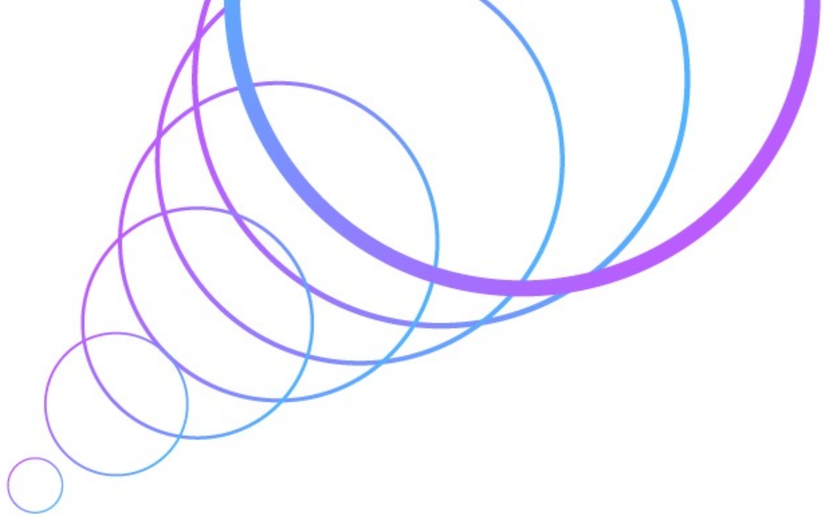


300억 벡터를 서빙하라!

네이버 검색은 ColBERT 벡터 유사도 검색 도전 중

반정호 NAVER
전보성 NAVER

CONTENTS

DEVIEW 2021

1.ColBERT 소개

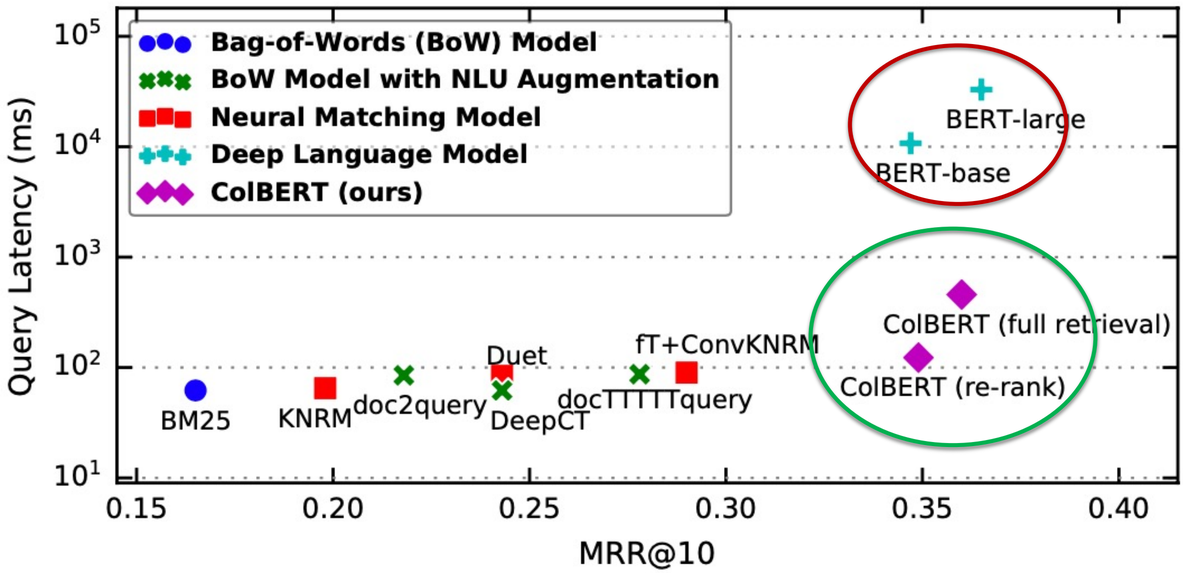
1.1 ColBERT

ColBERT : Contextualized Late Interaction over BERT

- BERT 기반 랭킹 모델

- BERT에 비해 훨씬 빠른 검색 속도

- BERT만큼 좋은 품질

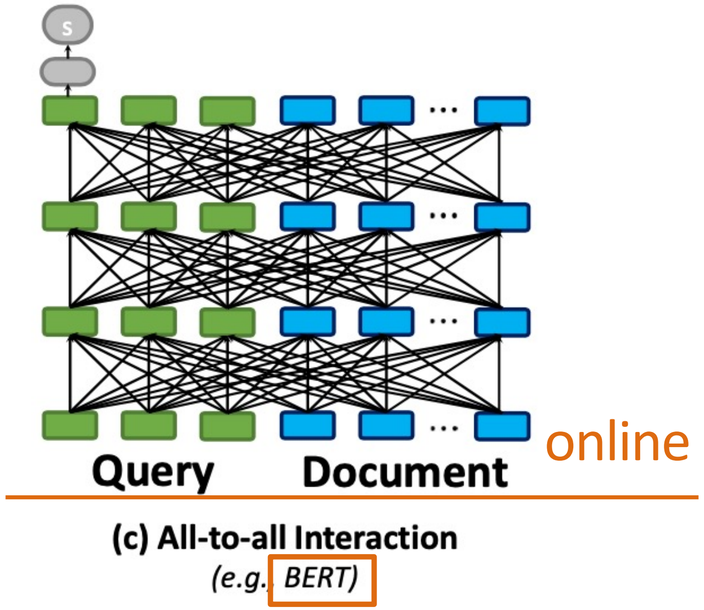


Khattab, O., Zaharia, M.: Colbert: Efficient and effective passage search via contextualized late interaction over BERT. SIGIR'20

1.2 ColBERT Late Interaction

All-to-all Interaction (BERT)

- 질의와 문서를 함께 계산함
- 검색에서 쓰기에 비현실적인 연산량

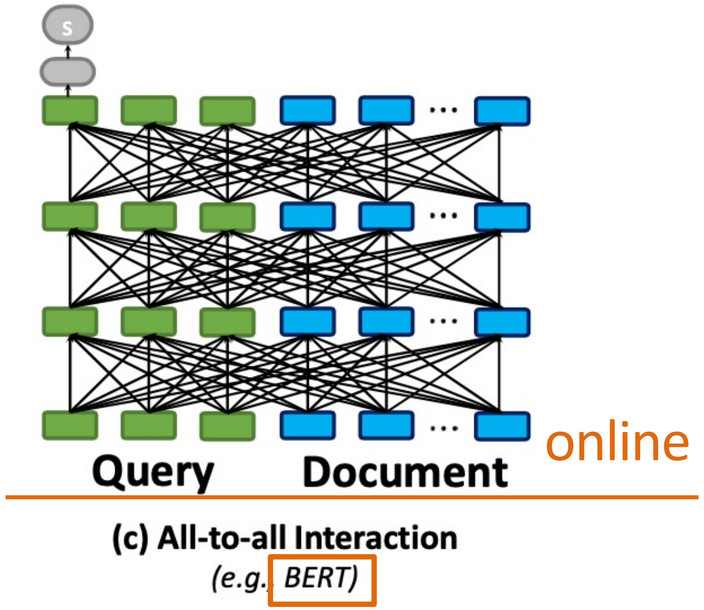


Khattab, O., Zaharia, M.: Colbert: Efficient and effective passage search via contextualized late interaction over BERT. SIGIR'20

1.2 ColBERT Late Interaction

All-to-all Interaction (BERT)

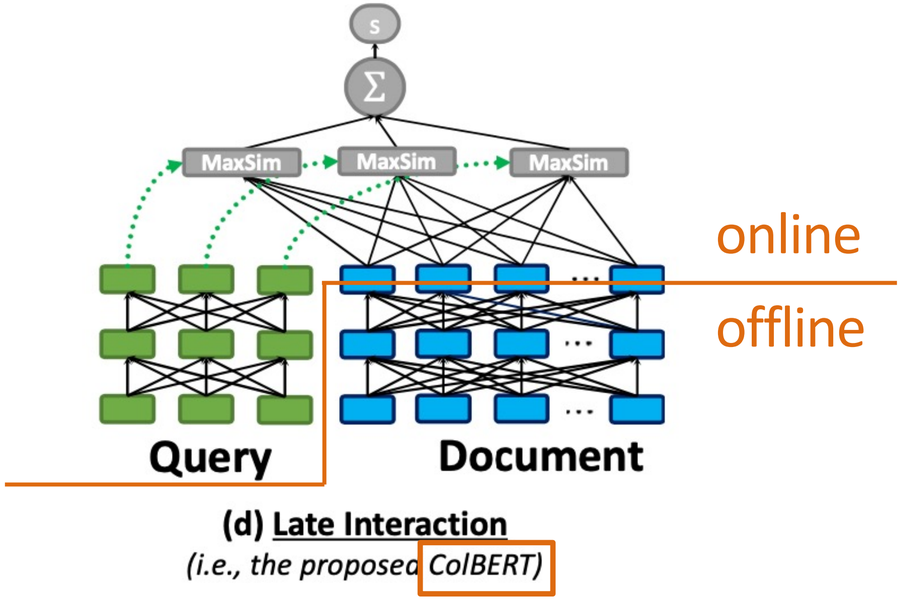
- 질의와 문서를 함께 계산함
- 검색에서 쓰기에 비현실적인 연산량



(c) All-to-all Interaction
(e.g., BERT)

Late Interaction (ColBERT)

- 질의와 문서를 분리하여 계산
- 문서의 임베딩만 미리 구축 가능
- 질의와 문서 연관도를 계산하기 위해 빠르고 간단한 수식 제안



(d) Late Interaction
(i.e., the proposed ColBERT)

Khattab, O., Zaharia, M.: Colbert: Efficient and effective passage search via contextualized late interaction over BERT. SIGIR'20

1.3 ColBERT architecture

DEVIEW 2021

MaxSim operation

질의 임베딩(벡터)과 가장
유사도가 높은 문서 임베딩(벡터)

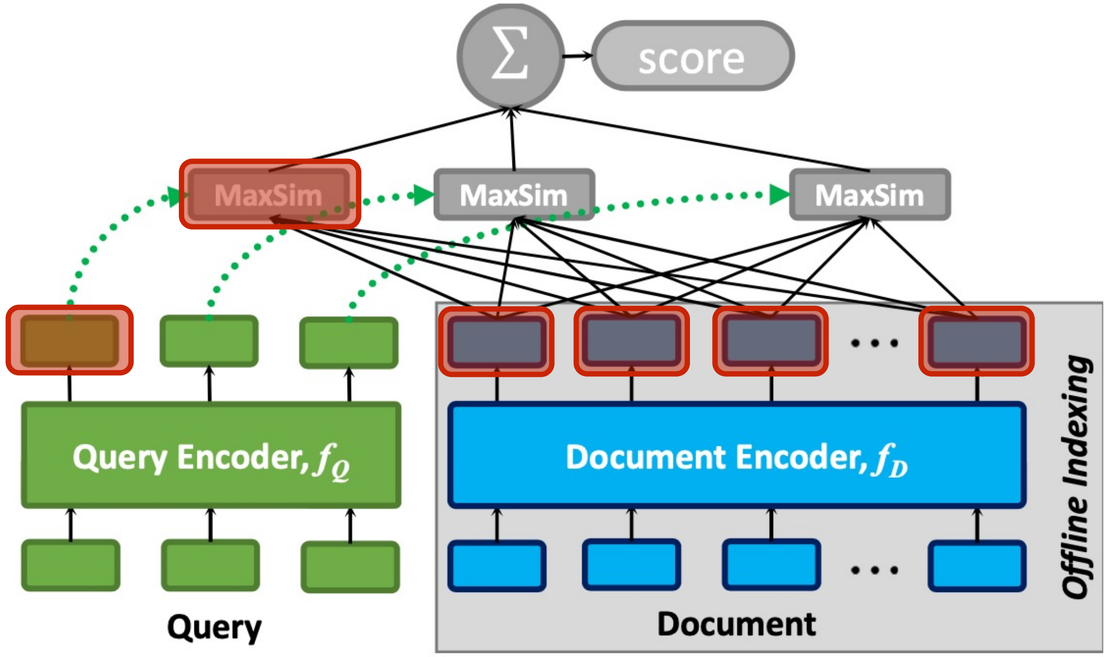


Figure 3: The general architecture of ColBERT given a query q and a document d.

Khattab, O., Zaharia, M.: Colbert: Efficient and effective passage search via contextualized late interaction over BERT. SIGIR'20

1.3 ColBERT architecture

MaxSim operation

질의 임베딩(벡터)과 가장
유사도가 높은 문서 임베딩(벡터)

score = ∑ MaxSim

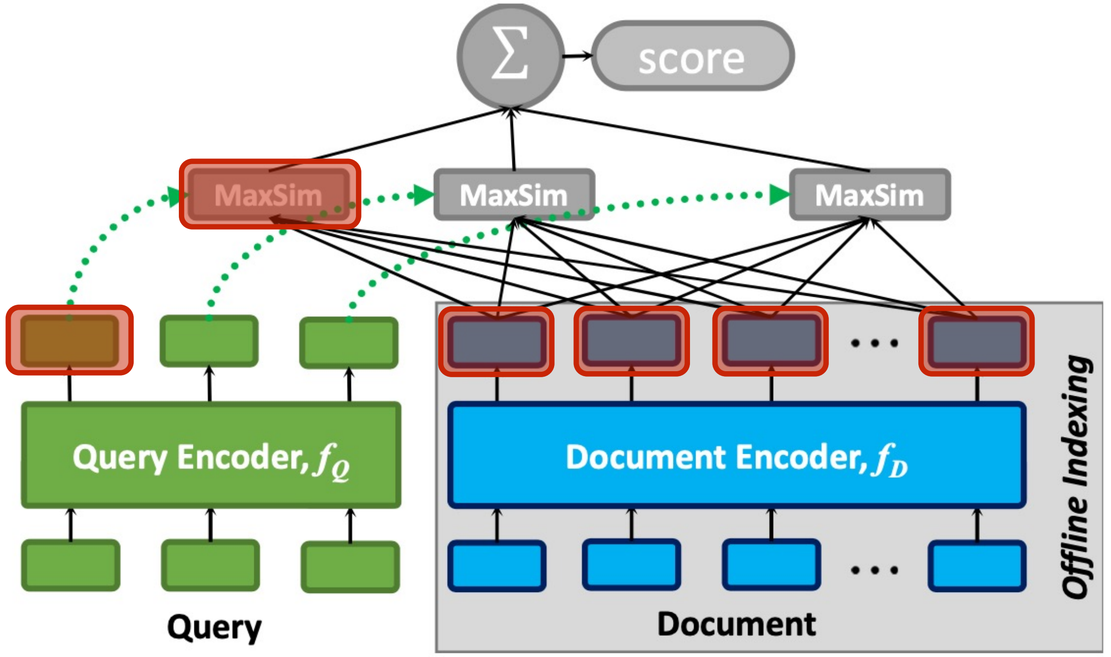


Figure 3: The general architecture of ColBERT given a query q and a document d.

Khattab, O., Zaharia, M.: Colbert: Efficient and effective passage search via contextualized late interaction over BERT. SIGIR'20

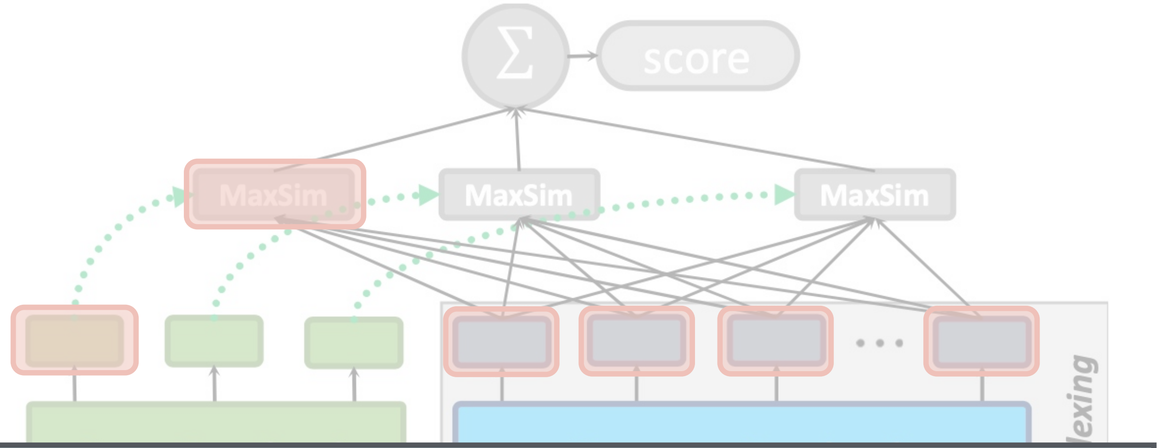
1.3 ColBERT architecture

DEVIEW 2021

MaxSim operation

질의 임베딩(벡터)과 가장
유사도가 높은 문서 임베딩(벡터)

score = ∑ MaxSim

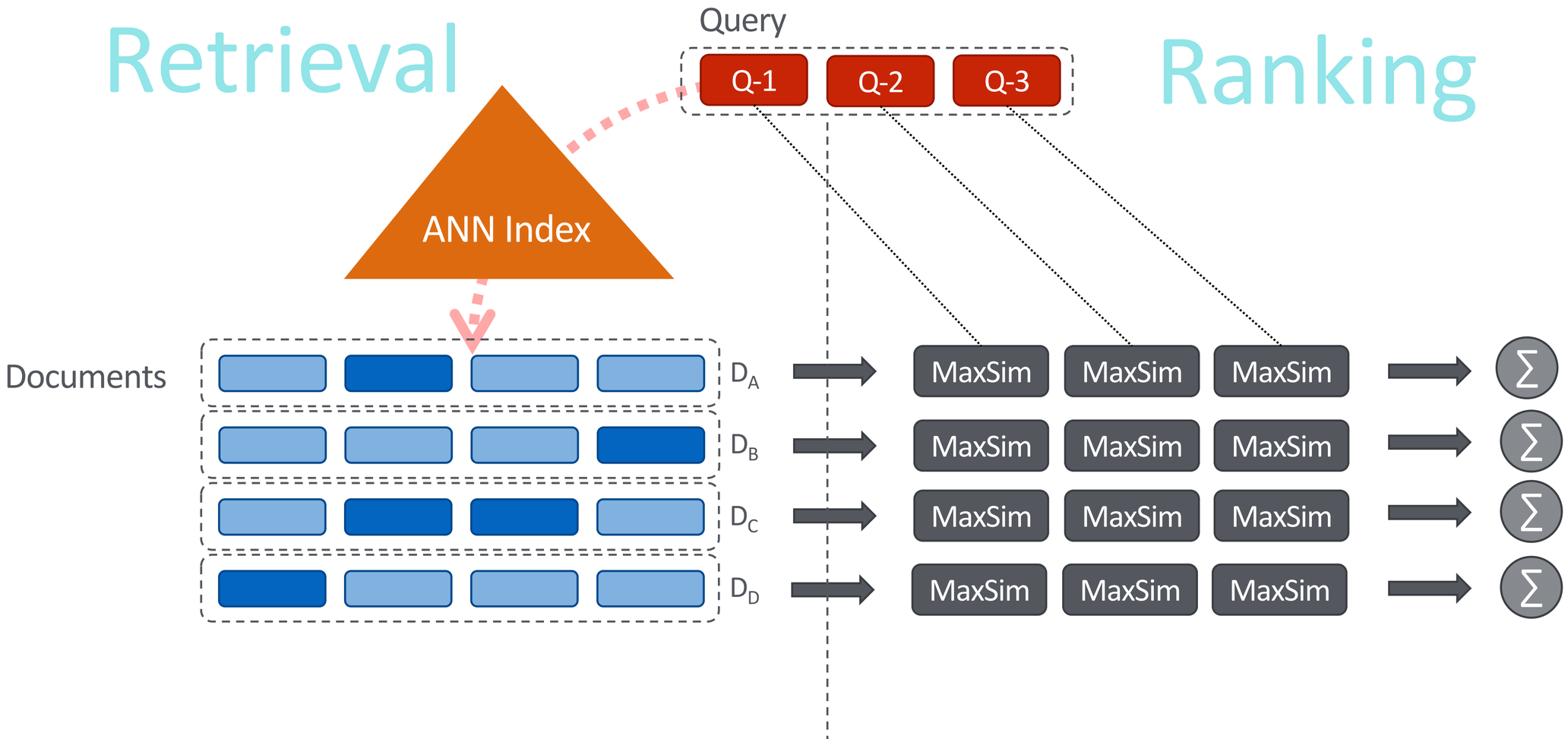


ColBERT의 랭킹 방식은 (BM25랭킹의 IDF처럼)
중요한 텀에 높은 가중치를 부여하는 효과

Thibault Formal et al., “A White Box Analysis of ColBERT”, ECIR 2021

1.4 ColBERT e2e

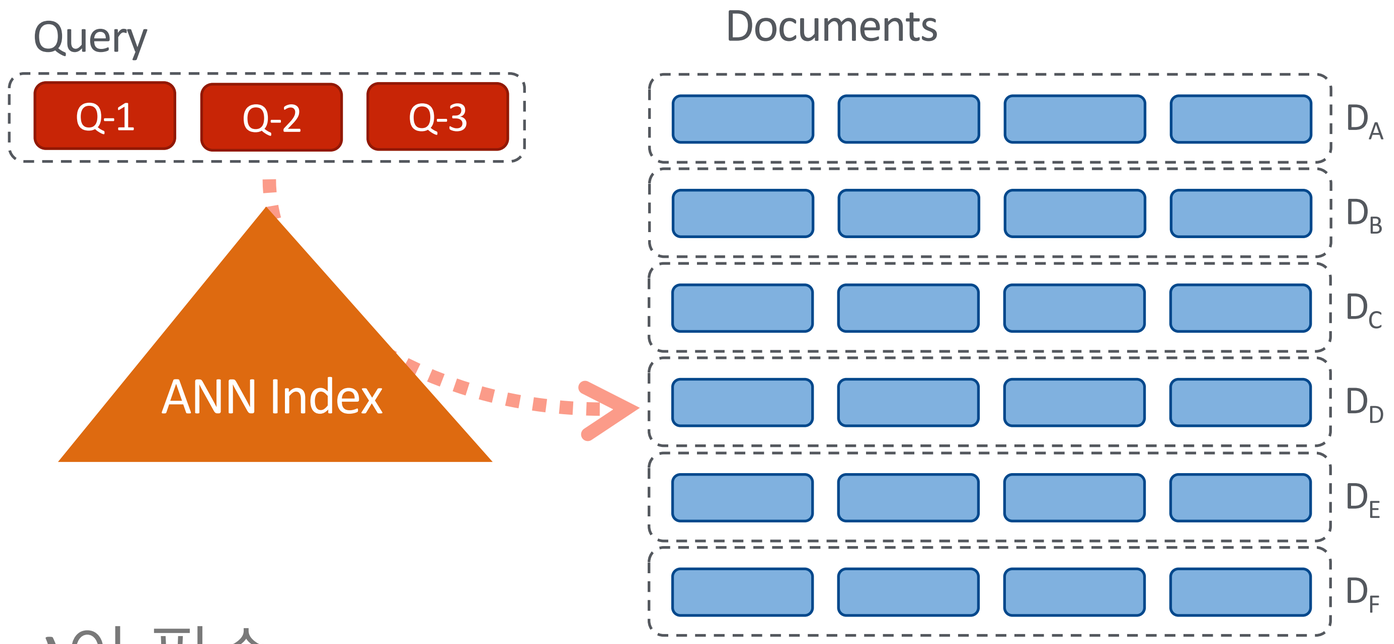
ColBERT e2e = Retrieval + Ranking



1.5 ColBERT Retrieval

ColBERT Retrieval

- ColBERT score가 높을 가능성이 큰 후보군을 선별하는 과정



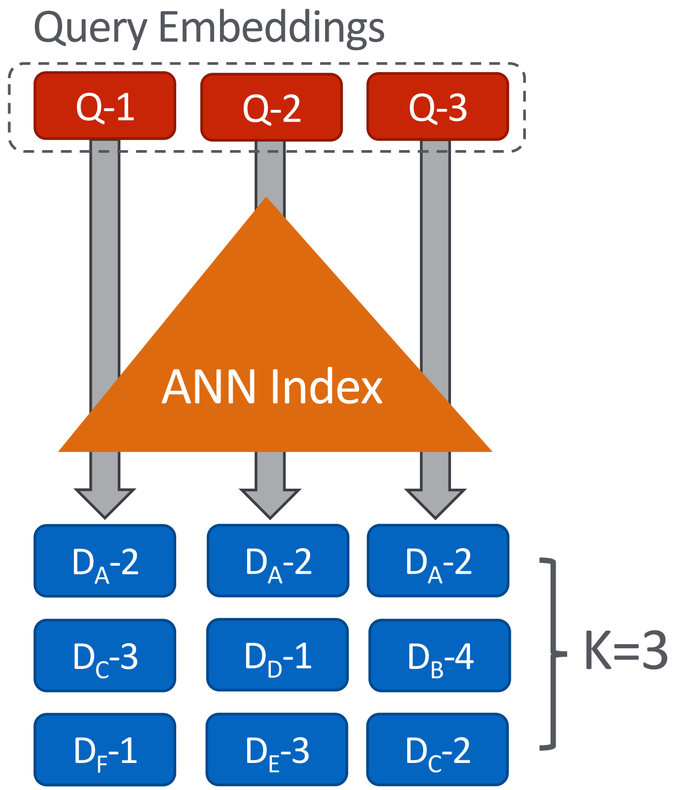
- 빠른 Retrieval을 위해

ANN(Approximate Nearest Neighbors)이 필수

1.5 ColBERT Retrieval

ColBERT Retrieval

1. 질의 임베딩과 유사한
   문서 임베딩을 ANN에서 찾음

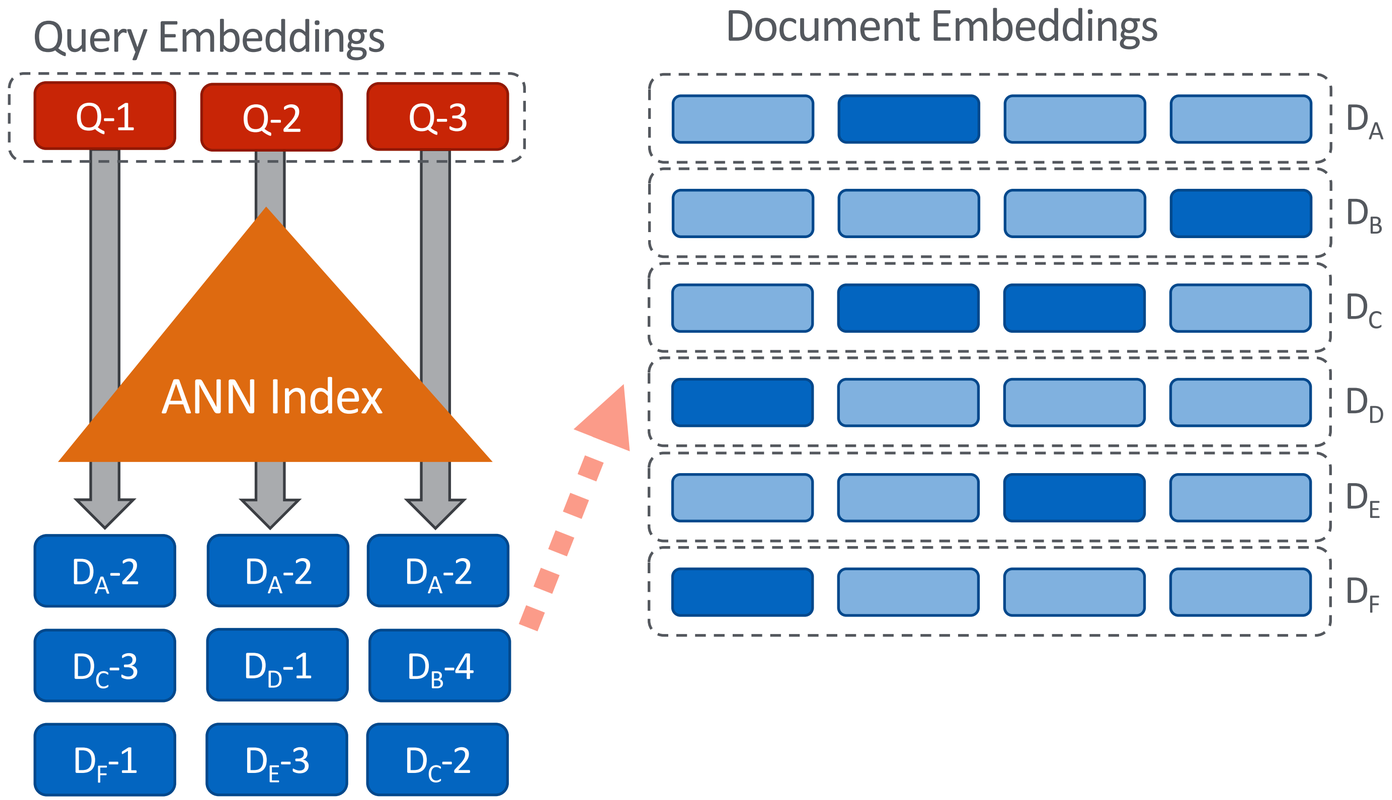


1.5 ColBERT Retrieval

ColBERT Retrieval

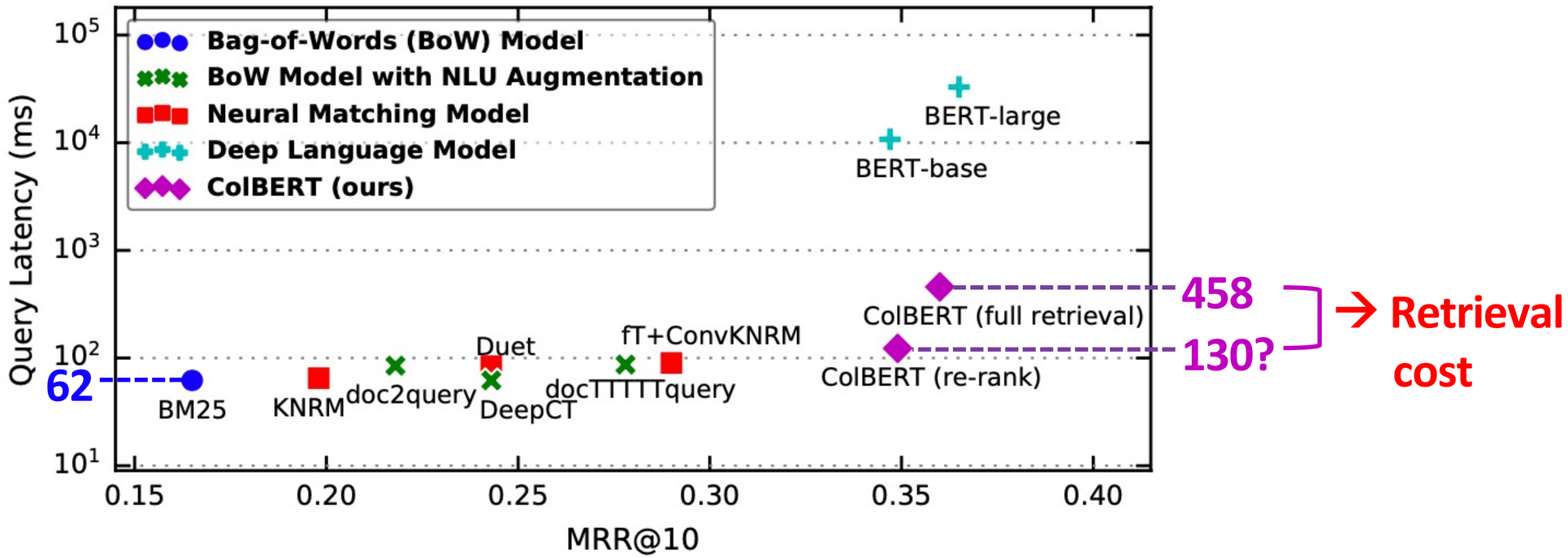
1. 질의 임베딩과 유사한
   문서 임베딩을 ANN에서 찾음

2. 문서 임베딩이 나온
   모든 문서의 목록을 구함 (union)



1.5 ColBERT Retrieval

Retrieval이 ColBERT에서 가장 큰 비용을 차지함



Khattab, O., Zaharia, M.: Colbert: Efficient and effective passage search via contextualized late interaction over BERT. SIGIR'20

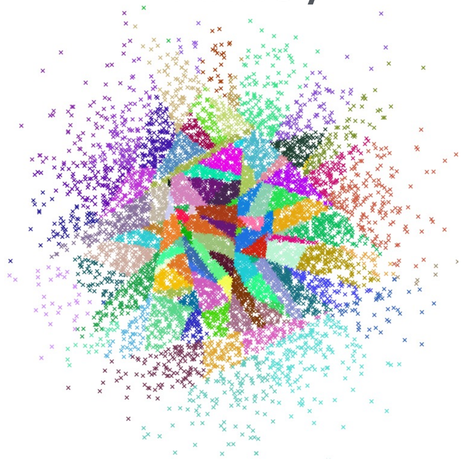
2. ColBERT Retrieval 핵심기술 : ANN

2.1 ANN?

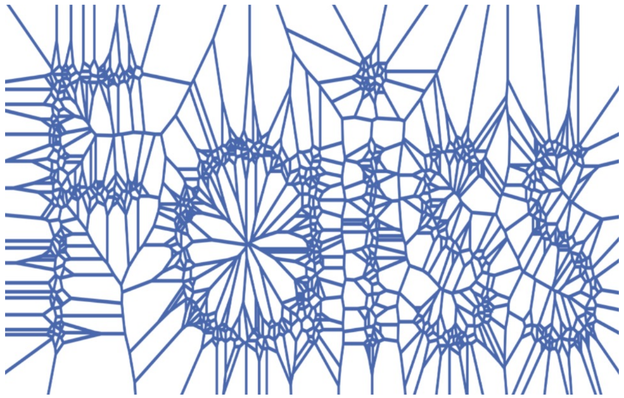
ANN : Approximate Nearest Neighbors

- 근사적으로 가까운 벡터를 찾는 알고리즘
- 정확한 탐색을 포기하는 대신에 매우 빠르게 벡터를 찾을 수 있음

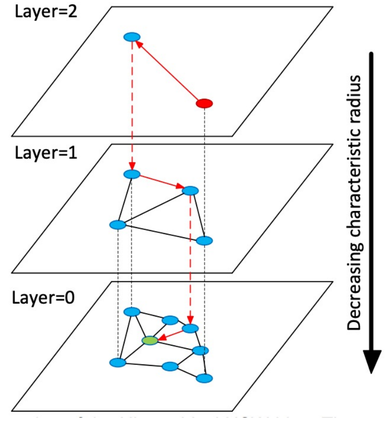
Annoy



Faiss



Hnswlib



https://github.com/spotify/annoy

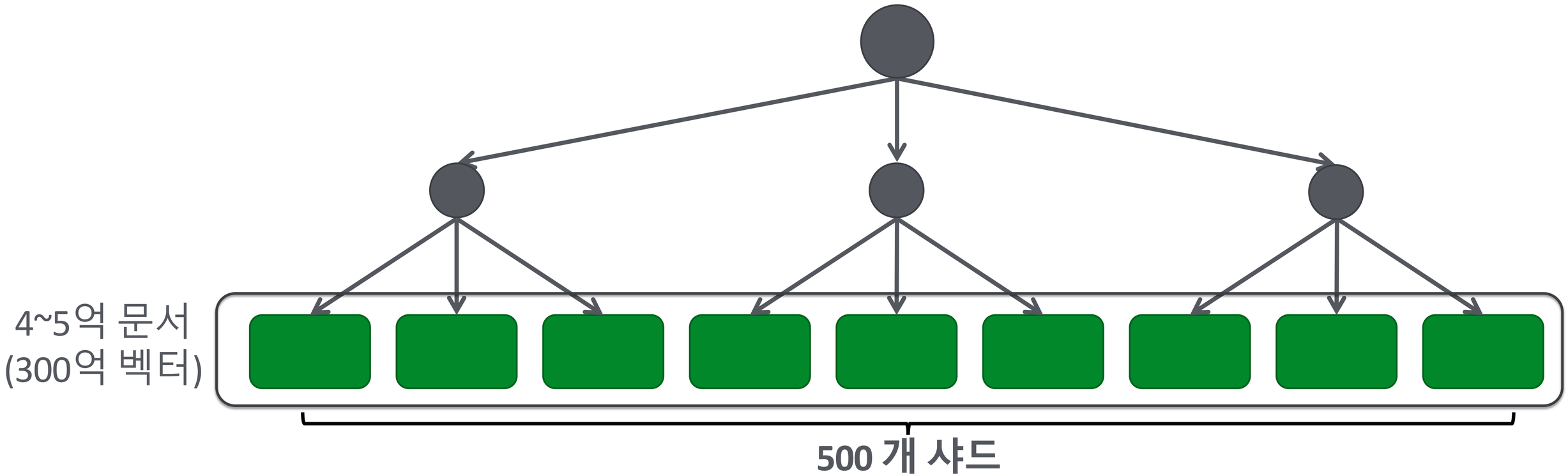
https://engineering.fb.com/2017/03/29/data-infrastructure/faiss-a-library-for-efficient-similarity-search/

Yu. A. Malkov, D. A. Yashunin. Efficient and robust approximate nearest neighbor search using Hierarchical Navigable Small World graphs. 2016.

2.2 ANN Index 크기

분산 검색 시스템

여러 장비로 전체 데이터를 분산

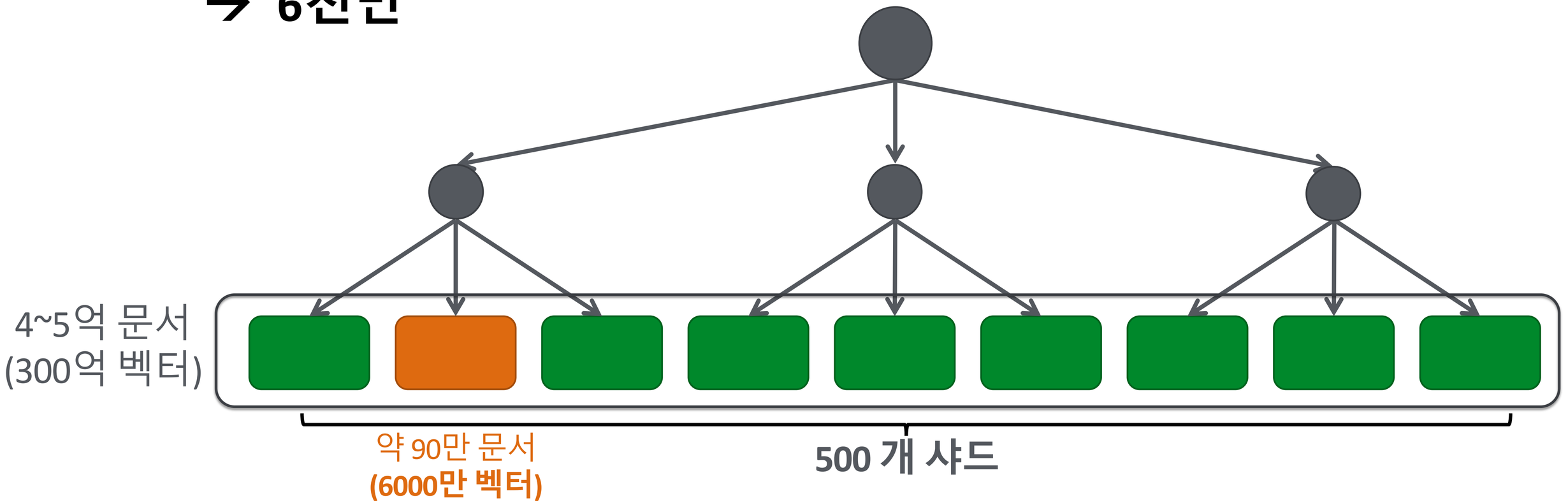


2.2 ANN Index 크기

분산 검색 시스템

300억 ÷ 500

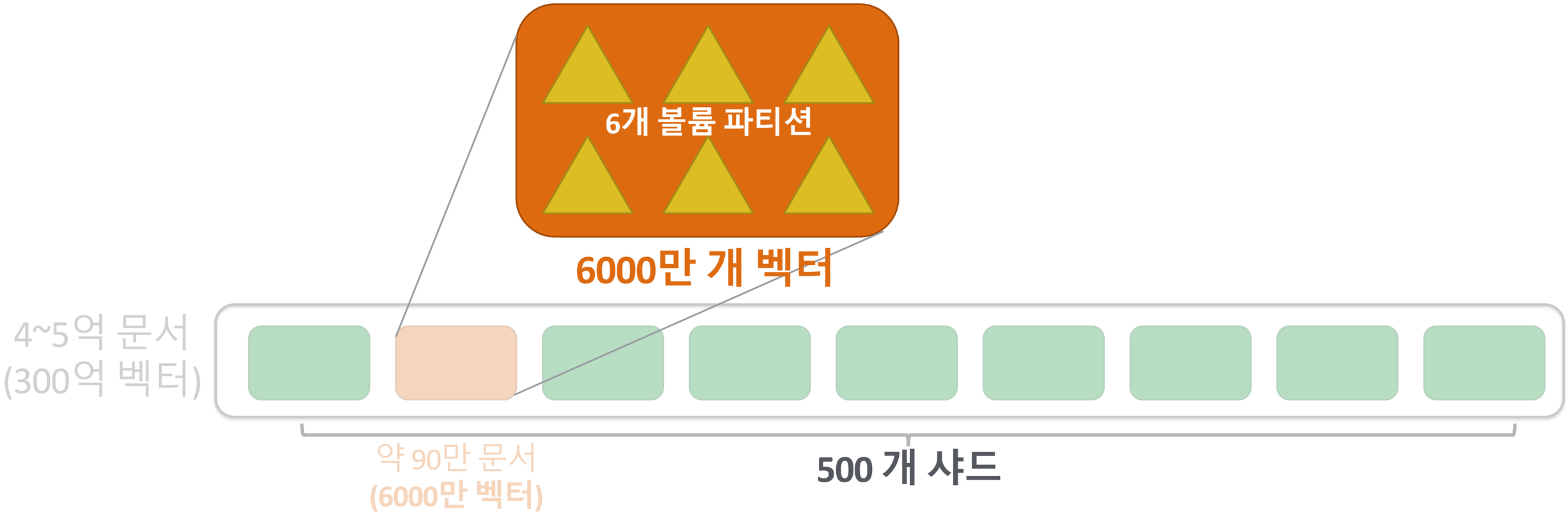
→ 6천만



2.2 ANN Index 크기

증분을 위해 작은 단위로 나눠서 저장함

6천만 ÷ 6

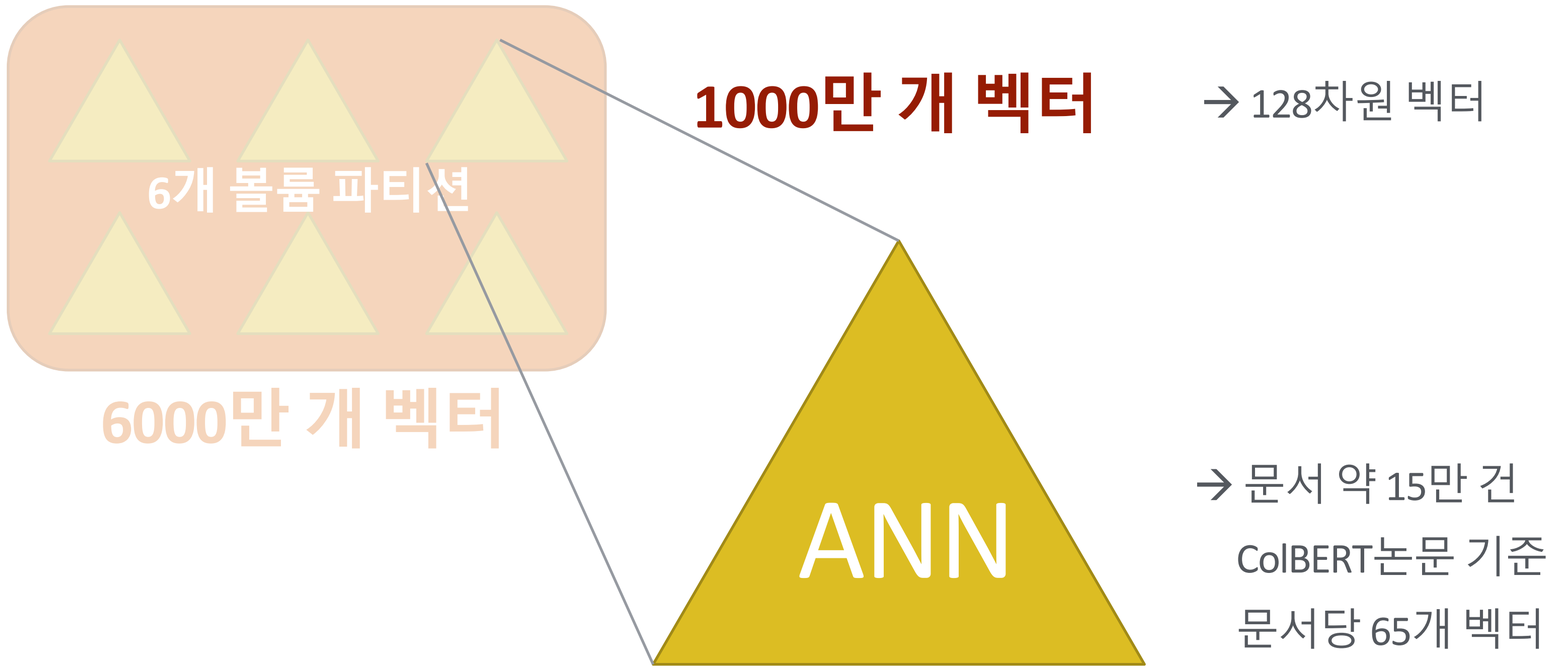


2.2 ANN Index 크기

단위 ANN Index 크기

6천만 ÷ 6

→ 1000만



2.3 ANN 응답 시간

검색 유입이 1500 QPS일 때, 검색 응답 시간

동시 처리 가능 리소스(코어) 수

4 x 24 → 96

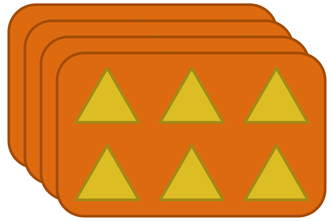
코어 당 QPS

1500 / 96 → 15.6

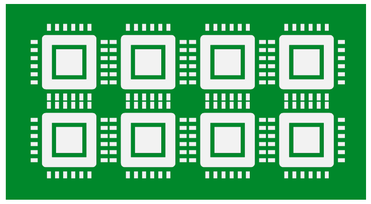
검색 응답 시간

1 sec / 15.6 → 64 ms

복제 서버 × 4 ⋯⋯→ **장비 2000대**
= 500 x 4



Multi-core × 24



2.3 ANN 응답 시간

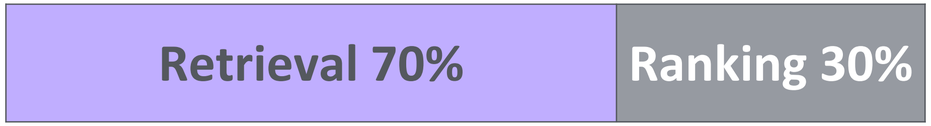
검색 유입이 1500 QPS일 때, ColBERT Retrieval 응답 시간

검색 응답시간 : 64 ms

ColBERT Retrieval 응답 시간

64 ms × 70% → 44.8 ms

ColBERT e2e 수행 시간 비율 (예상)



2.3 ANN 응답 시간

질의 당 ANN 검색 횟수

ColBERT 질의 당 임베딩 수 : 16

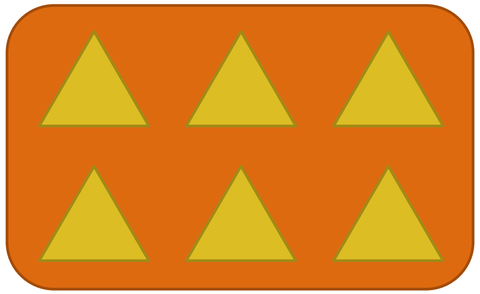
ANN Index 수 : 6

질의 당 ANN 검색 횟수

16 x 6 → 96

Query Embeddings × 16

ANN Indexes × 6



2.3 ANN 응답 시간

검색 유입이 1500 QPS일 때, ANN 응답 시간

ColBERT Retrieval 응답 시간 : 44.8 ms

질의 당 ANN 검색 횟수 : 96

ANN 응답 시간

44.8 ms / 96 → 0.47 ms

1500 QPS

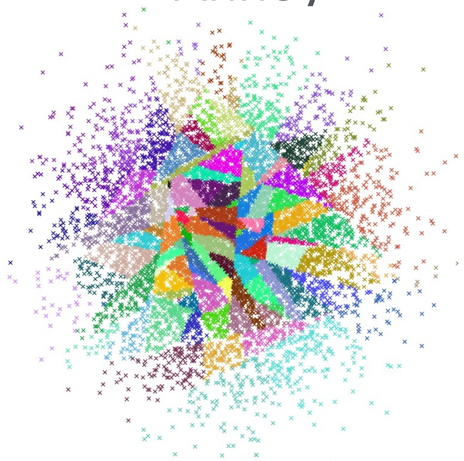
장비 2000대 (500 x 4)

문서 4 ~ 5억 (벡터 300억)

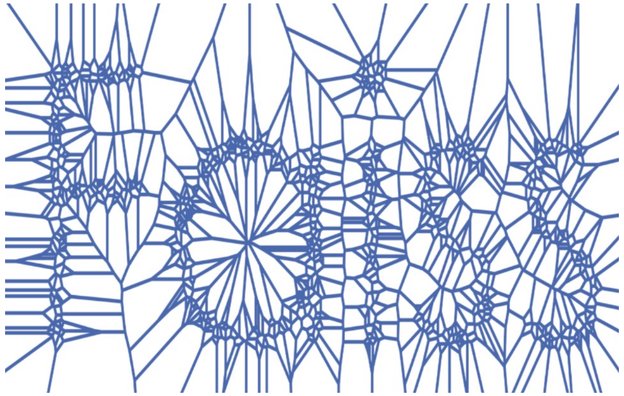
2.4 ANN 선택은?

DEVIEW 2021

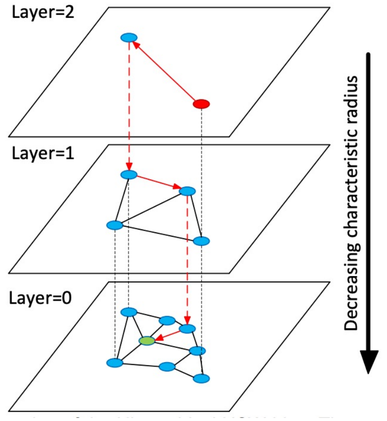
Annoy



Faiss



Hnswlib



https://github.com/spotify/annoy

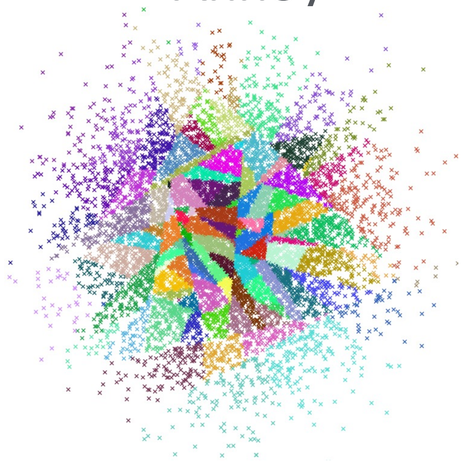
https://engineering.fb.com/2017/03/29/data-infrastructure/faiss-a-library-for-efficient-similarity-search/

Yu. A. Malkov, D. A. Yashunin. Efficient and robust approximate nearest neighbor search using Hierarchical Navigable Small World graphs. 2016.

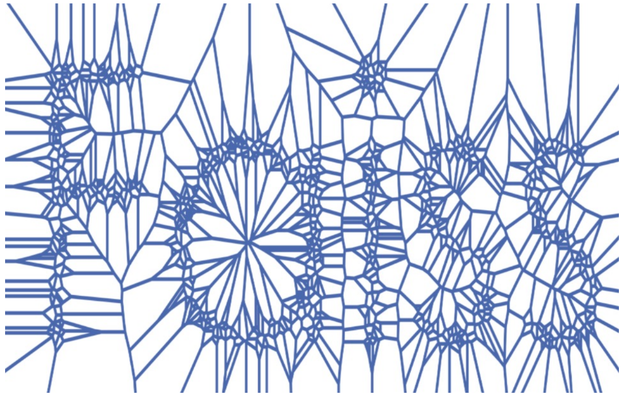
2.4 ANN 선택은?

DEVIEW 2021

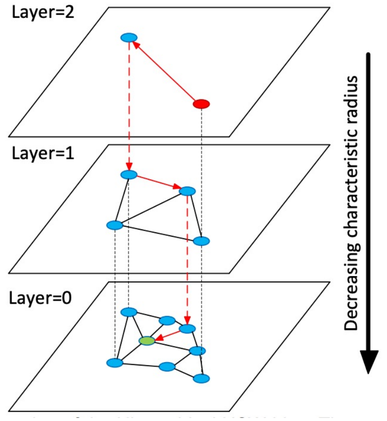
Annoy



Faiss



Hnswlib



🙂 인터페이스가 간단하고
파라미터 튜닝이 쉬움

https://github.com/spotify/annoy

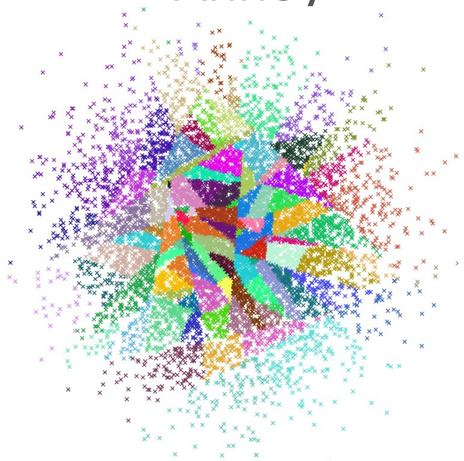
https://engineering.fb.com/2017/03/29/data-infrastructure/faiss-a-library-for-efficient-similarity-search/

Yu. A. Malkov, D. A. Yashunin. Efficient and robust approximate nearest neighbor search using Hierarchical Navigable Small World graphs. 2016.

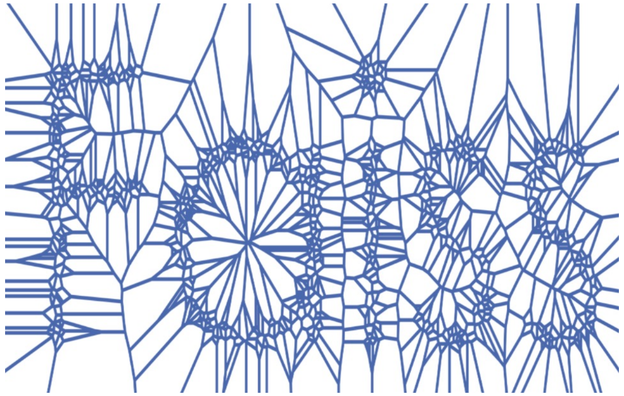
2.4 ANN 선택은?

DEVIEW 2021

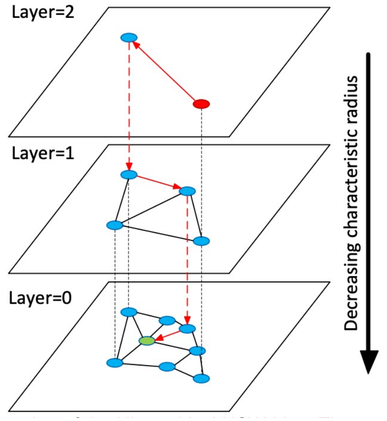
Annoy



Faiss



Hnswlib



🙂 인터페이스가 간단하고
파라미터 튜닝이 쉬움

😡 **탐색 속도 느림**

https://github.com/spotify/annoy

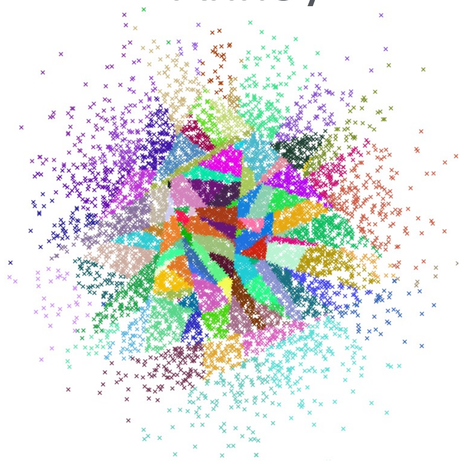
https://engineering.fb.com/2017/03/29/data-infrastructure/faiss-a-library-for-efficient-similarity-search/

Yu. A. Malkov, D. A. Yashunin. Efficient and robust approximate nearest neighbor search using Hierarchical Navigable Small World graphs. 2016.

2.4 ANN 선택은?

DEVIEW 2021

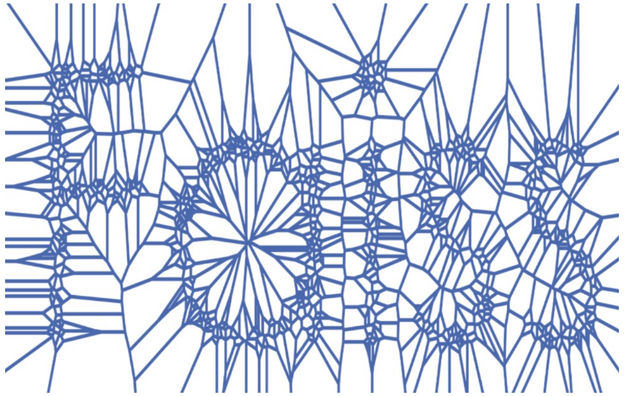
Annoy



🙂 인터페이스가 간단하고
파라미터 튜닝이 쉬움

😡 **탐색 속도 느림**

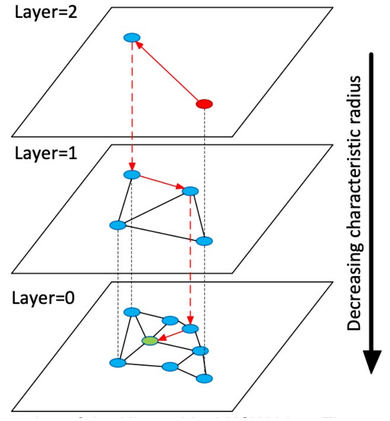
Faiss



😮 ColBERT 논문에서 사용
(faiss IVFPQ)

🙂 다양한 ANN 알고리즘 지원

Hnswlib



https://github.com/spotify/annoy

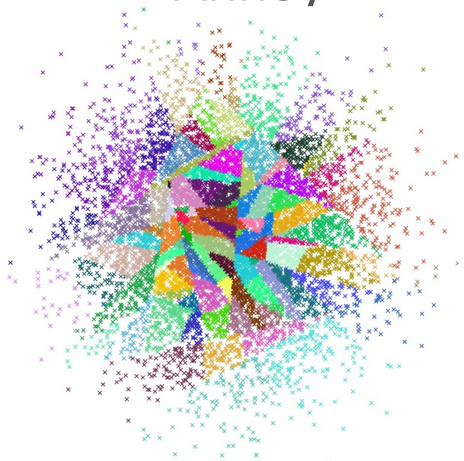
https://engineering.fb.com/2017/03/29/data-infrastructure/faiss-a-library-for-efficient-similarity-search/

Yu. A. Malkov, D. A. Yashunin. Efficient and robust approximate nearest neighbor search using Hierarchical Navigable Small World graphs. 2016.

2.4 ANN 선택은?

DEVIEW 2021

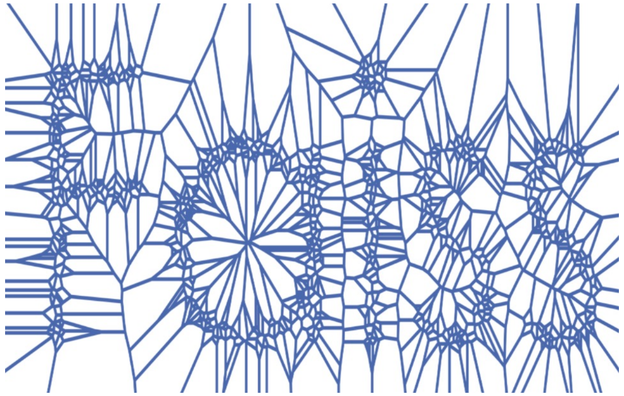
Annoy



🙂 인터페이스가 간단하고
파라미터 튜닝이 쉬움

😡 **탐색 속도 느림**

Faiss

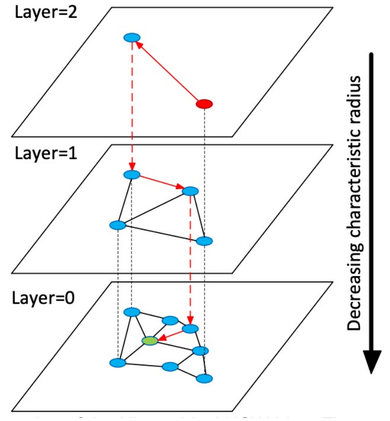


😮 ColBERT 논문에서 사용
(faiss IVFPQ)

🙂 다양한 ANN 알고리즘 지원

😡 **탐색 속도 느림** **≥ 4 ms**

Hnswlib



https://github.com/spotify/annoy

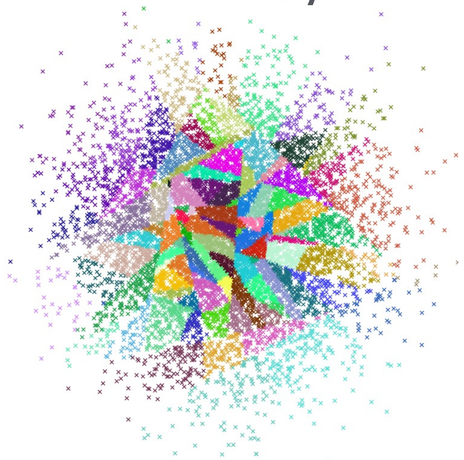
https://engineering.fb.com/2017/03/29/data-infrastructure/faiss-a-library-for-efficient-similarity-search/

Yu. A. Malkov, D. A. Yashunin. Efficient and robust approximate nearest neighbor search using Hierarchical Navigable Small World graphs. 2016.

2.4 ANN 선택은?

DEVIEW 2021

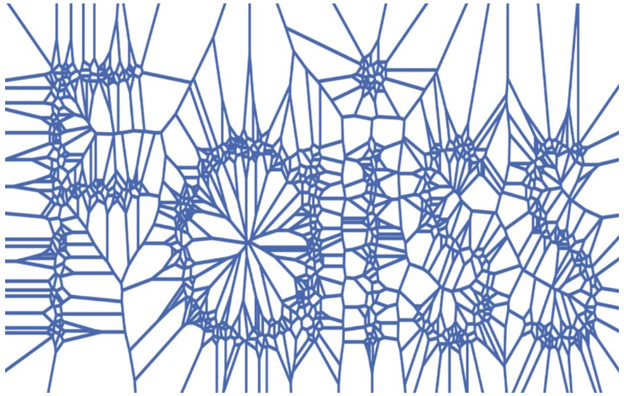
Annoy



🙂 인터페이스가 간단하고
파라미터 튜닝이 쉬움

😡 **탐색 속도 느림**

Faiss

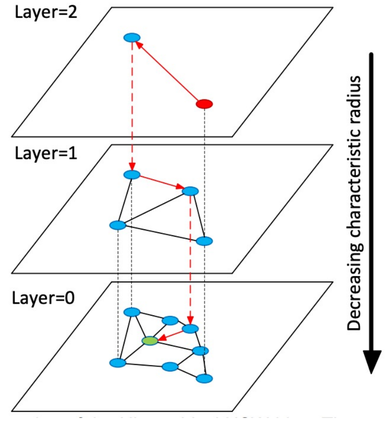


😮 ColBERT 논문에서 사용
(faiss IVFPQ)

🙂 다양한 ANN 알고리즘 지원

😡 **탐색 속도 느림** **≥ 4 ms**

Hnswlib



🙂 **빠른 검색** **≤ 0.2 ms**

https://github.com/spotify/annoy

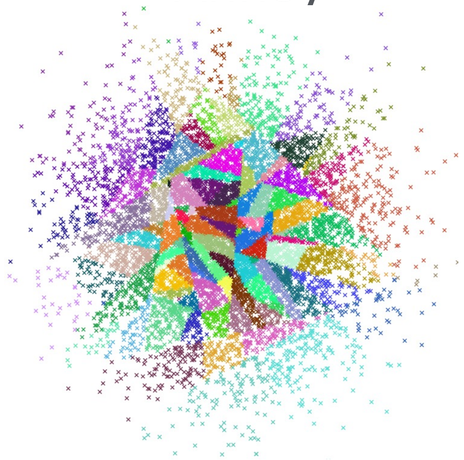
https://engineering.fb.com/2017/03/29/data-infrastructure/faiss-a-library-for-efficient-similarity-search/

Yu. A. Malkov, D. A. Yashunin. Efficient and robust approximate nearest neighbor search using Hierarchical Navigable Small World graphs. 2016.

2.4 ANN 선택은?

DEVIEW 2021

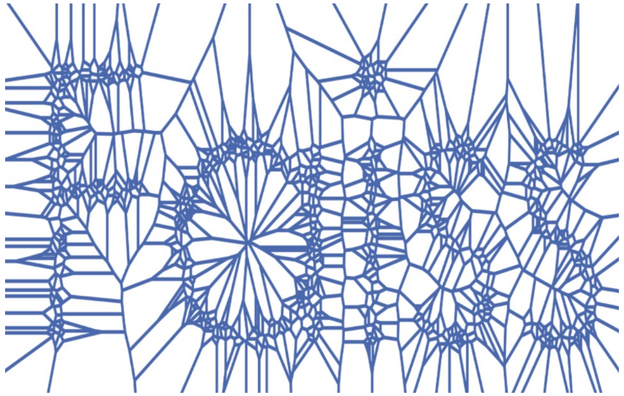
Annoy



🙂 인터페이스가 간단하고
파라미터 튜닝이 쉬움

😡 **탐색 속도 느림**

Faiss

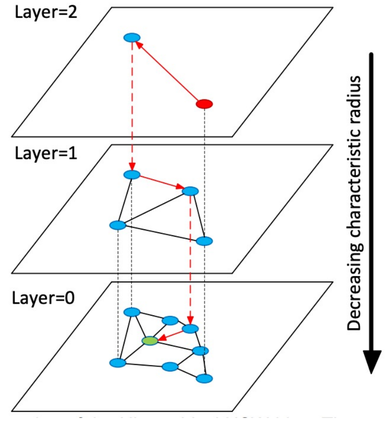


😮 ColBERT 논문에서 사용
(faiss IVFPQ)

🙂 다양한 ANN 알고리즘 지원

😡 **탐색 속도 느림** **≥ 4 ms**

Hnswlib



🙂 **빠른 검색** **≤ 0.2 ms**

😡 느린 빌딩 속도 **≈ 1 hour**

😡 mmap 지원 안 함

https://github.com/spotify/annoy

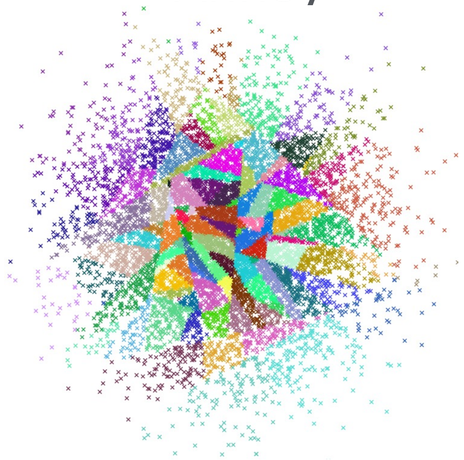
https://engineering.fb.com/2017/03/29/data-infrastructure/faiss-a-library-for-efficient-similarity-search/

Yu. A. Malkov, D. A. Yashunin. Efficient and robust approximate nearest neighbor search using Hierarchical Navigable Small World graphs. 2016.

2.4 ANN 선택은?

DEVIEW 2021

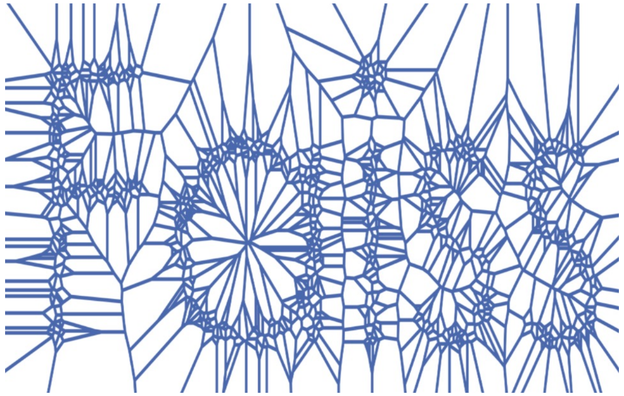
Annoy



🙂 인터페이스가 간단하고
파라미터 튜닝이 쉬움

😡 **탐색 속도 느림**

Faiss

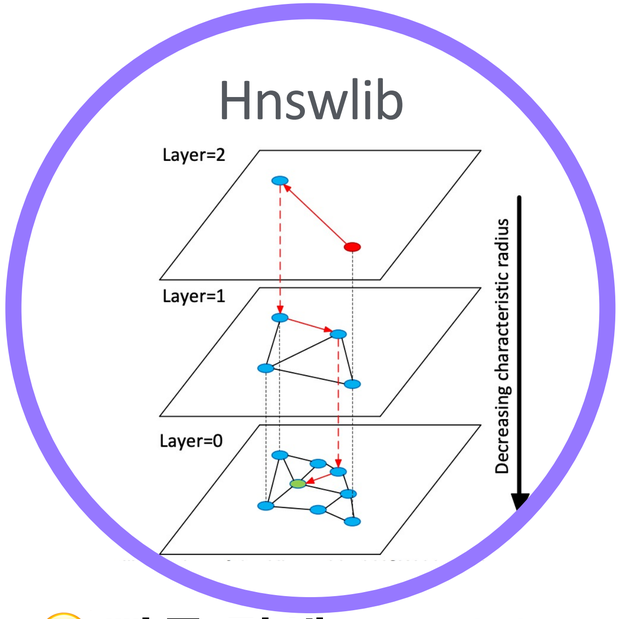


😮 ColBERT 논문에서 사용
(faiss IVFPQ)

🙂 다양한 ANN 알고리즘 지원

😡 **탐색 속도 느림** **≥ 4 ms**

Hnswlib



🙂 **빠른 검색** **≤ 0.2 ms**

😡 느린 빌딩 속도 **≈ 1 hour**

😡 mmap 지원 안 함

https://github.com/spotify/annoy

https://engineering.fb.com/2017/03/29/data-infrastructure/faiss-a-library-for-efficient-similarity-search/

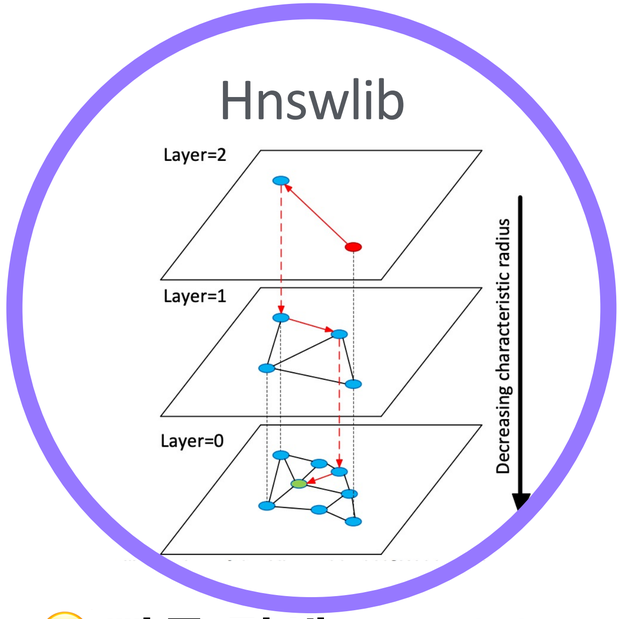
Yu. A. Malkov, D. A. Yashunin. Efficient and robust approximate nearest neighbor search using Hierarchical Navigable Small World graphs. 2016.

2.4 ANN 선택은?

빌딩 병렬화 지원

- 여러 스레드가 동시에 벡터를 추가할 수 있음

Thread 수	Build time
8	1 / 7
16	**1 / 12**
32	1 / 16



🙂 **빠른 검색** **≤ 0.2 ms**

~~😡 느린 빌딩 속도 ≈ 1 hour~~

≈ 5 min

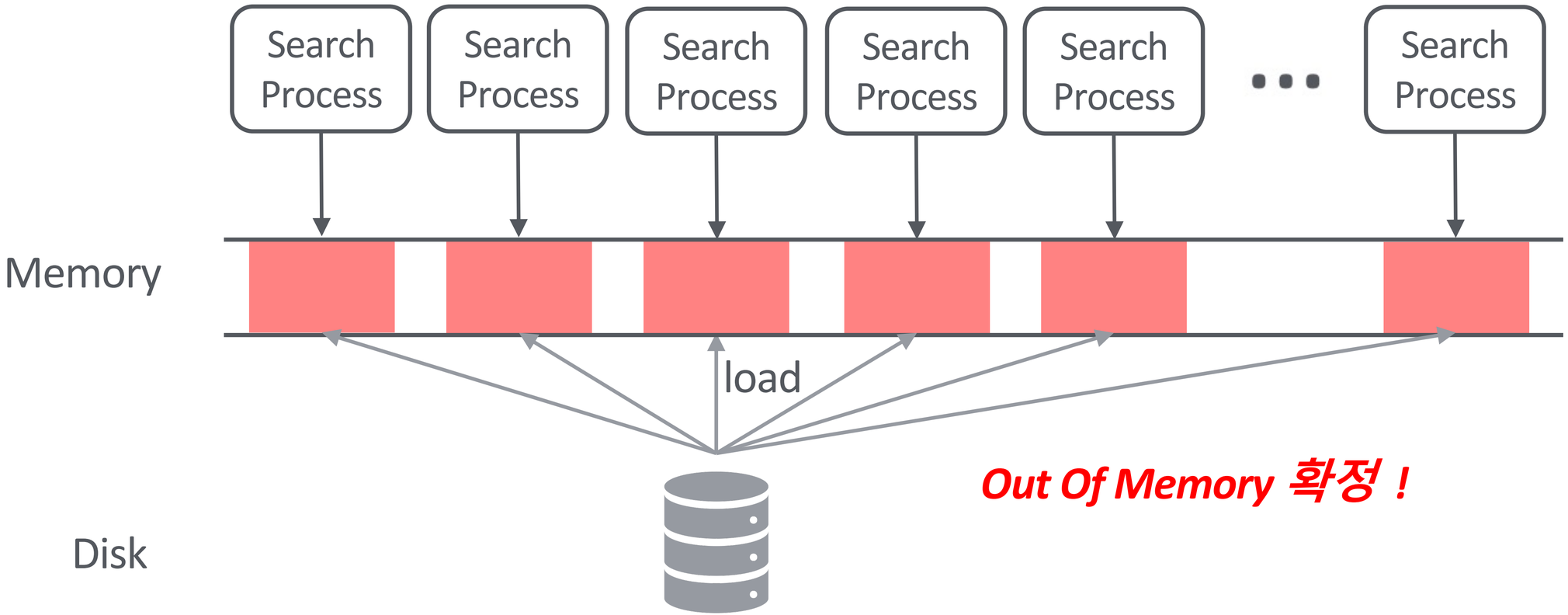
😡 mmap 지원 안 함

Yu. A. Malkov, D. A. Yashunin. Efficient and robust approximate nearest neighbor search using Hierarchical Navigable Small World graphs. 2016.

3. Customizing ANN

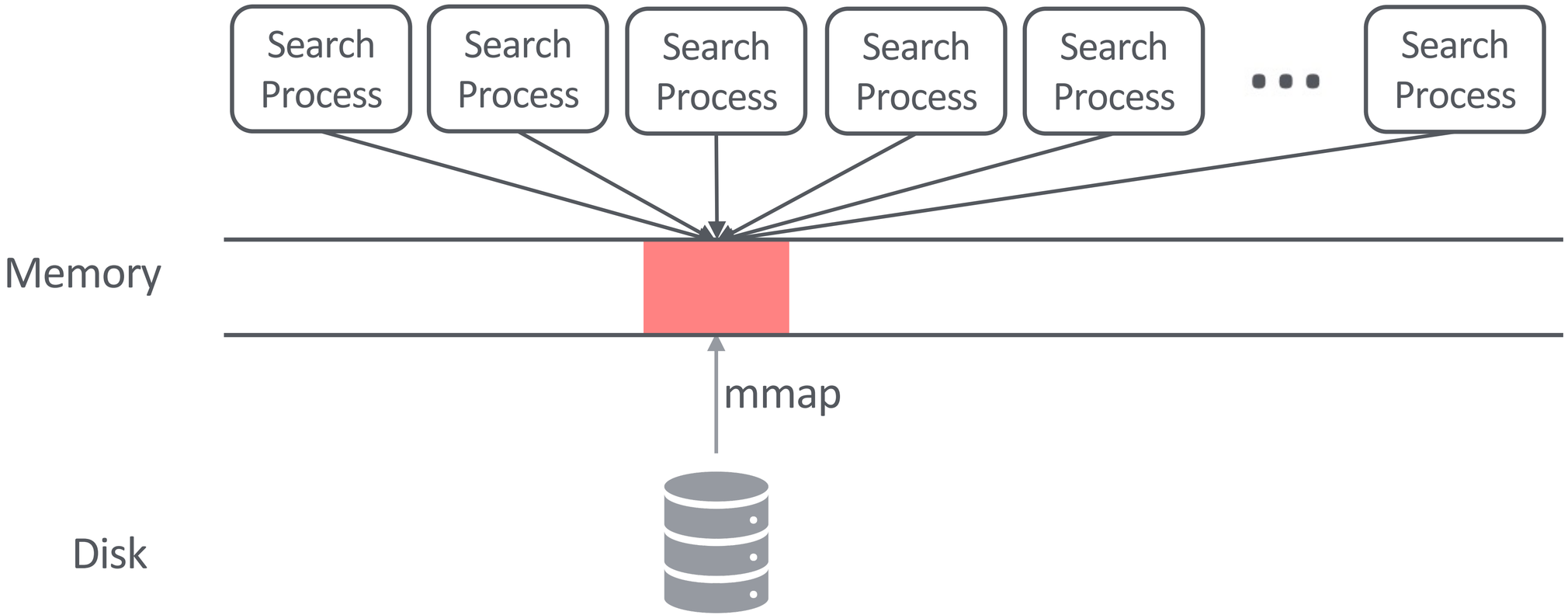
3.1 hnswlib original

mmap을 지원하지 않을 때



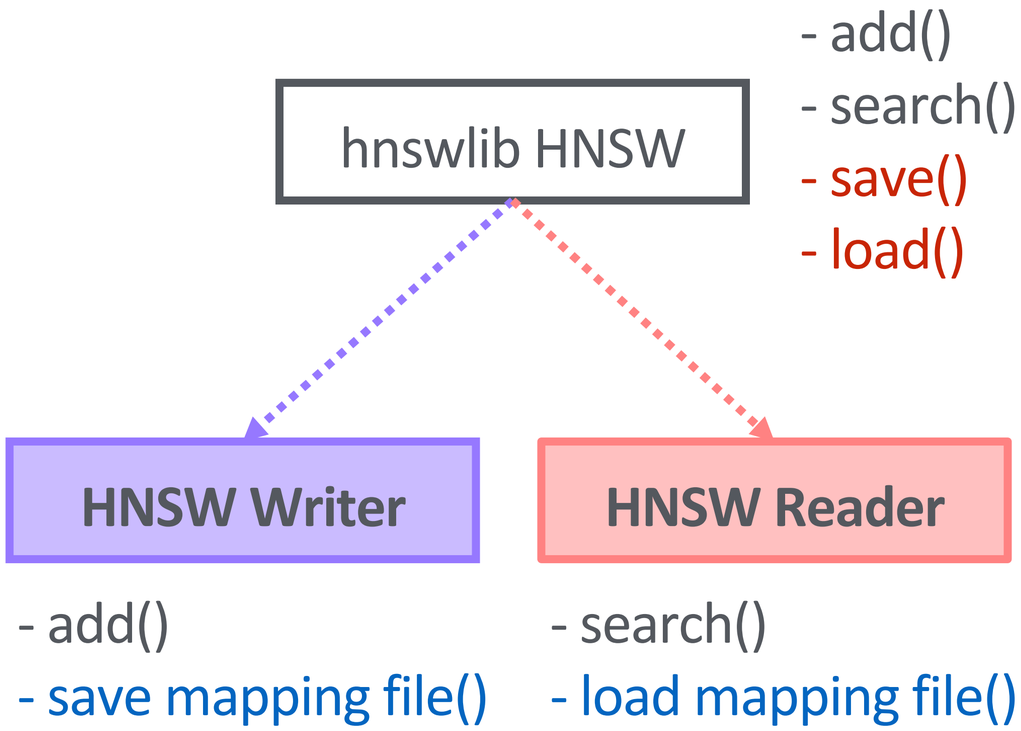
3.1 hnswlib original

mmap을 지원할 때



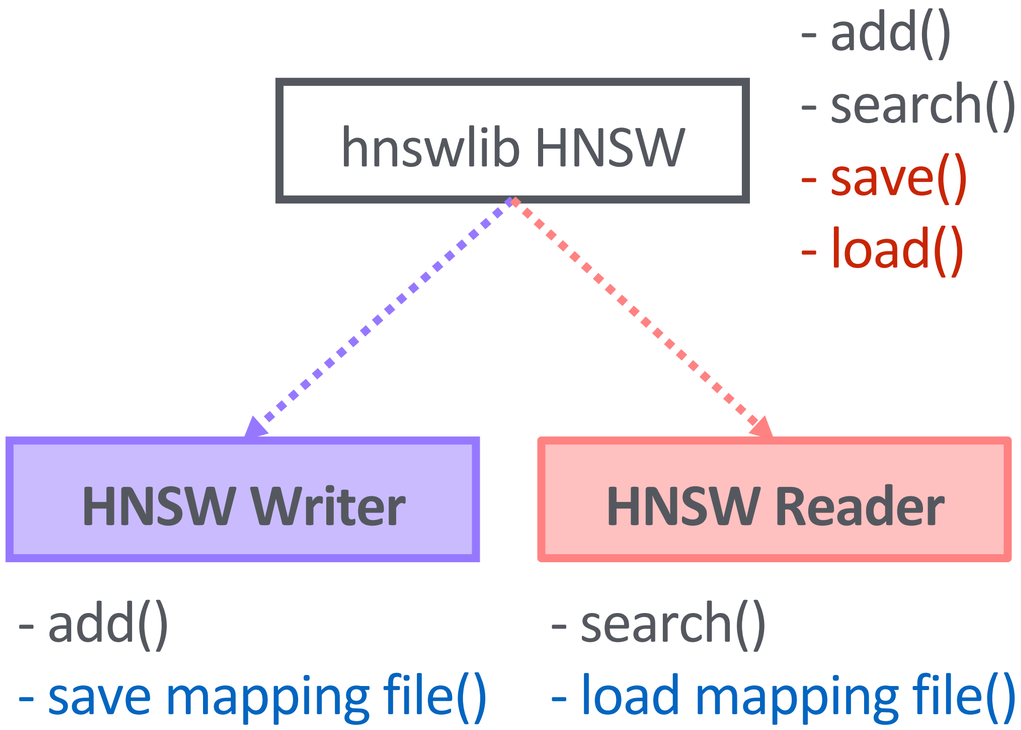
3.2 hnswlib + **mmap**

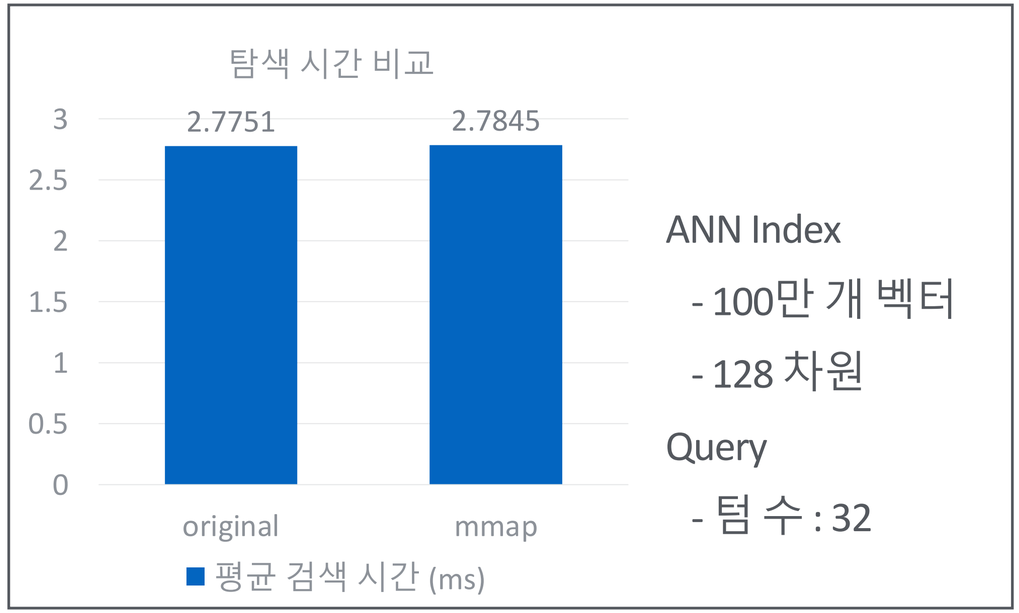
mapping file을 매개로 Writer/Reader 분리



3.2 hnswlib + **mmap**

mmap 버전 성능 차이 없음



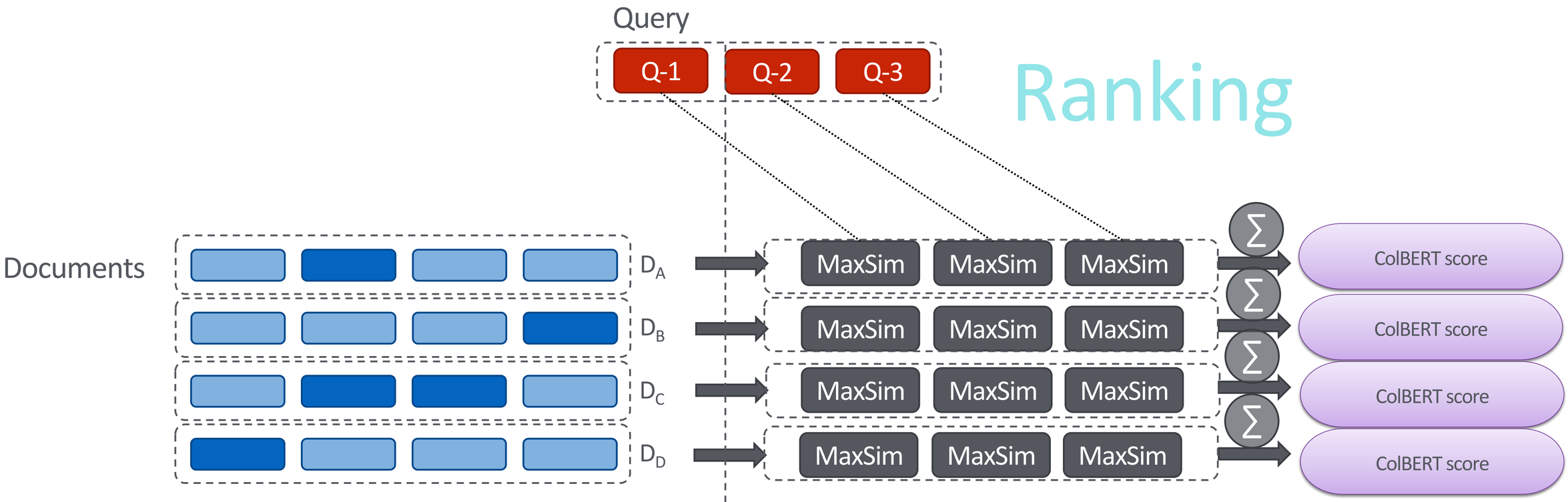


4. ColBERT 계산 복잡도

4.1 ColBERT score

Very expensive feature

- Sum of maximum similarity computation cost ≃
  query vector count, document vector count, dimension



4.2 BM25F 모델 소개

BM25

- 통계를 기반으로 하는 모델
- TF (Term Frequency), Dlen (Document Length), IDF(Inverted Document Frequency)

BM25F

- BM25와 달리 섹션 별로 정규화 및 파라미터를 적용
- 섹션은 정적이나 동적으로 제공될 수 있음

$$BM25 = dotProd(TermWeight, Imp)$$

$$Imp_s = normalization(\boldsymbol{IDF_s})$$

$$TermWeight_s = \frac{\boldsymbol{TF_s}\ (k + 1.0)}{TF_s + k(1.0\ - b + \frac{\boldsymbol{Dlen}}{avgDlen})}$$

4.3 BM25F score

$$bm25f = \frac{DotProd\left(\frac{STW_{sum}}{STW_{sum} + coef_{norm}}, Imp\right)}{QtermCount}$$

weight

(Pre-computable part)

$$STW_{sum} = STW_1 + \cdots$$

$$STW_1 = k_s \frac{secTF_1}{1.0 - b_s(1 - \frac{slen + 1.0}{avgSlen_1 + 1.0})}$$

Cheap static BM25F

- Computation cost ≃

  query term count

- Using Postings (Inverted index)

Expensive dynamic BM25F

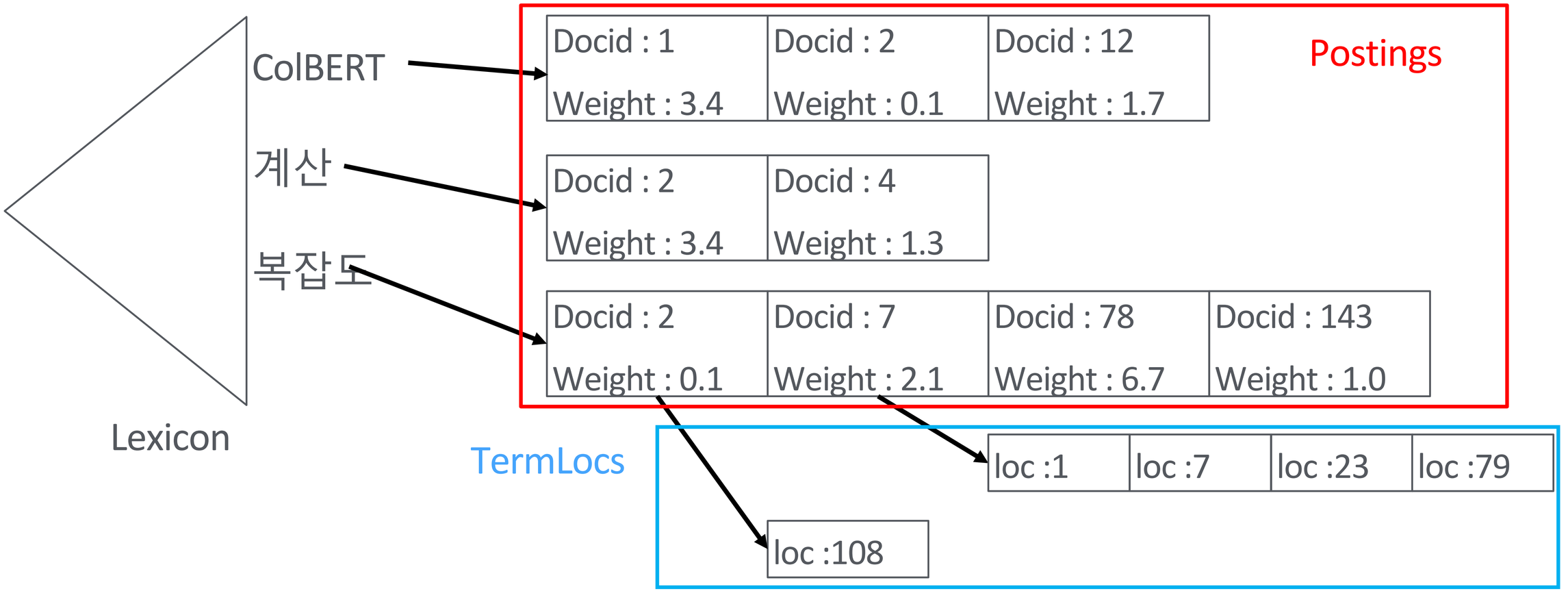
- Computation cost ≃

  query term count, term frequency, section count

- Using Termlocs (Inverted index)

4.4 Inverted Index structure

Inverted Index 구조를 통해 빠르게 BM25F 계산 가능

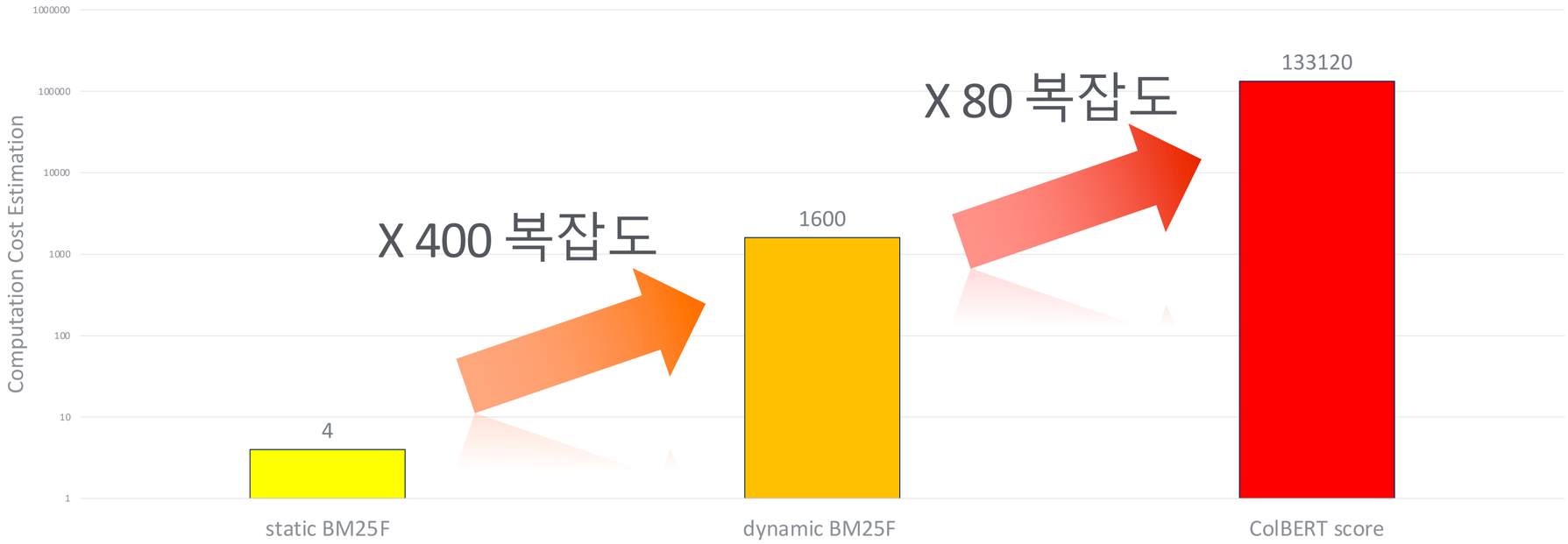


4.5 Computational Complexity

BM25F VS ColBERT

Assume

- query term count : 4 , section count : 8 , term frequency : 50
- query vector count : 16, document vector count : 65, dimension : 128



≃ query term count

≃ query term count
term frequency
section count

≃ query vector count
document vector count
dimension

4.6 Computation Complexity and Cluster Size

DEVIEW 2021

Assume

- 1500 QPS required
- Number of matched docs or candidates = 25만 Docs (300억 벡터에 대한 문서수의 0.5%)

Static BM25F

Dynamic BM25F

수십~수백 Tera flops 연산 가능한 클러스터

ColBERT

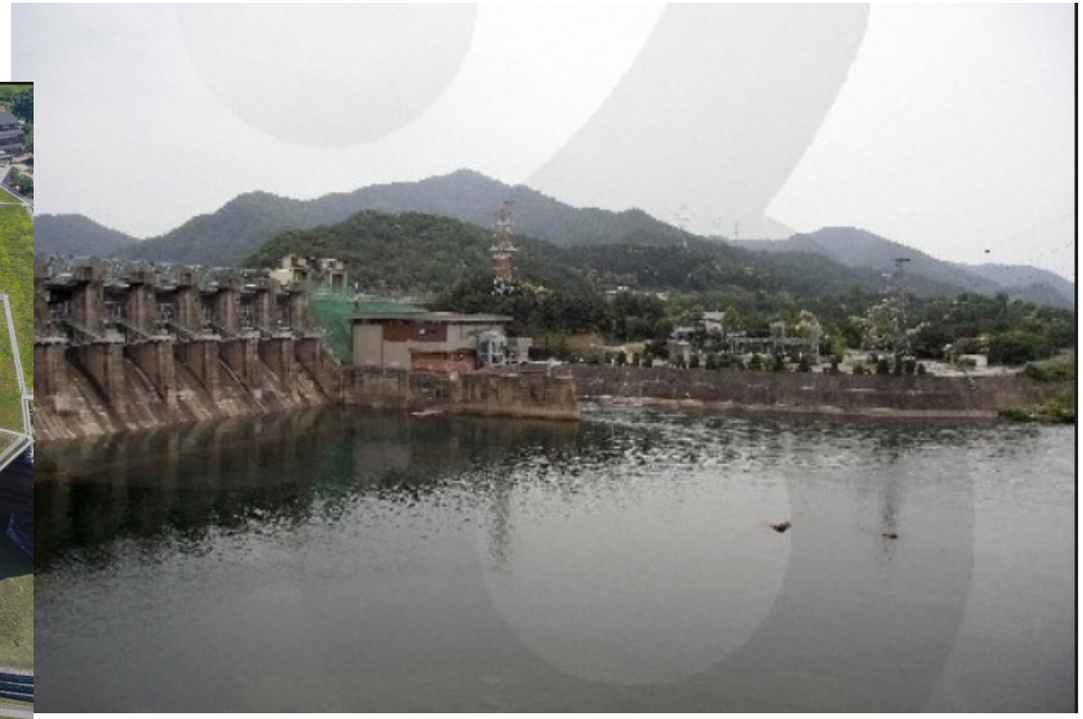
Peta flops 연산이 가능한 클러스터

X 수십 ~ 수백 규모 증가 필요

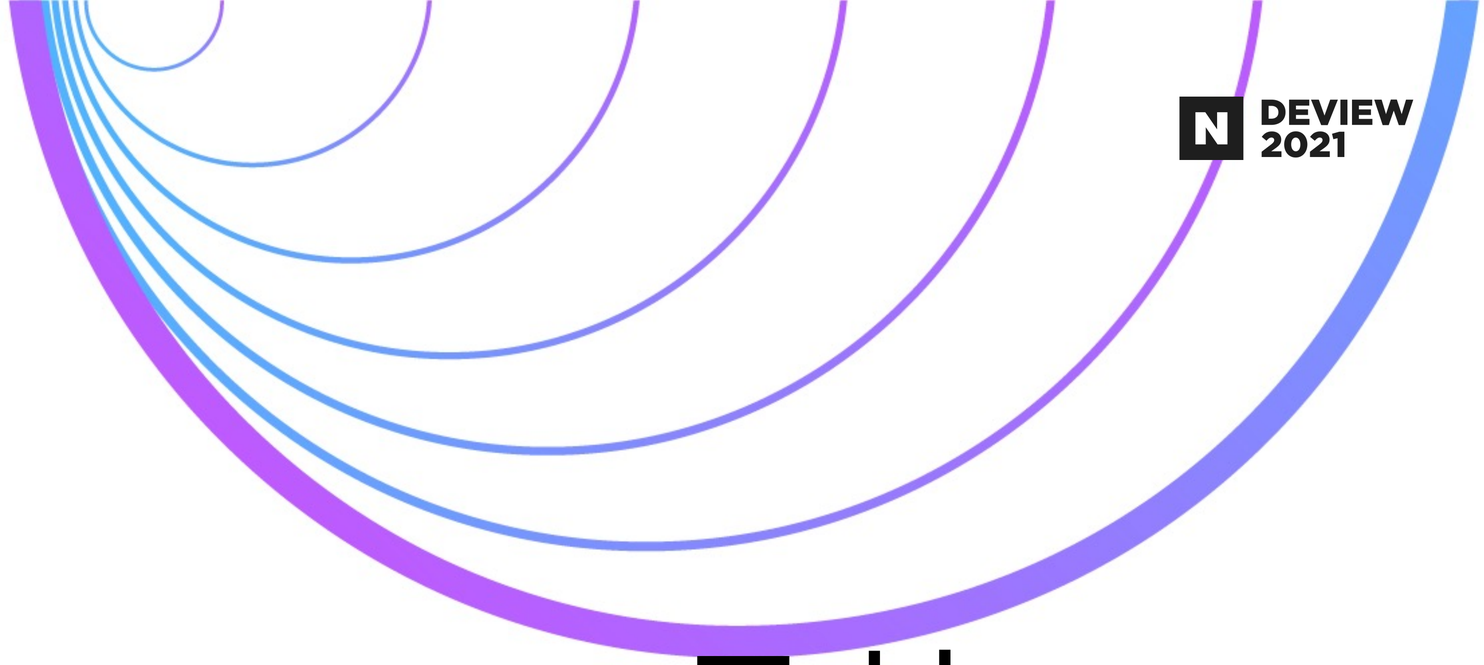
4.7 장비/공간/에너지 문제 발생

데이터 센터 필요?

발전소 필요?



춘천댐 (출처 : 대한민국 구석구석)



DEVIEW 2021

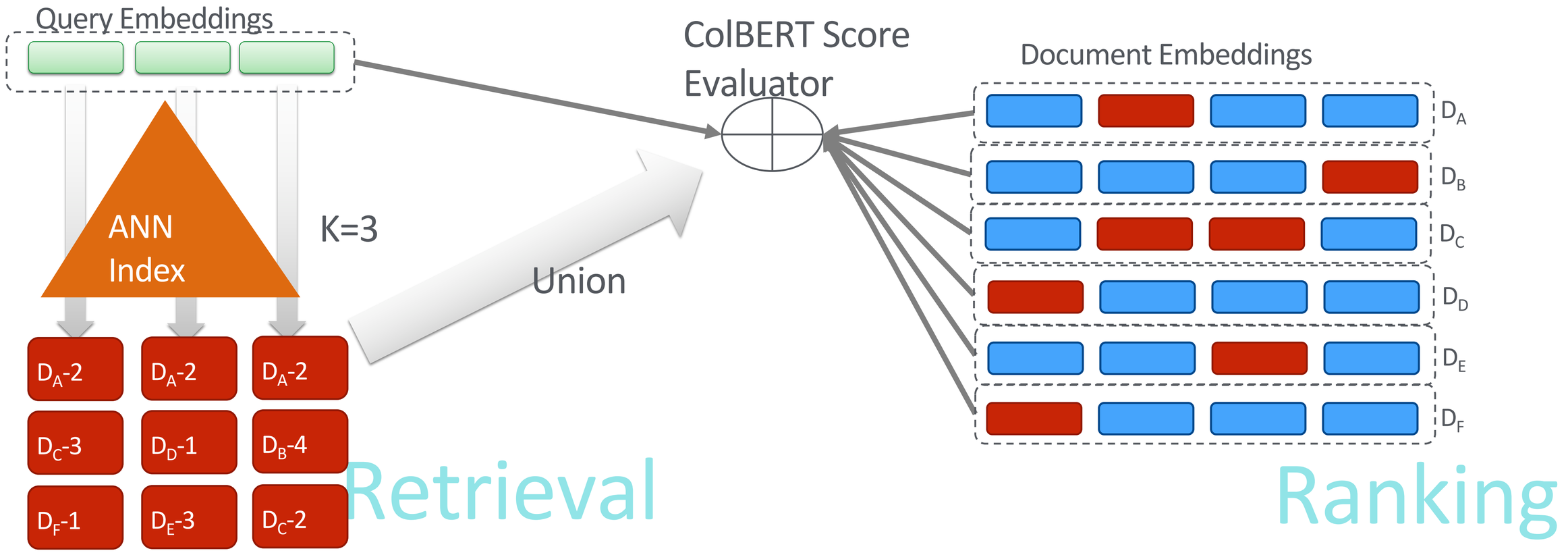
5. Late Interaction 구현

5.1 Vector Access in Retrieval / Ranking

ANN 으로 접근 가능한 벡터로는 ColBERT score 계산 불가

문서 키로부터 document embeddings를 모두 접근 가능한 구조 필요

메모리 복잡도 큼 : ColBERT score memory access ≃ candidate count , document vector count , dimension



5.2 요구조건과 완화 조건

DEVIEW 2021

1. ANN Index

(요) 빠르고 정확해야 함

(요) DocID가 반환되어야 함

(완) 벡터 유사도나 벡터값 반환 필요 없음

(완) 결과가 정렬 될 필요 없음

2. Vector Storage

(요) 문서 벡터를 빠르게 접근 가능해야 함

(요) Data 는 메모리에 있어야 함

(완) 문서의 특정 벡터를 접근할 필요 없음

3. ColBERT Score Evaluator

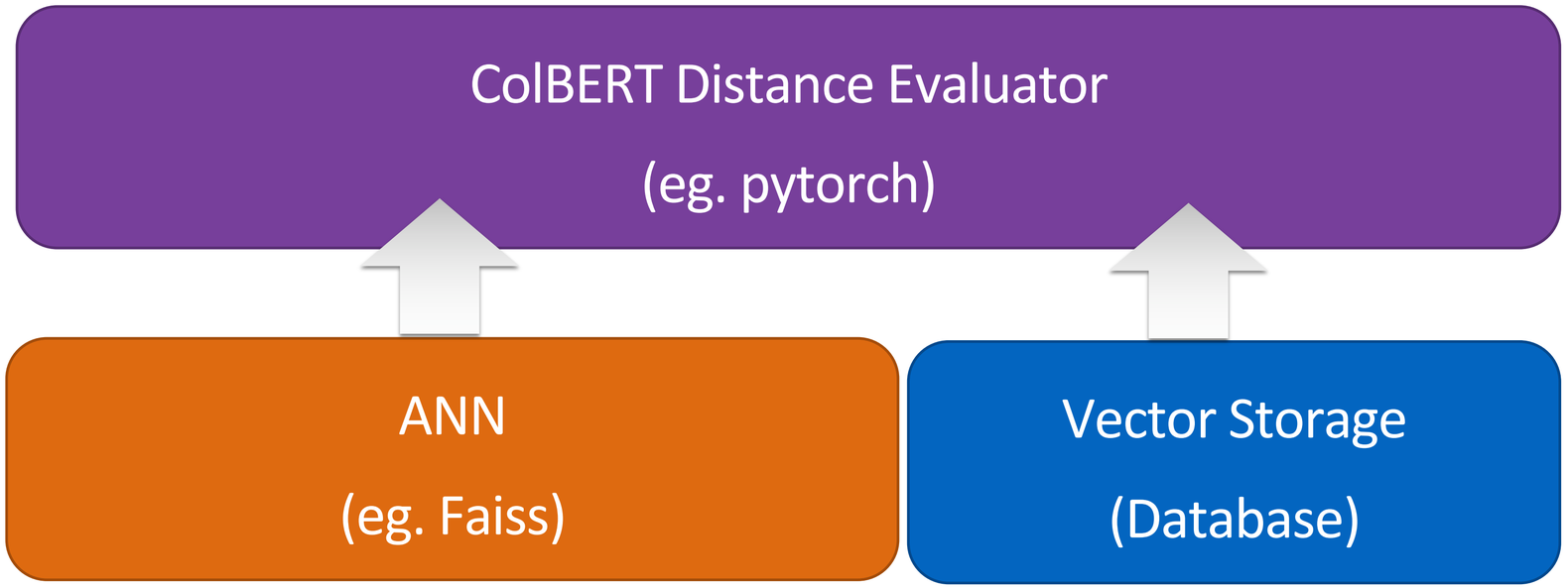
(요) 빠르게 연산이 되어야 함

(요) 요구 조건, (완) 완화 조건

5.3 성능 요건을 만족하지 못한 Prototype

Implement using existing components

- 5.3 만 Docs
- Elapsed time : 30~40 sec



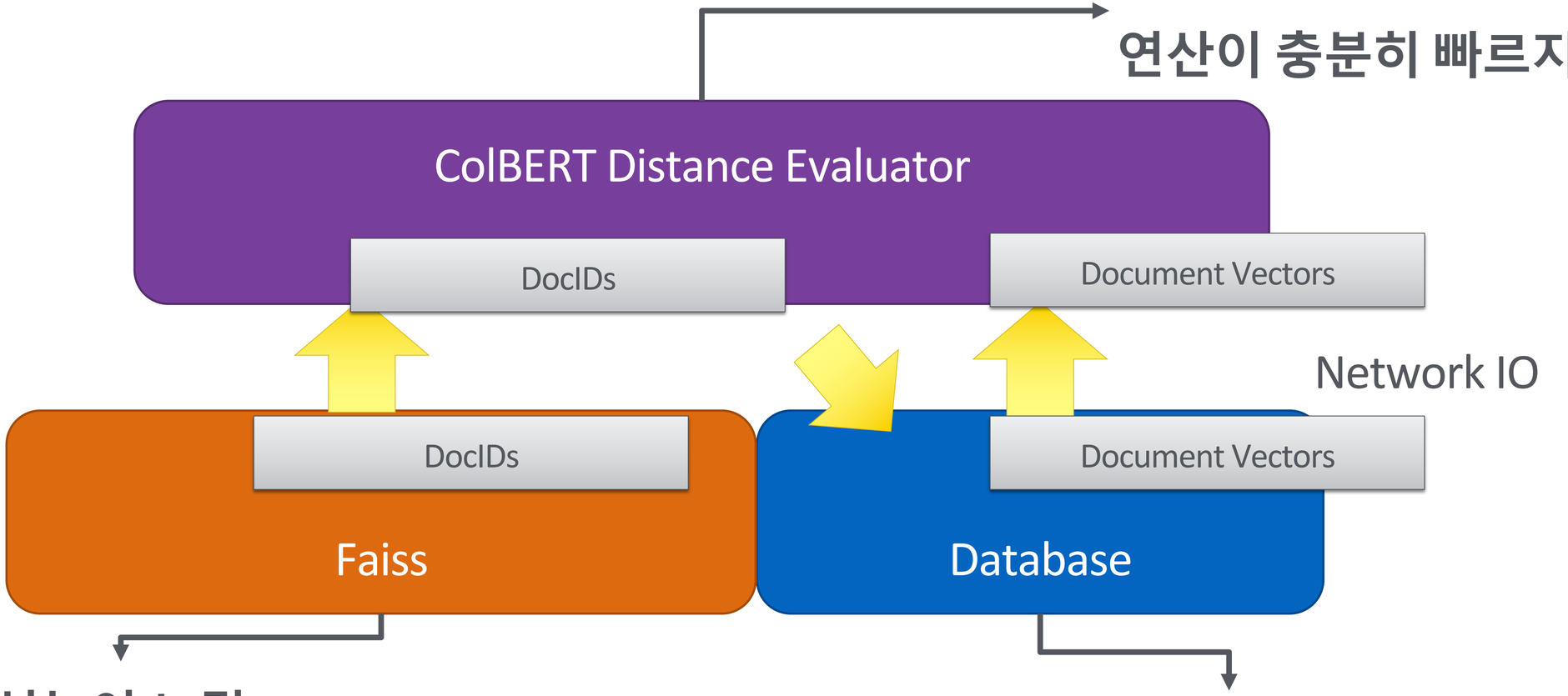
품질 : 응답속도가 너무 느림

리소스 : 원하는 문서 규모와 QPS를 만족하려면 수백 만대 필요

5.4 가능한 성능저하 포인트들

데이터를 복사 해옴

연산이 충분히 빠르지 않음



CPU에서는 성능이 느림

GPU노드에서 돌리기엔 메모리 한계

분산 ANN Index 요구

문서 키로 부터 데이터 접근이 복잡

Global DocID = (Shard#, Datafile#, Segment#, Offset)

In-memory ?

5.5 검색엔진 구현 하이라이트

1. 모든 Component가 검색 엔진에 들어가도록 구현

2. Simple document ID를 이용한 columnar structure

3. ColBERT distance evaluator의 고도의 성능 최적화

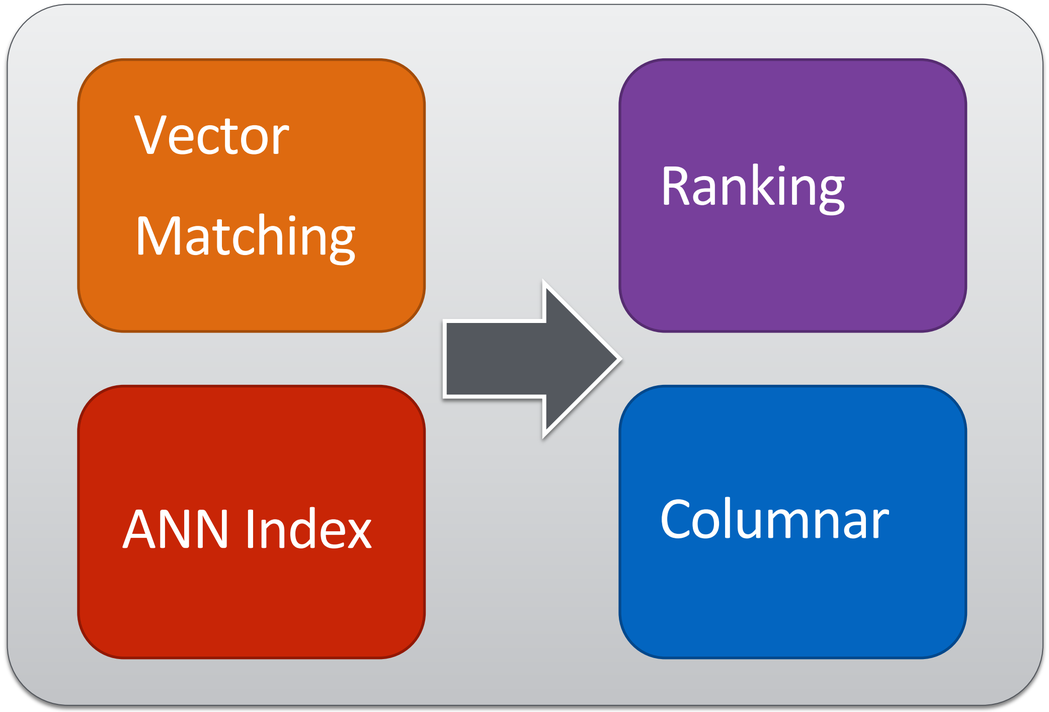
5.6 All-in-one structure

All components in search engine

- Vector matching (ANN)
- ColBERT score evaluator + ranking
- Vector storage

Zero communication between components

Zero copy of document vectors



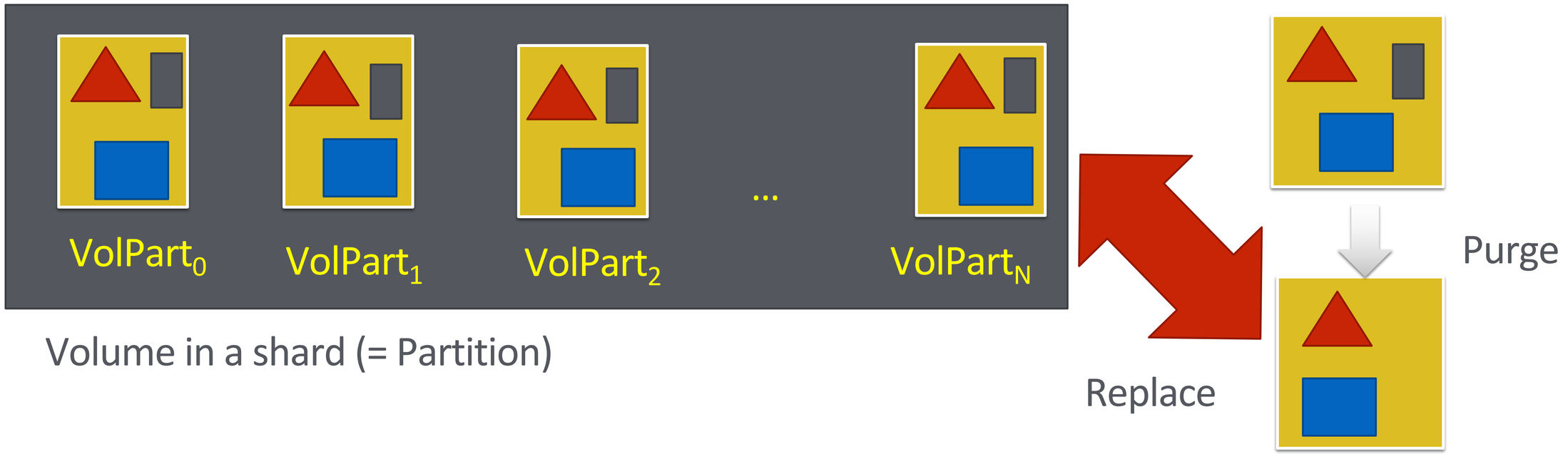
Search Engine

5.7 Multi-partitioned Volume Structure

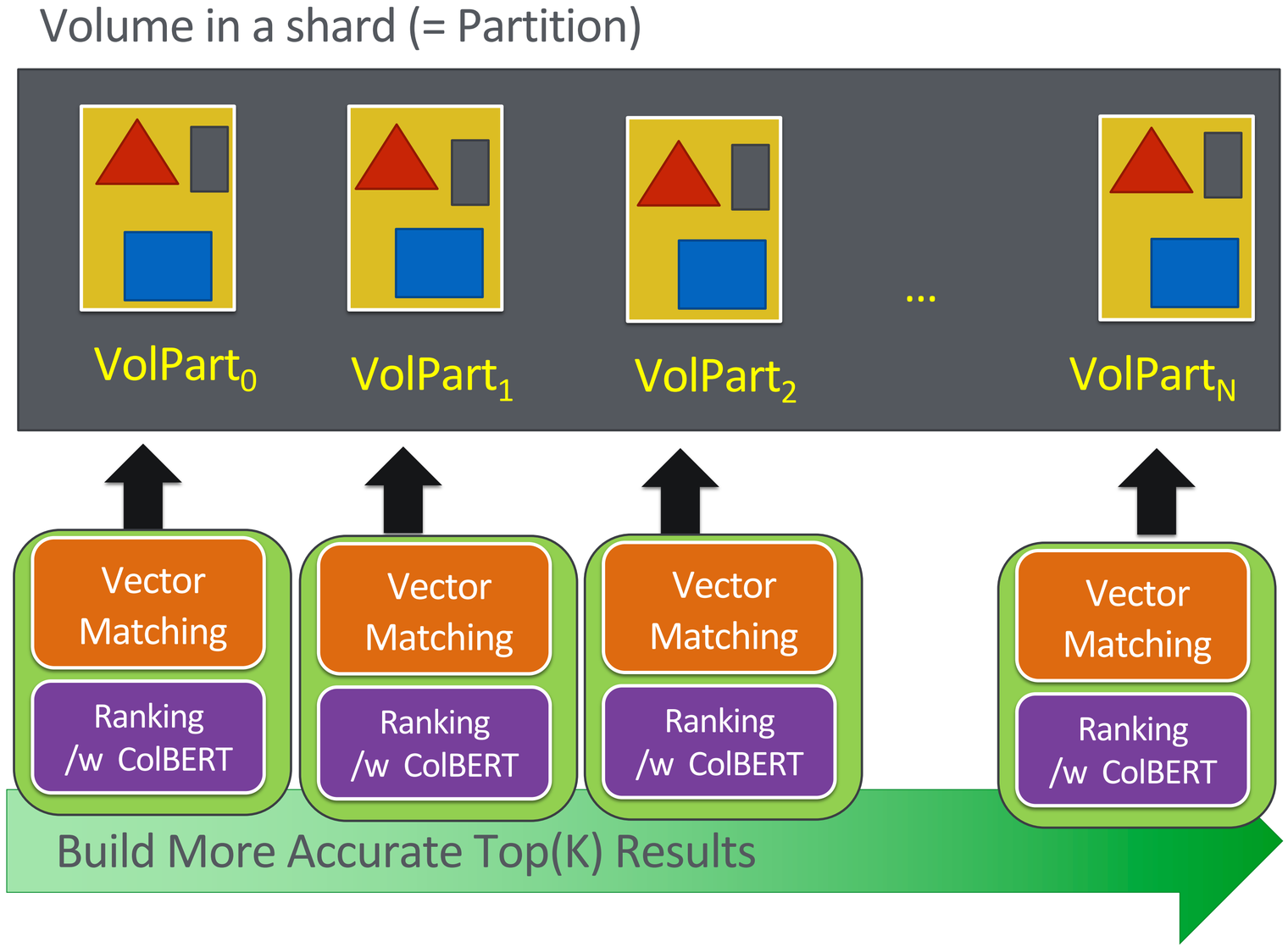
Volume은 적당한 규모의 volume partition으로 구성

Volume purge impact를 줄이기 위함

- 한번에 하나의 volume partition 에만 볼륨 변경



5.8 Multi-partition Iterator / local DocID



파티션에 따른 루틴

- Volume Partition 내에서
  Retrieval 된 문서들에 대하여
  ColBERT score 계산

Top(K) 결과는 Partition 외부에 유지

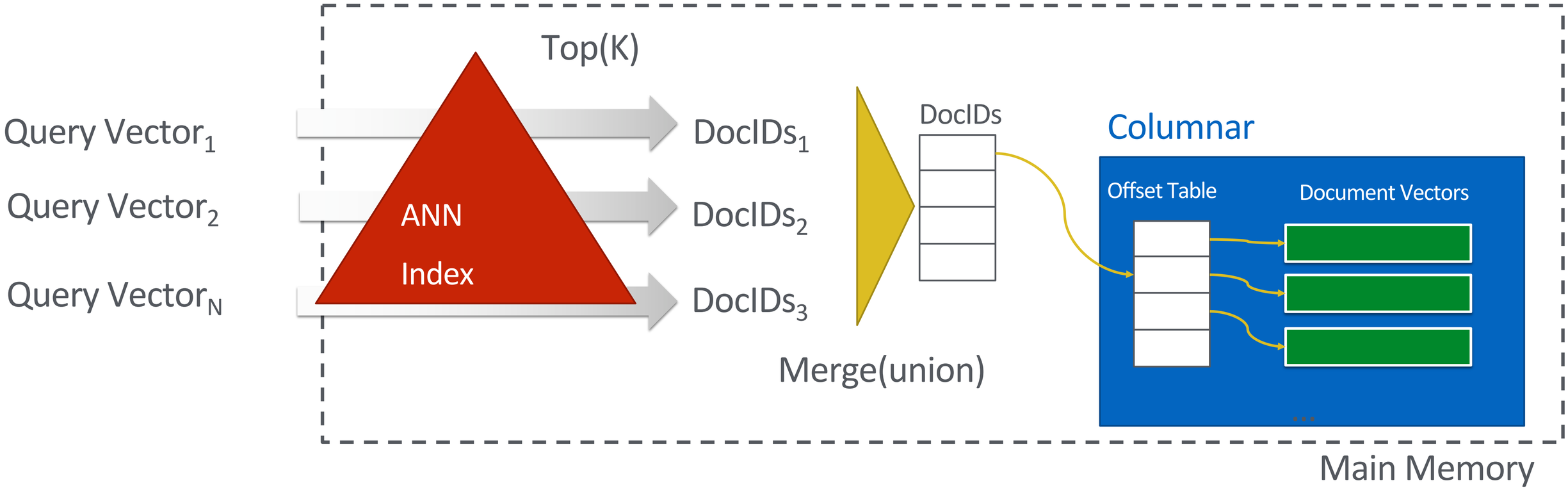
Local DocID 가능

(size of DocID = 4bytes)

5.8 In-memory Columnar Structure

Simpler DocID, faster search structure

No data copy of In-memory Data



5.8 Score Evaluator 성능 최적화 기법

SIMD(AVX512F) Intrinsic function을 이용하여 한땀한땀 최적화

- Instruction level의 Profiling (그럴 가치가 있음)
- Compiler Automatic Vectorization 대비 5.63(CosSim), 6.12(L2Dist) 배 성능 향상

벡터의 Norm값 이용

- 문서 벡터의 Norm 값을 미리 계산하여 추가 저장

Normalized Vector 이용

- SumOfMax CosSim 경우 효과적

Score의 특성에 따른 최적화

- SumOfMax L2Dist 만 적용 가능 / Max를 구하고 난 후에 sqrt 적용

5.9 Performance Optimization Results

DEVIEW 2021

CPU

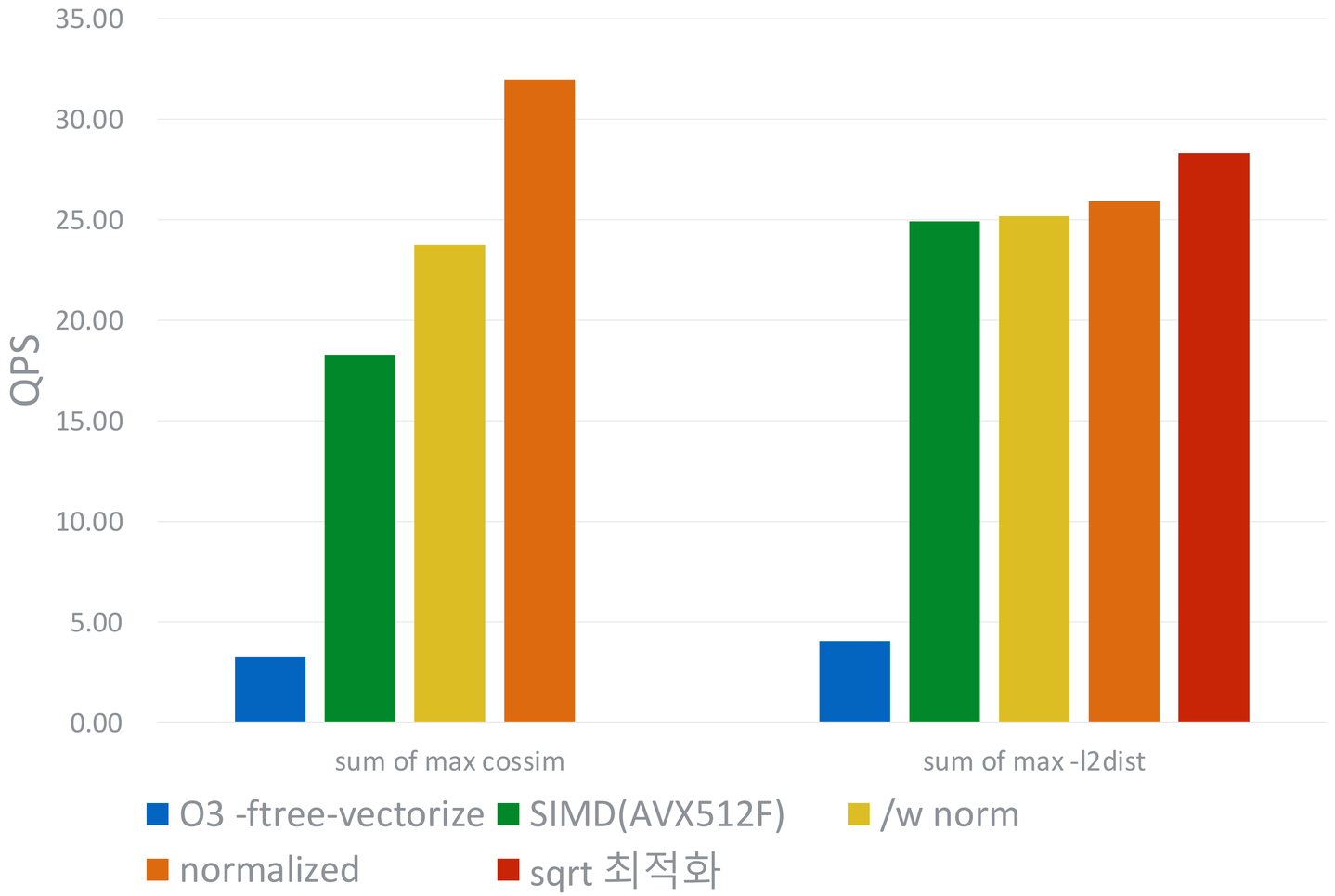
- Intel(R) Xeon(R) Silver 4214

Volume

- 4 Volume parts
- Number of documents : 141K
- Dimension of vector : 128
- Number of document vectors : 65

Query

- Number of query vectors : 16
- Number of candidates : 1000
- TopK : 10



6. 검색 성능 평가와 품질 개선점

6.1 성능 평가

DEVIEW 2021

Elapsed time

- Average : **62.4ms**
- Minimum : 43.8ms
- Maximum : 85.0ms

Volume

- Single shard / 4 slots
- 82만 documents
- Document vector(dimension=128, count=65)

Query

- ANN retrieval option (K =50, ef_search=100)
- TopK(K=10)
- Query vector (dimension=128, count=16)

6.2 300억 벡터를 위한 Cluster 규모 추정

측정 수치

예상 수치

1500 QPS 조건 만족을 위한 Replica 수 추정

- 평균 쿼리 수행 시간 = 62.4ms
- 1000ms / 62.4ms * 24 Cores * 4 Replicas = 1538 > 1500

30억 벡터 만족을 위한 파티션 수 추정

- Number of documents in a shard = 82만 문서
- 5000만 문서 / 82만 문서 ≒ 600 Partitions

Cluster 규모 추정

- 600 Partitions * 4 Replicas = 2400 Nodes

6.3 클러스터 구축 사례

ColBERT cluster A (for experiment)

- 3800만 Docs (7.6%)
- 40 Partitions X 1 Replica

ColBERT cluster B (for experiment)

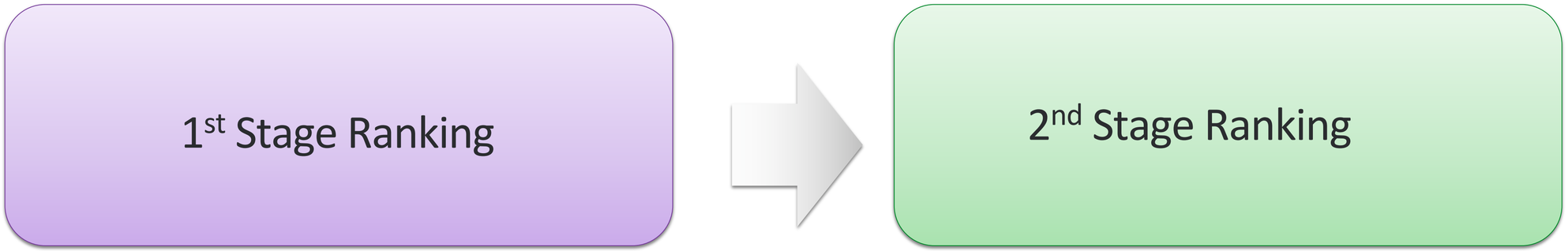
- 3.6억 Docs (72%)
- 500 Partitions X 2 Replica

6.5 최근의 검색 모델 소개

DEVIEW 2021

Two Stage LTR(Learning To Rank) Model

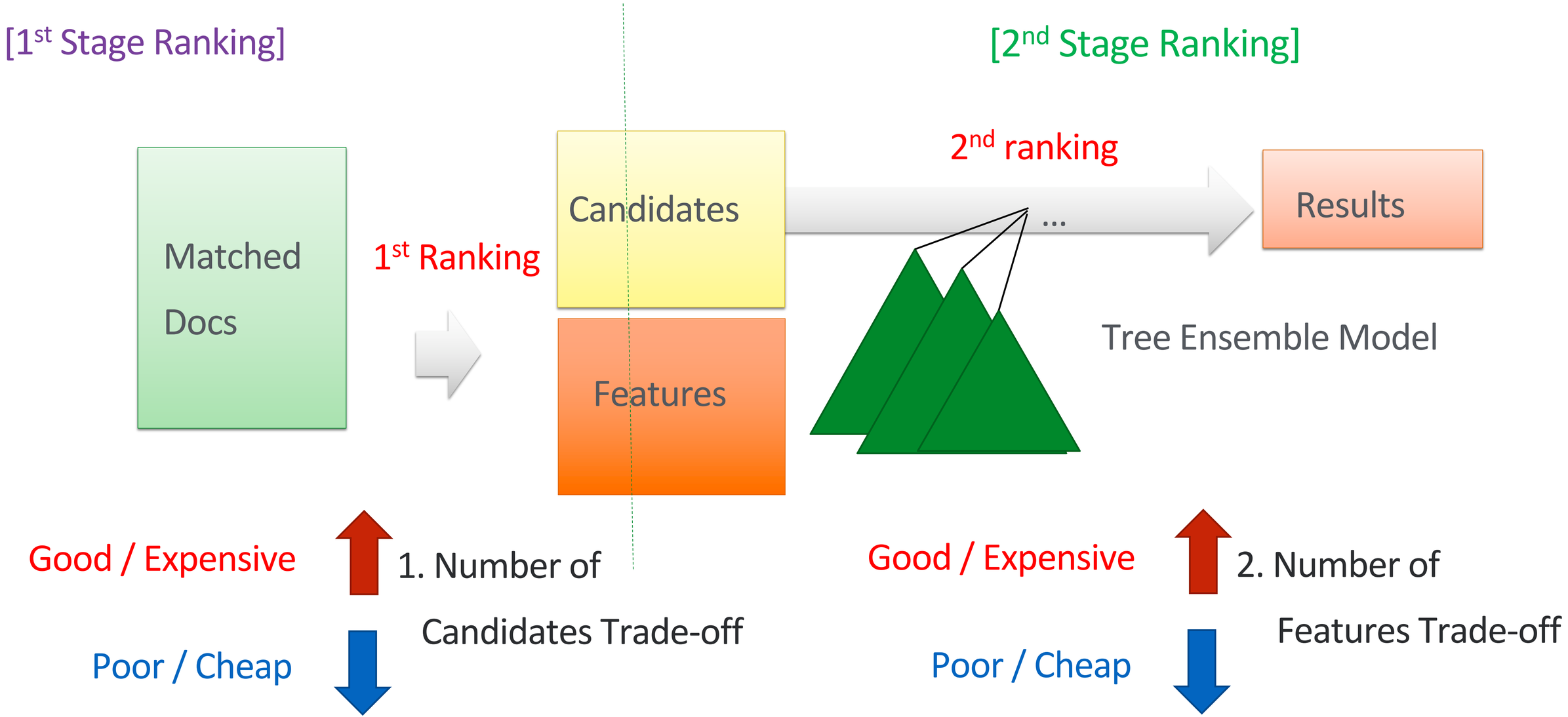
Van Dang, Michael Bendersky, and W. Bruce Croft. Two-stage learning to rank for information retreival. ECIR 2013: Advances in Information Retrieval, pp 423-434. Springer, 2013.



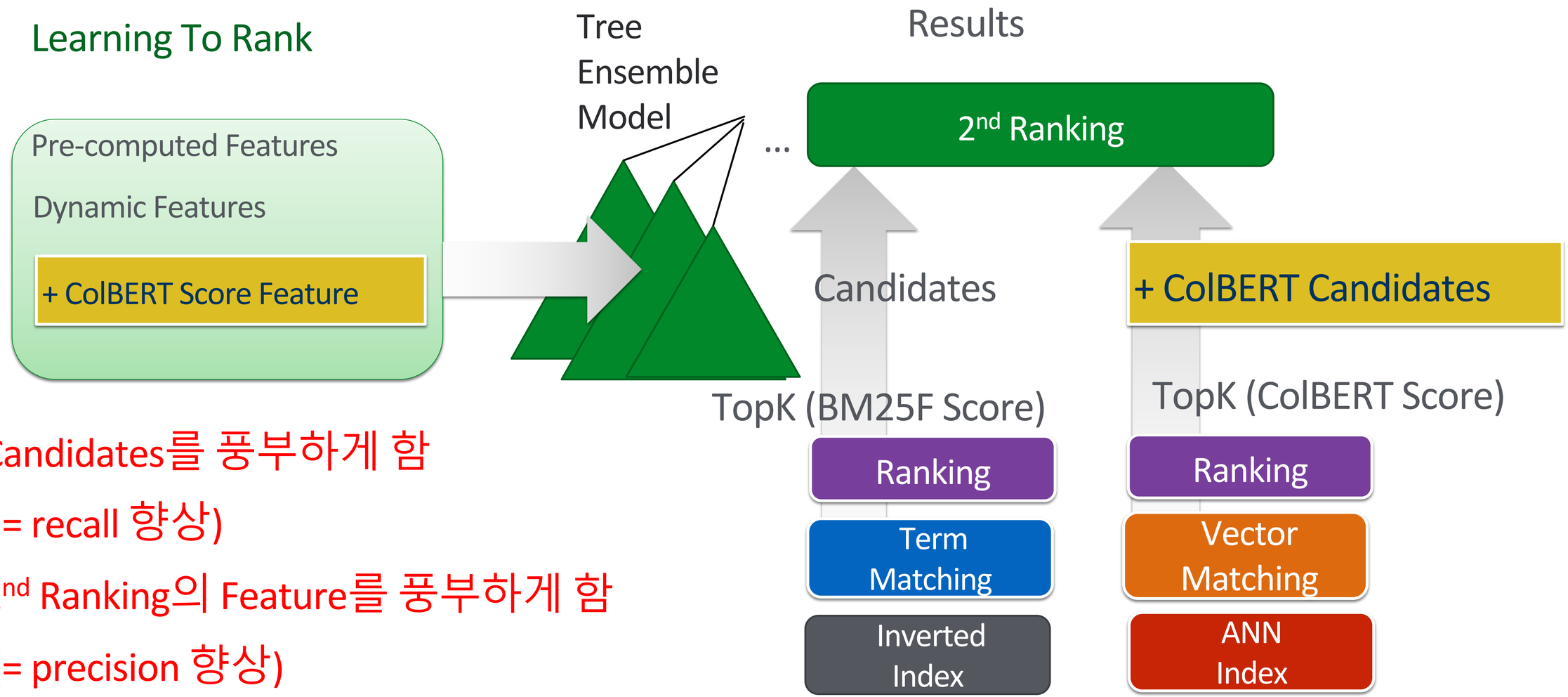
- recall 기반 : 대상을 풍부하게 하는게 목적

- precision 기반 : 순위를 정교하게 하는게 목적
- 기계 학습을 기반으로 동작 (= Learning to rank)

6.5 Trade-offs in Two Stage LTR Model



6.4 ColBERT의 검색 품질 향상 포인트

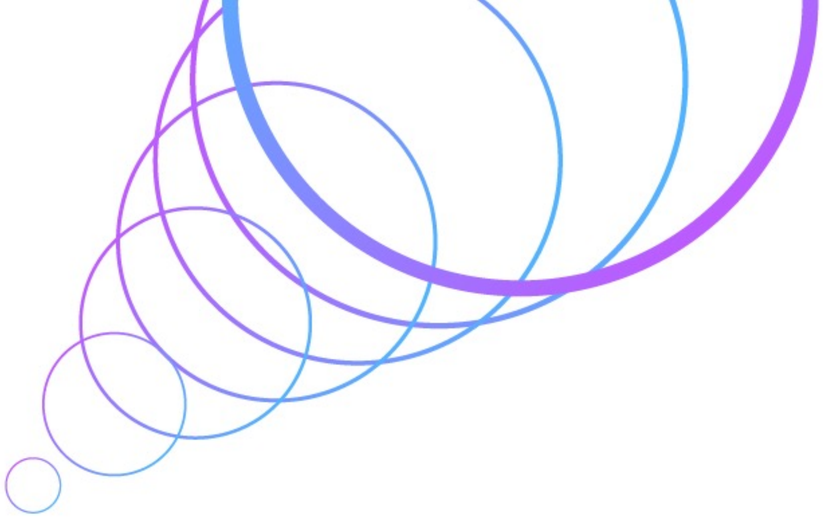


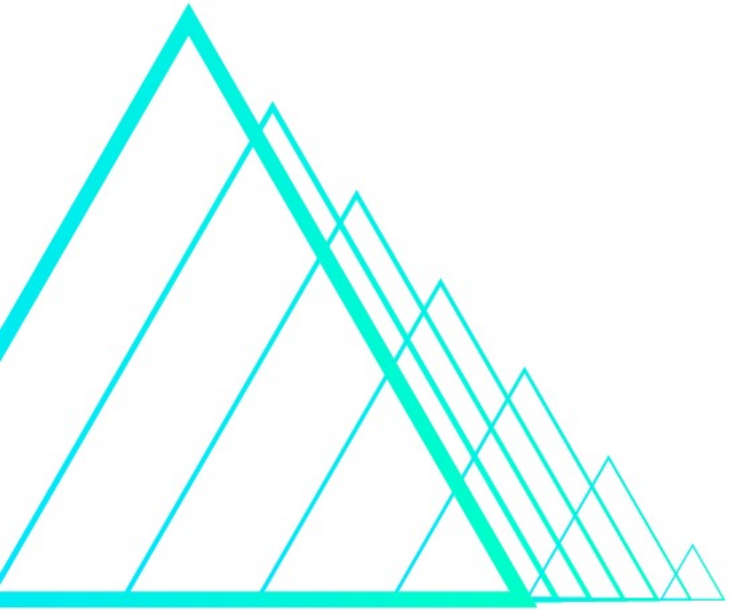
Candidates를 풍부하게 함

(= recall 향상)

2nd Ranking의 Feature를 풍부하게 함

(= precision 향상)





Q & A

Thank You