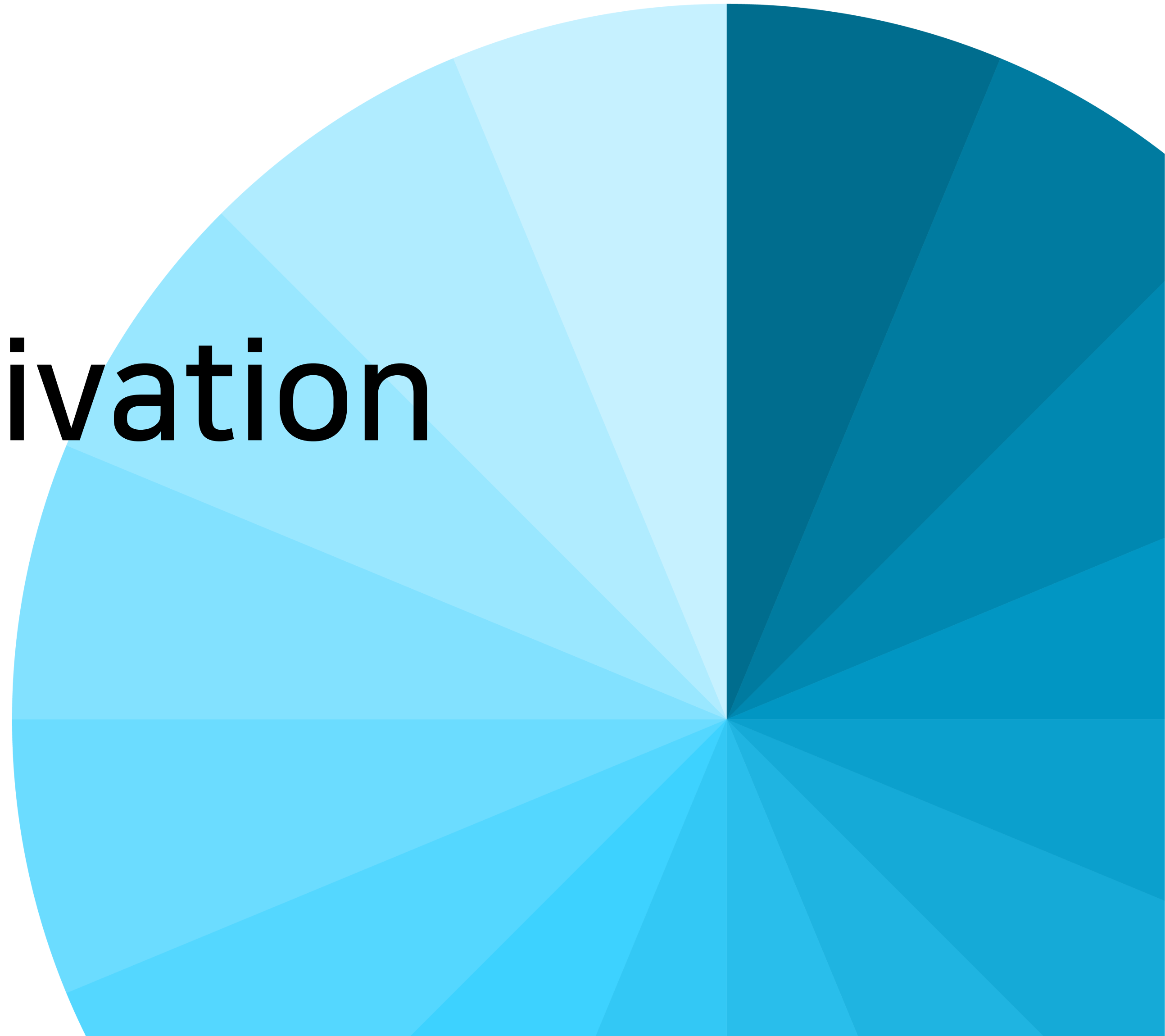


C++로 다시 쓴 ES Ultimate Search

김두용 NAVER

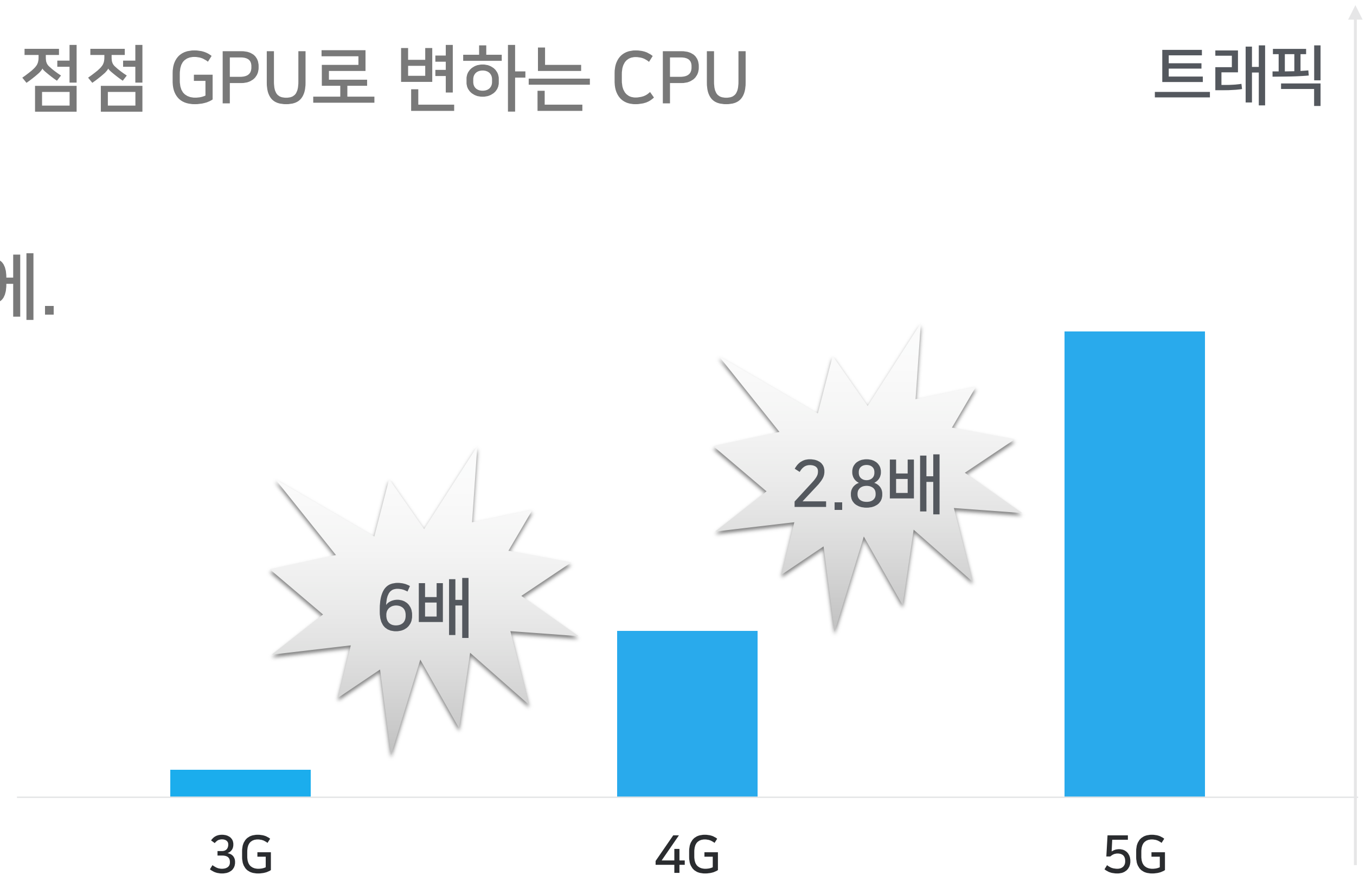
1. Motivation



1.1 빅데이터, 그리고 GC

점점 더 빨라지는 연결 속도, 데이터는 폭발하고 있다.

- 5G : 유래없는 속도, 초연결 시대
- 프로세서의 진화 : NPU, GPU 그리고 점점 GPU로 변하는 CPU
- GC : 대규모 프로세싱의 병목.
- 클라우드 : 모든 계산은 저 구름 너머에.



1.2 더 빨라질 수 있다.

하드웨어 진화적 알고리즘 그리고 GC만 없다면 더 빨라질 수 있다.

- GPU, NPU : 문제를 풀기 위한 하드웨어의 진화
- 결국 성능은 소프트웨어 + 하드웨어
- 빨라지는 RAM, DDR5 : 점점 커지는 메모리, 500G 그 너머로

"삼성, 비메모리 반도체에 **130조 투자한다**"

"엔비디아, ARM 단독 인수 협상 나서. **최대 62조** 메가딜"

"SK하이닉스 2세대 10나노급 DDR5 D램 출시"

1.3 AI, AI, AI

모든 것이 검색 대상이다!

미디어 뿐만 아니라 감정, 상황 등 그야말로 모든 것.



Voice search



Visual search



Video search

2. Ultimate 프로젝트

어떻게 GC없는 검색을 할 수 있을까?

어떻게 하드웨어를 잘 활용 할 수 있을까?

어떻게 검색 대상을 확대 할 수 있을까?

2.1 GC 없는 더 빠른 검색

C++. 결국 너였구나. 엔진 전체를 C++로 재작성

- Ultimate Lucene : 색인, 검색 핵심 로직을 C++로 재작성¹
- SIMD : CPU의 SIMD 명령어셋으로 프로세싱
- Ultimate Search : 클러스터링, 그리고 THE WORLD NEVER STOPS

나는 왜 루씬을 C++로 재작성했는가?

2.2 비정형 검색

모든 것은 벡터로 통한다.

- Vector space model : 어떻게 벡터를 연관 시킬 것 인가?
- Hierarchical Navigable Small World : 벡터를 색인하고 검색하다.

2.3 왜 ES를 재작성했는가?

ES는 검증된 엔진. 동시에 많은 개발자에게 사랑받는 엔진.

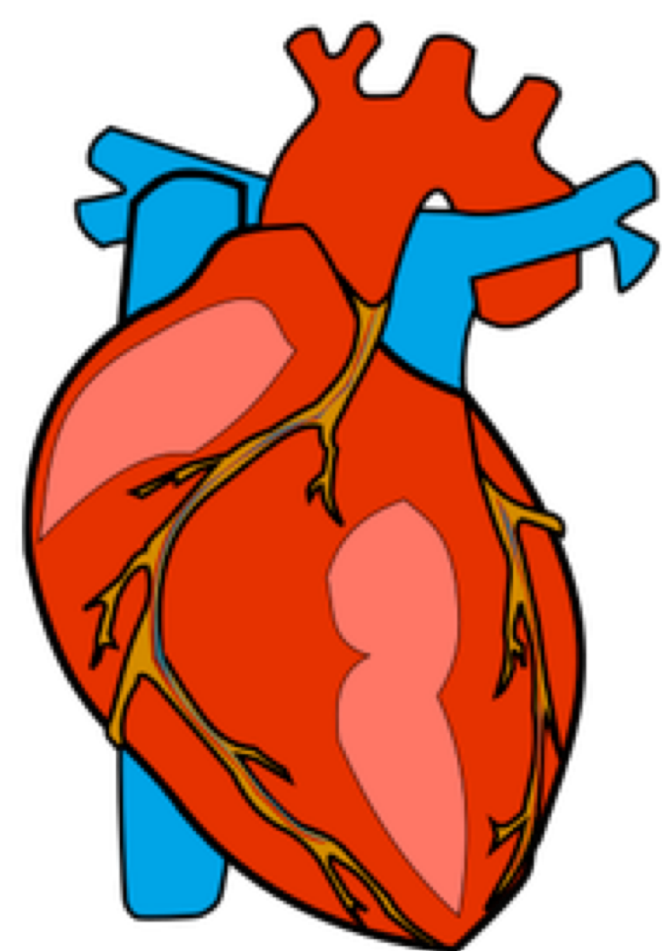
- 편하다! 정말로
- 확장성있는 플러그인 구조
- 독립적인 아키텍처, 별도 의존 인프라 필요 없음
- 우아한 코드 구조, 꼼꼼한 코딩 스타일
- Scalable 클러스터. Raft consensus

3. Ultimate Lucene

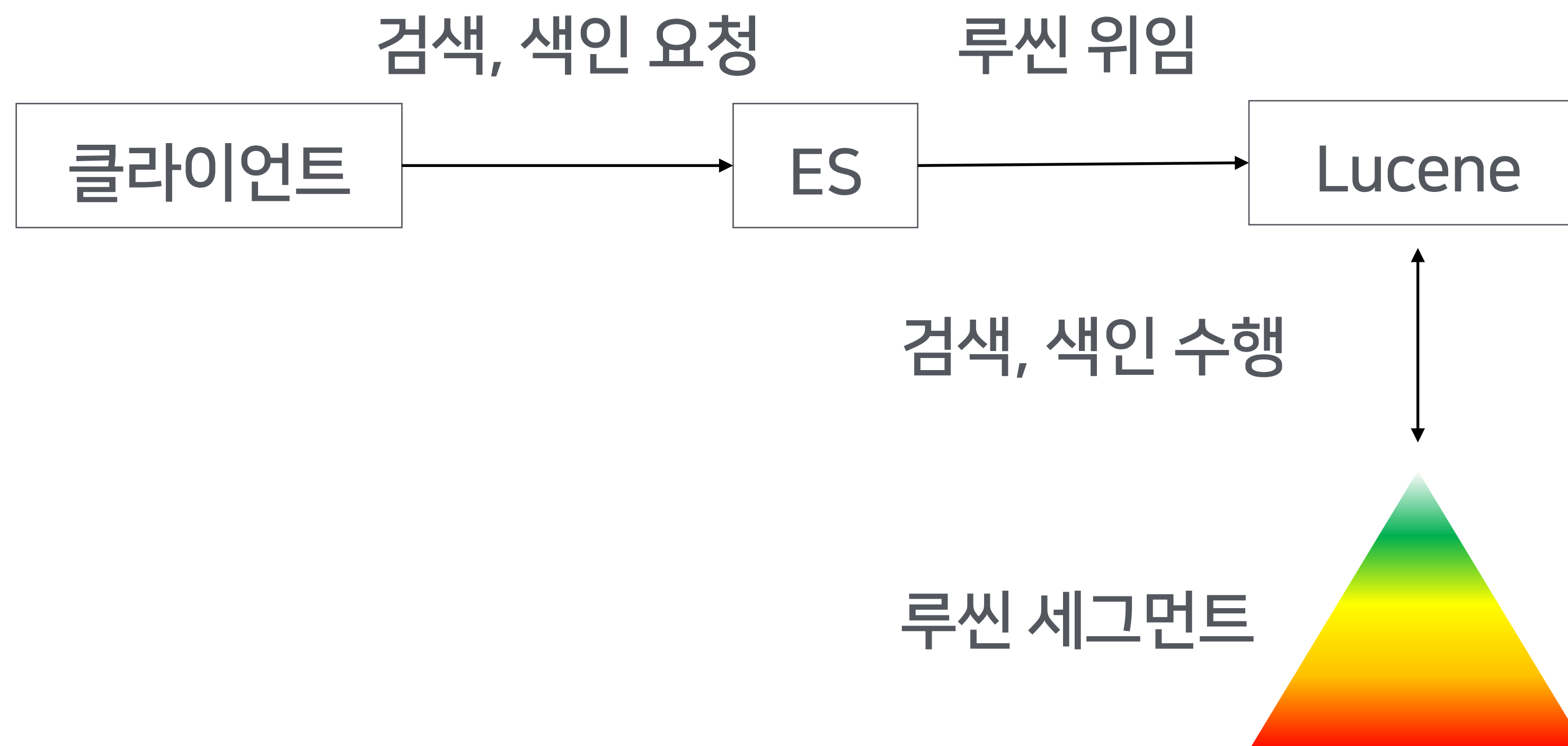
C++로 다시 쓴 Lucene

3.1 Ultimate Lucene

엔진의 심장. 색인과 검색을 책임진다.



Lucene



3.1 Ultimate Lucene

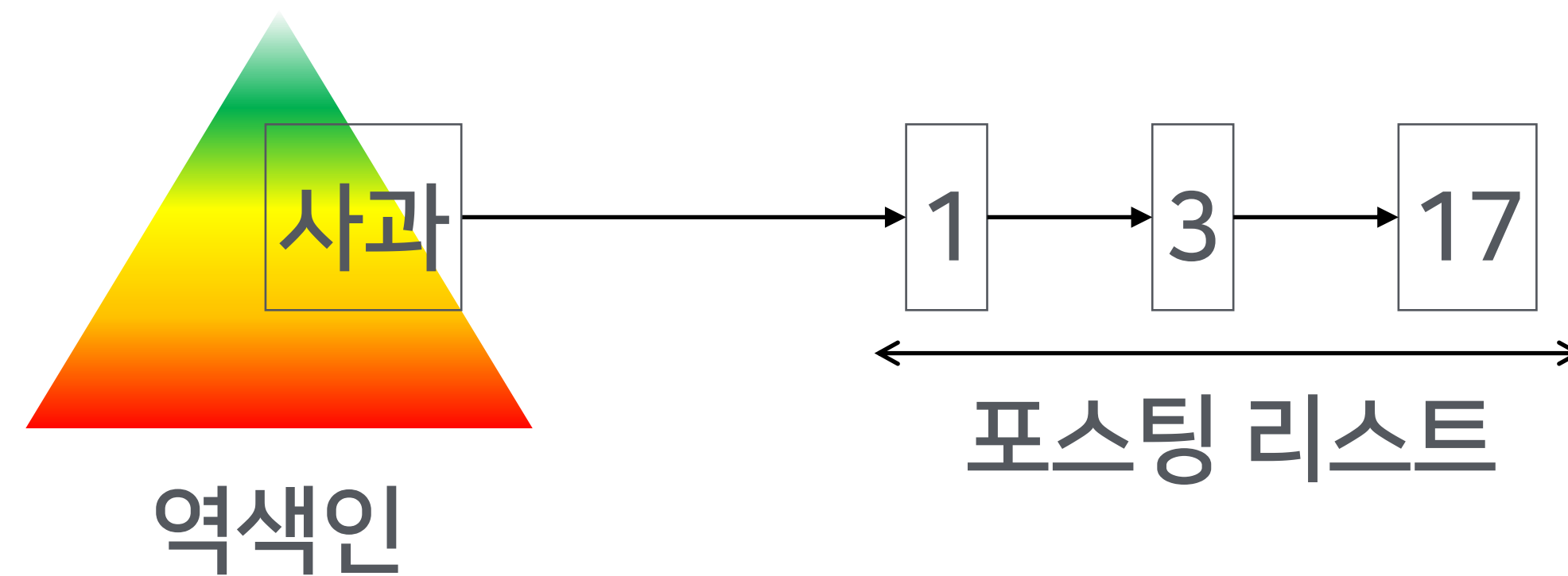
검색, 색인 로직 전체를 C++로 재작성

- 검색 비용의 대다수를 차지한다
- CPU를 100% 활용하여 검색 로직 수행. SIMD
- GC 없음, 평균 2배 성능 Or more? ;)
- 벡터 검색 가능, 비정형 검색으로의 대상 확장

3.2 핵심은 CPU 활용

포스팅 리스트. 모든것이 시작되는 출발점.

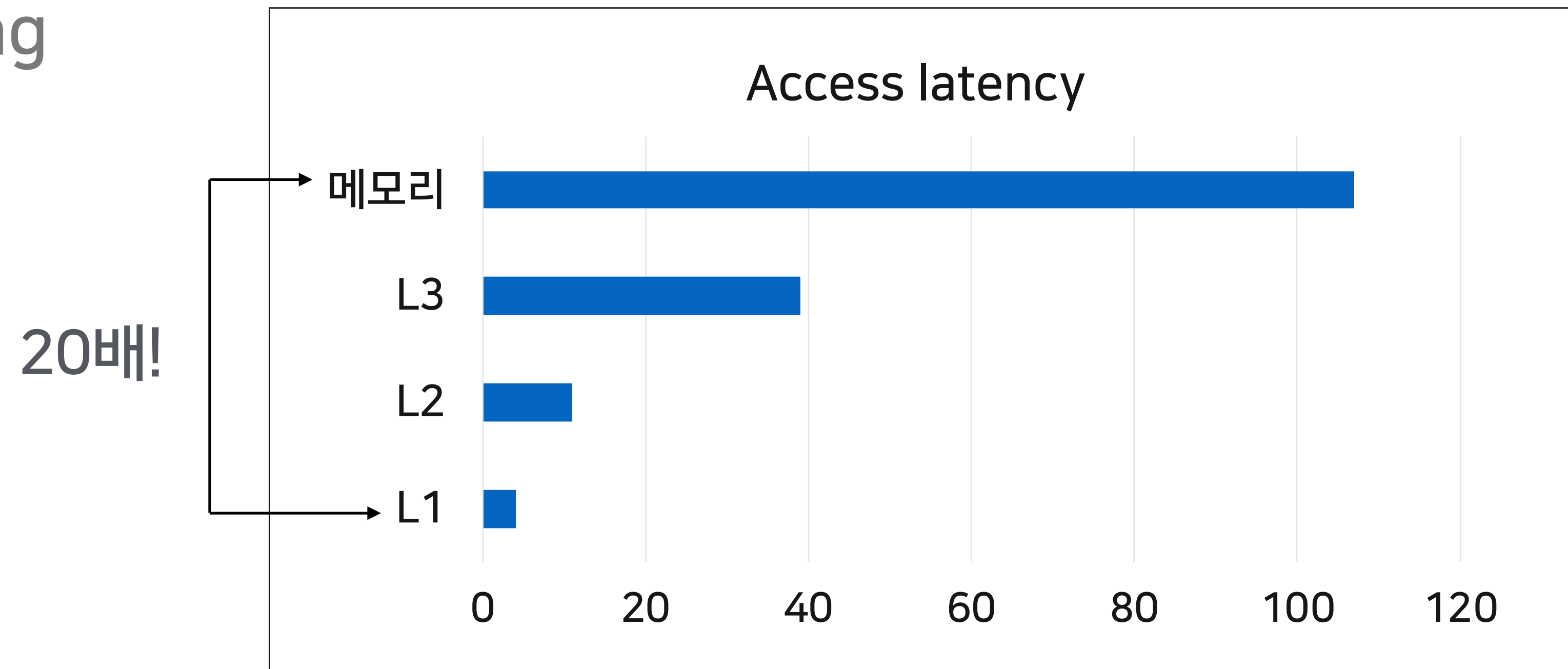
- 20배 빠르다. L1 캐시는. Memory보다.
- 캐시 + SIMD 프로세싱
- Encoding + Decoding



3.2 핵심은 CPU 활용

포스팅 리스트. 모든것이 시작되는 출발점.

- 20배 빠르다. L1 캐시는. Memory보다.
- 캐시 + SIMD 프로세싱
- Encoding + Decoding



3.2 핵심은 CPU 활용

포스팅 리스트. 모든것이 시작되는 출발점.

- 20배 빠르다. L1 캐시는. Memory보다.
- 캐시 + SIMD 프로세싱
- Encoding + Decoding

```
for i in range(0, len)
  c[i] = a[i] + b[i]
```

vs

+

A1	A2	A3	A4
----	----	----	----

B1	B2	B3	B4
----	----	----	----

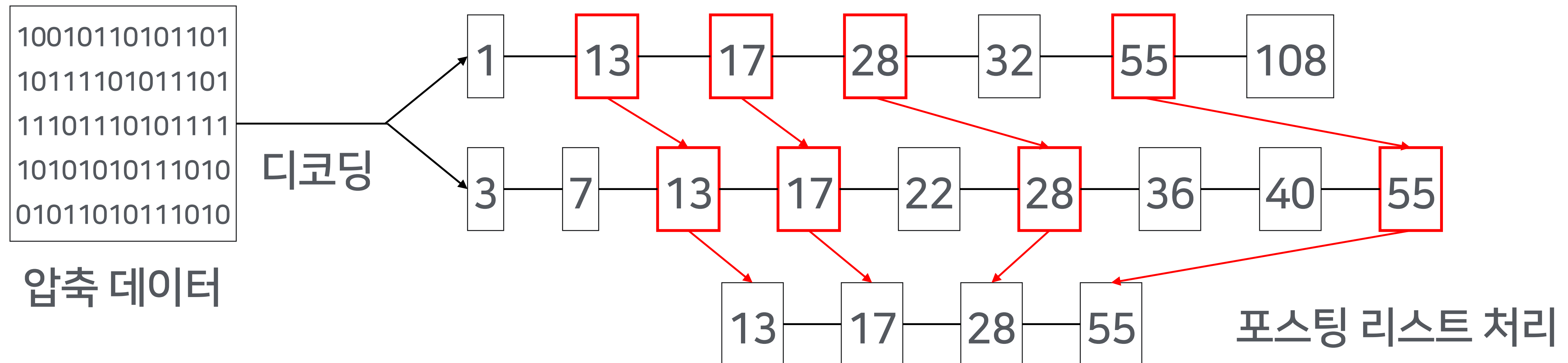
=

A1 + B1	A2 + B2	A3 + B3	A4 + B4
---------	---------	---------	---------

3.2 핵심은 CPU 활용

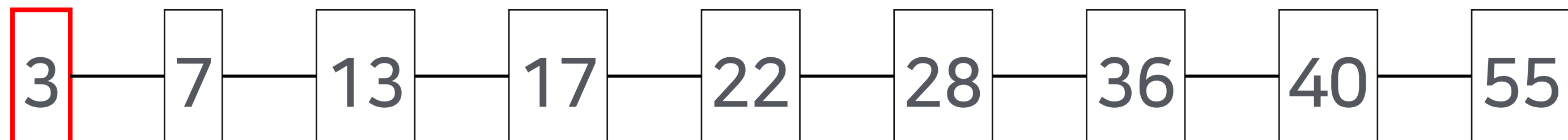
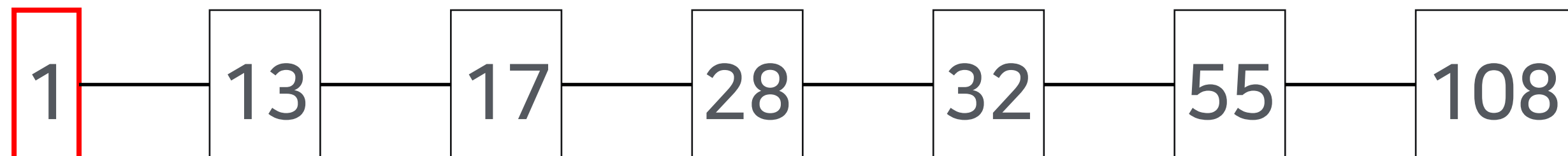
포스팅 리스트. 모든것이 시작되는 출발점.

- 20배 빠르다. L1 캐시는. Memory보다.
- 캐시 + SIMD 프로세싱
- Encoding + Decoding



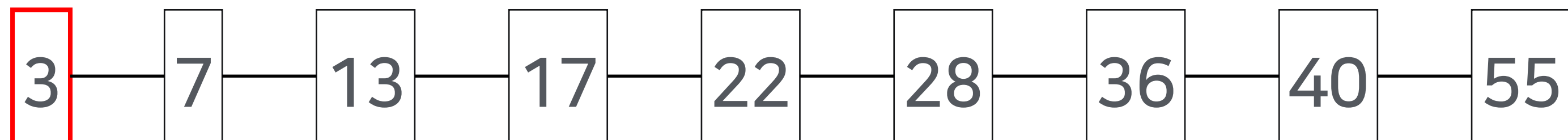
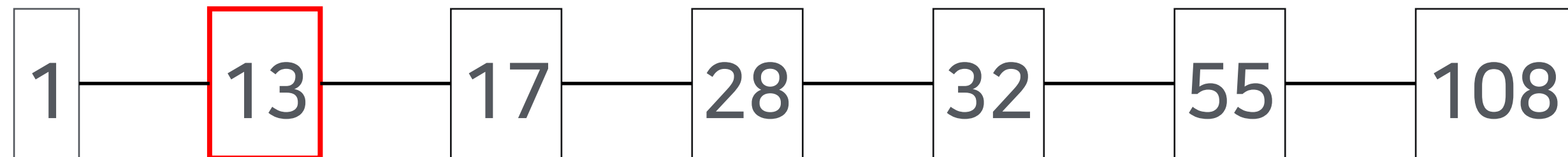
3.3 Document At A Time

프로세싱 매커니즘의 확대 : 한번에 여러개씩 비교



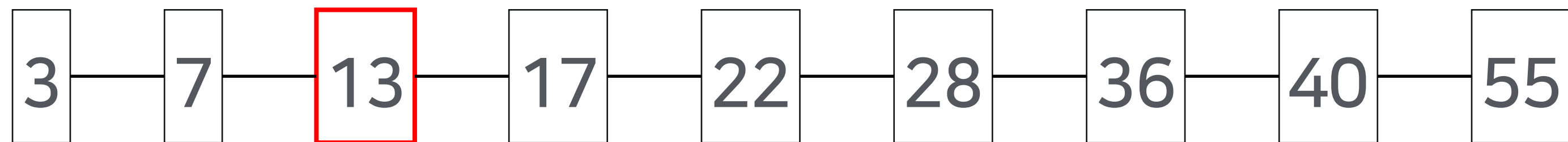
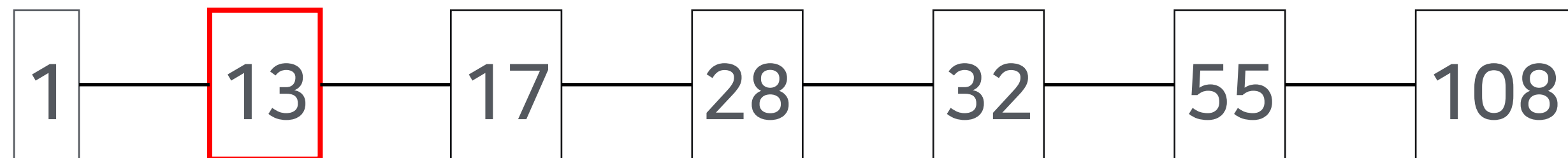
3.3 Document At A Time

프로세싱 매커니즘의 확대 : 한번에 여러개씩 비교



3.3 Documenet At A Time

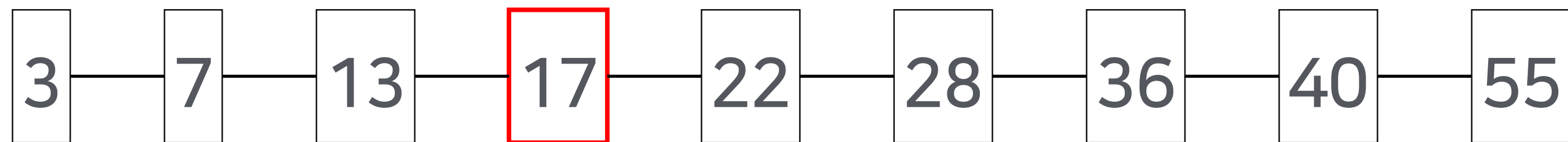
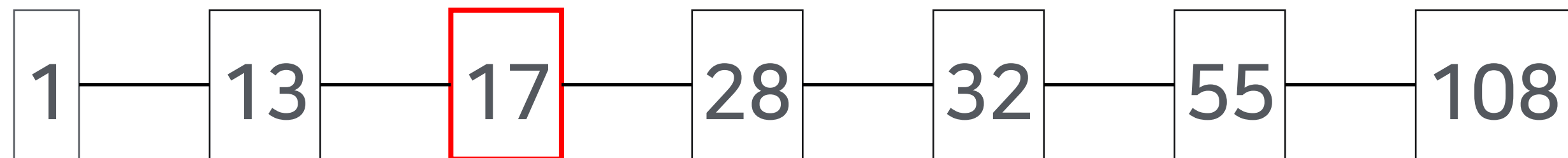
프로세싱 매커니즘의 확대 : 한번에 여러개씩 비교



13번 문서 Match!

3.3 Documenet At A Time

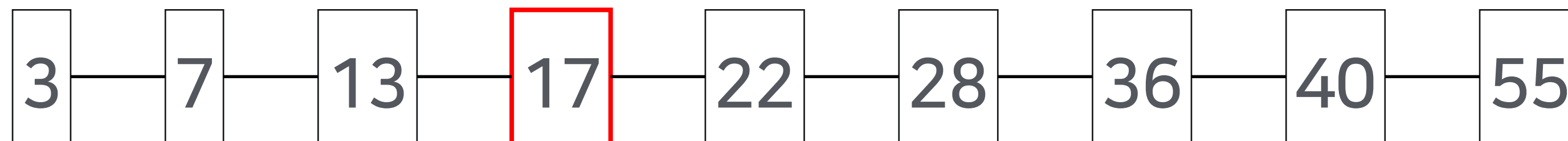
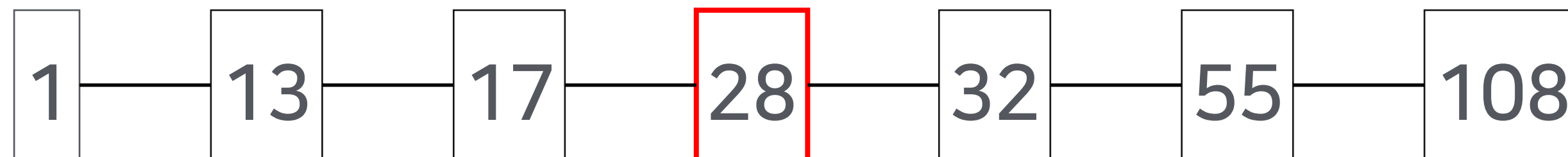
프로세싱 매커니즘의 확대 : 한번에 여러개씩 비교



17번 문서 Match!

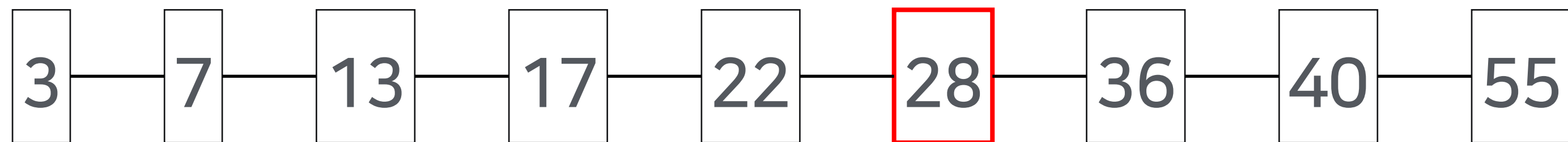
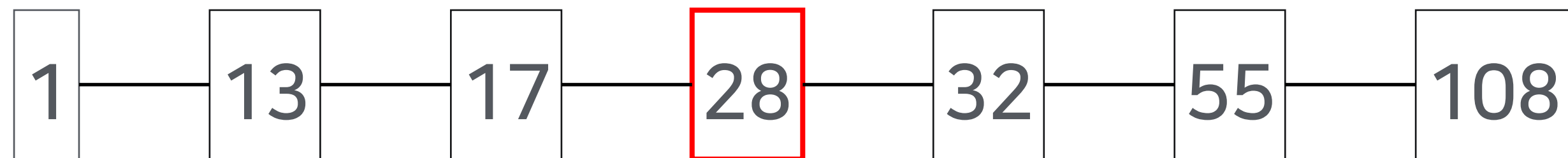
3.3 Document At A Time

프로세싱 매커니즘의 확대 : 한번에 여러개씩 비교



3.3 Document At A Time

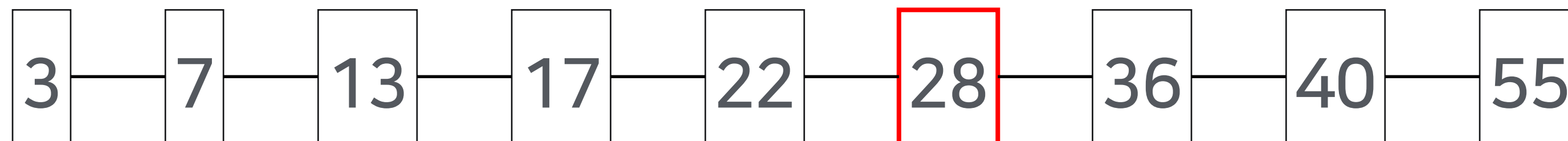
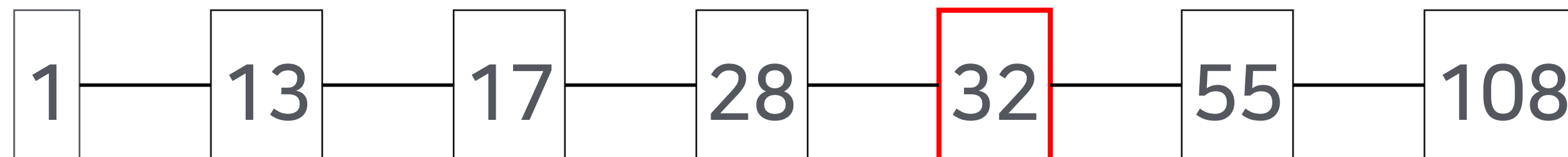
프로세싱 매커니즘의 확대 : 한번에 여러개씩 비교



28번 문서 Match!

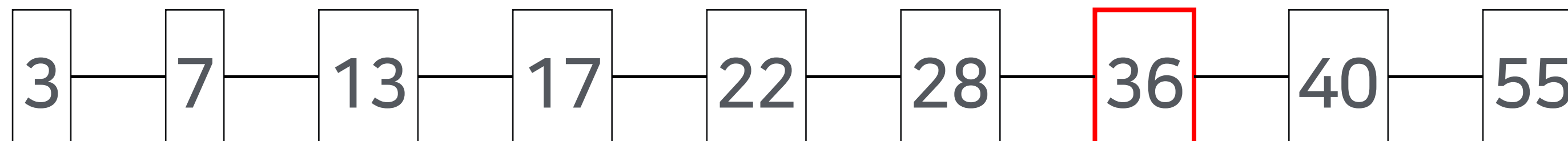
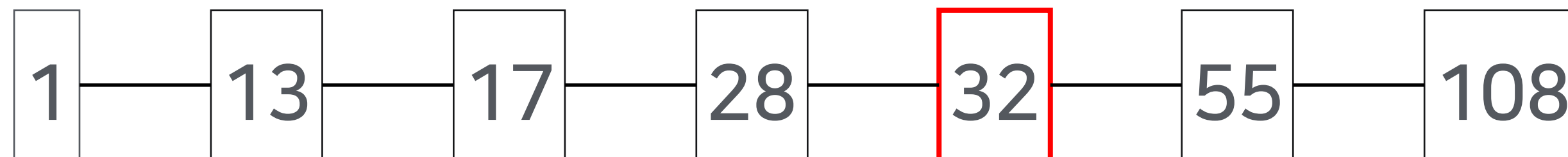
3.3 Document At A Time

프로세싱 매커니즘의 확대 : 한번에 여러개씩 비교



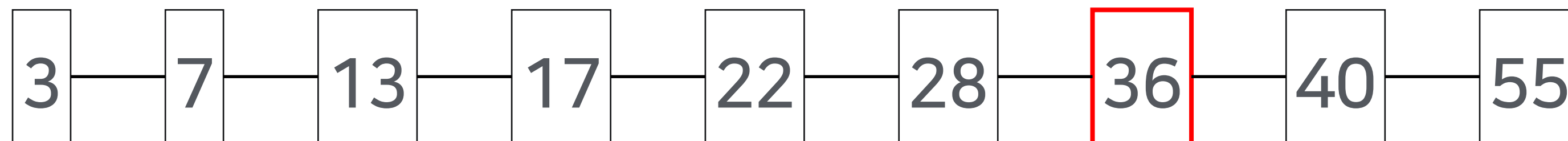
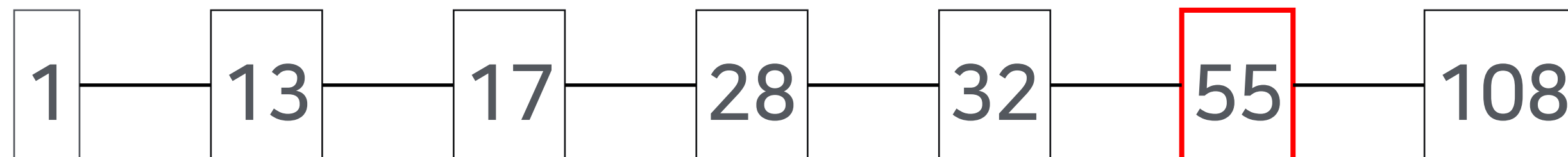
3.3 Document At A Time

프로세싱 매커니즘의 확대 : 한번에 여러개씩 비교



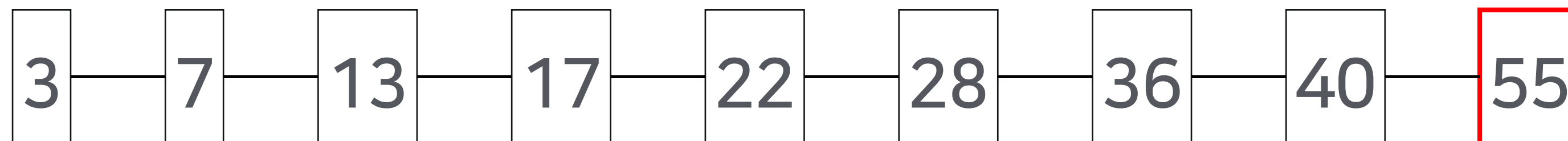
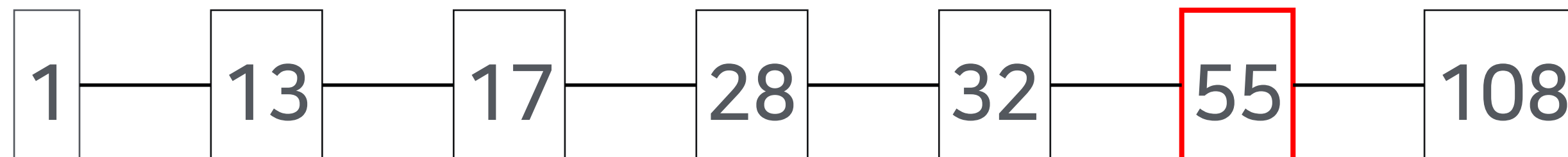
3.3 Document At A Time

프로세싱 매커니즘의 확대 : 한번에 여러개씩 비교



3.3 Document At A Time

프로세싱 매커니즘의 확대 : 한번에 여러개씩 비교



55번 문서 Match!

3.3 Document Vector At A Time

프로세싱 매커니즘의 확대 : 한번에 여러개씩 처리

1	13	17	28	32	55	108
---	----	----	----	----	----	-----

벡터 비교

3	3	3	3	3
---	---	---	---	---



0	0	0	0	0
---	---	---	---	---

3.3 Document Vector At A Time

프로세싱 매커니즘의 확대 : 한번에 여러개씩 처리

1	13	17	28	32	55	108
---	----	----	----	----	----	-----

벡터 비교

7	7	7	7	7
---	---	---	---	---



0	0	0	0	0
---	---	---	---	---

3.3 Document Vector At A Time

프로세싱 매커니즘의 확대 : 한번에 여러개씩 처리

1	13	17	28	32	55	108
---	----	----	----	----	----	-----

벡터 비교

13	13	13	13	13
----	----	----	----	----

0	1	0	0	0
---	---	---	---	---

13번 문서 Match!

3.3 Document Vector At A Time

프로세싱 매커니즘의 확대 : 한번에 여러개씩 처리

1	13	17	28	32	55	108
---	----	----	----	----	----	-----

벡터 비교

17	17	17	17	17
----	----	----	----	----

0	0	1	0	0
---	---	---	---	---

17번 문서 Match!

No CPU Branch!

4. ES + Lucene의 큰 그림 검색과 집계를 동시에

4.1 집계와 검색 : 포스팅리스트

포스팅리스트 + 프로세싱 = 대상 문서들 ID 확보.

- 결국 Set operation

질의 = {t1=네이버, t2=궁극, t3=검색엔진}

$P1 \cap (P2 \cup P3) \cap (P4 \cup P5) \cap P6$

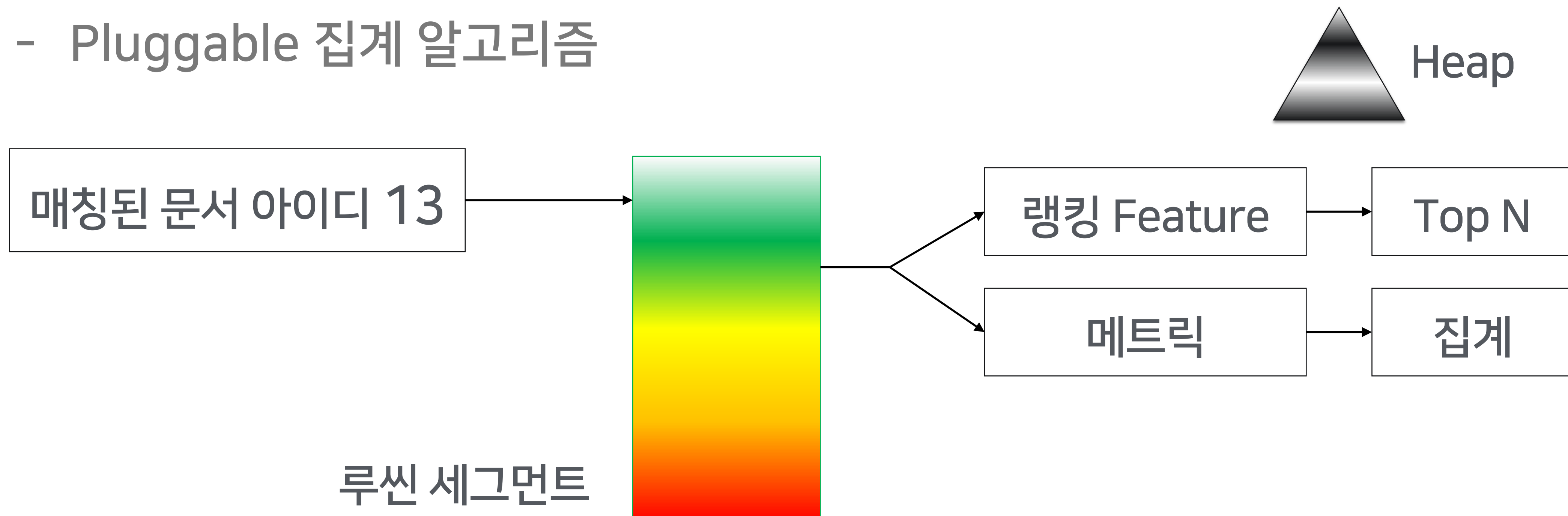
where P1, P2, P3 모두 텀 매칭 포스팅 리스트

그 외 P4, P5, P6 은 확장된 포스팅 리스트

4.2 집계와 검색 : Doc value

Column oriented format + 집계 프로세싱

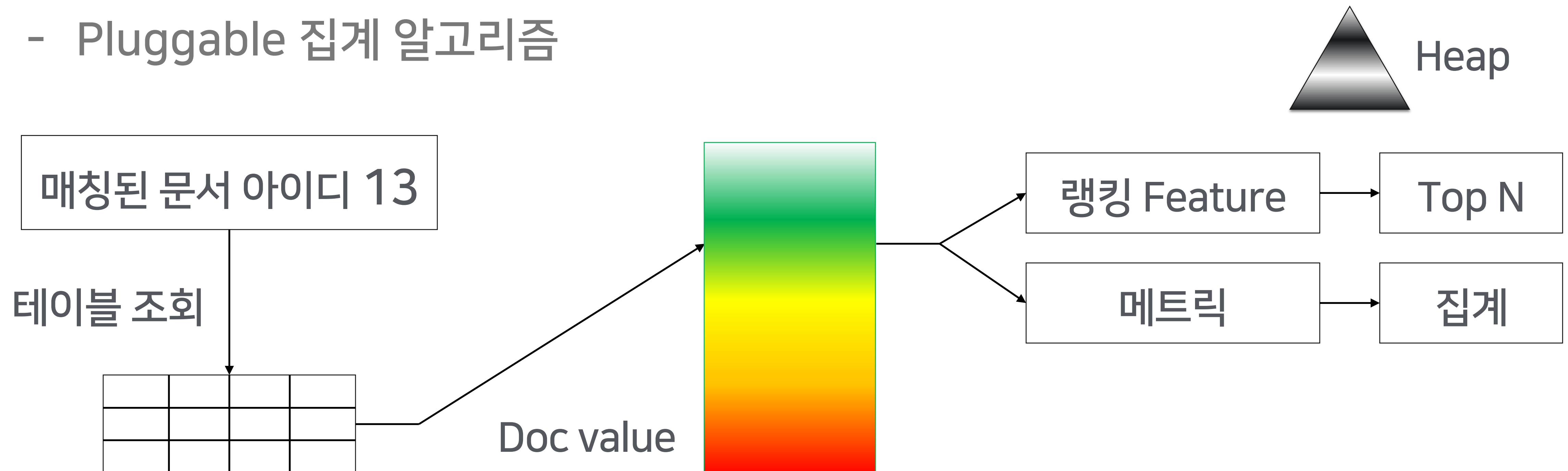
- Pluggable 집계 알고리즘



4.2 집계와 검색 : Doc value

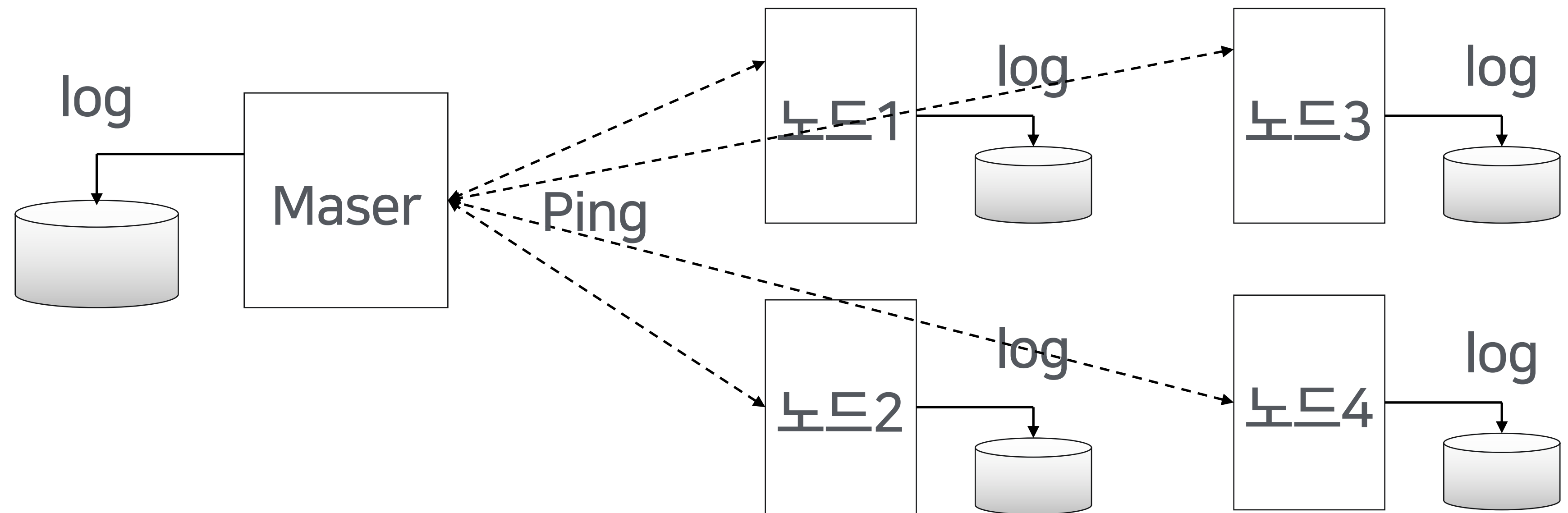
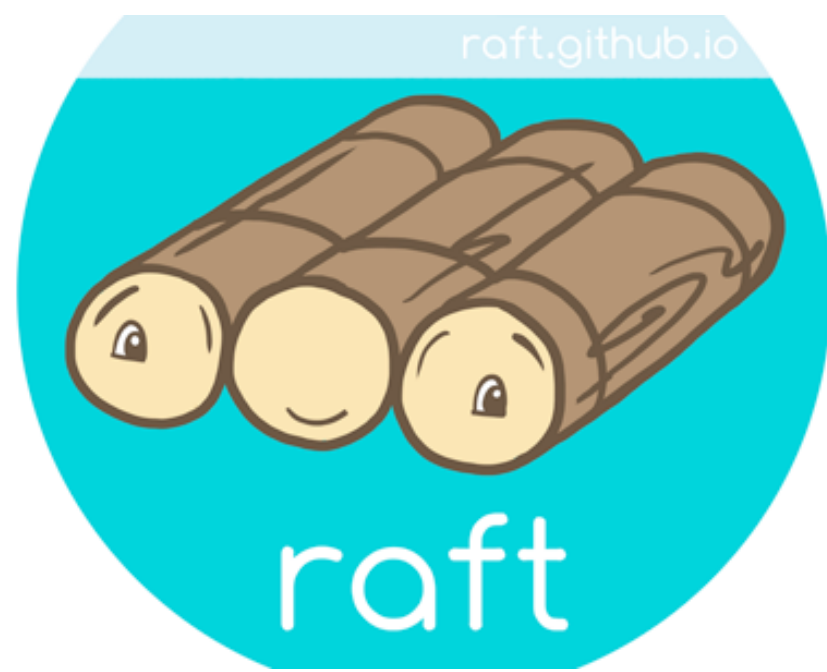
Column oriented format + 집계 프로세싱

- Pluggable 집계 알고리즘



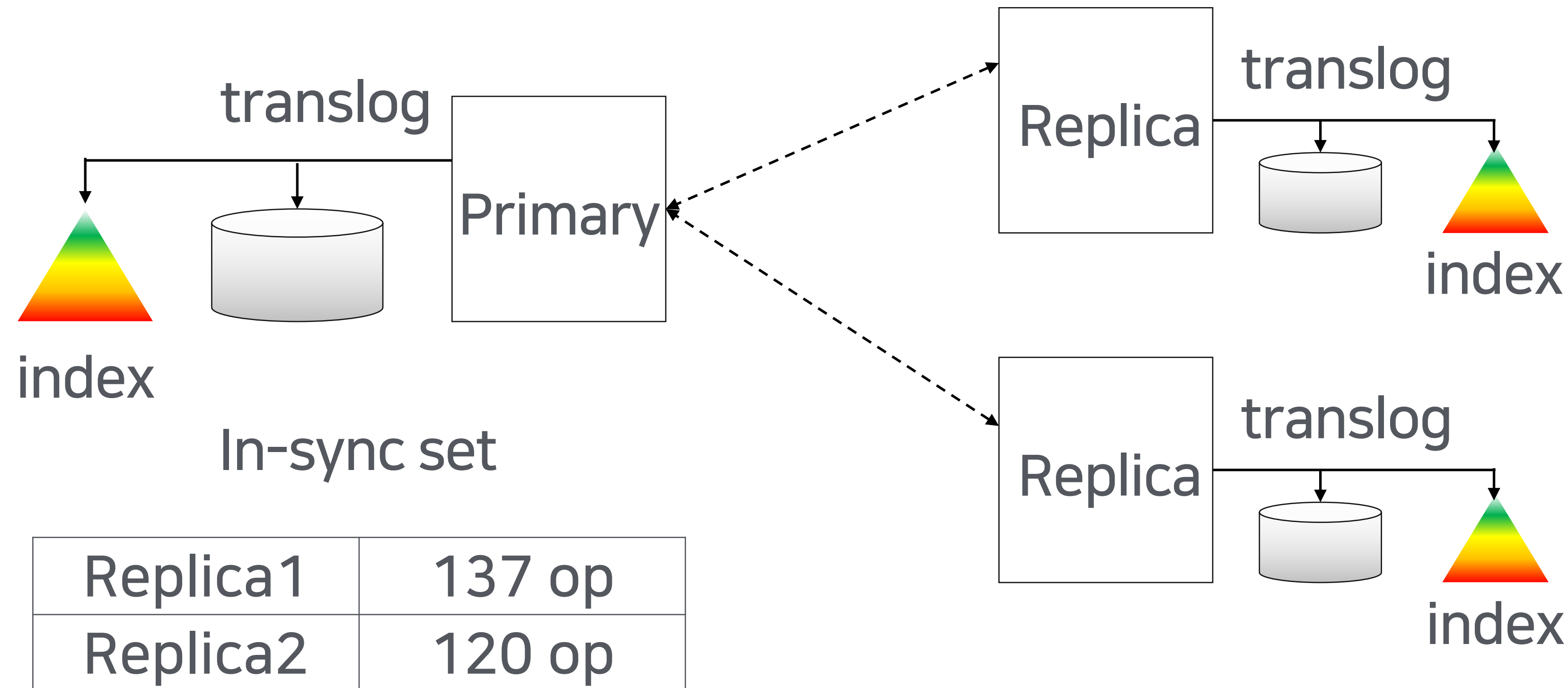
4.1 Raft와 Primary backup model

Raft, ES 클러스터링의 백본



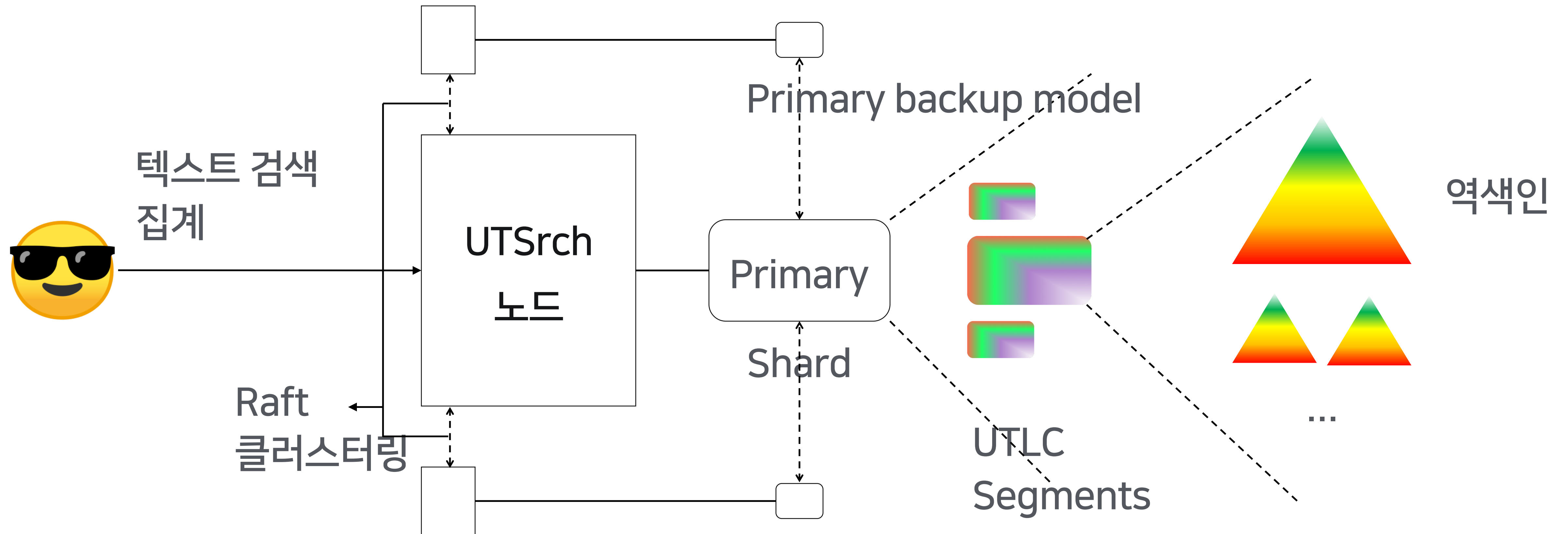
4.1 Raft와 Primary backup model

Raft위에 올려진 데이터 분산, 리커버리

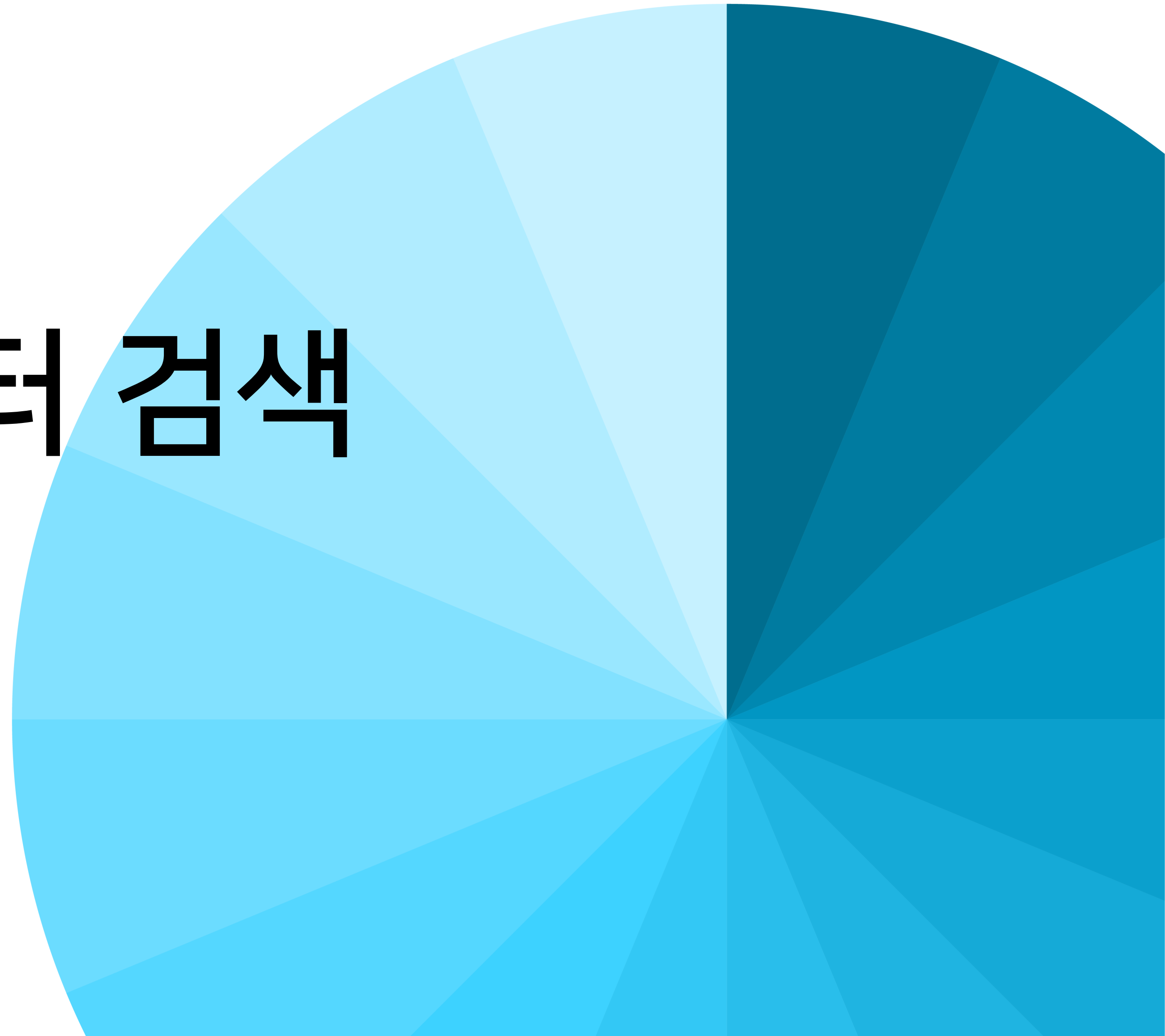


4.2 전체 뷰, Ultimate Search

UTSrch는 포르쉐, UTLC는 터보 엔진



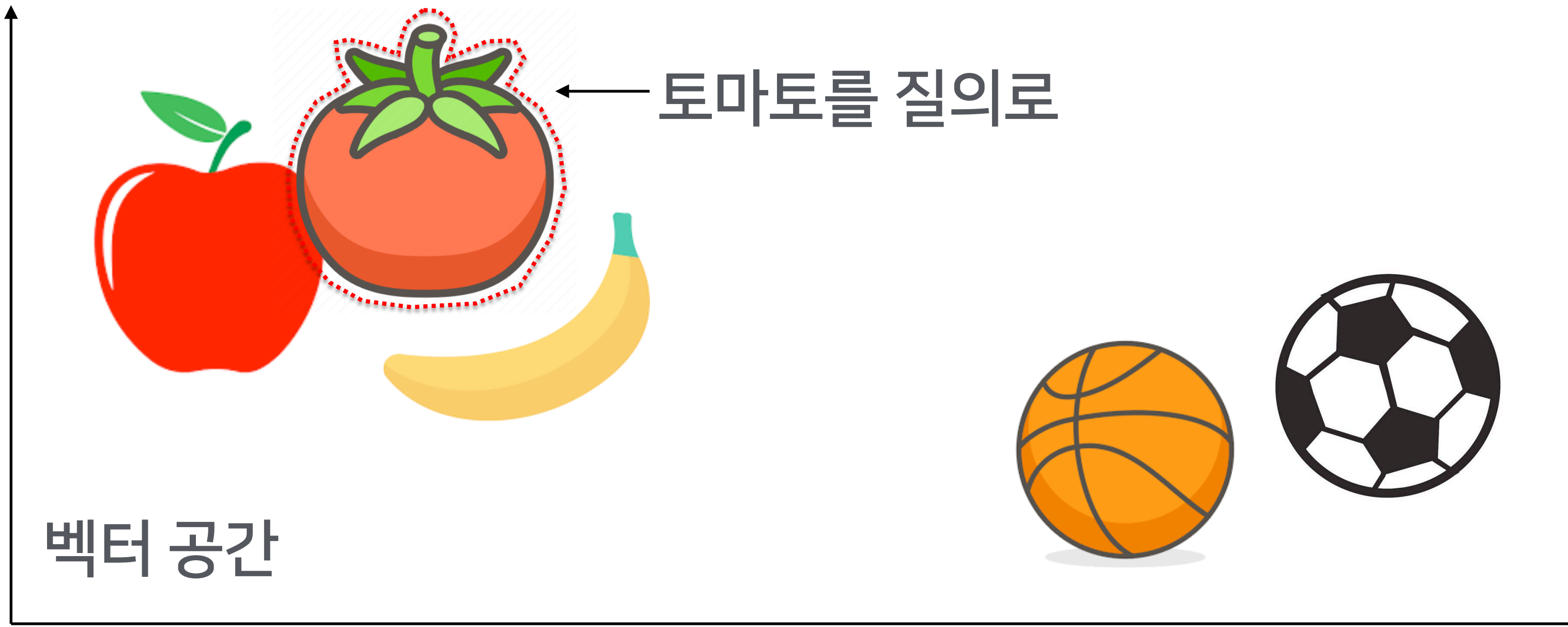
5. 벡터 검색



5.1 벡터 검색

어떤것이든 공간에 사상시킨다.

- 딥러닝의 발달, AI의 일상화
- 핵심 질문 : 어떻게 질의 벡터와 유사한 것을 찾을 수 있을까?



5.1 Nearest neighbor search

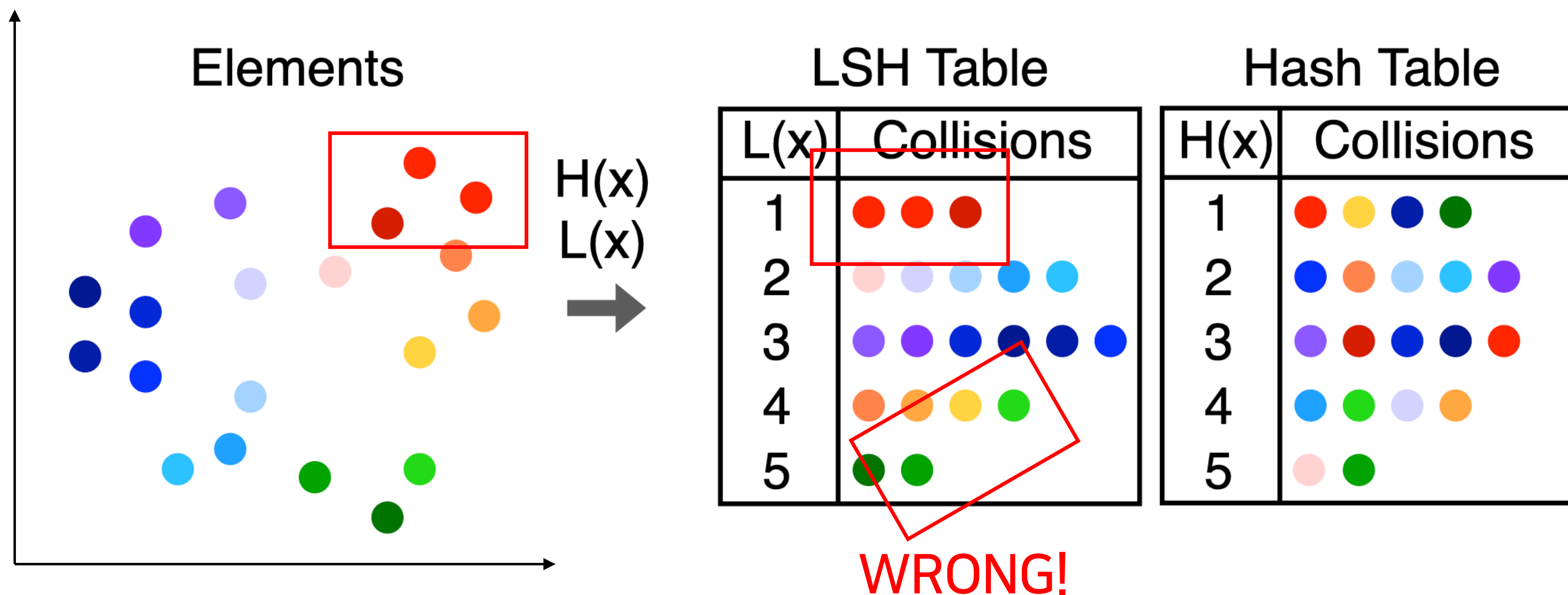
97% 정확도로 빠른 성능을!

검색 공간을 빠르게 줄여 나가는 것이 핵심

- Local sensitive hashing
- Annoy, 공간 forest
- HNSW : 멀티 레이어 공간 그래프 (선택)

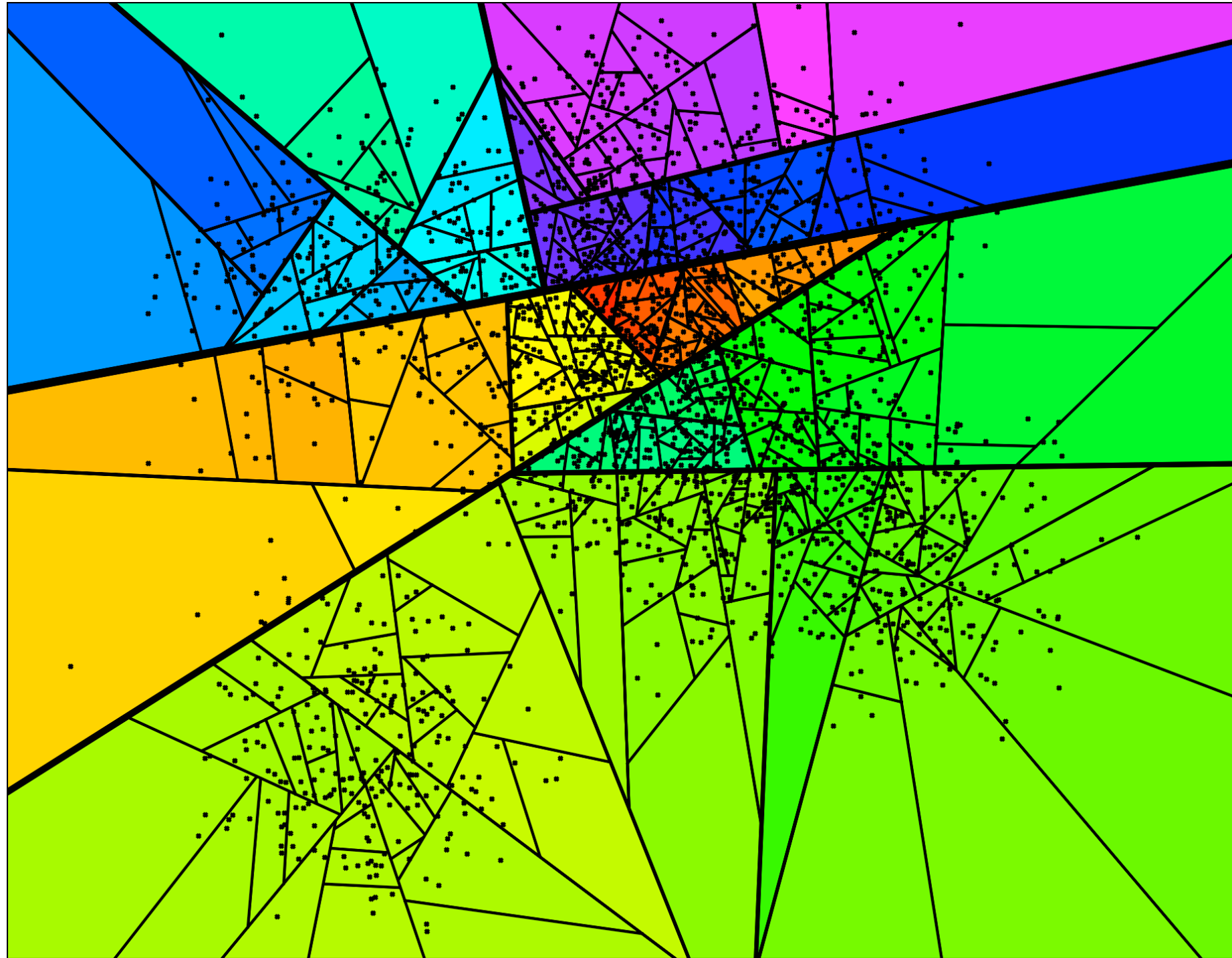
5.2 Local sensitive hashing, LSH

유사한 벡터들의 해시값은 대체로 같도록 하자

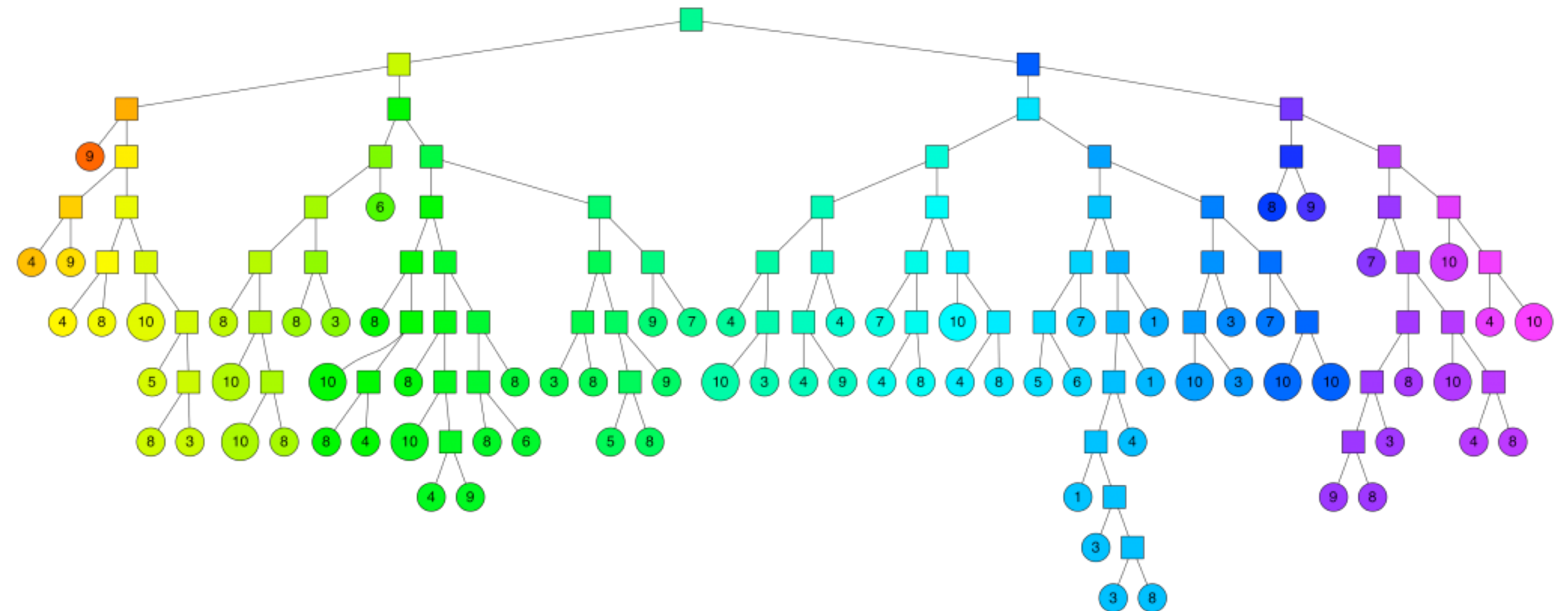


5.3 공간 forest, Annoy

랜덤 분할한 공간 트리가 N개 = 공간 forest

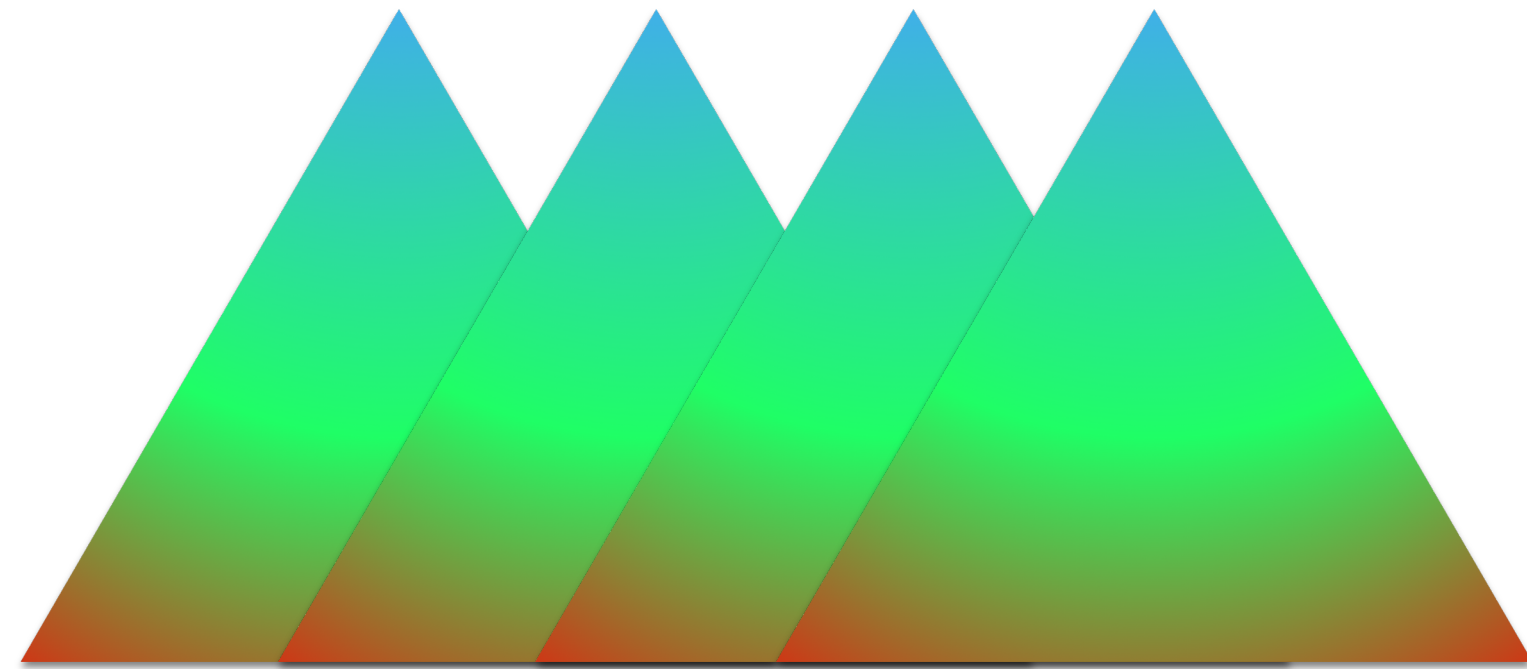


Annoy tree

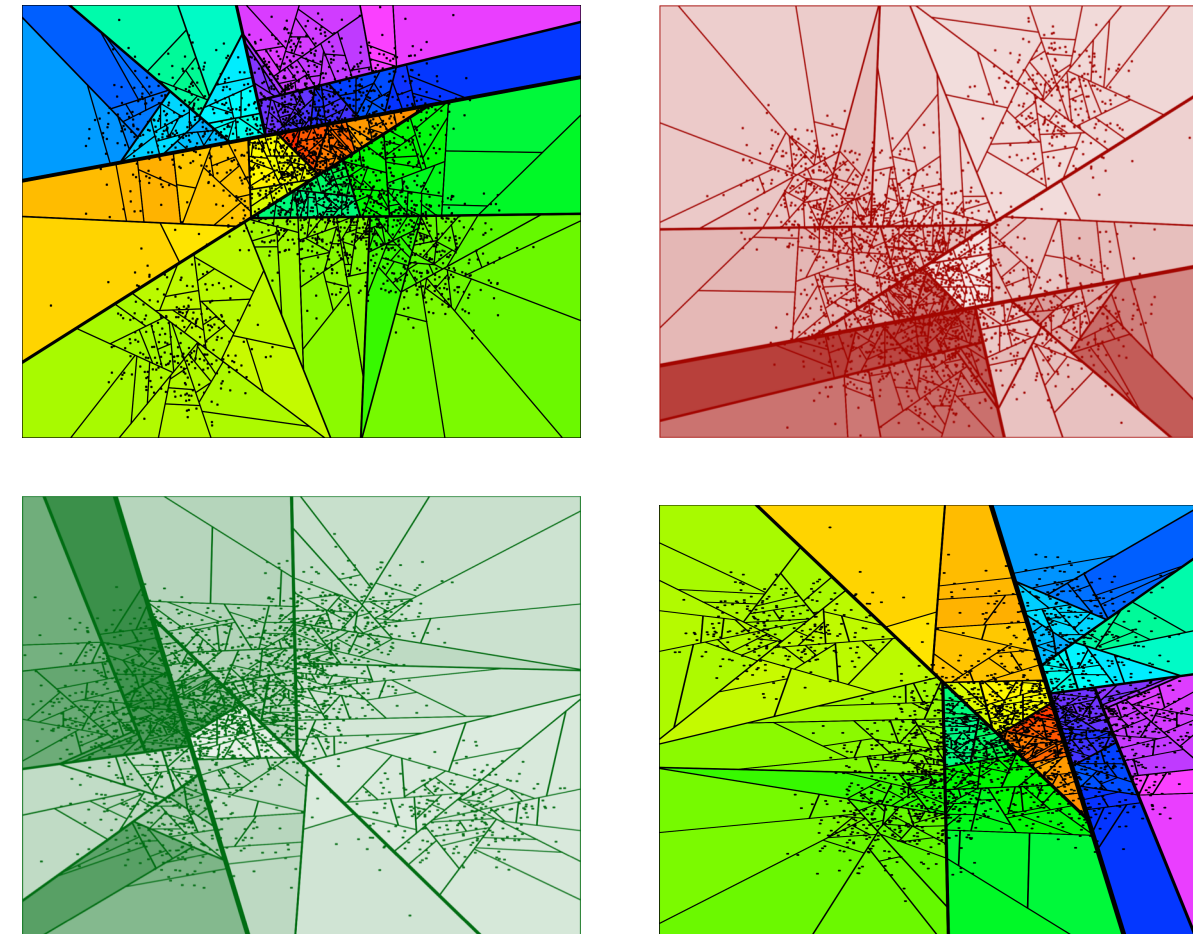


5.3 공간 forest, Annoy

랜덤 분할한 공간 트리가 N개 = 공간 forest



N개의 Random Annoy tree



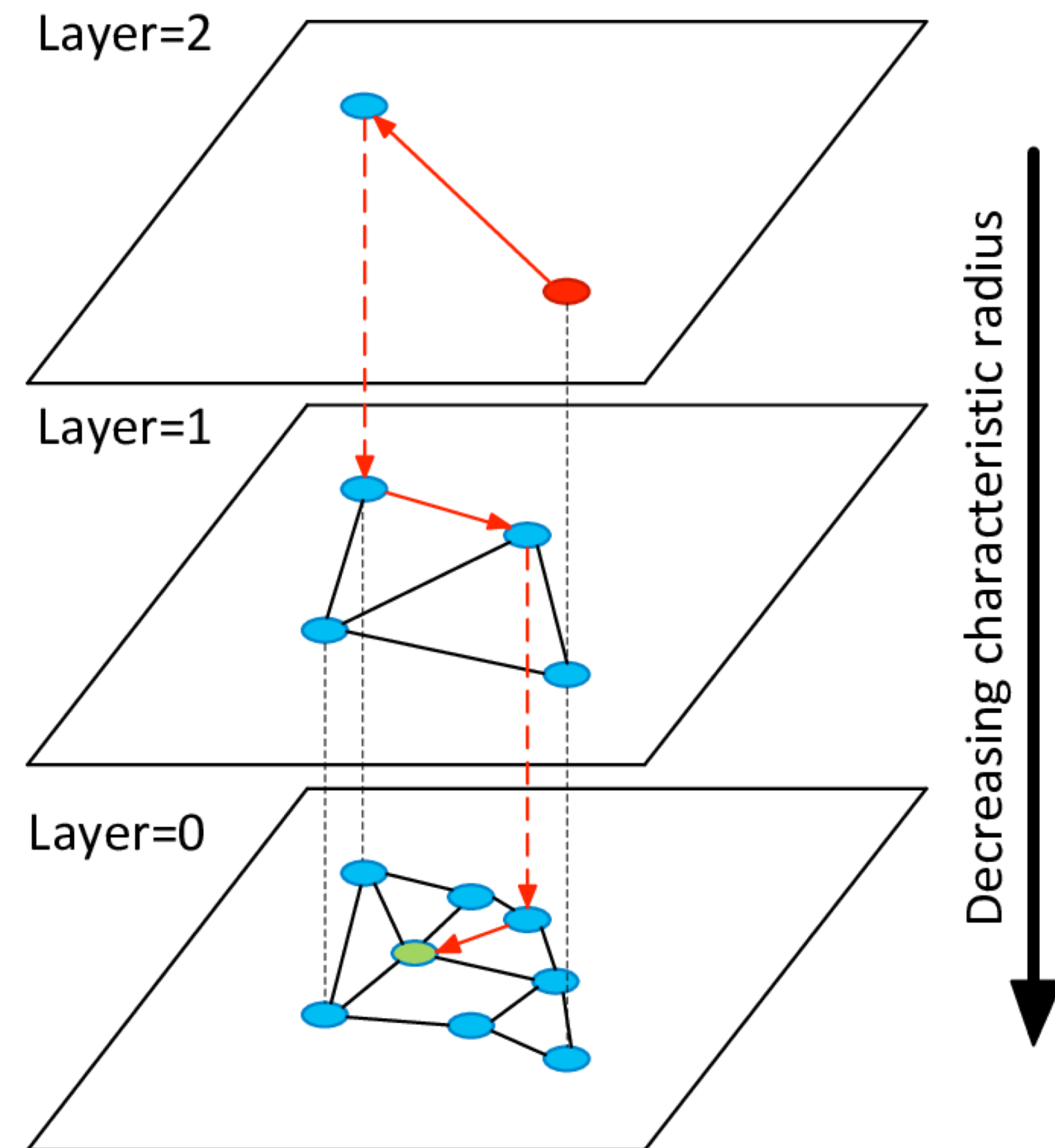
N개의 Random split spaces

5.2 HNSW

멀티 레이어 그래프 + 여러 휴리스틱

- 확률 자료구조 : Layer level
- 검색 휴리스틱
- **매우 빠르다 : 빠르게 목적지에 도달 한다.**
- 병렬 색인 구조

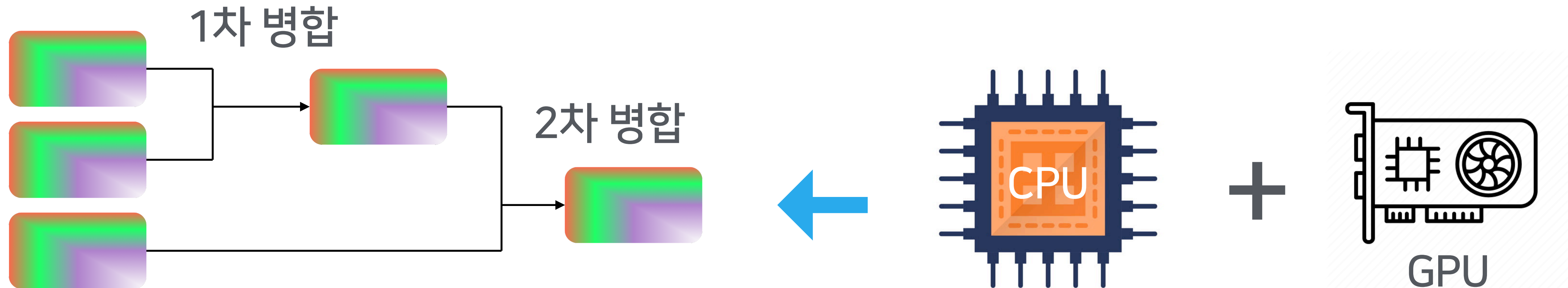
계층이 깊어질 수록 조밀하다



5.3 UTLC와의 통합

HNSW를 UTLC Codec에 추가

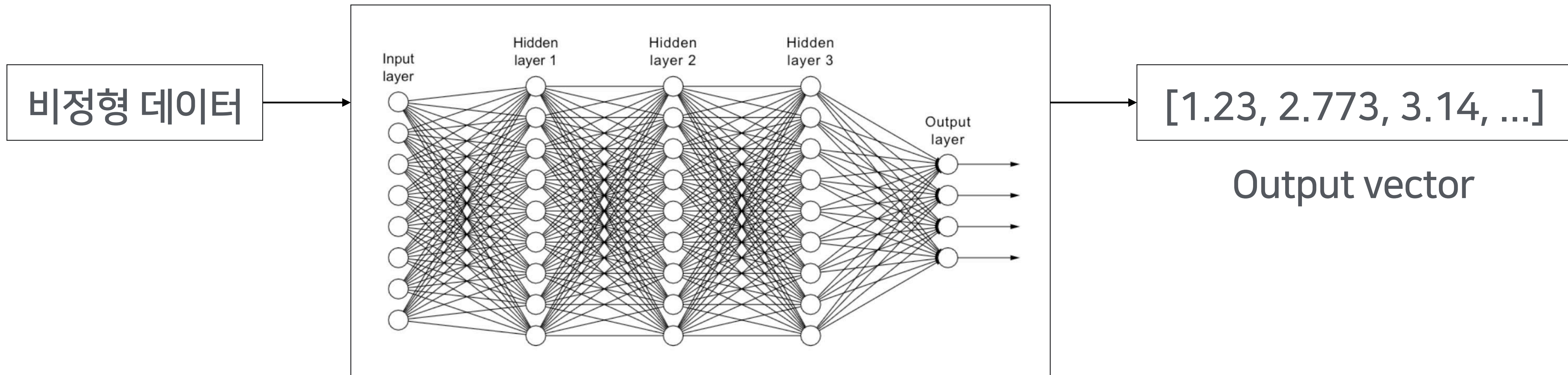
- Sub milliseconds 검색 성능. 빠르다! (600 차원, 300만건, Consine similarity)
- 루씬 증분 병합 구조의 문제 : GPU가 답인가?
- 목표 : CPU + GPU 하이브리드 모델



루씬, 증분 세그먼트 병합, 싱글코어

5.4 벡터 검색엔진

벡터를 색인하고 벡터를 검색한다!



5.4 벡터 검색엔진

벡터를 색인하고 벡터를 검색한다!



6. 검백집, 삼위일체

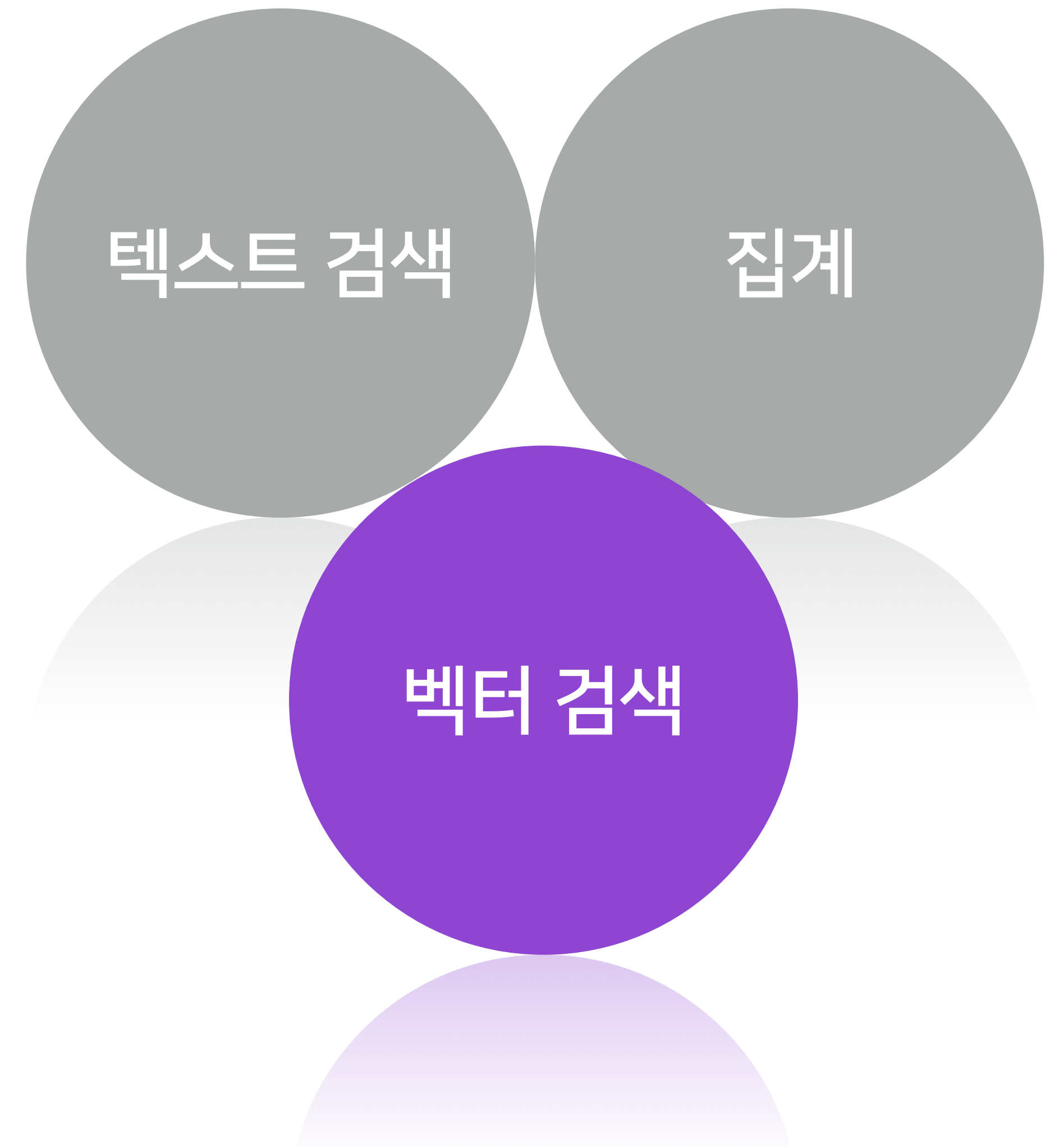
6.1 삼위 일체, Ultimate Search

텍스트 검색 & 집계

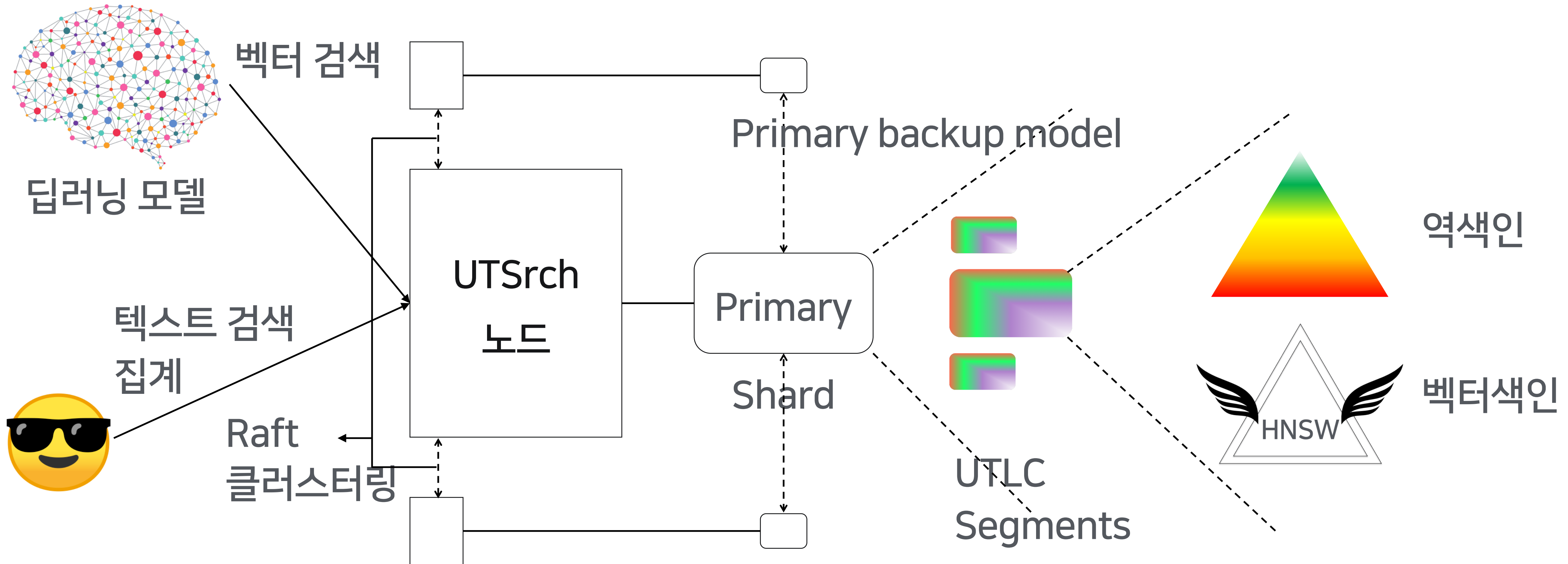
- 풍부한 Query
- No GC, 빠른 성능
- 다양한 Analyzer와 Similarity는 과제

벡터 검색 : 벡터를 던져요 벡터를 줄거요

- 모델러는 좋은 벡터 생성 모델에 집중
- 낮은 허들, ES와 동일한 매커니즘
- 다양한 플러그인은 과제



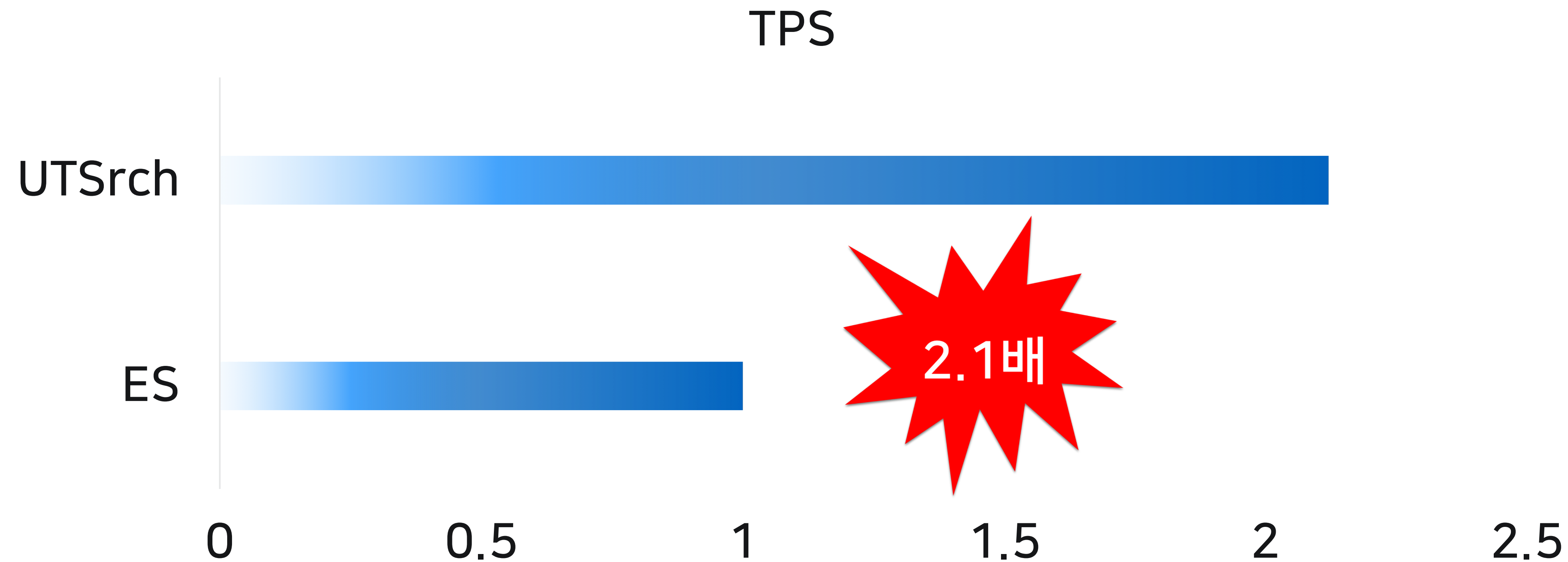
6.1 삼위 일체, Ultimate Search



7. 성능평가

7.1 Conjunction 평가

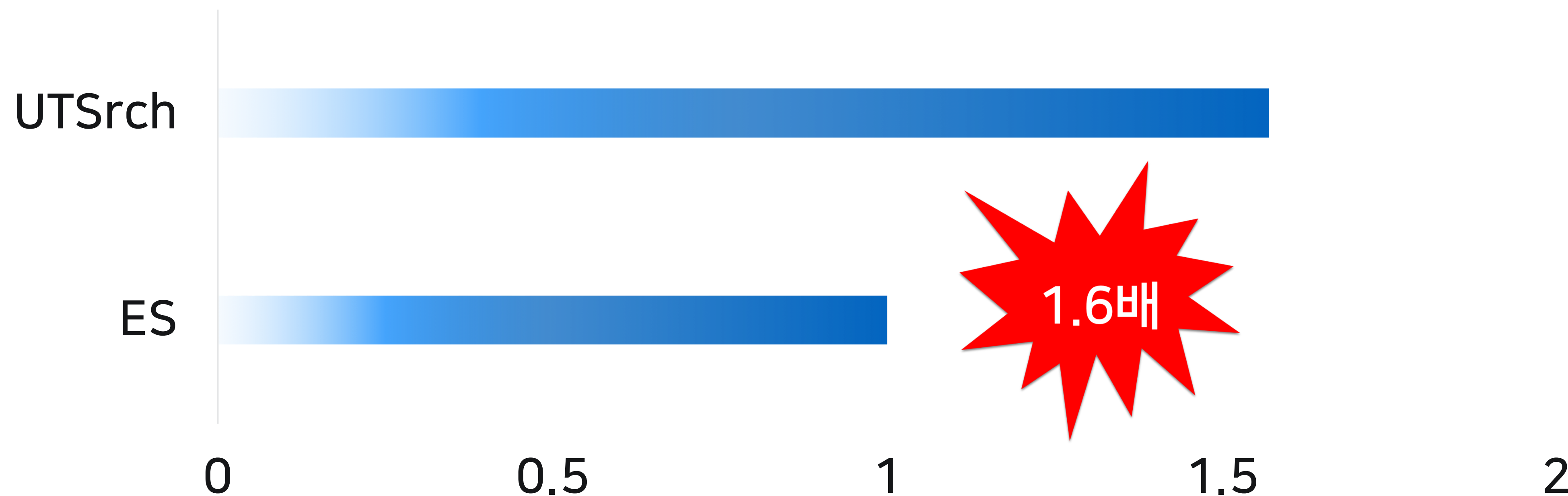
2,000만건, 128G, 32 Cores



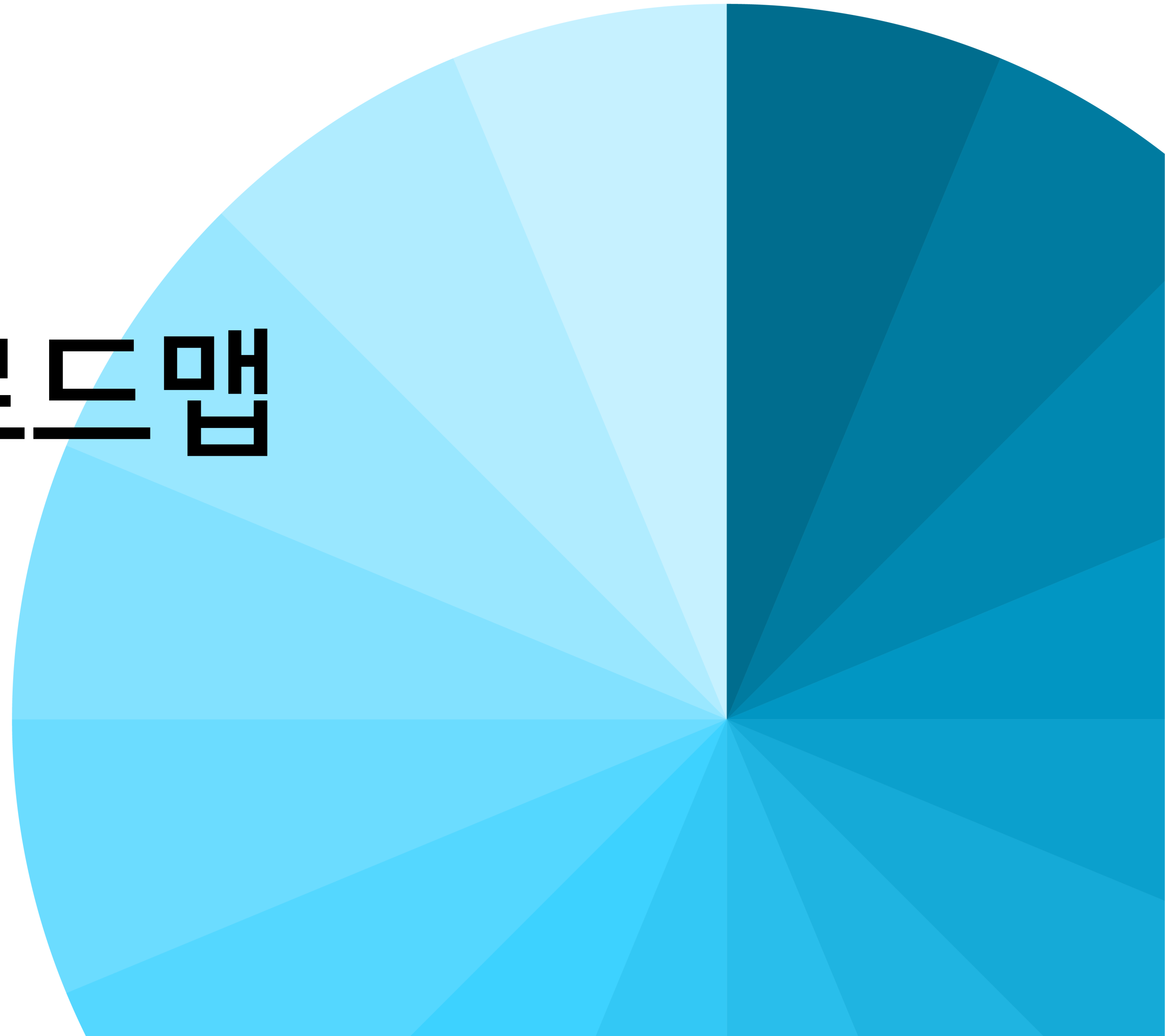
7.2 Disjunction 평가

2,000만건, 128G, 32 Cores

TPS



8. 로드맵



8.1 테스트 테스트 테스트

어떻게 테스트를 해야 할까?

어떻게 메모리 Leak을 점검할까?

어떻게 프로파일링 할 수 있을까?

어떻게 연속성있게 버전업을 할 수 있을까?

어떻게 통합 테스트를 해야 할까?

...

8.2 플러그인

어떻게 플러그인을 쉽게 작성할 수 있을까?

어떻게 개발자 친화적인 엔진이 될 수 있을까?

어떻게 문서를 작성할까?

어떻게 채널을 효율적으로 개설할까?

어떻게 많은 훌륭한 개발자들과 같이 개발 할 수 있을까?

...

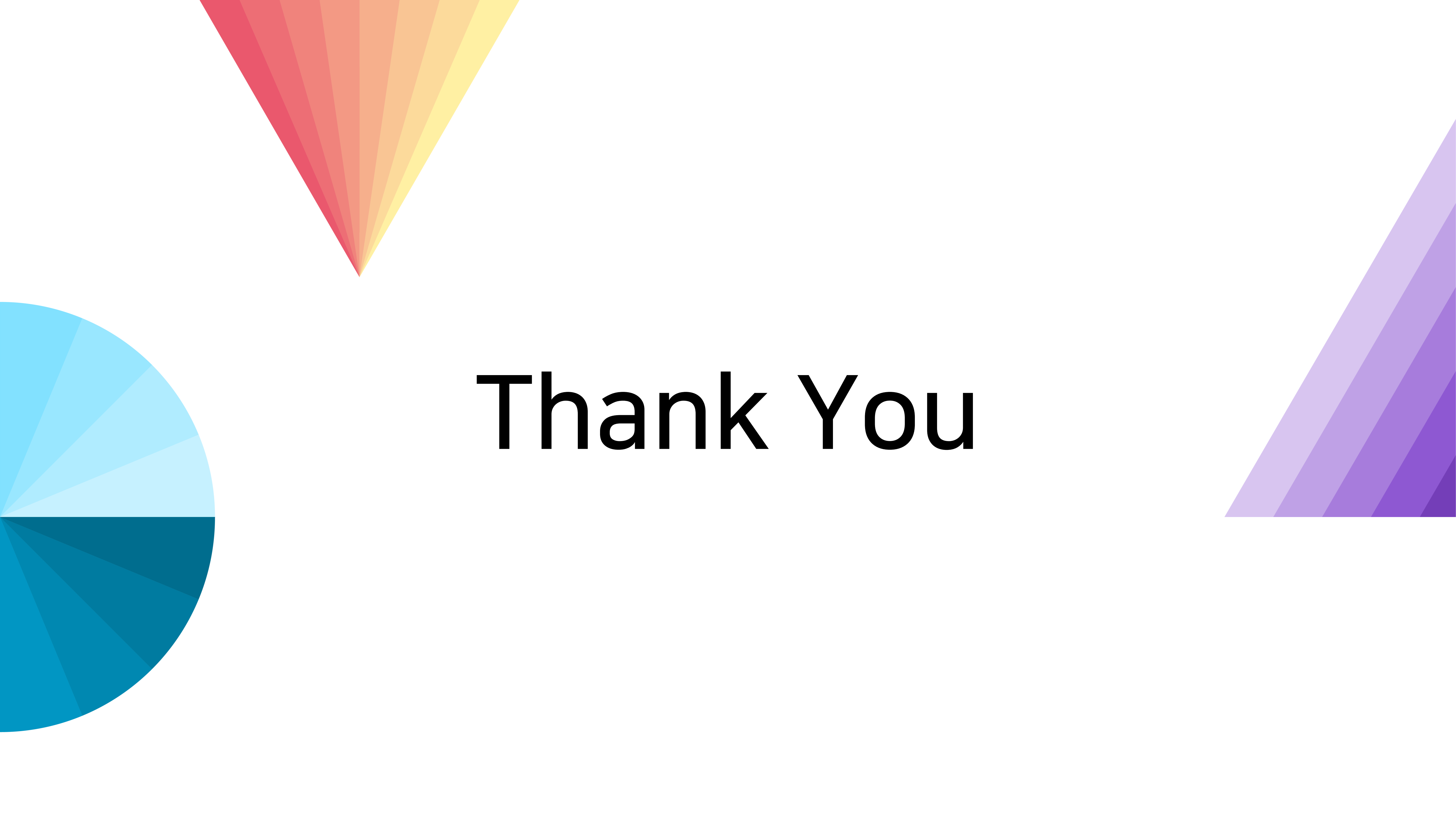
세상은 더이상 멈추지 않는다
GC없는 더 빠른 검색

이제, 무엇이든 검색할 것이다.
벡터 검색

Ultimate Search 2021 Coming Soon



Q & A



Thank You