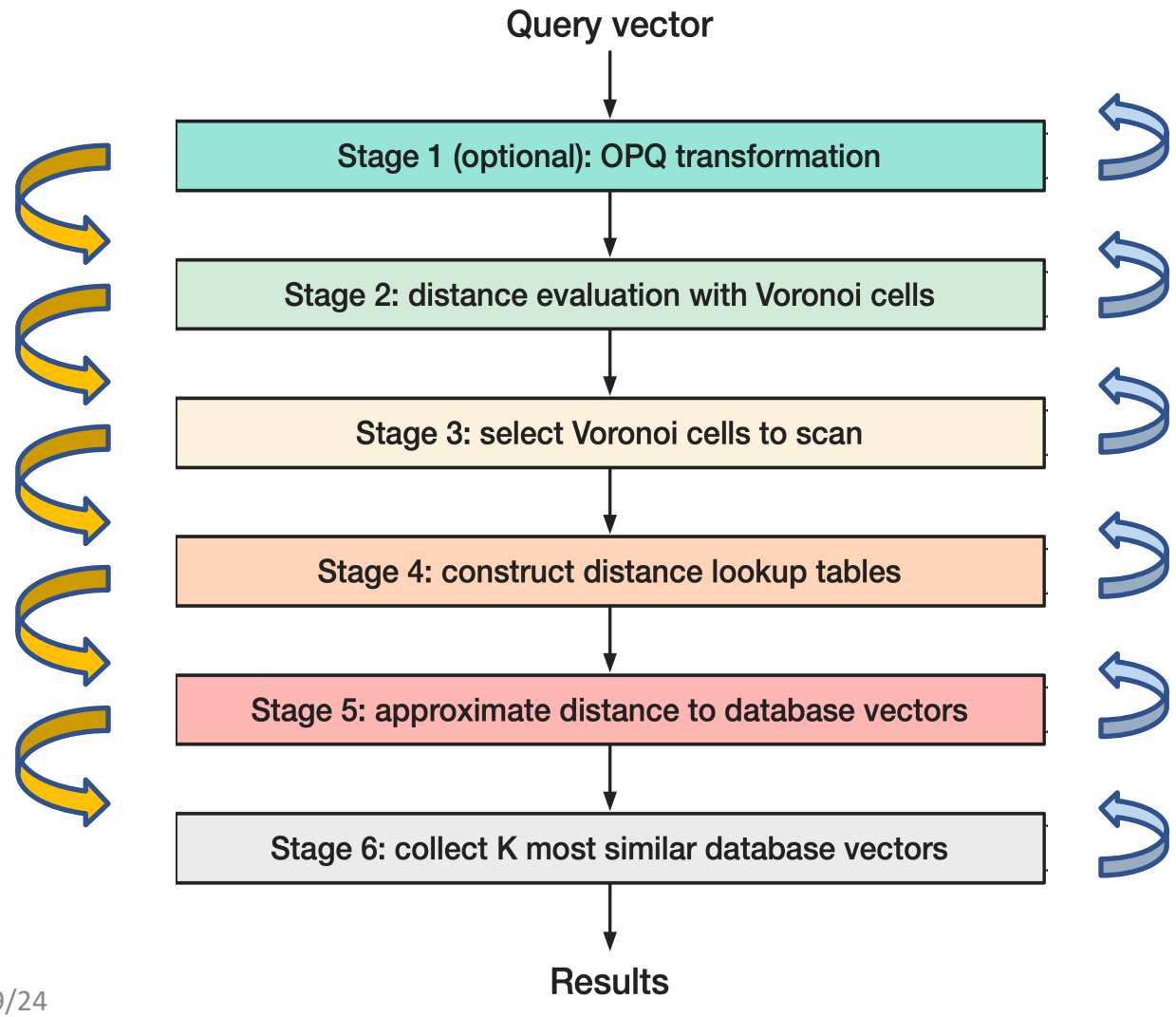


# Putting them together: IVF-PQ



# How to build a specialized hardware accelerator for IVF-PQ-based vector search?

# Complexities in designing IVF-PQ accelerators




Accelerator design involves:

Inter-stage heterogeneity

Intra-stage heterogeneity

Resource allocation

# Complexities in designing IVF-PQ accelerators

Should identify the bottleneck stage first

But there are many parameters in IVF-PQ…

*nlist*: the number of clusters in the index

*nprobe*: the number of clusters to visit per search

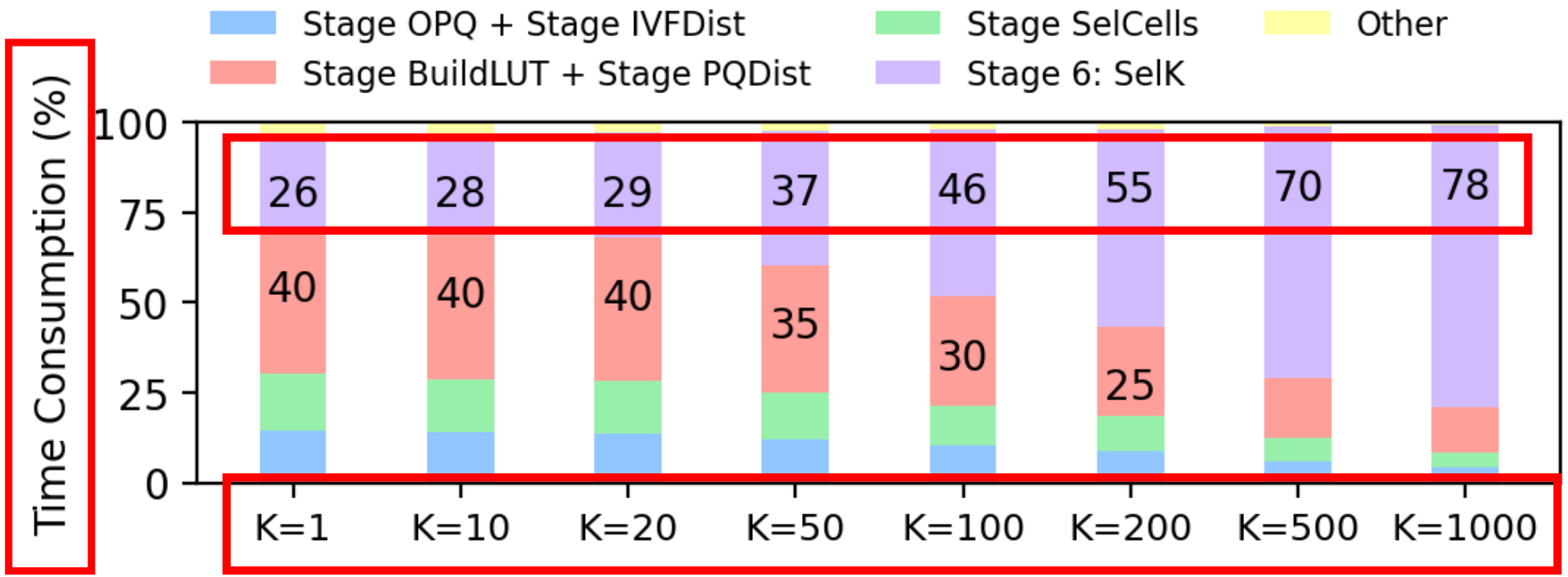
*K*: the number of results to return per query

…

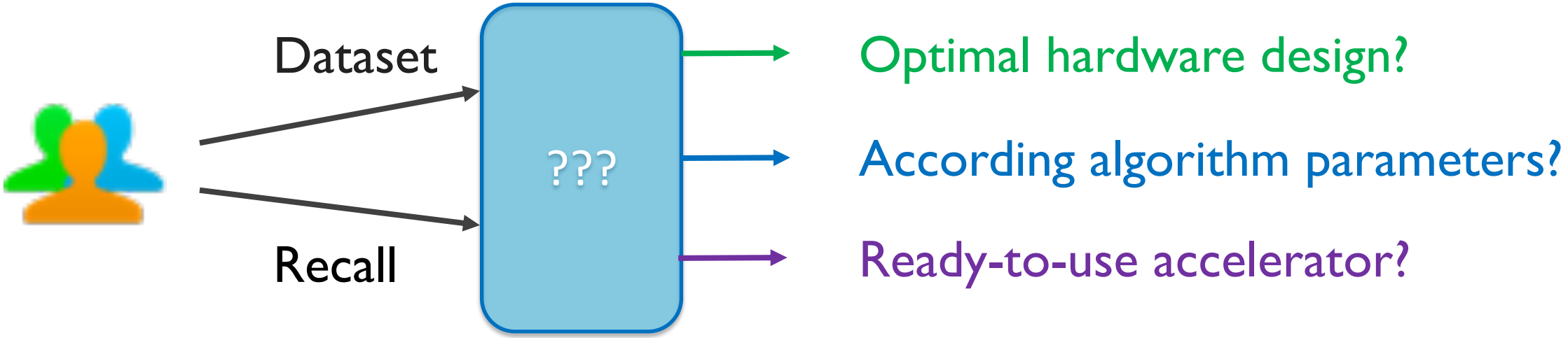
All these parameters can change bottlenecks dramatically!

# The effect of K on performance bottlenecks

*K:* the number of results to return per query



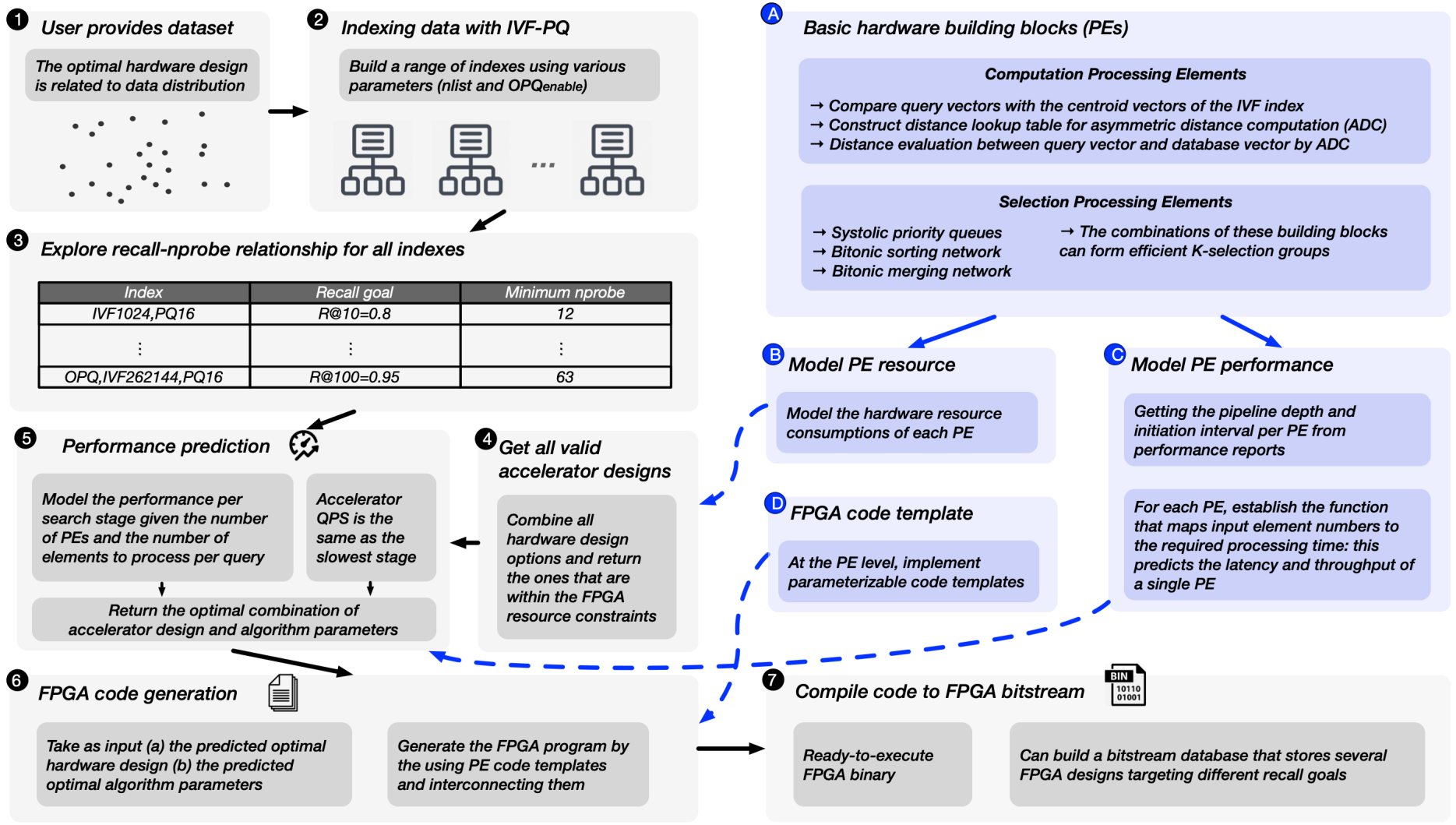
# What is the need in a deployment scenario?



# FANNS: co-design hardware and algorithm

1. Consider various algorithm parameters
2. Consider various valid hardware designs
3. Model the performance of each combination
4. Choose the most performant combination
5. Generate the respective hardware on FPGA

# FANNS: co-design hardware and algorithm



# FANNS: the algorithm side

Given user-provided dataset, try various IVF-PQ parameters

Explore the relationship between parameters and recall

| Index | Recall goal | Minimum nprobe |
|---|---|---|
| IVF1024,PQ16 | R@10=0.8 | 12 |
| ⋮ | ⋮ | ⋮ |
| OPQ,IVF262144,PQ16 | R@100=0.95 | 63 |

# FANNS: the hardware side

For each stage, build the hardware processing elements (PE)

If there are multiple valid designs, build multiple types of PEs

Model the resource consumption per type of PE

Model the performance per type of PE

$$QPS_{PE} = freq/(L + (N - 1) * II)$$

# Get all valid accelerator designs

Combine hardware building blocks of all six stages

An accelerator is valid if it fits on the FPGA

$$\sum_i C_r(PE_i) + \sum_i C_r(FIFO_i) + C_r(infra) \leq Constraint_r,$$
$$\forall r \in \{BRAM, URAM, LUT, FF, DSP\}$$

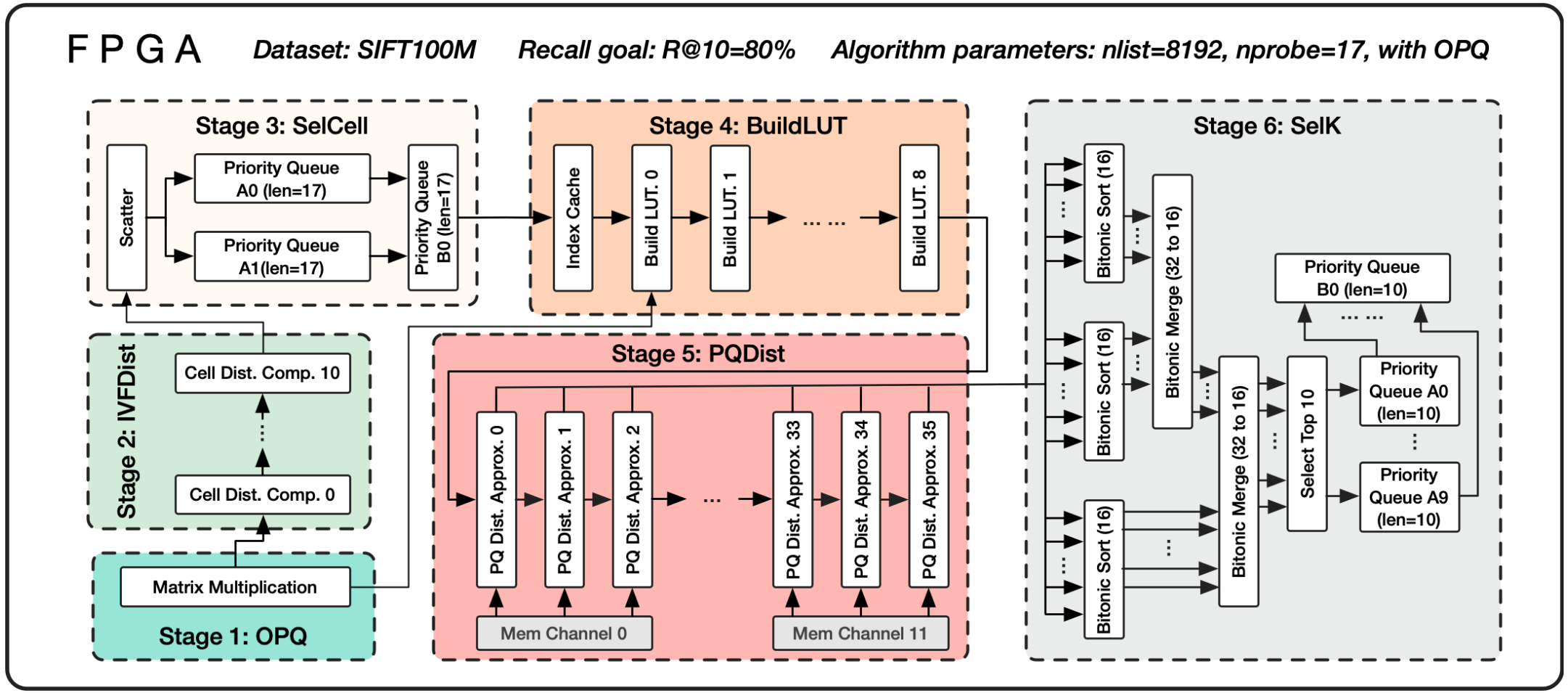
# Search performance prediction

We have data on both algorithm and hardware sides:

- Algorithm: the parameter set to achieve the recall goal
- Hardware: the performance model of a single PE

Putting them together: the search performance of each parameter-hardware combination

# End-to-end generation of the optimal accelerator



# The designs differ across user requirements

baseline

| | Index | nprobe | Stage OPQ | | Stage IVFDist | | | Stage SelCells | | | Stage BuildLUT | | | Stage PQDist | | Stage SelK | | | Pred. QPS (140 MHz) |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | #PE | LUT.(%) | #PE | Index store | LUT.(%) | Arch. | #InStream | LUT.(%) | #PE | Index store | LUT.(%) | #PE | LUT.(%) | Arch. | #InStream | LUT.(%) | |
| K=1 (Baseline) | N/A | N/A | 1 | 0.2 | 10 | HBM | 6.9 | HPQ | 2 | 6.4 | 5 | HBM | 6.9 | 36 | 15.2 | HPQ | 72 | 1.8 | N/A |
| K=10 (Baseline) | N/A | N/A | 1 | 0.2 | 10 | HBM | 6.9 | HPQ | 2 | 6.4 | 4 | HBM | 6.3 | 16 | 6.7 | HPQ | 32 | 5.7 | N/A |
| K=100 (Baseline) | N/A | N/A | 1 | 0.2 | 10 | HBM | 6.9 | HPQ | 2 | 6.4 | 4 | HBM | 6.3 | 4 | 1.7 | HPQ | 8 | 15.0 | N/A |
| K=1 (*FANNS*) | IVF4096 | 5 | 0 | 0 | 16 | on-chip | 11.0 | HPQ | 2 | 0.3 | 5 | on-chip | 2.6 | 57 | 24.0 | HPQ | 114 | 2.9 | 31,876 |
| K=10 (*FANNS*) | OPQ+IVF8192 | 17 | 1 | 0.2 | 11 | on-chip | 7.6 | HPQ | 2 | 0.9 | 9 | on-chip | 5.2 | 36 | 15.2 | HSMPQG | 36 | 12.7 | 11,098 |
| K=100 (*FANNS*) | OPQ+IVF16384 | 33 | 1 | 0.2 | 8 | on-chip | 5.5 | HPQ | 1 | 0.6 | 5 | on-chip | 3.6 | 9 | 3.8 | HPQ | 18 | 31.7 | 3,818 |

FANNS

# The designs differ across user requirements

| | Index | nprobe | Stage OPQ | | Stage IVFDist | | | Stage SelCells | | | Stage BuildLUT | | | Stage PQDist | | Stage SelK | | | Pred. QPS (140 MHz) |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | #PE | LUT.(%) | #PE | Index store | LUT.(%) | Arch. | #InStream | LUT.(%) | #PE | Index store | LUT.(%) | #PE | LUT.(%) | Arch. | #InStream | LUT.(%) | |
| K=1 (Baseline) | N/A | N/A | 1 | 0.2 | 10 | HBM | 6.9 | HPO | 2 | 6.4 | 5 | HBM | 6.9 | 36 | 15.2 | HPO | 72 | 1.8 | N/A |
| K=10 (Basel | | | | | | | | | | | | | | | | | 32 | 5.7 | N/A |
| K=100 (Base | | | | | | | | | | | | | | | | | 8 | 15.0 | N/A |
| K=1 (*FANN* | | | | | | | | | | | | | | | | | 114 | 2.9 | 31,876 |
| K=10 (*FANN* | | | | | | | | | | | | | | | | QG | 36 | 12.7 | 11,098 |
| K=100 (*FAN* | | | | | | | | | | | | | | | | | 18 | 31.7 | 3,818 |

| | Index | nprobe | Stage OPQ | | Stage IVFDist | | |
|---|---|---|---|---|---|---|---|
| | | | #PE | LUT.(%) | #PE | Index store | LUT.(%) |
| K=1 (Baseline) | N/A | N/A | 1 | 0.2 | 10 | HBM | 6.9 |
| K=10 (Baseline) | N/A | N/A | 1 | 0.2 | 10 | HBM | 6.9 |
| K=100 (Baseline) | N/A | N/A | 1 | 0.2 | 10 | HBM | 6.9 |
| K=1 (*FANNS*) | IVF4096 | 5 | 0 | 0 | 16 | on-chip | 11.0 |
| K=10 (*FANNS*) | OPQ+IVF8192 | 17 | 1 | 0.2 | 11 | on-chip | 7.6 |
| K=100 (*FANNS*) | OPQ+IVF16384 | 33 | 1 | 0.2 | 8 | on-chip | 5.5 |

# The designs differ across user requirements

| | Index | nprobe | Stage OPQ | | Stage IVFDist | | | Stage SelCells | | | Stage BuildLUT | | | Stage PQDist | | Stage SelK | | | Pred. QPS (140 MHz) |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | #PE | LUT.(%) | #PE | Index store | LUT.(%) | Arch. | #InStream | LUT.(%) | #PE | Index store | LUT.(%) | #PE | LUT.(%) | Arch. | #InStream | LUT.(%) | |
| K=1 (Baseline) | N/A | N/A | 1 | 0.2 | 10 | HBM | 6.9 | HPO | 2 | 6.4 | 5 | HBM | 6.9 | 36 | 15.2 | HPQ | 72 | 1.8 | N/A |
| K=10 (Baseline) | N/A | N/A | 1 | 0.2 | 10 | HBM | 6.9 | HPO | 2 | 6.4 | 5 | HBM | 6.9 | 16 | 6.7 | HPQ | 32 | 5.7 | N/A |
| K=100 (Baseline) | N/A | N/A | 1 | 0.2 | 10 | HBM | 6.9 | HPO | 2 | 6.4 | 5 | HBM | 6.9 | 4 | 1.7 | HPQ | 8 | 15.0 | N/A |
| K=1 (*FANNS*) | IVF4096 | 5 | 1 | 0.2 | 10 | HBM | 6.9 | HPO | 2 | 6.4 | 5 | HBM | 6.9 | 57 | 24.0 | HPQ | 114 | 2.9 | 31,876 |
| K=10 (*FANNS*) | OPQ+IVF8192 | 1 | 1 | 0.2 | 10 | HBM | 6.9 | HPO | 2 | 6.4 | 5 | HBM | 6.9 | 36 | 15.2 | HSMPQG | 36 | 12.7 | 11,098 |
| K=100 (*FANNS*) | OPQ+IVF16384 | 3 | 1 | 0.2 | 10 | HBM | 6.9 | HPO | 2 | 6.4 | 5 | HBM | 6.9 | 9 | 3.8 | HPQ | 18 | 31.7 | 3,818 |

# Evaluation

Hardware

- CPU: Intel(R) Xeon(R) Platinum 8259CL (14 nm), 16vCPU, 64GB
- GPU: NVIDIA V100 (12 nm), 16 GB
- FPGA: AMD Alveo U55c (16 nm), 16 GB

Software

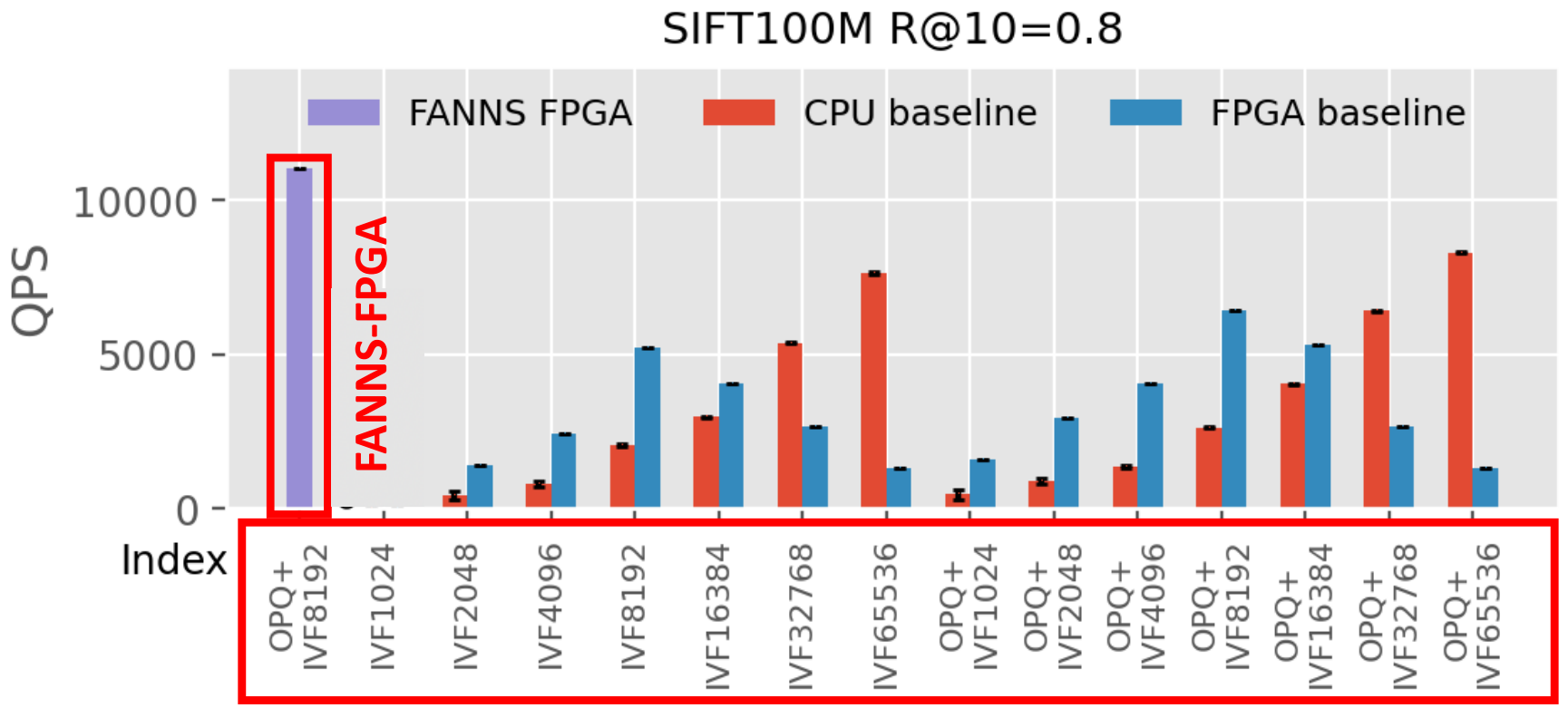
- Faiss: the most popular library for PQ-based ANN search
- Vitis HLS: for FPGA accelerator development

Datasets

- SIFT: 128-dimensional, 100 million vectors
- Deep: 96-dimensional, 100 million vectors

# Throughput speedups over CPU and FPGA baselines

**Up to 20.79x and 29.98x speedup over the FPGA/CPU baselines**



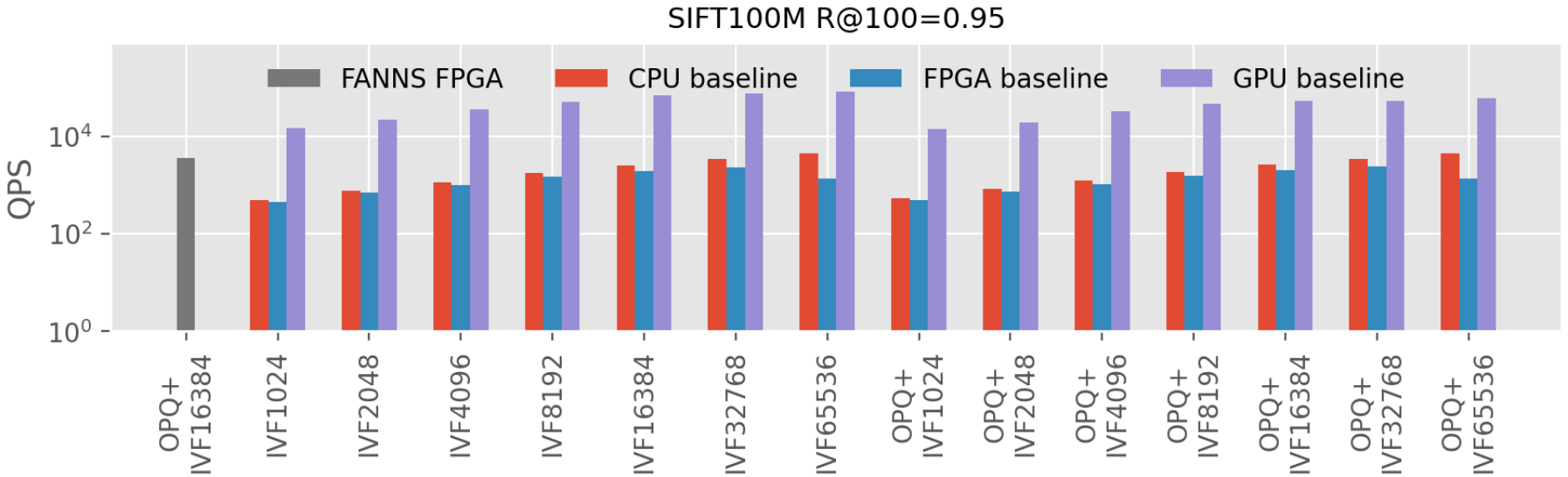

**The generated accelerators achieve >90% of the predicted performance**

# Throughput comparison to GPU baselines

Current-generation GPUs are much more powerful than FPGAs

- two orders of magnitude higher flop/s
- one order of magnitude higher bandwidth

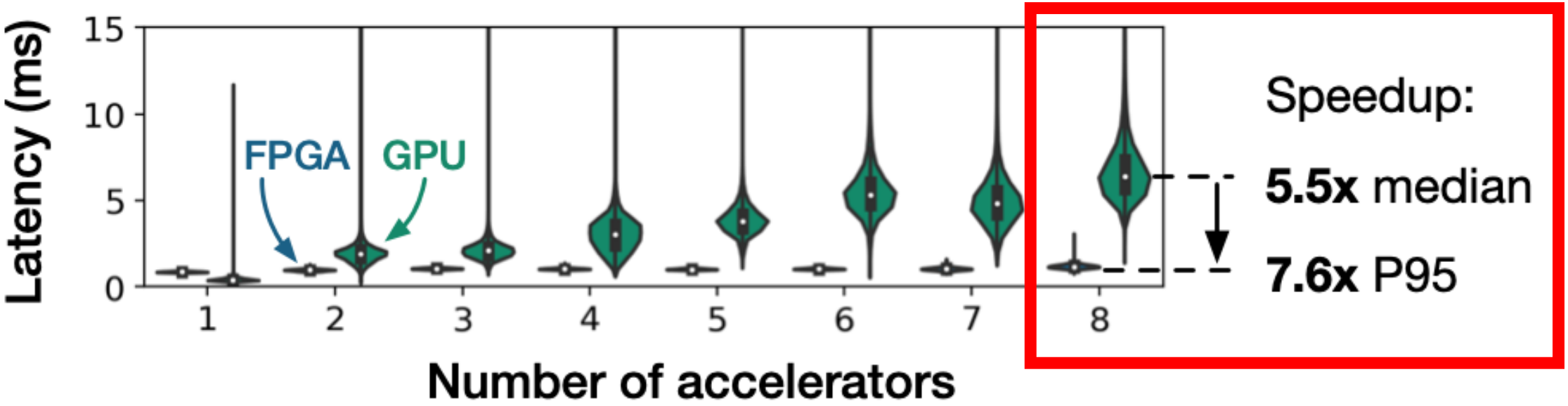
GPUs outperform FANNS in throughput (5.3~22.0x)



# FANNS versus GPUs in latency

GPUs have shown high variance in terms on latency (long tails)

In scale-out settings, the latency is limited by the slowest run



# Future ASIC designs?

The shifting bottlenecks is problematic for ASIC designs

Algorithm-level improvements to mitigate the problem?

- Scan many PQ codes are essential to achieve high recall
- But index scan can be treated as ANN search instead

Hybrid architecture:

- CPU for fast index traversal
- ASIC for scanning quantized DB vectors and collecting K results

# Conclusion

Vector search is the core of many AI applications including LLMs

The IVF-PQ algorithm have multiple stages and shifting bottlenecks

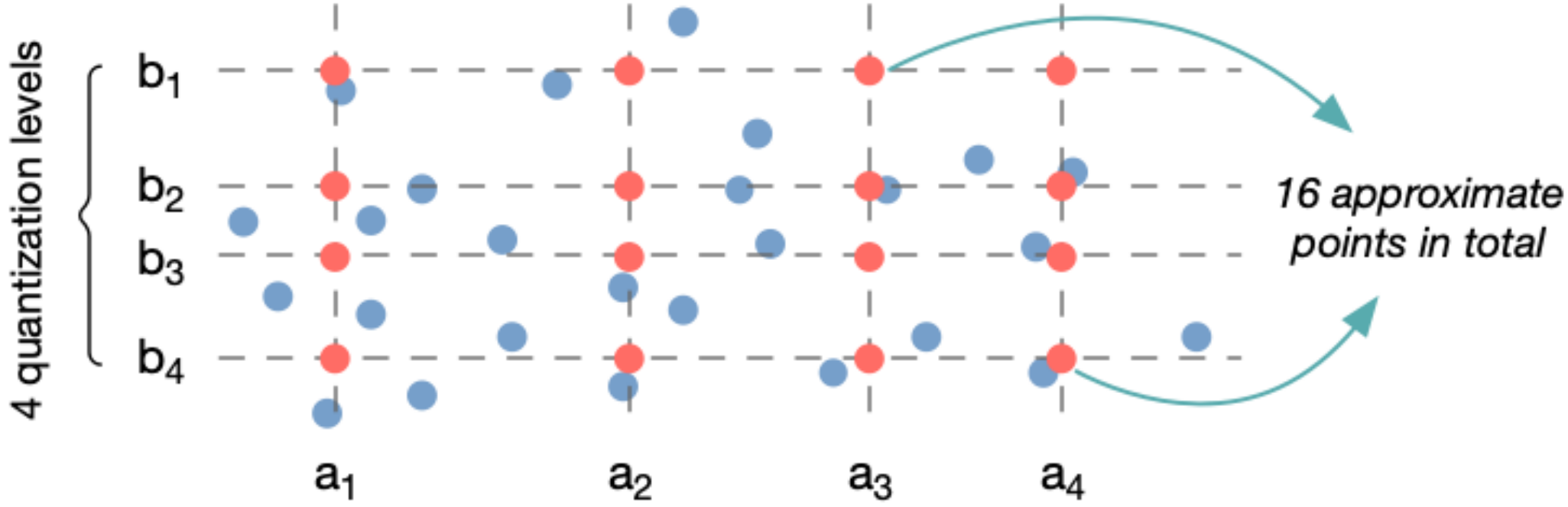
FANNS: design-space exploration by hardware-algorithm co-design

- Given a recall target on a dataset
- Use a performance-model to guide accelerator design
- Use a code-generator to make the design transparent to users

Up to 20.79x and 29.98x speedup over the FPGA/CPU baselines

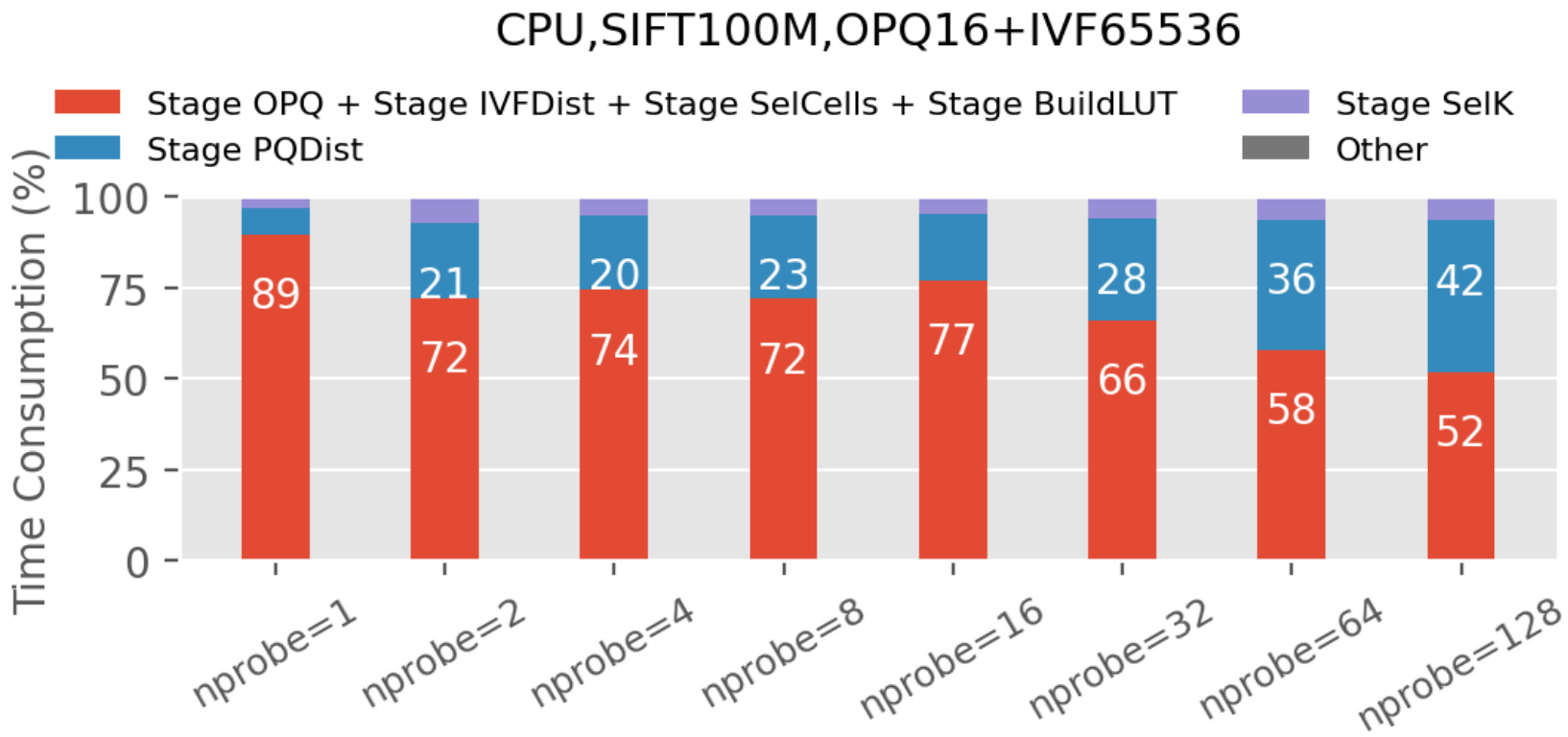
# Backup slides

# PQ on two-dimensional vectors



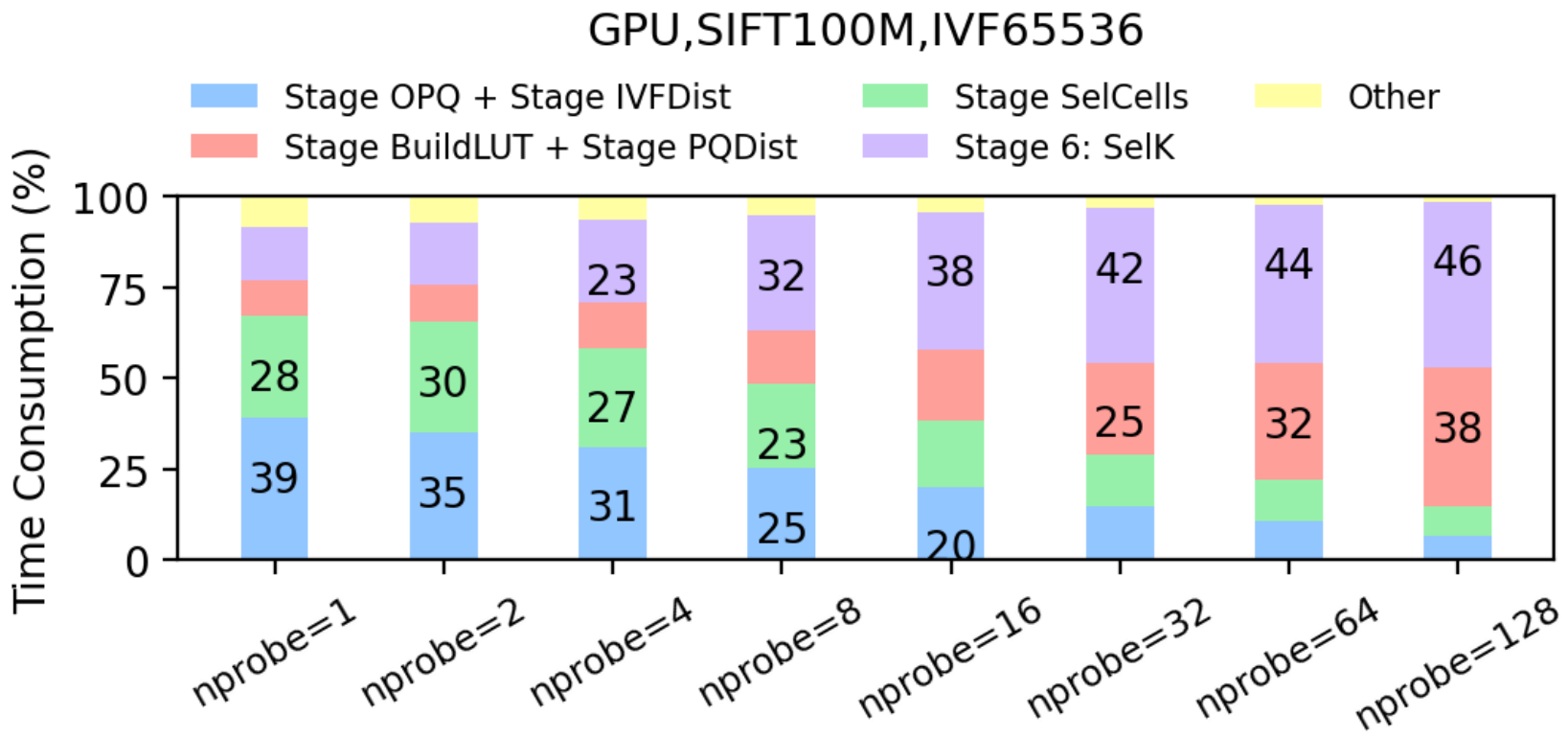
# Bottleneck shifts with different *nprobe*

*nprobe* = number of clusters to scan



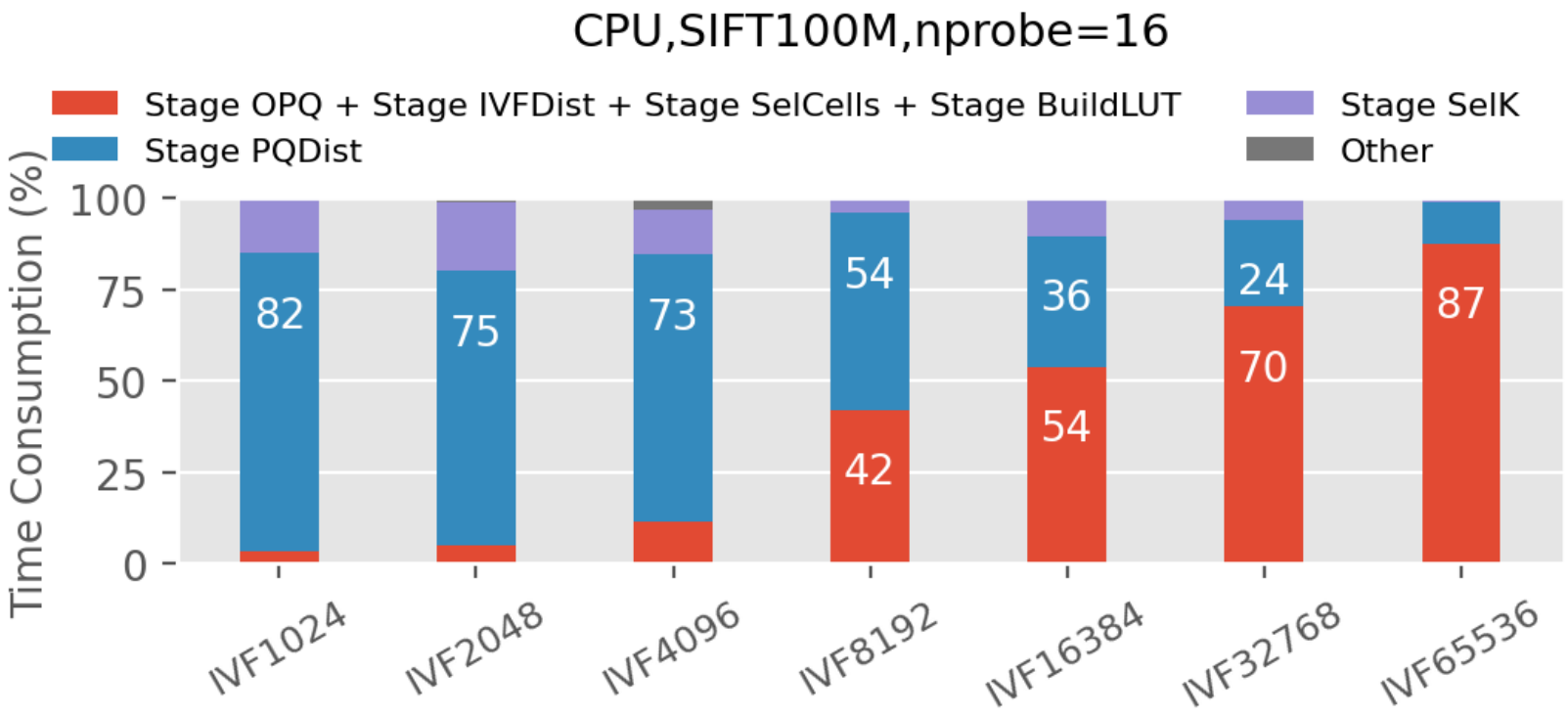
# Bottleneck shifts with different *nprobe*

*nprobe* = number of clusters to scan



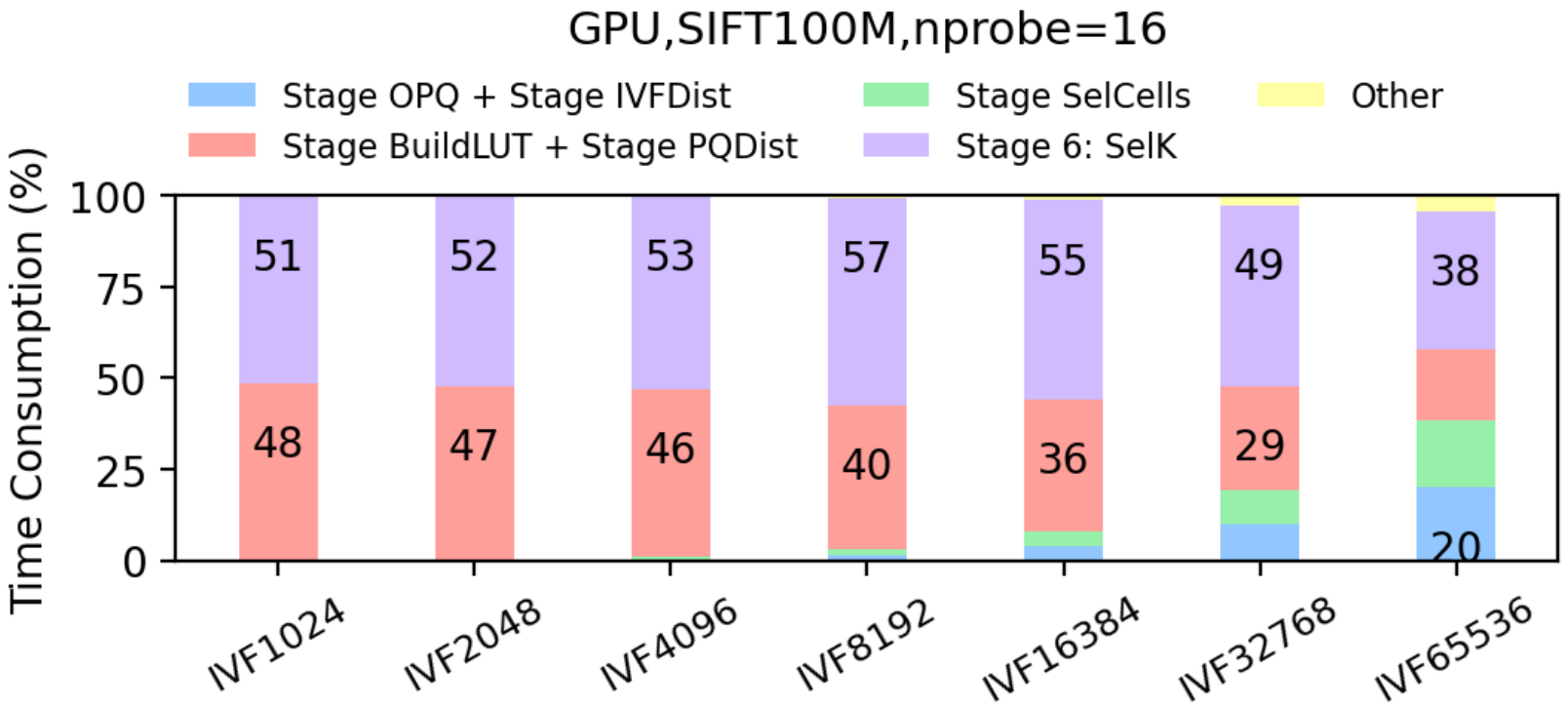
# Bottleneck shifts with different *nlist*

*nlist* = number of clusters in total



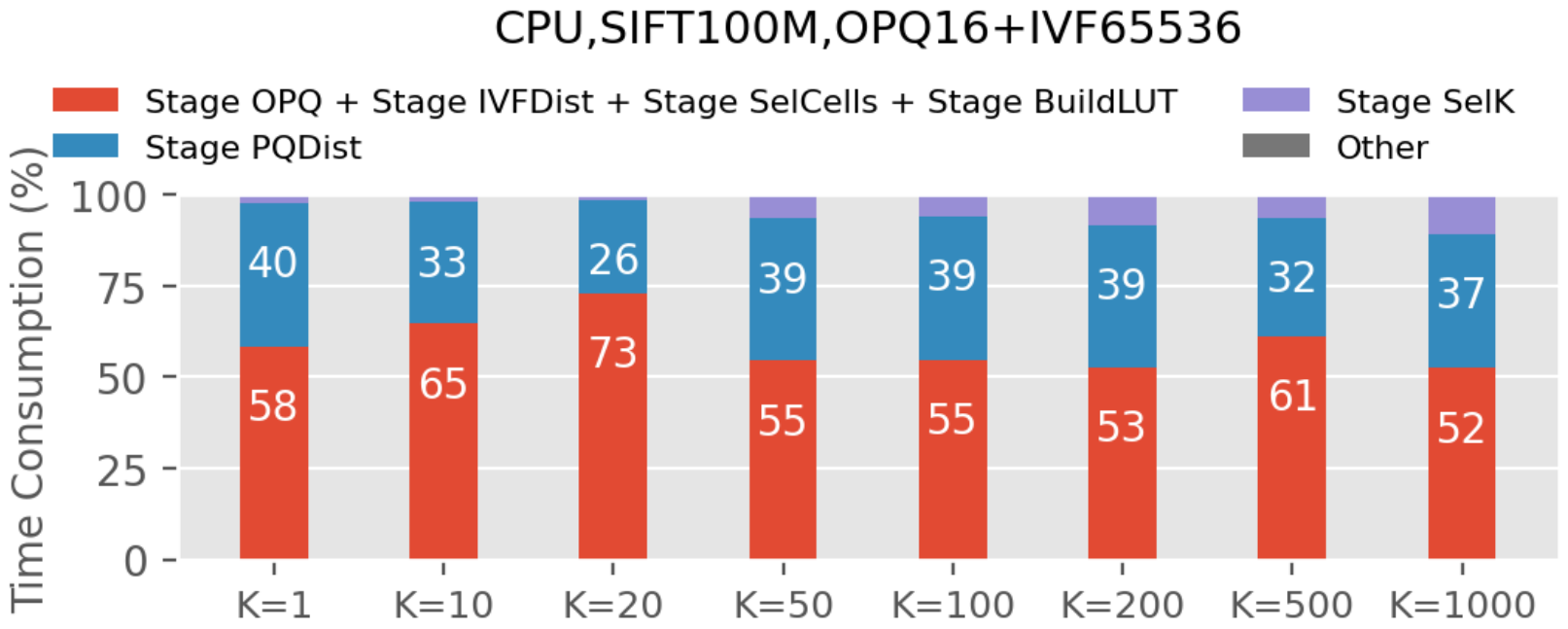
# Bottleneck shifts with different *nlist*

*nlist* = number of clusters in total



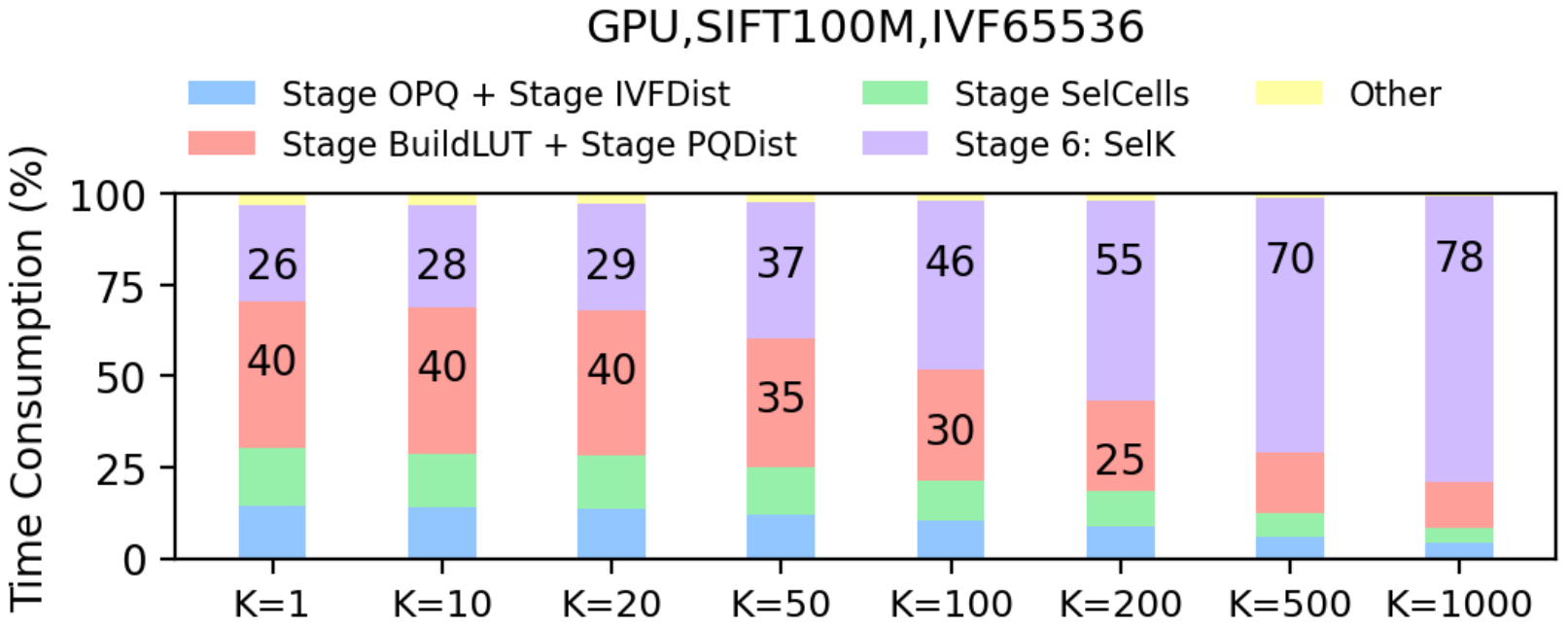
# Bottleneck shifts with different *K*

*K* = number of results to return



# Bottleneck shifts with different *K*

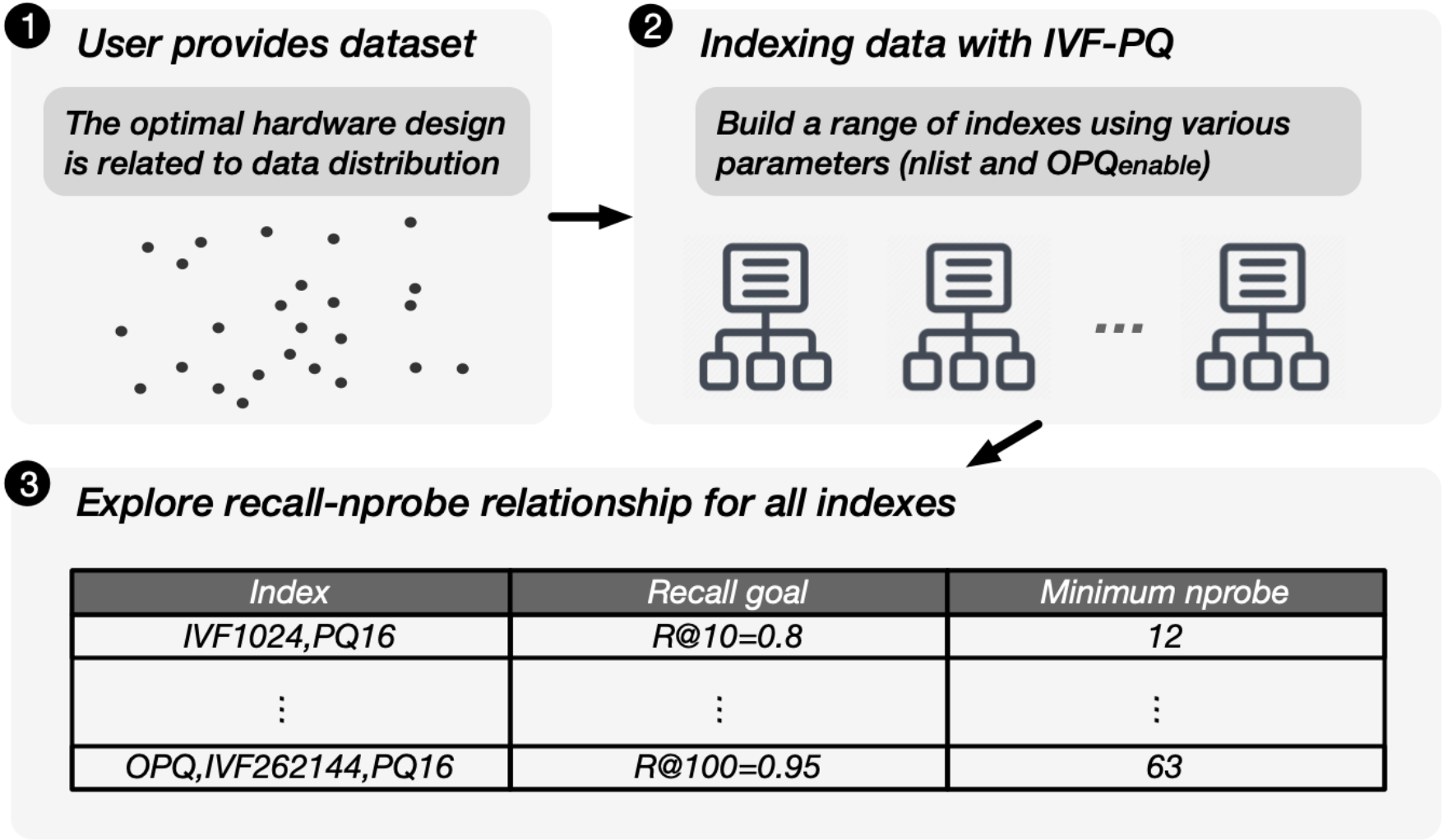
*K* = number of results to return



# Choices in design space exploration

| | |
|---|---|
| **Algorithm parameter space** | |
| $nlist$ | The totol Voronoi cell number. |
| $nprobe$ | The number of cells to be scanned per query. |
| $K$ | The number of most similar vectors to return. |
| $OPQ_{enable}$ | Whether to apply OPQ. |
| **Hardware design space** | |
| $Design_s$ | The microarchitecture design of stage $s$. |
| $\#PE_s$ | The number of processing elements in stage $s$. |
| $Cache_s$ | Cache index on-chip or store it off-chip for stage $s \in$ {Stage IVFDist, Stage BuildLUT}. |

# FANNS: the algorithm side




| Index | Recall goal | Minimum nprobe |
|---|---|---|
| IVF1024,PQ16 | R@10=0.8 | 12 |
| ⋮ | ⋮ | ⋮ |
| OPQ,IVF262144,PQ16 | R@100=0.95 | 63 |

# FANNS: the hardware side

A

***Basic hardware building blocks (PEs)***

***Computation Processing Elements***

→ *Compare query vectors with the centroid vectors of the IVF index*
→ *Construct distance lookup table for asymmetric distance computation (ADC)*
→ *Distance evaluation between query vector and database vector by ADC*

***Selection Processing Elements***

→ *Systolic priority queues*
→ *Bitonic sorting network*
→ *Bitonic merging network*

→ *The combinations of these building blocks can form efficient K-selection groups*

# FANNS: the hardware side

**B** *Model PE resource*

*Model the hardware resource consumptions of each PE*

**C** *Model PE performance*

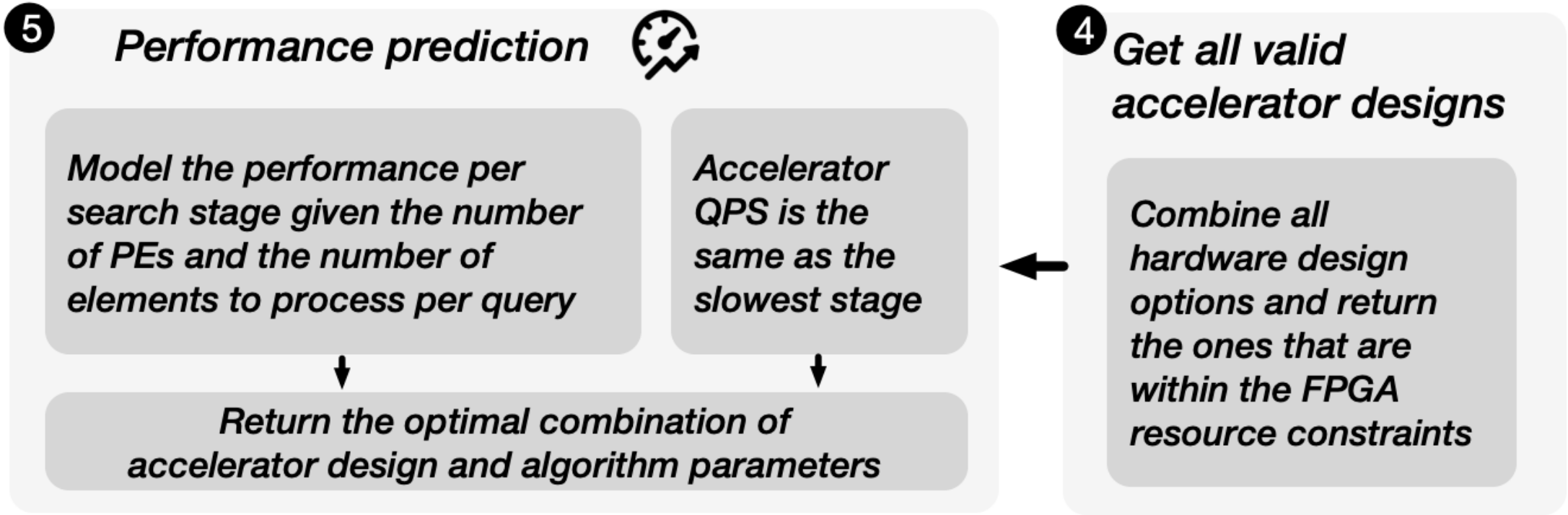
*Getting the pipeline depth and initiation interval per PE from performance reports*

*For each PE, establish the function that maps input element numbers to the required processing time: this predicts the latency and throughput of a single PE*

**D** *FPGA code template*

*At the PE level, implement parameterizable code templates*

# Predict optimal hardware-algorithm combination



# Performance prediction

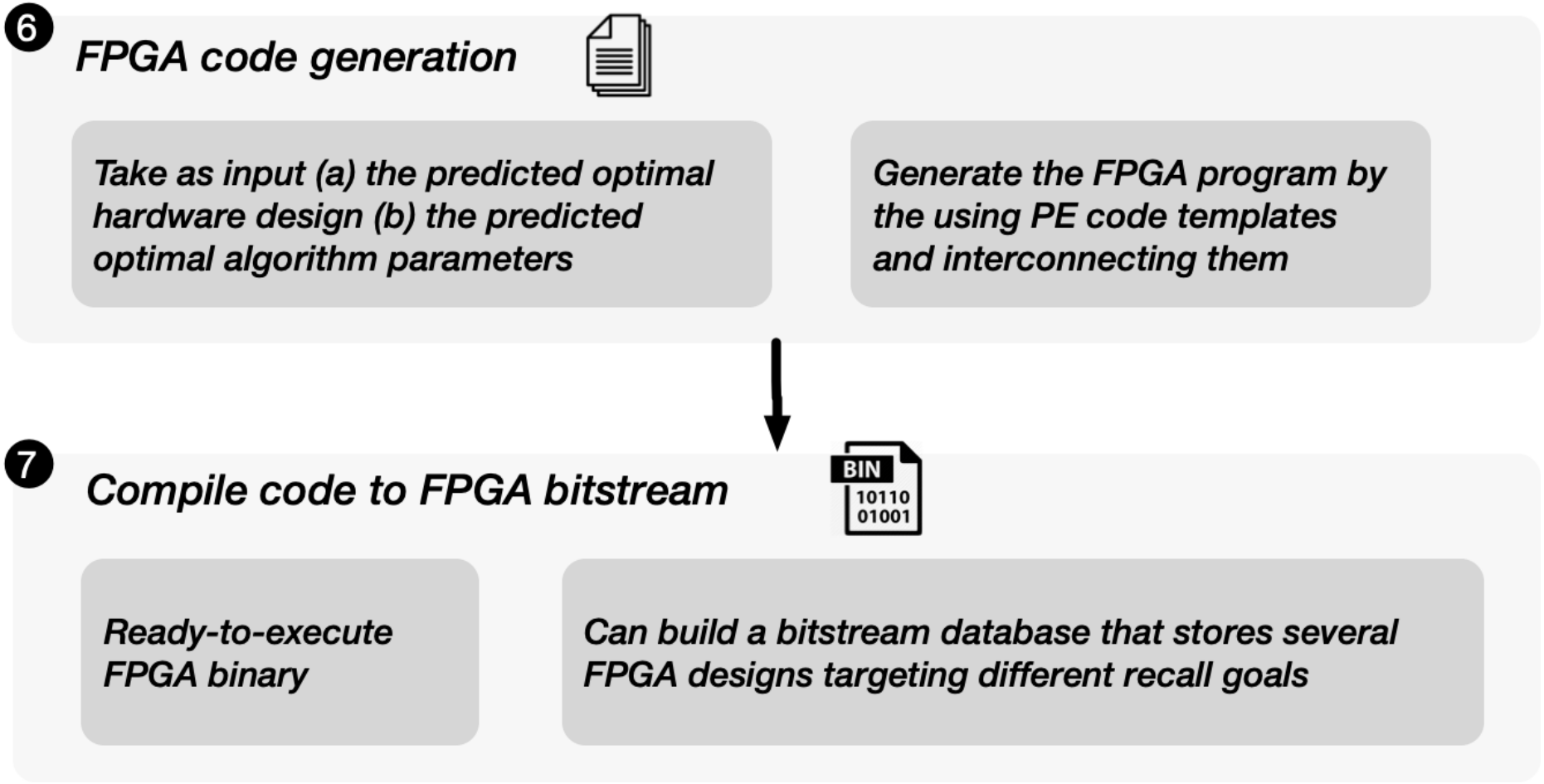
Performance of a single processing element:

$$QPS_{PE} = freq/(L + (N - 1) * II)$$

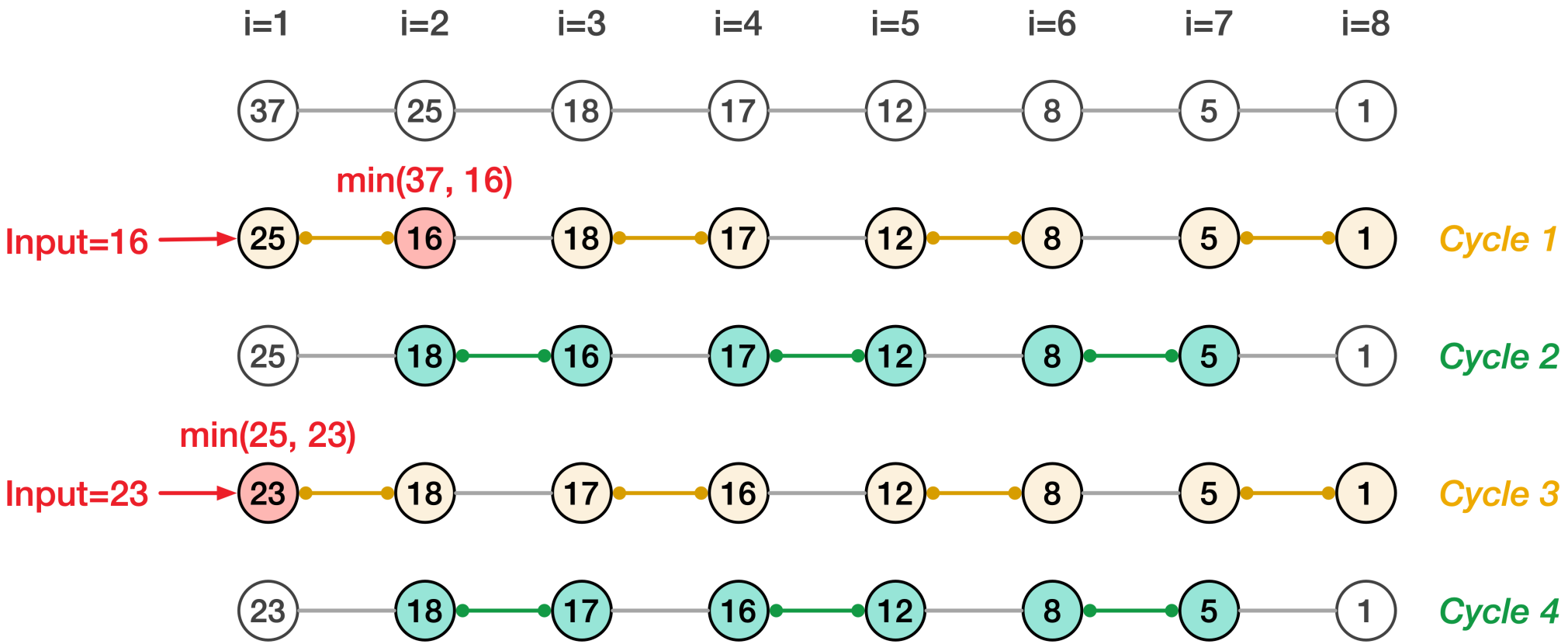
Performance of the entire accelerator depends on the slowest

$$QPS_{accelerator} = min(QPS_s), \text{where } s \in \{\text{Stages}\}$$

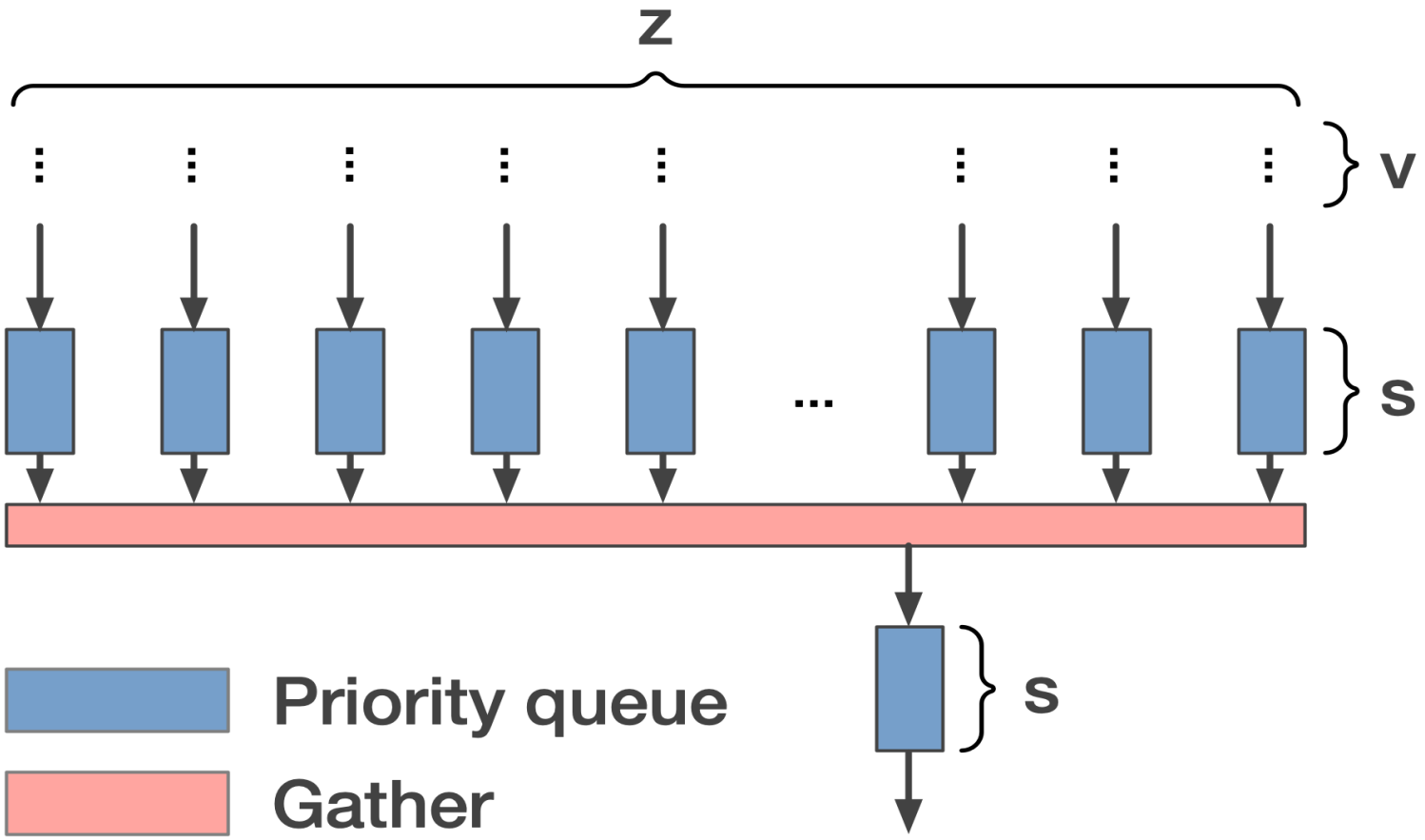
# Automatically generate the hardware



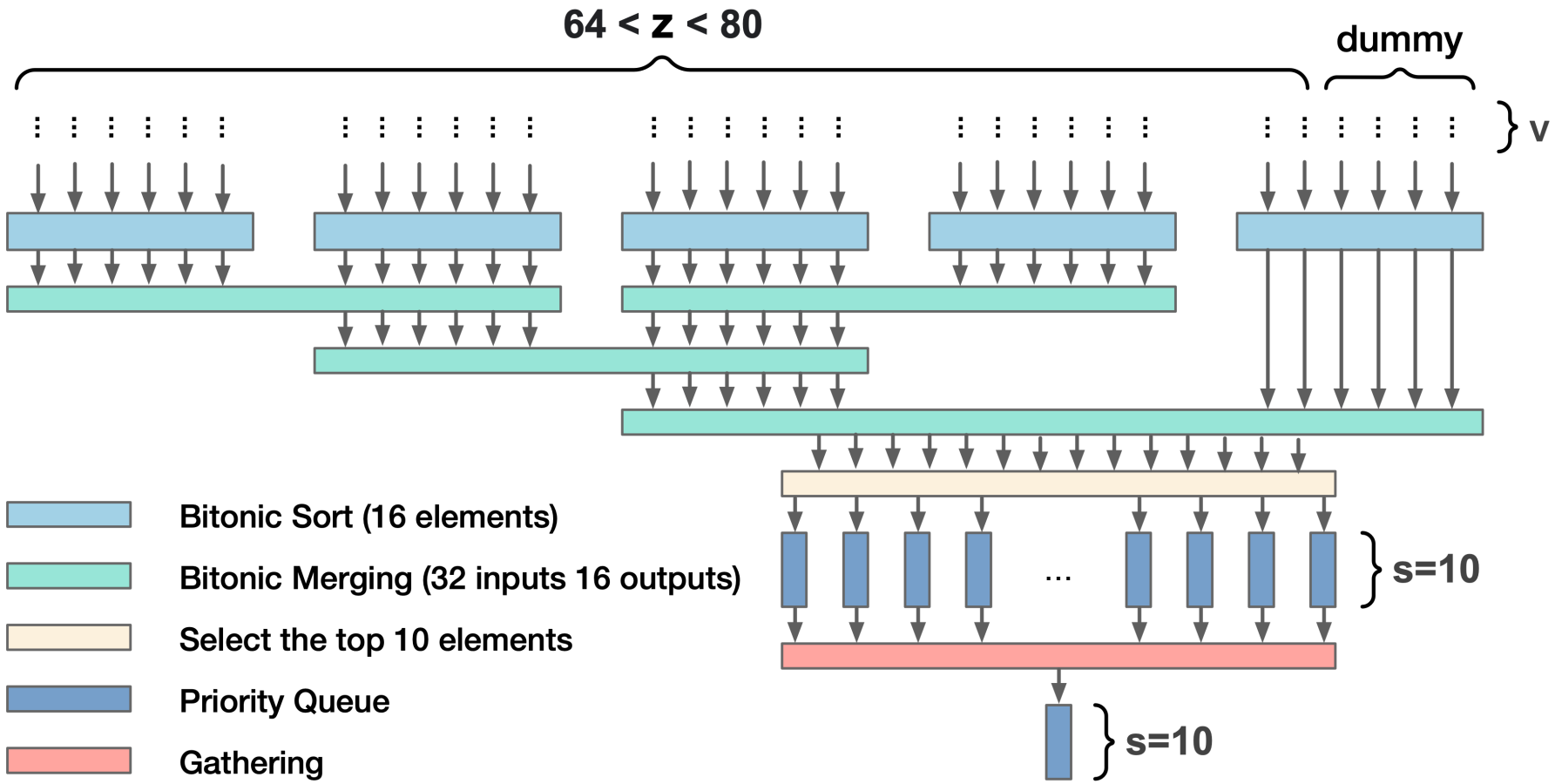
# Systolic Priority Queue



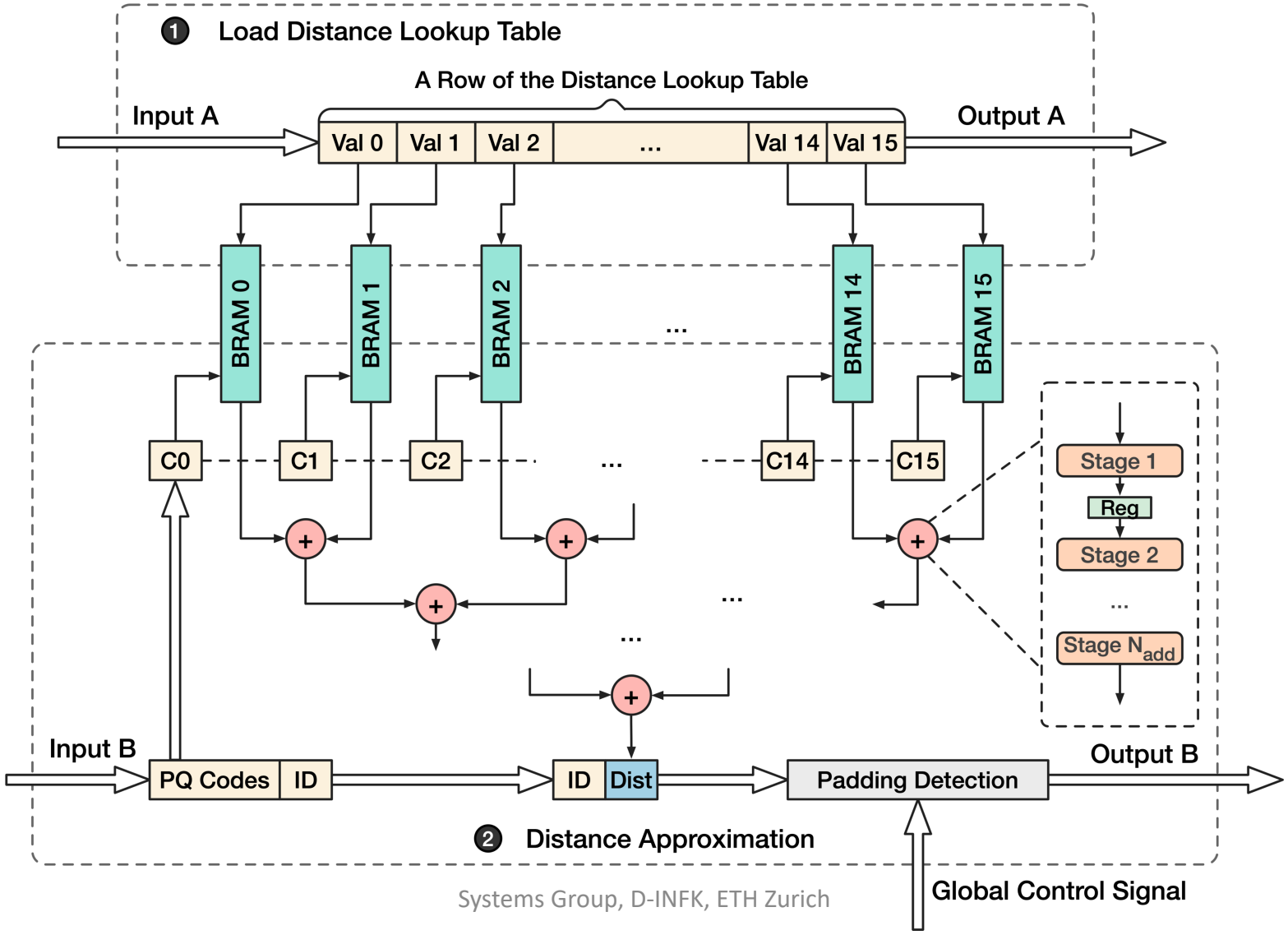
# Hierarchical priority queue group



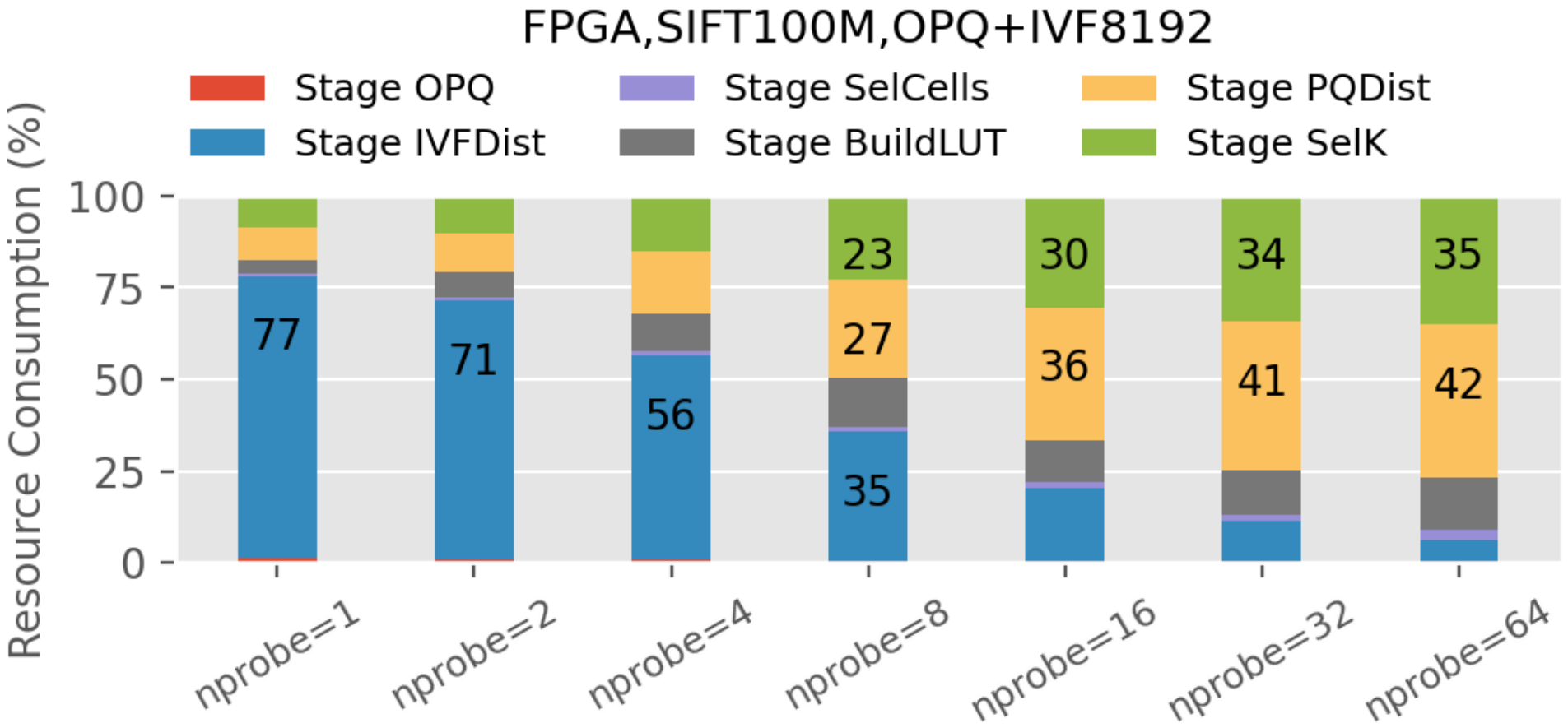
# Hybrid bitonic sorting, merging, and priority queue



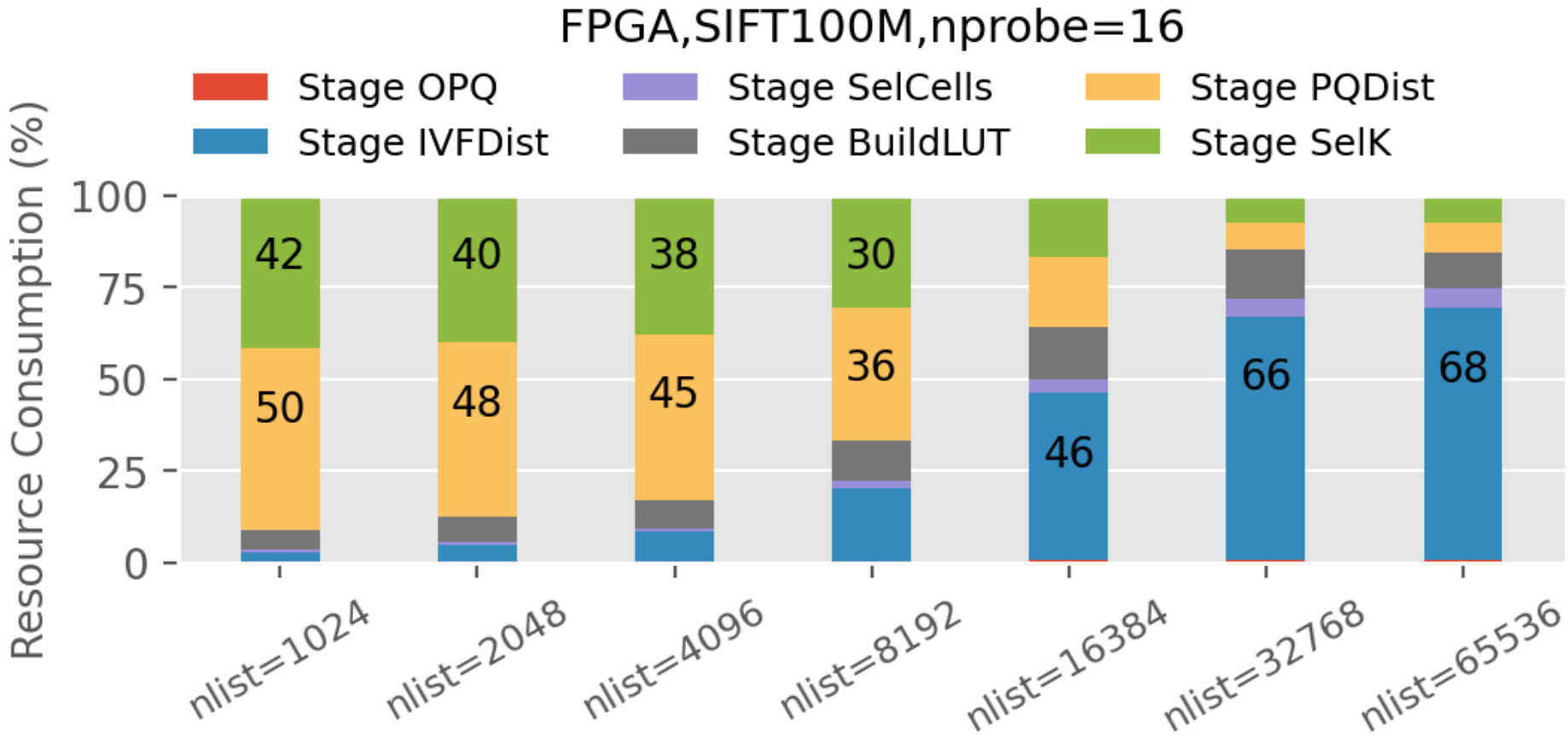
# PQ distance computation unit



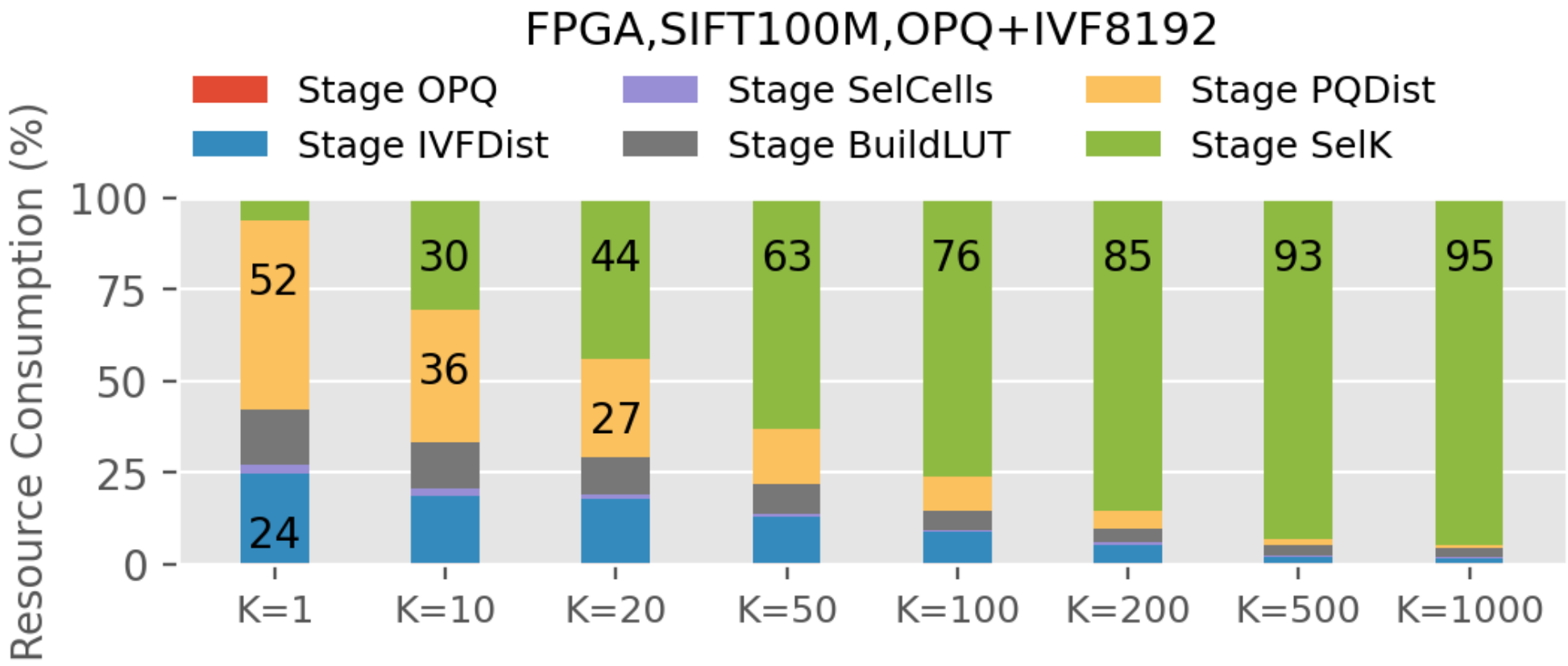
# Optimal FPGA design shifts with *nprobe*



# Optimal FPGA design shifts with *nlist*

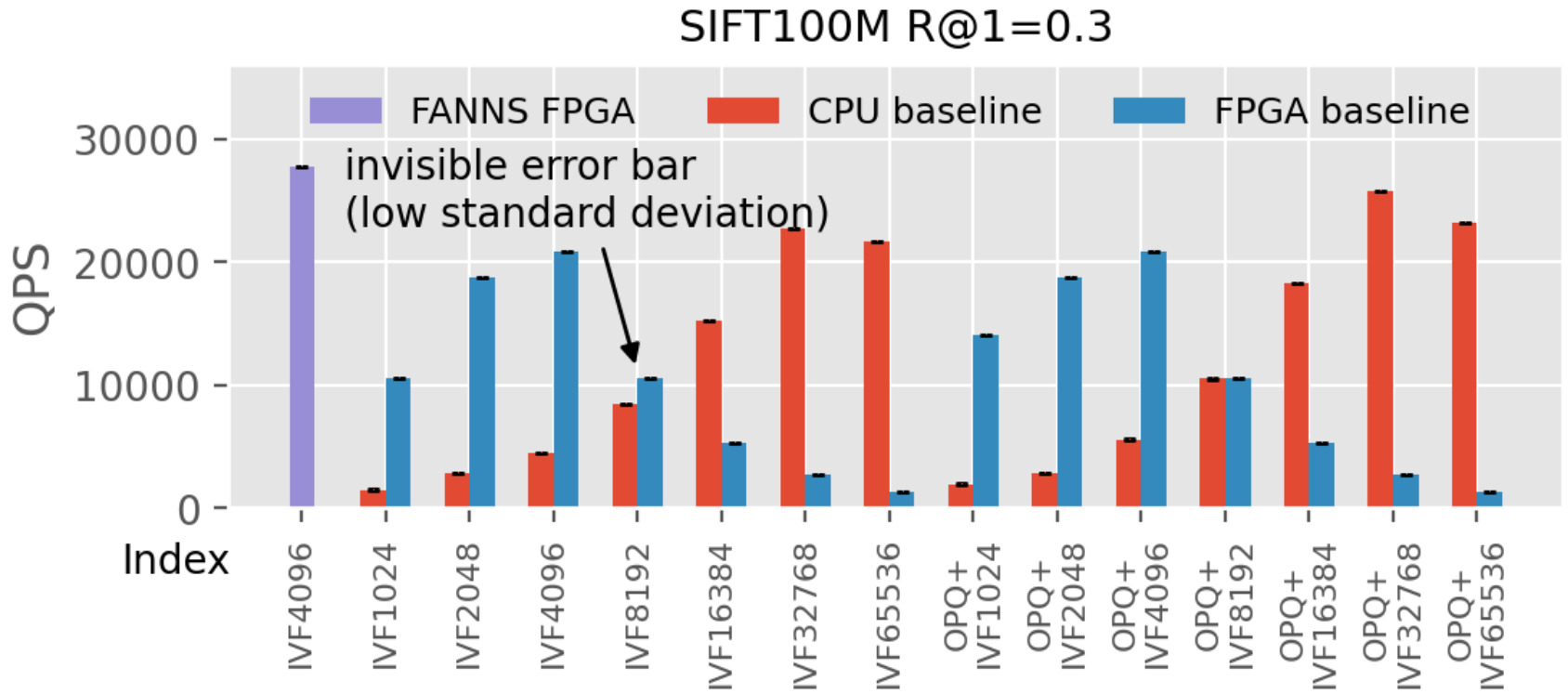


# Optimal FPGA design shifts with *K*



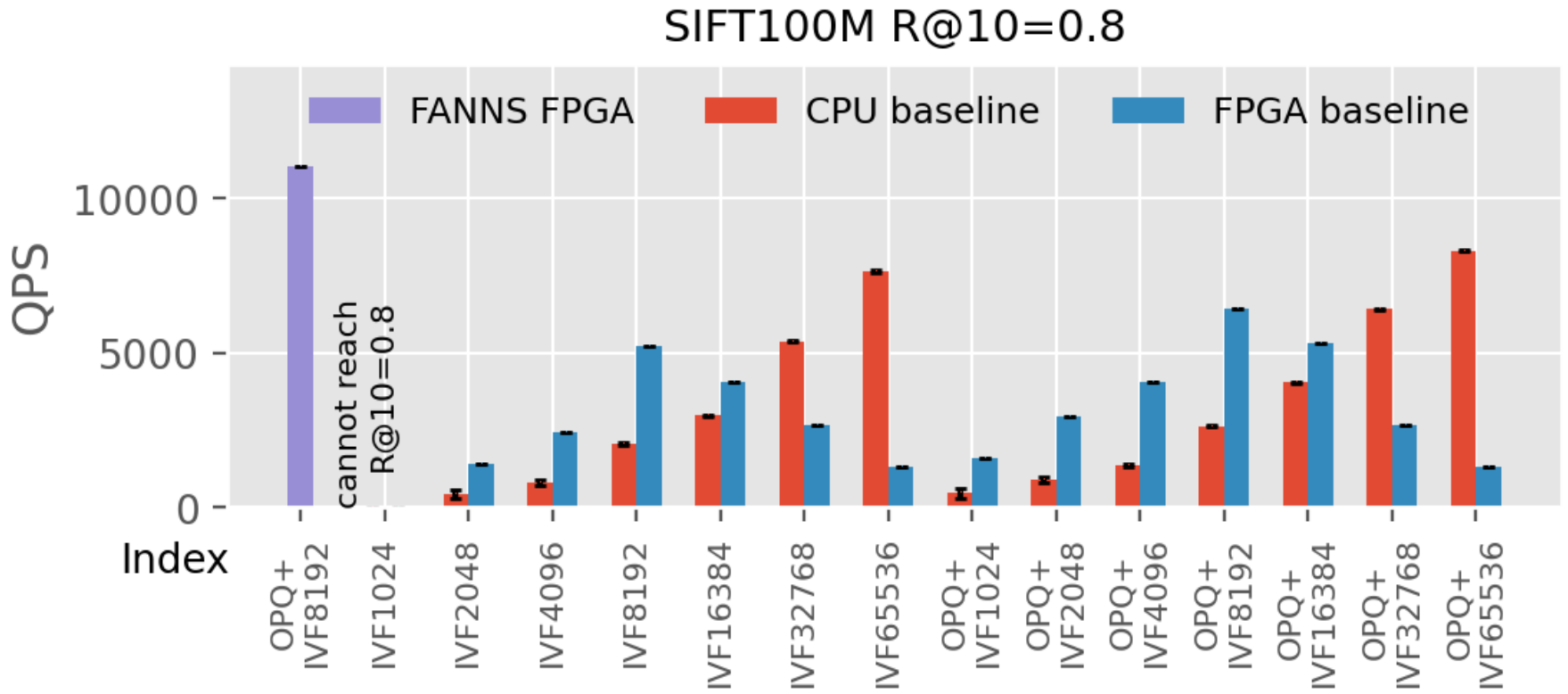
# Throughput speedups over CPU and FPGA baselines

**Up to 20.79x and 29.98x speedup over the FPGA/CPU baselines**



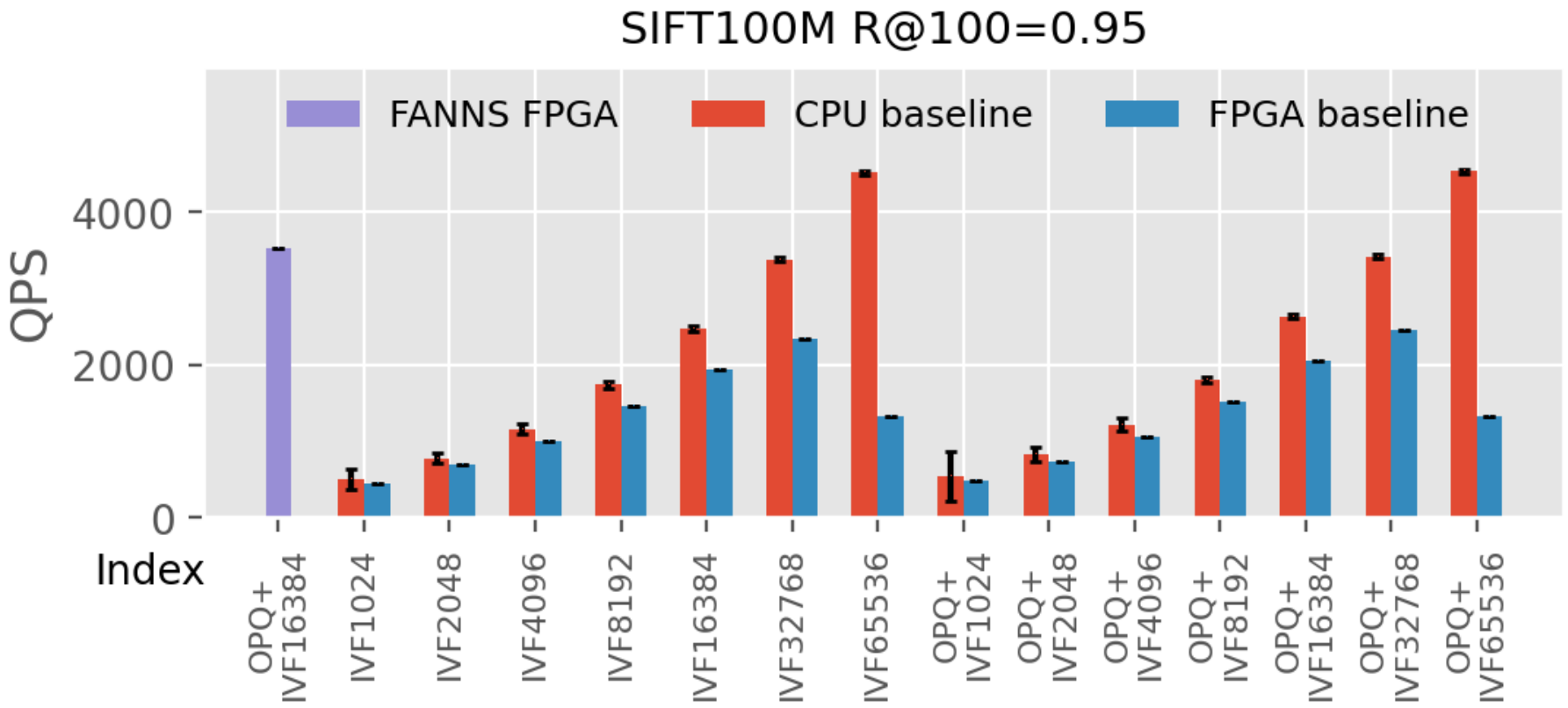
# Throughput speedups over CPU and FPGA baselines

**Up to 20.79x and 29.98x speedup over the FPGA/CPU baselines**

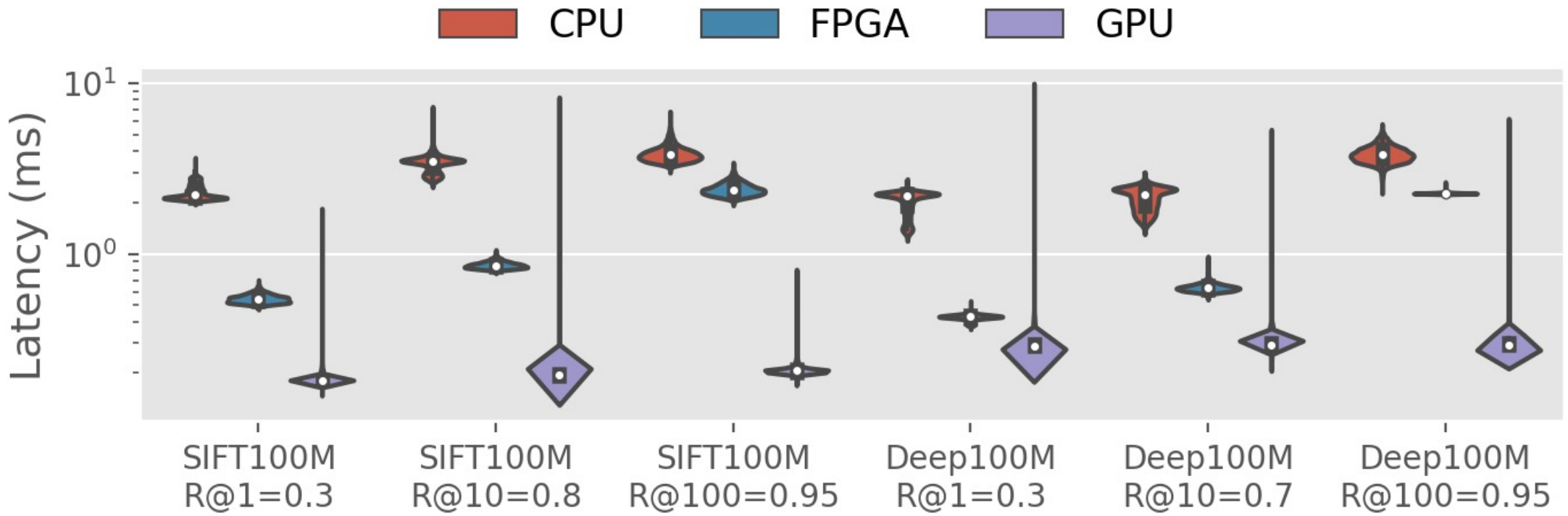


# Throughput speedups over CPU and FPGA baselines

**Up to 20.79x and 29.98x speedup over the FPGA/CPU baselines**

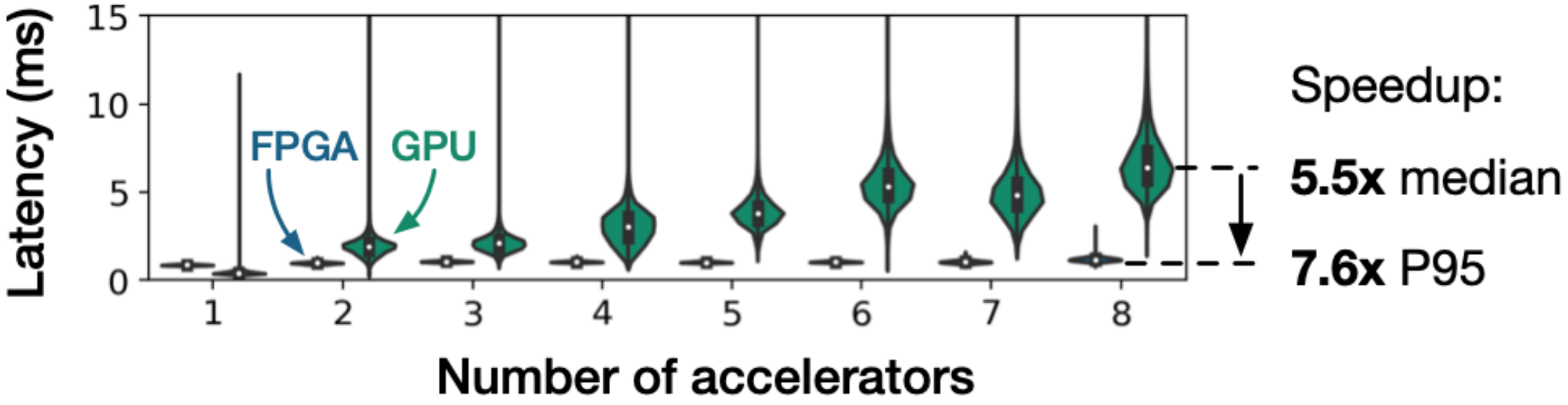


# Single-node latency comparison



# Latency scalability

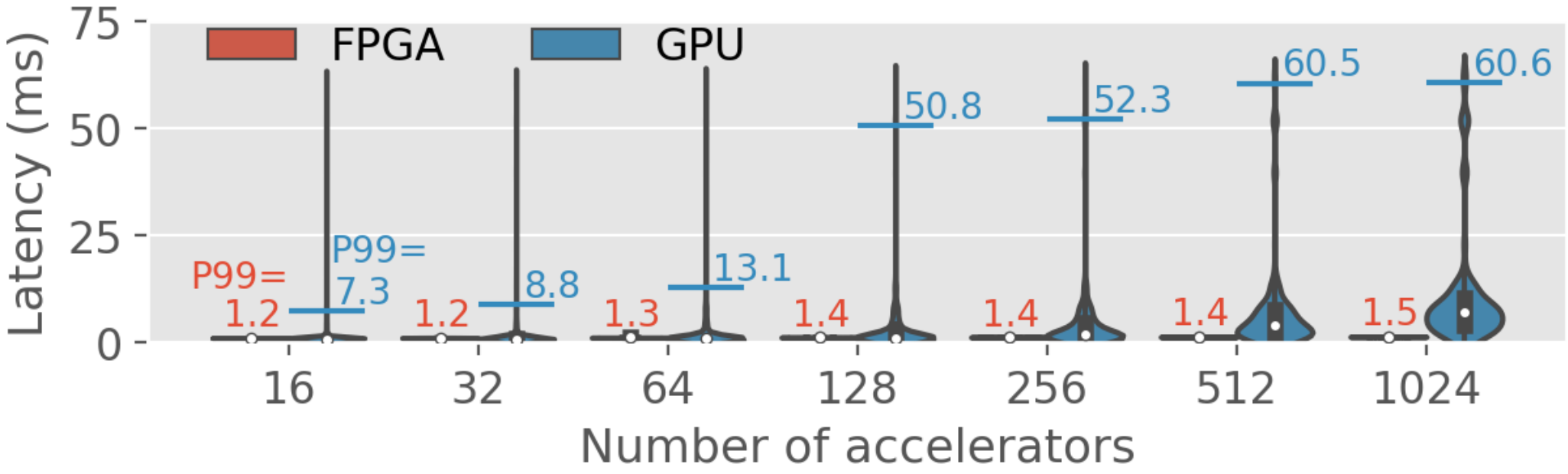
SIFT100M, nlist=8192, m=16, R@10=80%



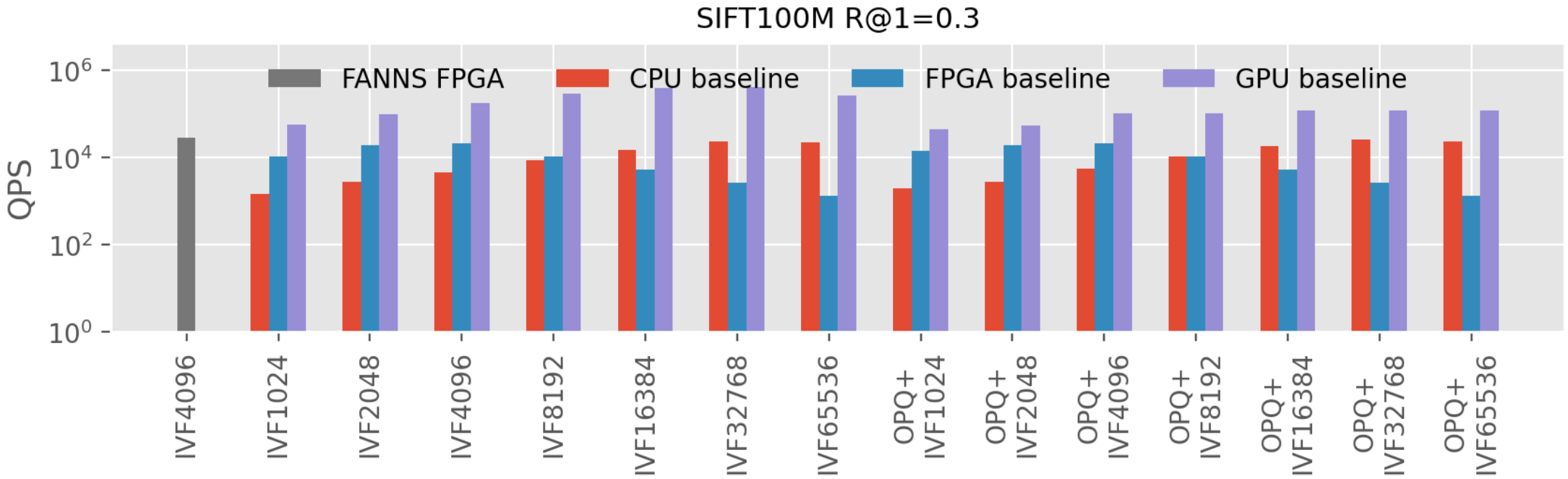
# Latency scalability trends

LogGP for modeling the network latency

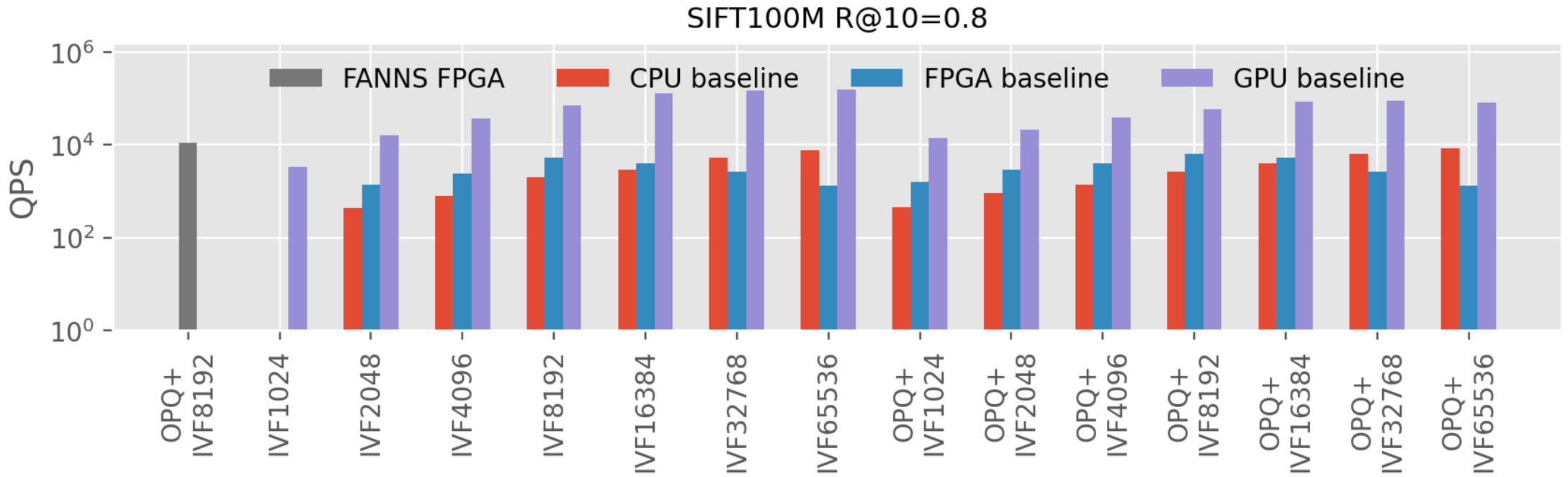
6.0 $\mu$s latency between two endpoints



# QPS comparison to CPU and GPU baselines



# QPS comparison to CPU and GPU baselines



# QPS comparison to CPU and GPU baselines