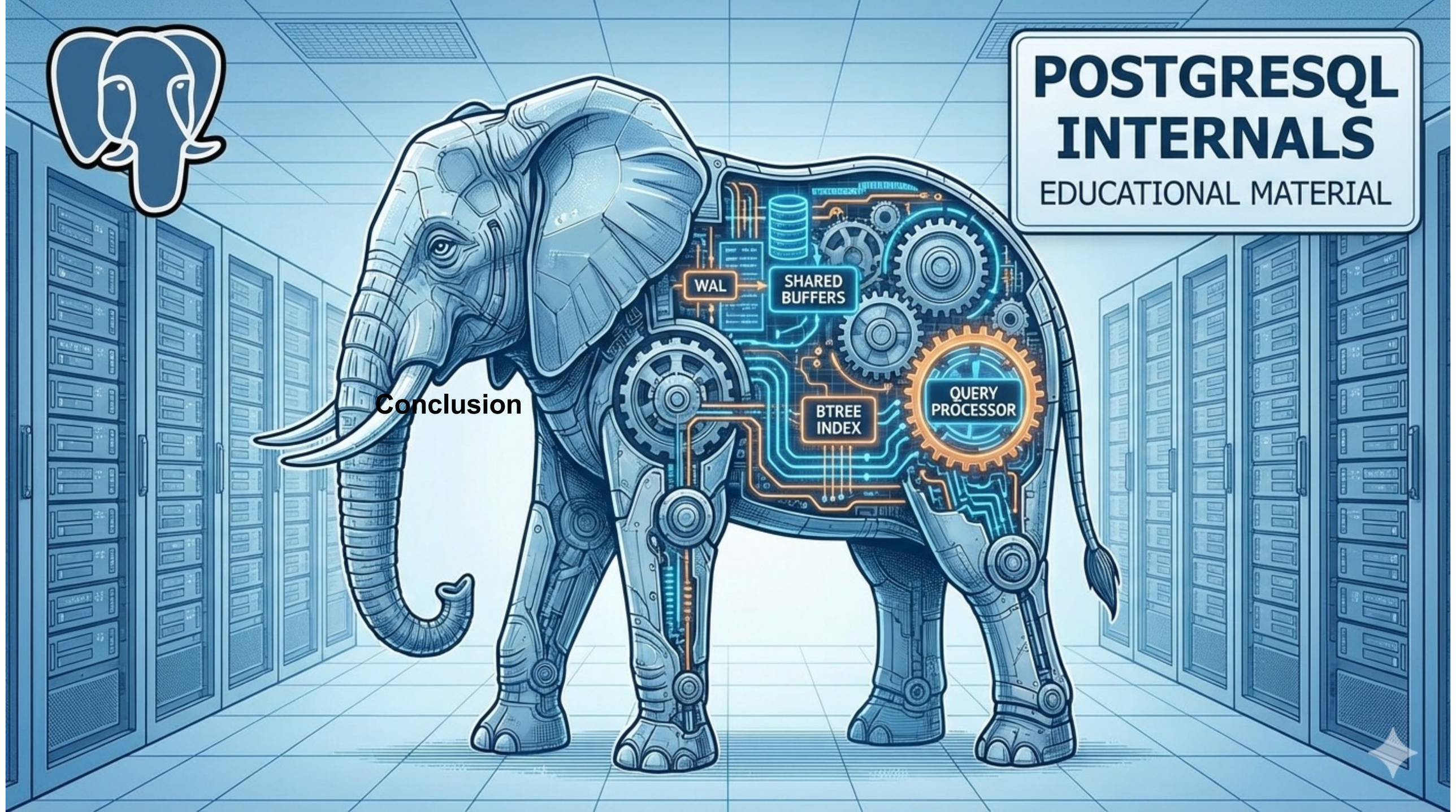

POSTGRESQL
INTERNALS
EDUCATIONAL MATERIAL
WAL
SHARED BUFFERS
BTREE INDEX
QUERY PROCESSOR
Conclusion

[PostgreSQL 교육 자료]
데이터베이스 전문가 윤명식 강사 프로필 요약

25+

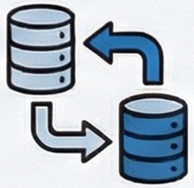
25년 이상의 DB/DW/BI 컨설팅 및 구축 속편도

아키텍처 설계부터 데이터 모델링, 설우 구현까지 데이터베이스 생평무기 전변에 뭘친 25년 이상의 목보처인 전문성을 보유하고 있습니다.



70% 이상의 데이터베이스 성능 최적화 달성

SQL 돔닝 및 아키택처 게선율 통해 60~70%의 성농 양성을 실뱐하고, 고비웅 레거서 DB(Oracle)를 오돈소스 기만으로 성공적으로 전형한 설적을 보유하고 있습니다.



EnterpriseDB 및 PostgreSQL 마이그레이션 전문가

EDB BDR을 활용한 99.999% 가응성 설계과 Oracle에서 Postgres로의 무용단 마이그레이션 전략 수립에 톡화되어 있습니다.



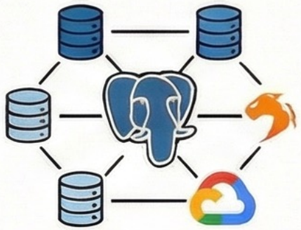
프로그래밍 및 자동화 역량

SQL, Shell Scripting, Java, Python들 활용하여 데이러 로팀, 변환 및 자동화 업무를 효율적으로 수쵕합니다.



주요 프로젝트 성과

삼성전자 GCP PoC, 신완은행 및 신한카드 상능 화적화 지원, 하나운행 차세대 시스템 EDB DBA 등 금융 및 대형 제조사 프로줴트를 성공적으로 이믐앴습니다.



RDBMS 및 NoSQL을 아우르는 기술 스벡트럼

PostgreSQL, Oracle과 같은 관계협 DB는 물론 MongoDB, TigerGraph, Couchbase 등 다양한 비관계할 DB 및 믈라부드(AWS, GCP, Azure) 환경에 능숙합니다.



기술 에반젤리스트 및 검증된 강사

SECaaS, Devfest 등 주요 IT 캔피런스에서 그래프 DB 및 데이터 기술 트랜드에 대한 갑연을 진행히며 지식 공믝에 망정서고 있습니다.

강사의 전문성을 증영하는 주요 인증 현황 및 학력

학력	한섬대학교 전산통계공학 학사
주요 자격동	AWS Certified Solution Architect, Snowflake Pro Core, SQL Professional (SQLP)
특화 자격	TigerGraph Associate, Couchbase Certified Architect/Developer/Admin

소통 채널 및 포트플리오

블로그(www.ninanolabs.com)

LinkedIn을 통해 최신 기술 동향과 프로젝트 인사이트를 공유하고 있습니다.

연락처 정보

이메일(jazzlian@gmail.com)

모바일(82-10-8948-9592)

전문적인 기술 문의 및 협업이 가능합니다.

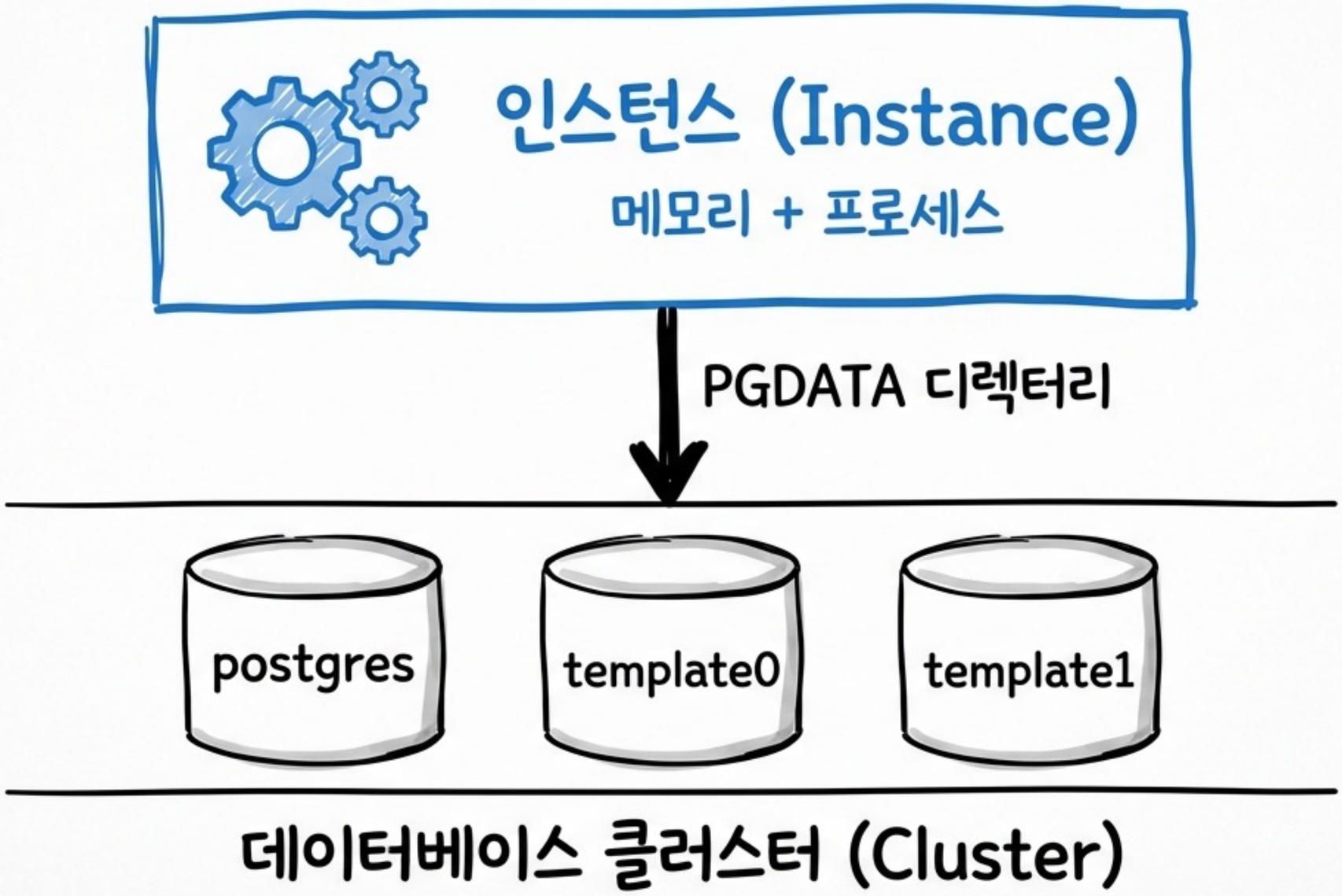
"지속적인 학습과 공유를 즐기는 기술 에반젤리스트"

IT 도서 번역 및 베타러딩, 제조와 목서를 사랑하는 인문학적 소량을 깃춘 기술 리더로서 소총합니다.

NotebookLM

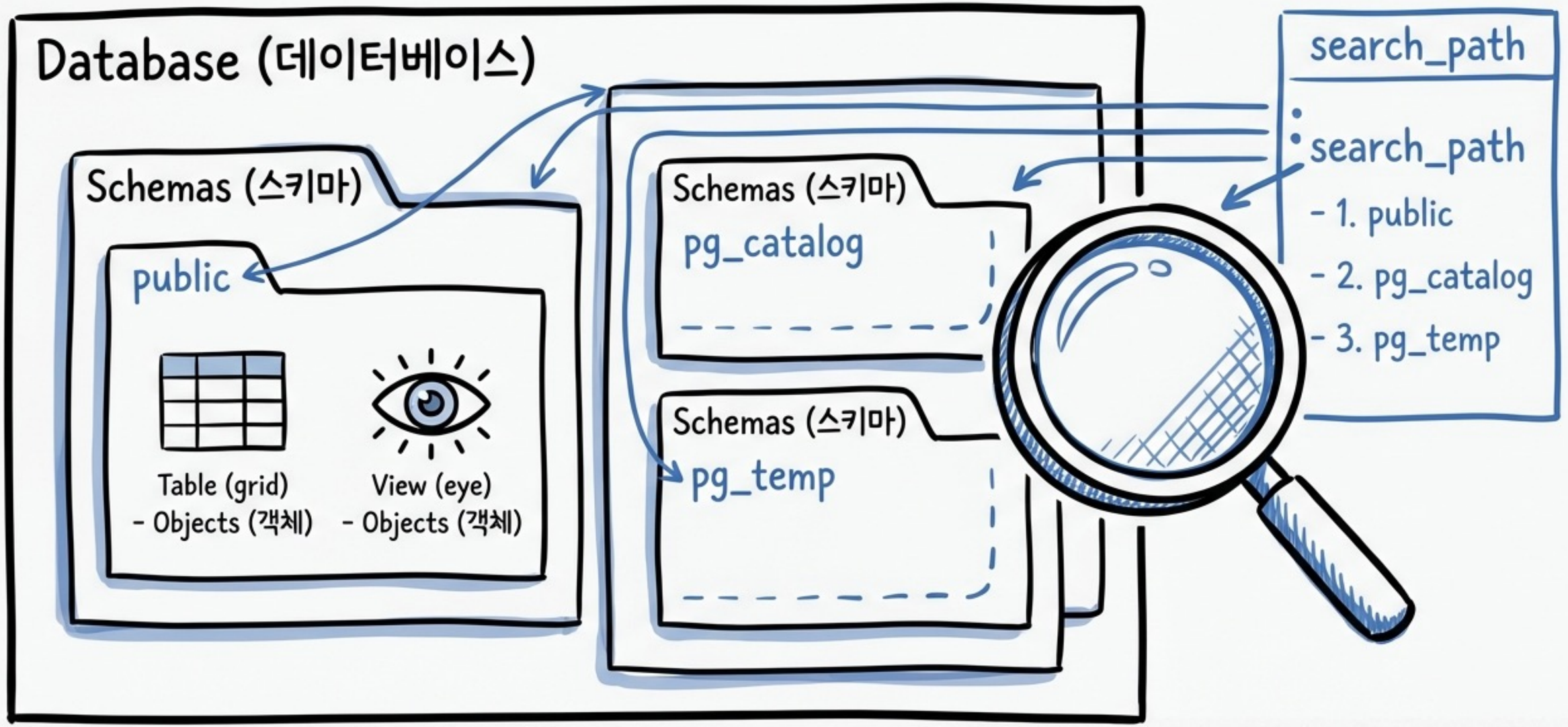
1. Introduction

PostgreSQL의 본질: 인스턴스와 클러스터



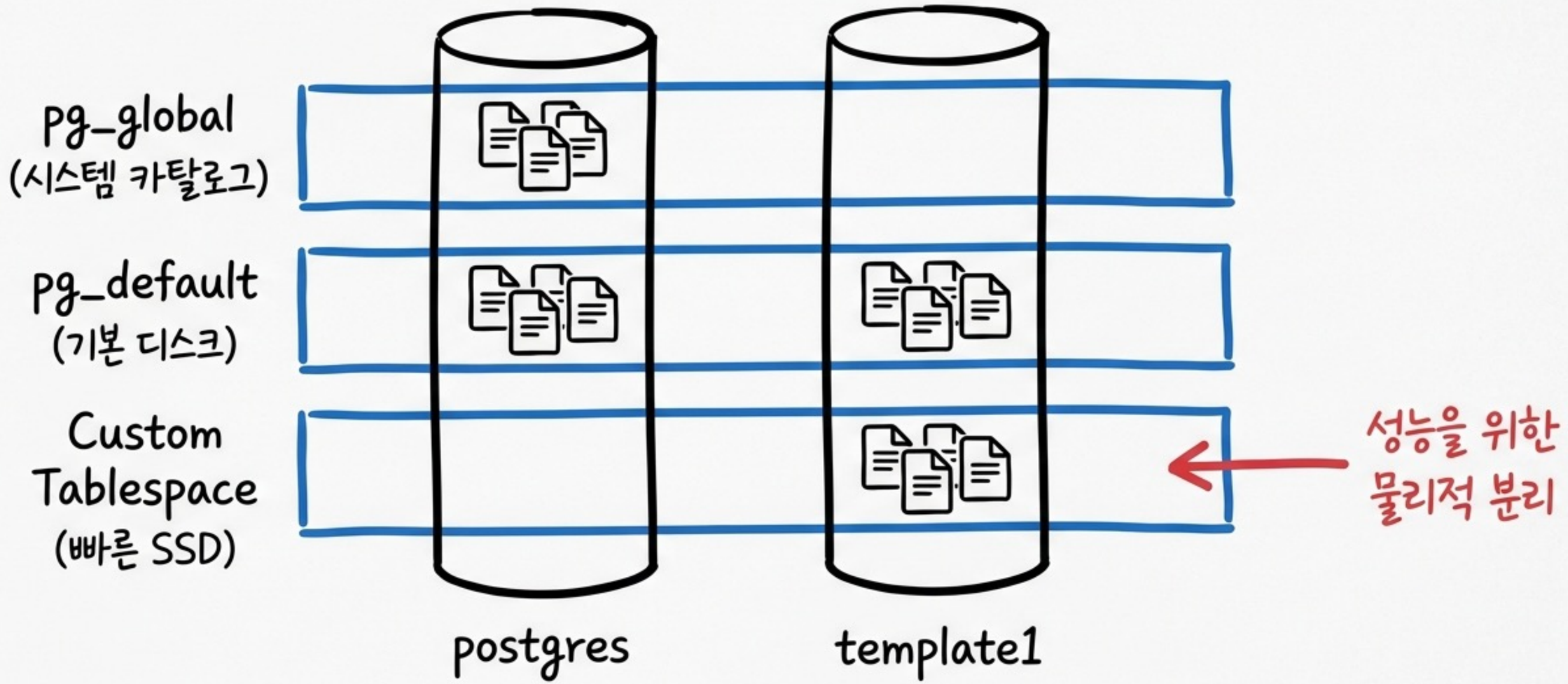
NotebookLM

논리적 구조: 데이터베이스와 스키마



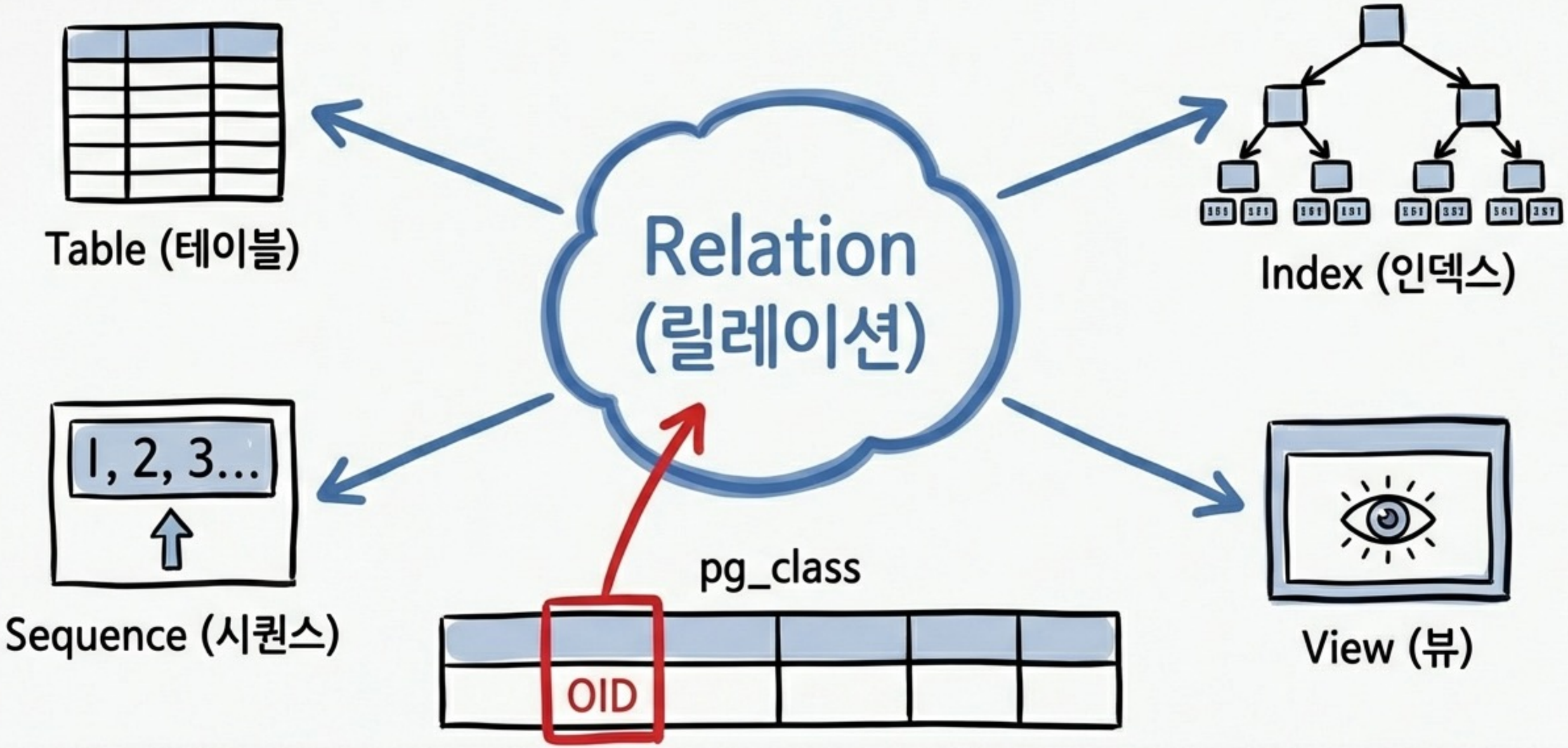
NotebookLM

물리적 배치: 테이블 스페이스



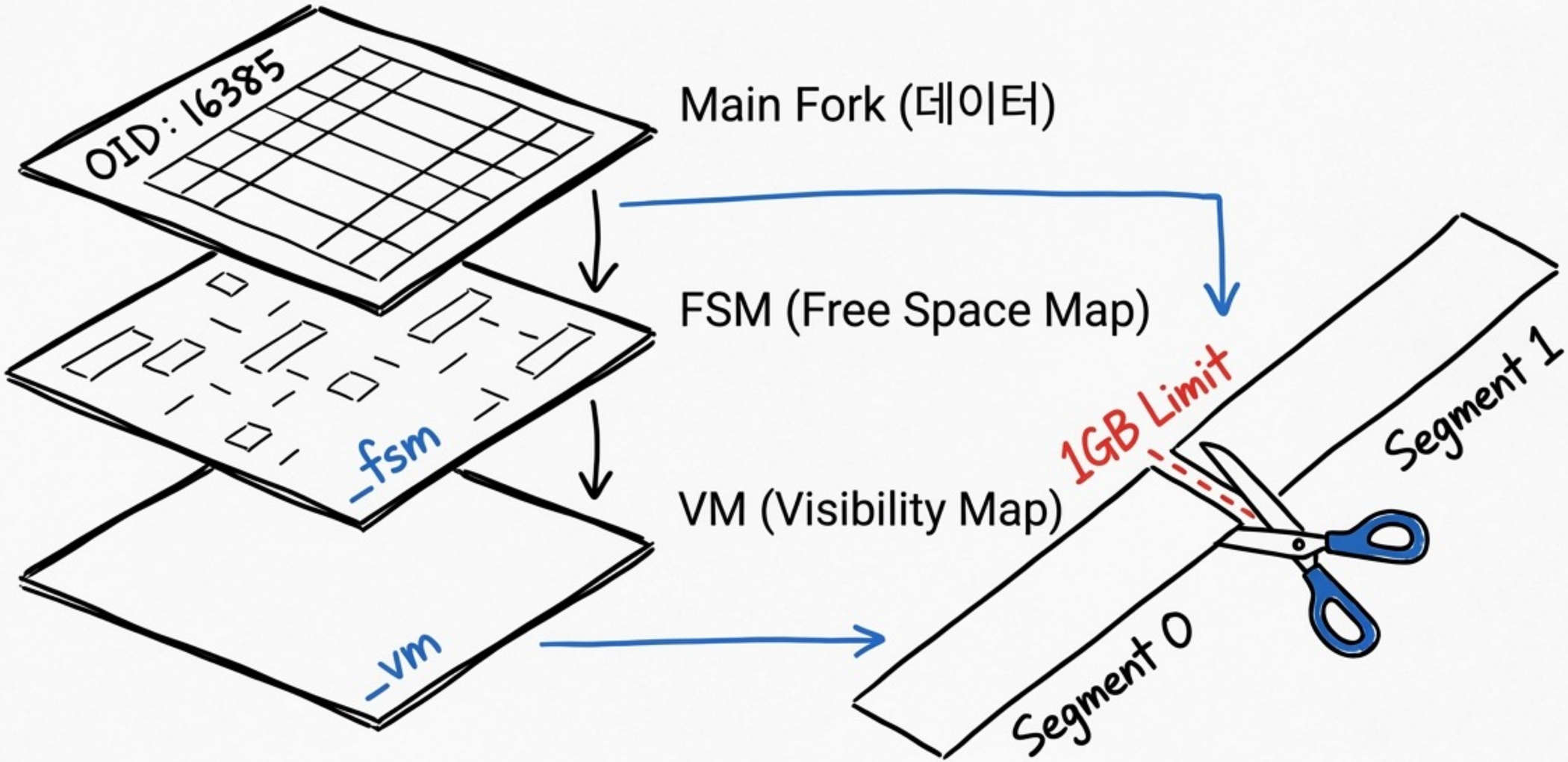
NotebookLM

모든 것은 '릴레이션'이다



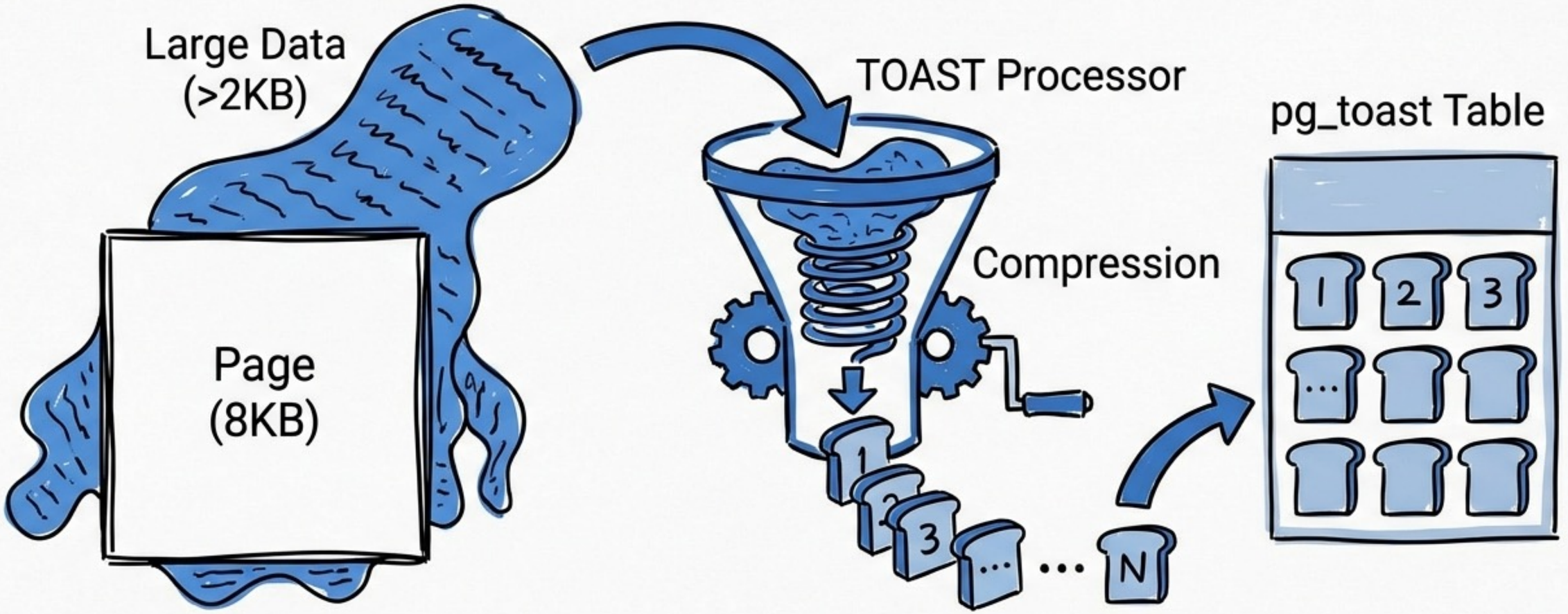
NotebookLM

파일과 포크: 릴레이션의 내부



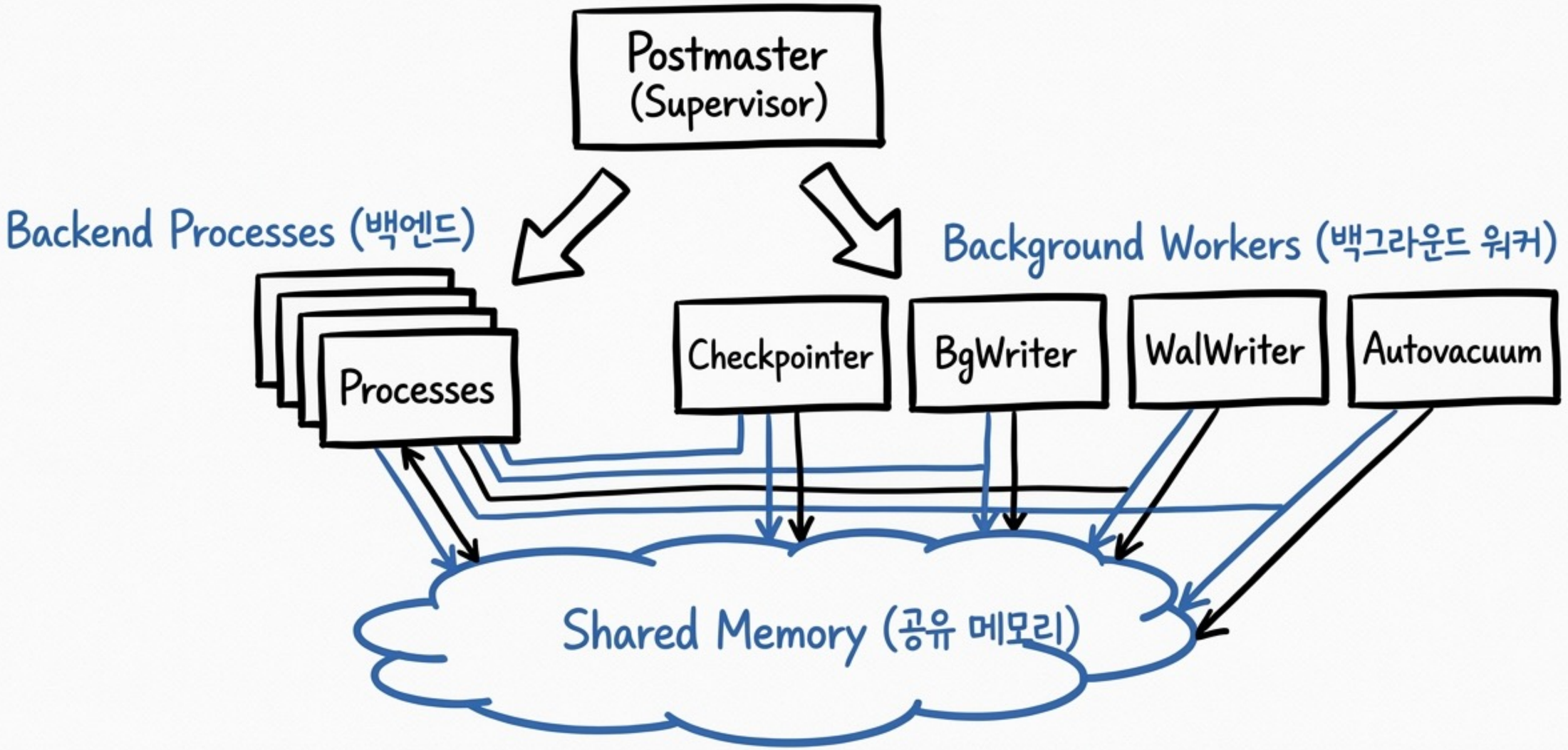
NotebookLM

페이지 구조와 TOAST



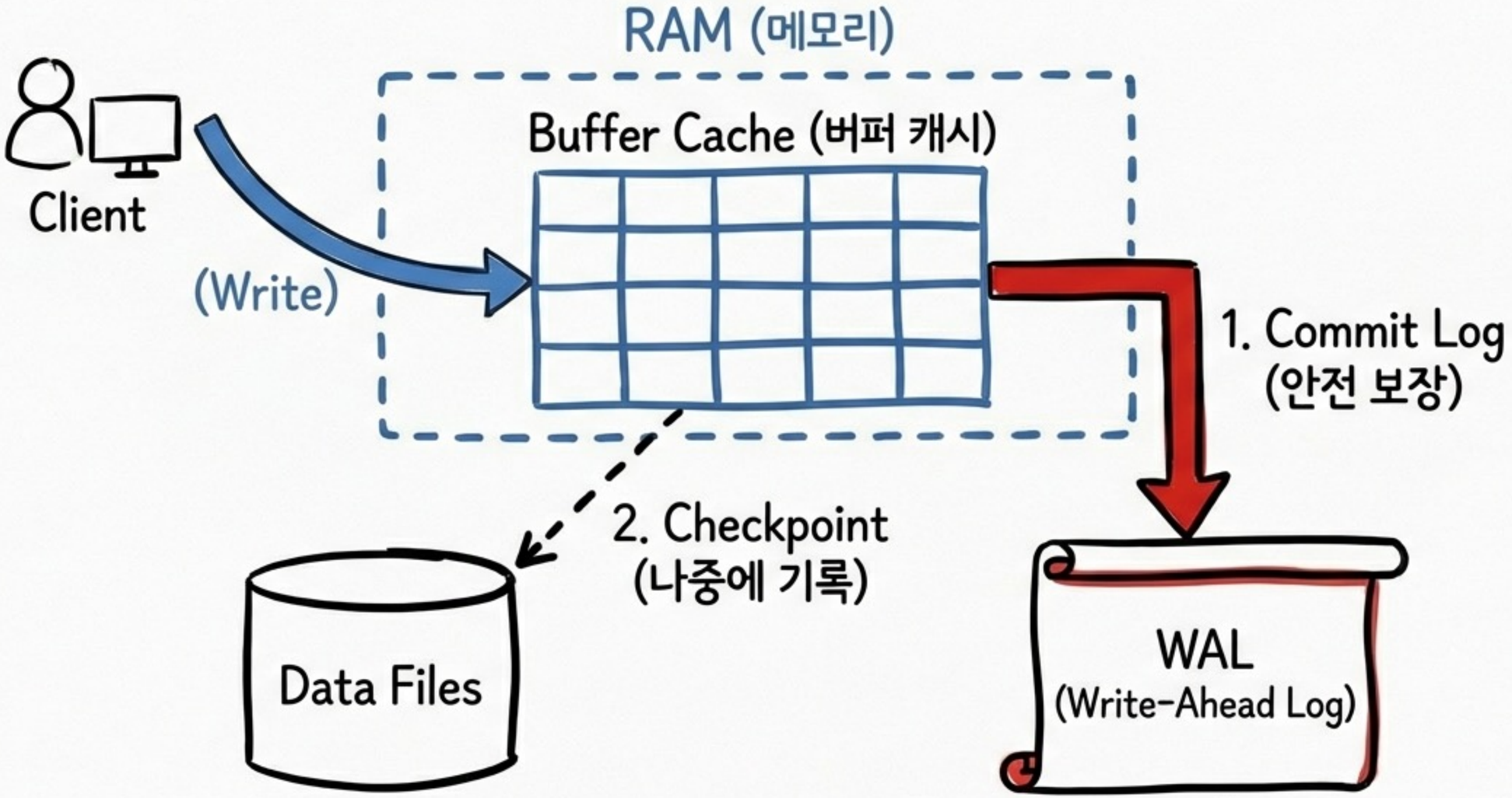
NotebookLM

프로세스 아키텍처



NotebookLM

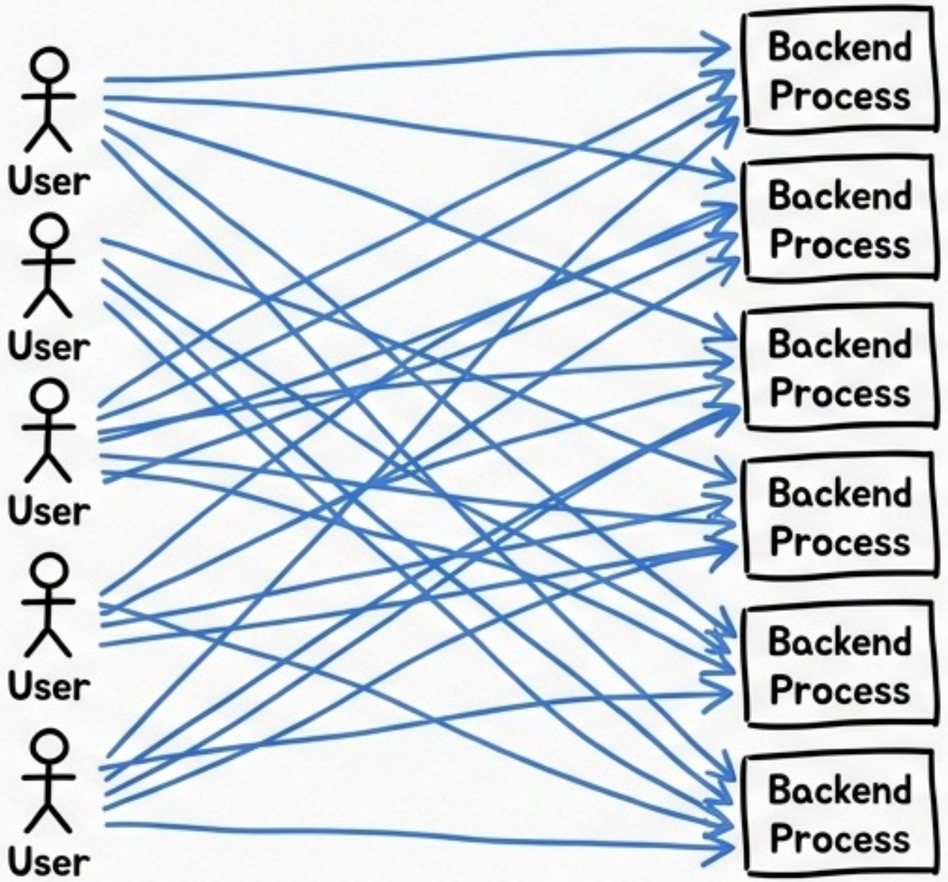
메모리와 데이터 안전성: 버퍼와 WAL



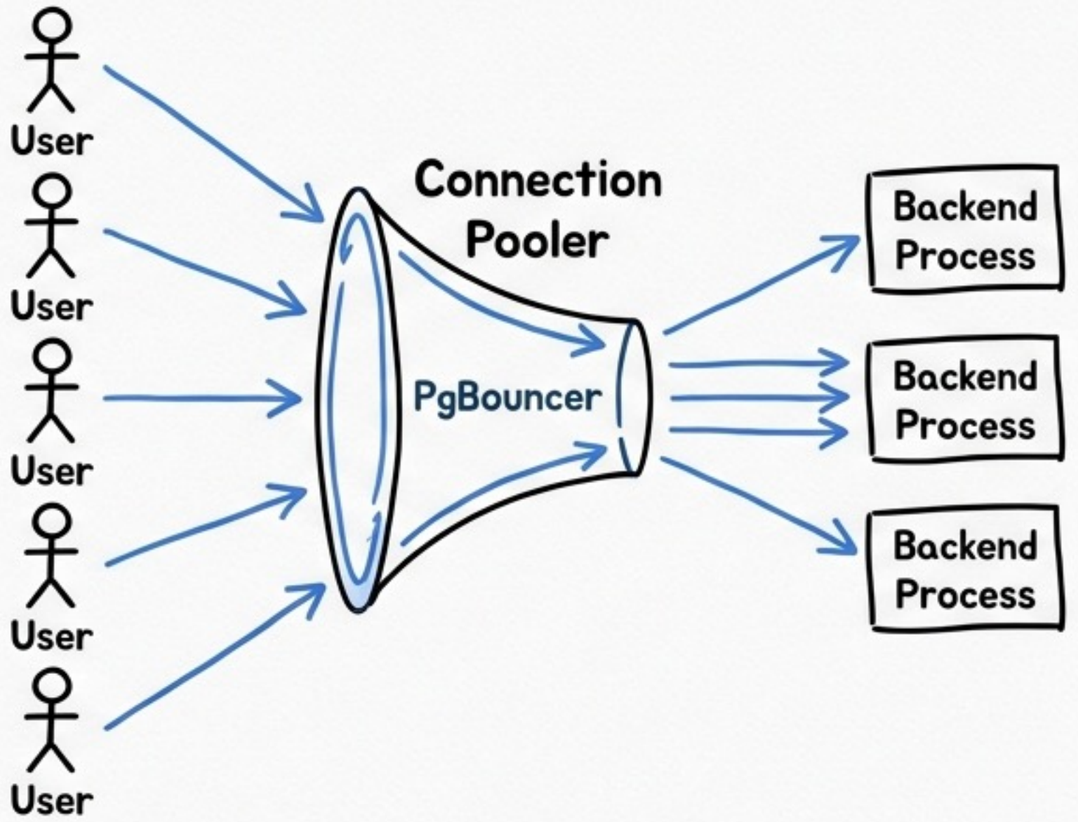
NotebookLM

클라이언트 연결 모델

No Pooling (비효율)

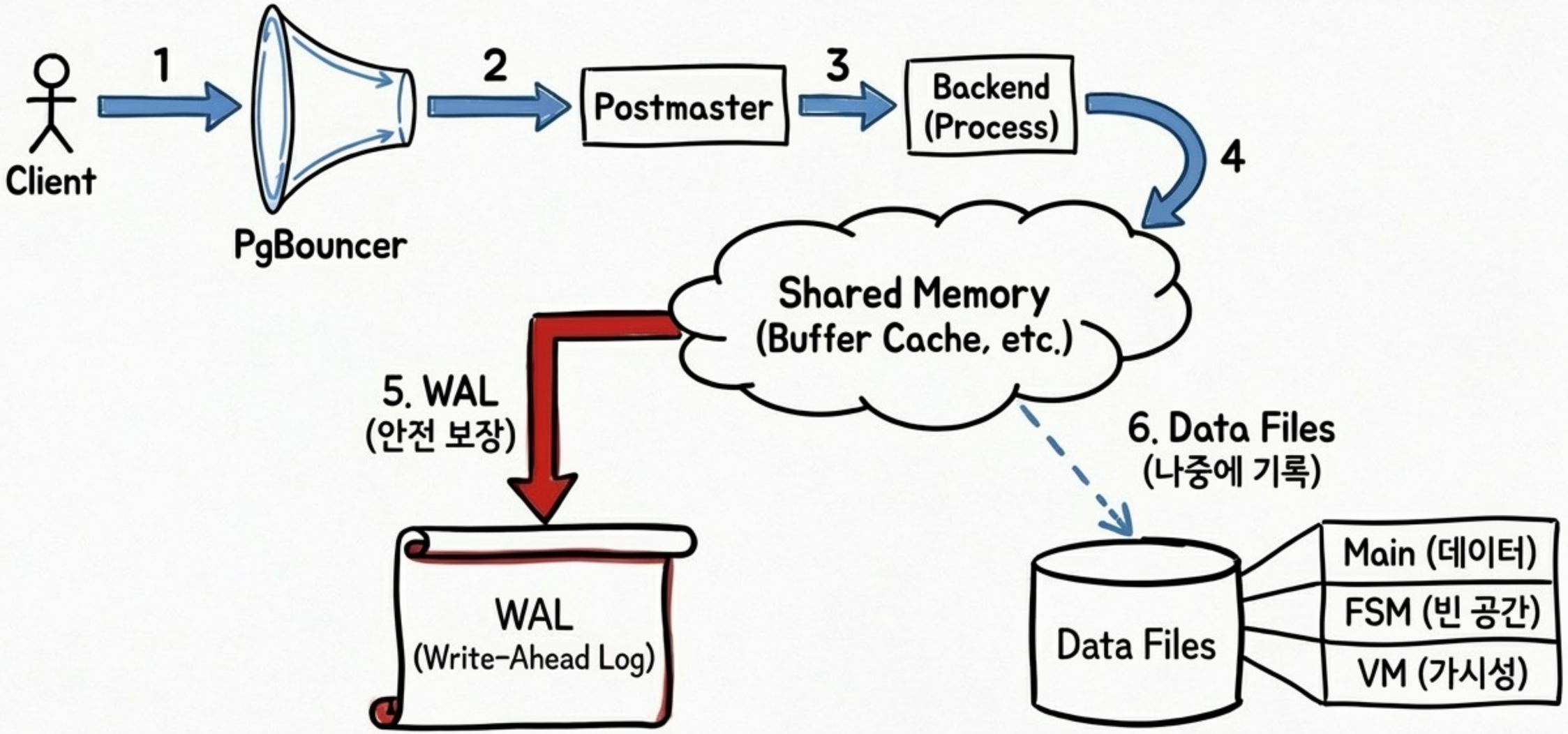


PgBouncer (효율적)



NotebookLM

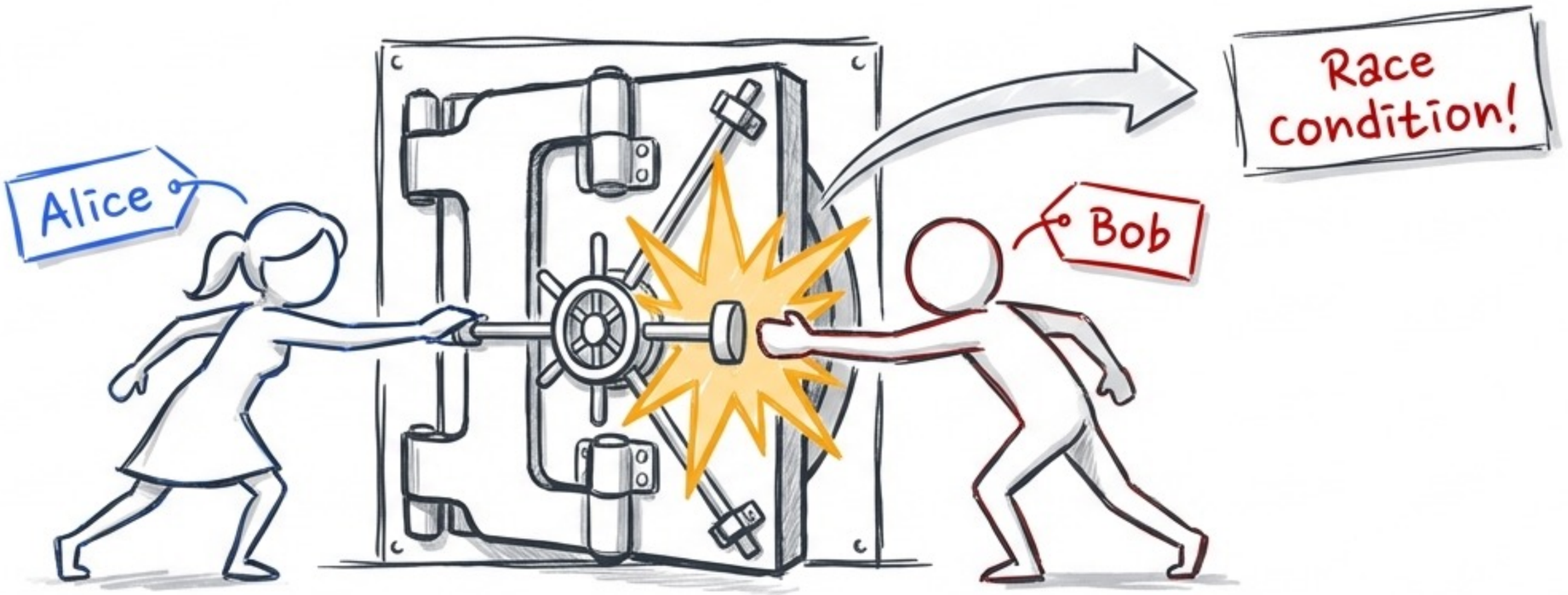
PostgreSQL 전체 아키텍처 요약



NotebookLM

- **2. Isolation**

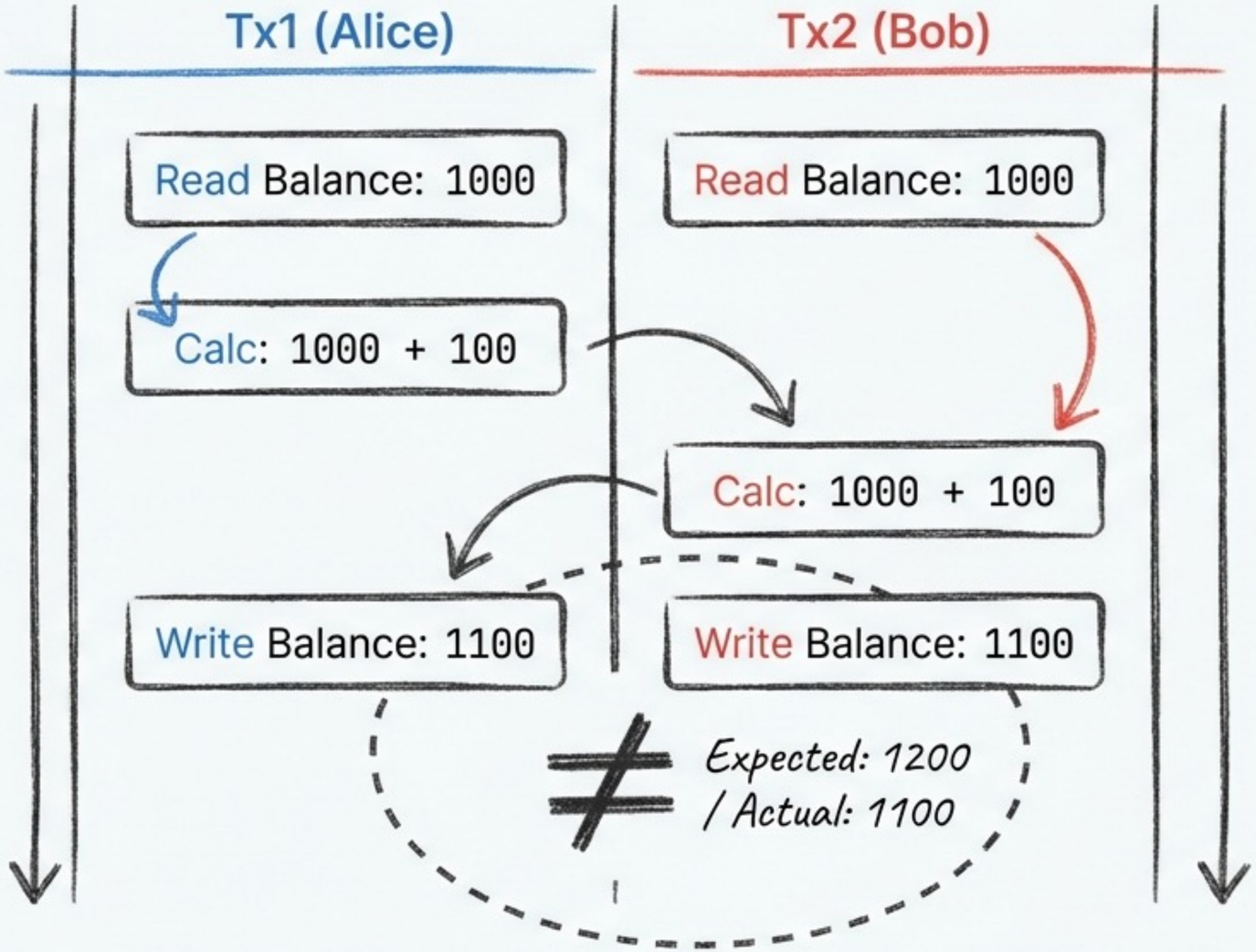
동시성은 어렵습니다: 격리(Isolation)와 일관성(Consistency)



- 관계형 데이터베이스의 핵심은 **데이터 일관성** (Data Consistency)입니다.
- 무결성 제약 조건(Integrity Constraints)만으로는 부족합니다.
- 문제: 애플리케이션이 올바르게 작동해도, **동시 실행** (Concurrent Execution)이 데이터를 파괴할 수 있습니다.

NotebookLM

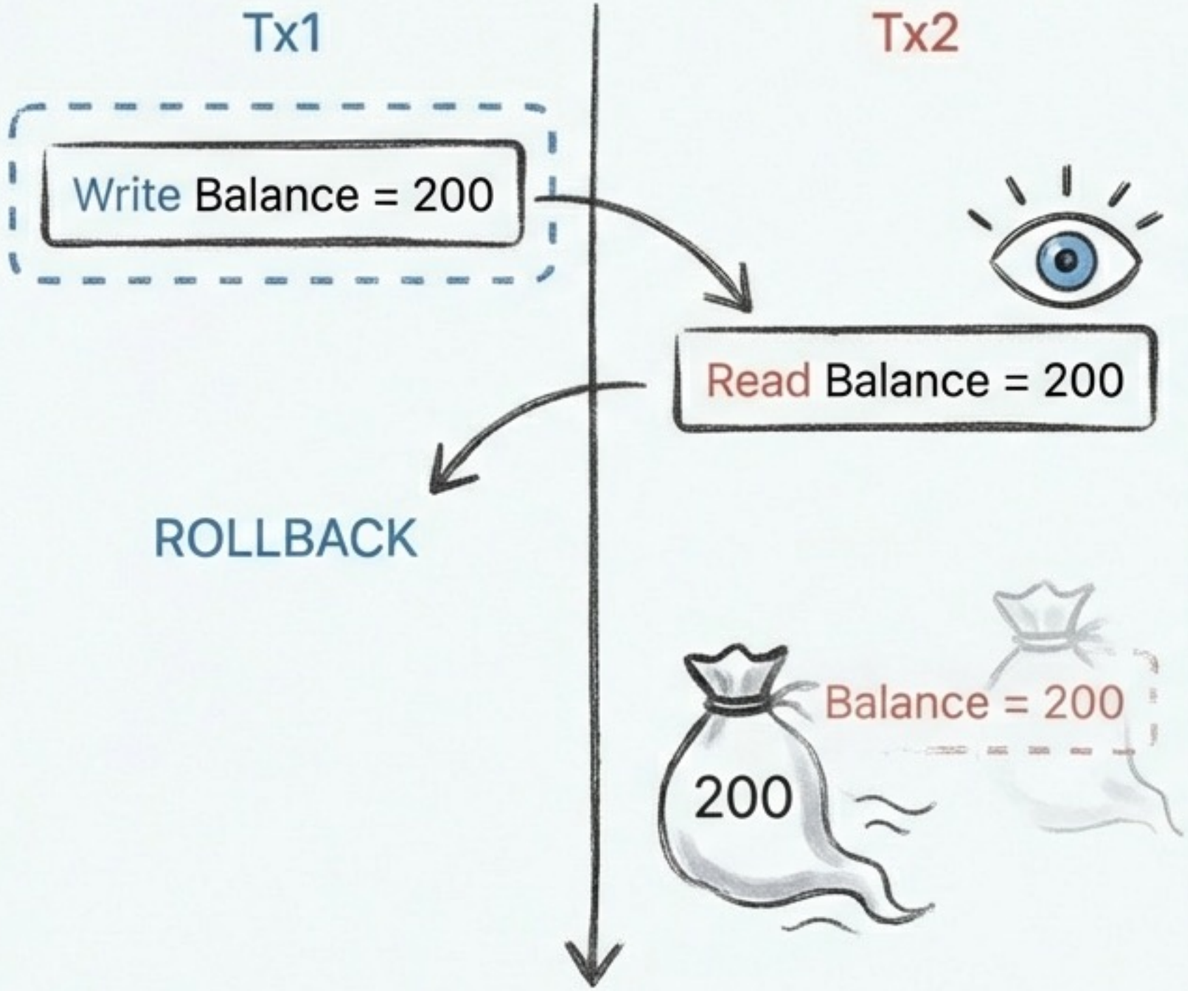
첫 번째 적: 잃어버린 수정 (Lost Update)



두 트랜잭션이 같은 데이터를 읽고 수정할 때, 나중의 수정이 이전 수정을 덮어씁니다. 결과적으로 **데이터 무결성**이 깨집니다.

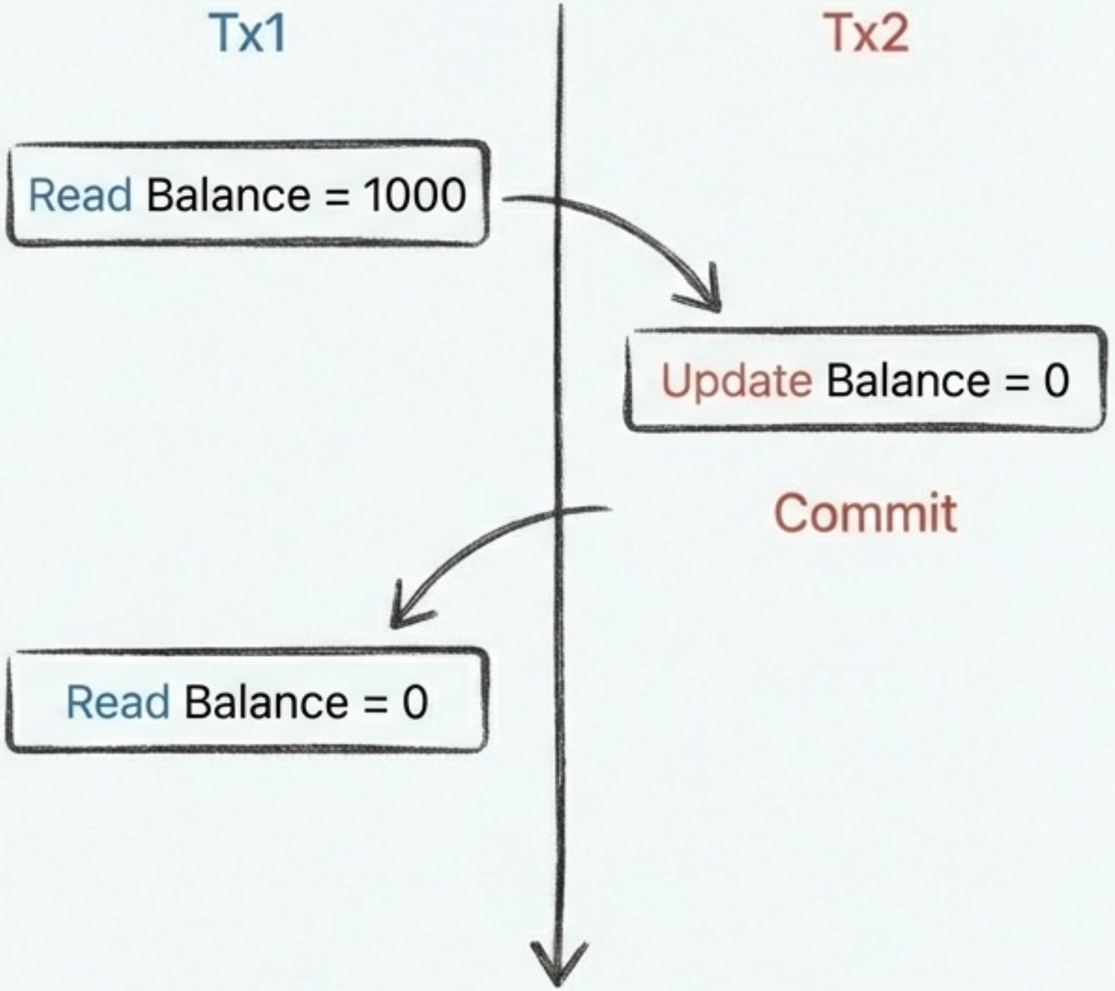
NotebookLM

더티 리드 (Dirty Read)



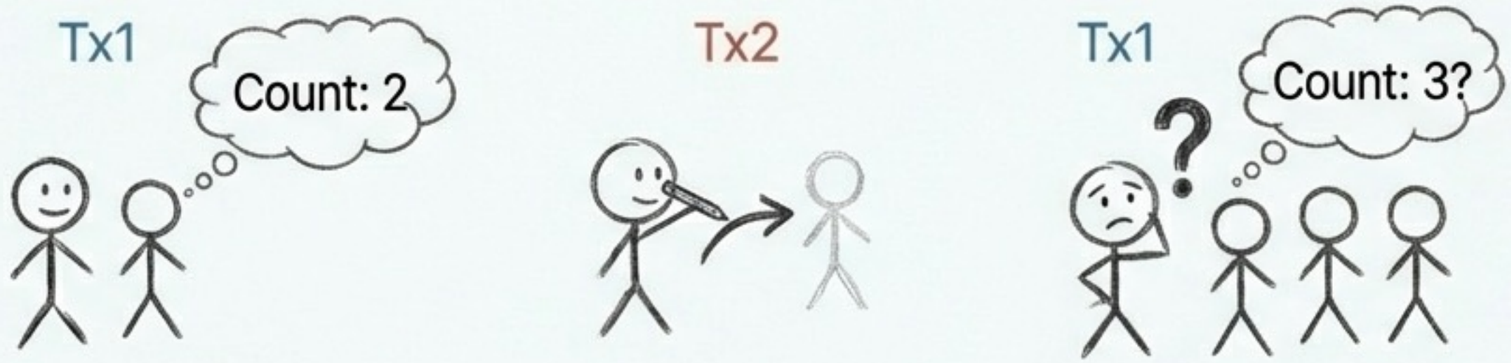
확정되지 않은(Uncommitted) 데이터를 읽음.

반복할 수 없는 읽기 (Non-Repeatable Read)



같은 트랜잭션 내에서 조회 결과가 달라짐.

NotebookLM



팬텀 리드 (Phantom Read): 유령 데이터의 등장

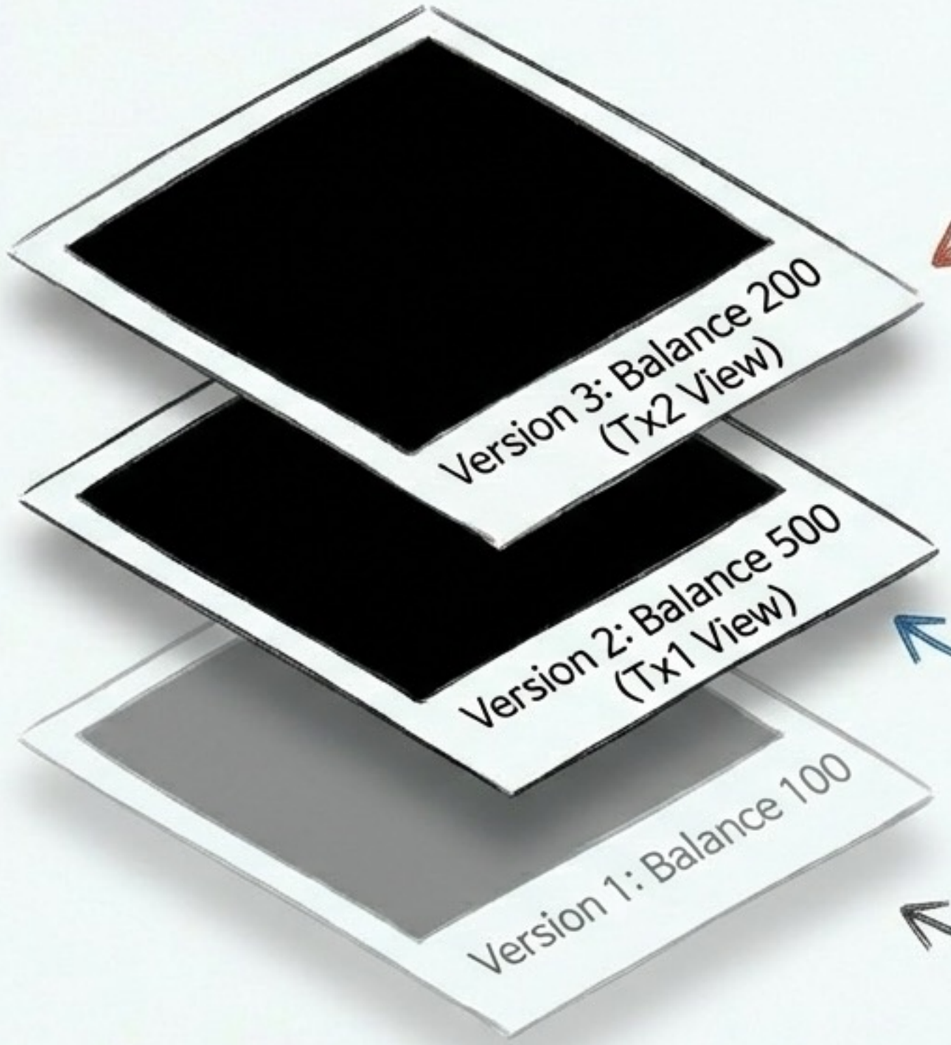
SQL 표준의 4가지 방어벽

Isolation Level	Dirty Read	Non-Repeatable Read	Phantom Read
Read Uncommitted	Yes	Yes	Yes
Read Committed	No	Yes	Yes
Repeatable Read	No	No	Yes
Serializable	No	No	No

아래로 갈수록 더 엄격하지만, 성능 비용이 발생합니다.

NotebookLM

PostgreSQL의 방식: MVCC와 스냅숏 격리



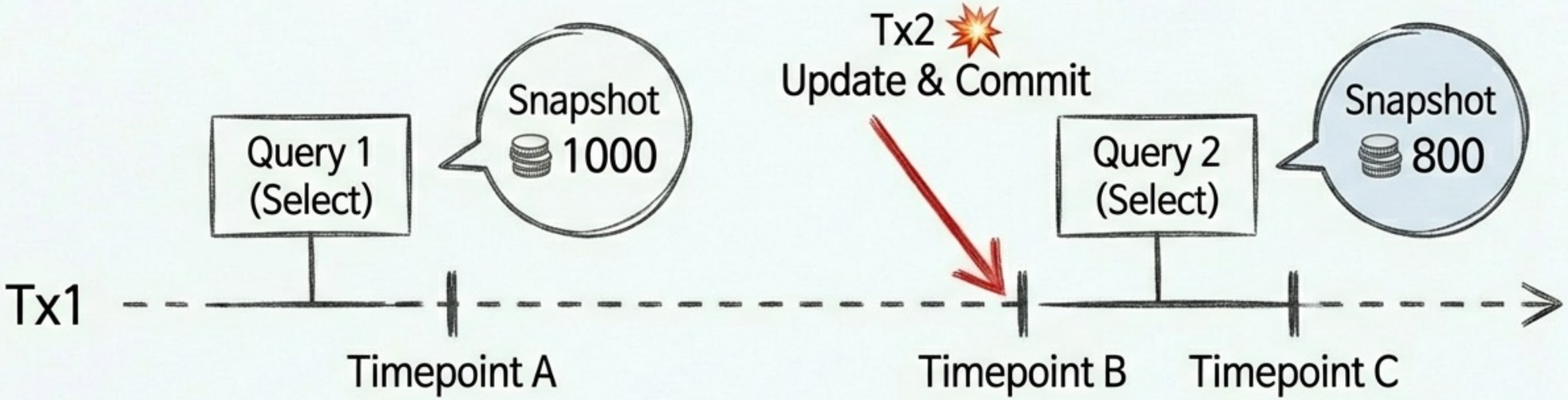
PostgreSQL은 단순 잠금(Locking) 대신 **스냅숏(Snapshot)**을 사용합니다.

MVCC: 읽는 쪽은 쓰는 쪽을 차단하지 않고, 쓰는 쪽도 읽는 쪽을 차단하지 않습니다.

각 트랜잭션은 특정 시점의 데이터 사진을 봅니다.

NotebookLM

기본값(Default): 커밋된 읽기 (Read Committed)




```
IF (잔액 >= 1000) UPDATE...
```

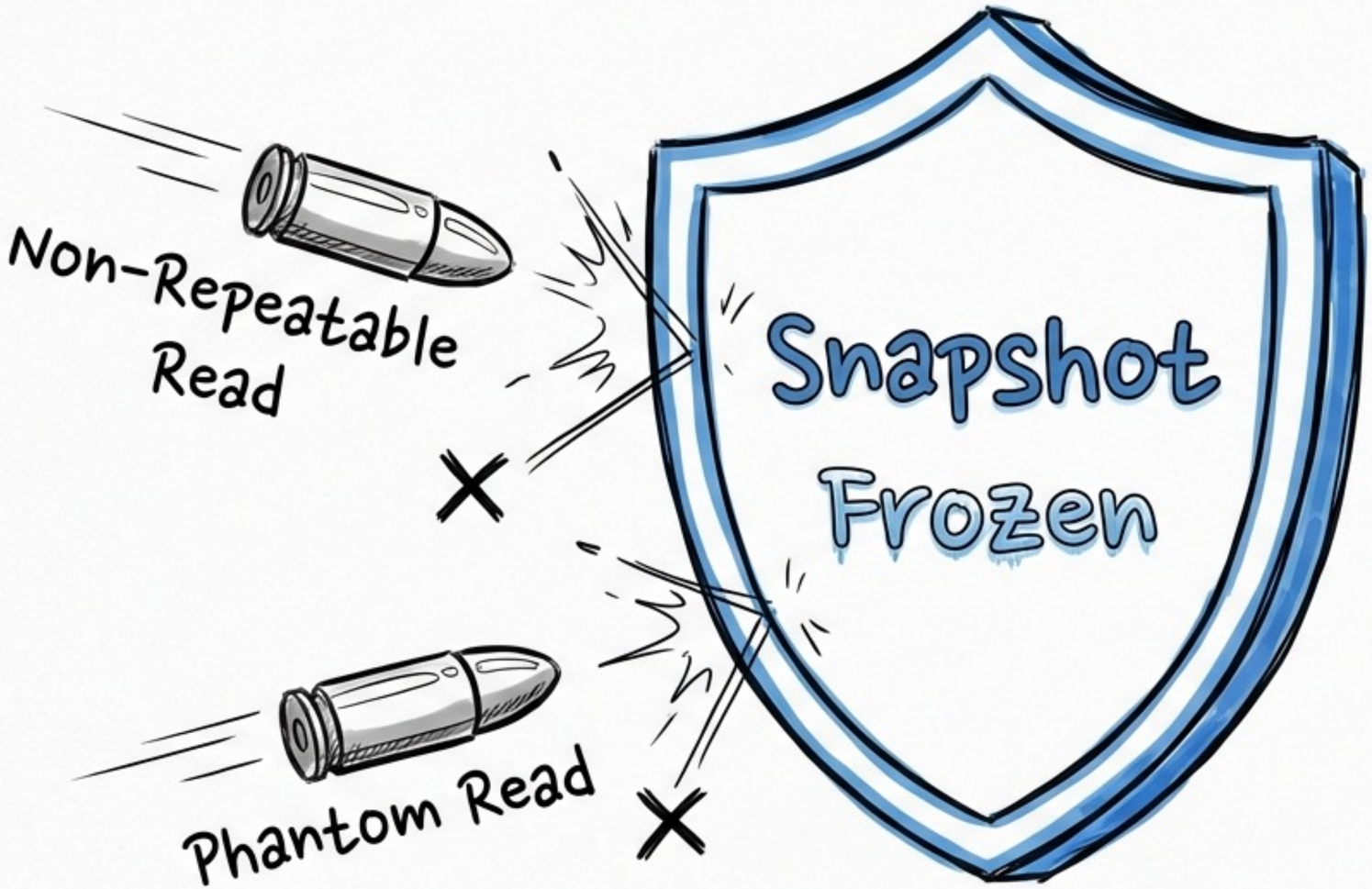
위험! Query 1과 Update 사이에 값이 변할 수 있음 (Anti-pattern).

✓ **보장:** 더티 리드(Dirty Read) 방지

✗ **한계:** 쿼리마다 새로운 스냅숏을 생성하므로, 쿼리 사이에 값이 변할 수 있음.

NotebookLM

PostgreSQL의 강력함: 반복 읽기 (Repeatable Read)

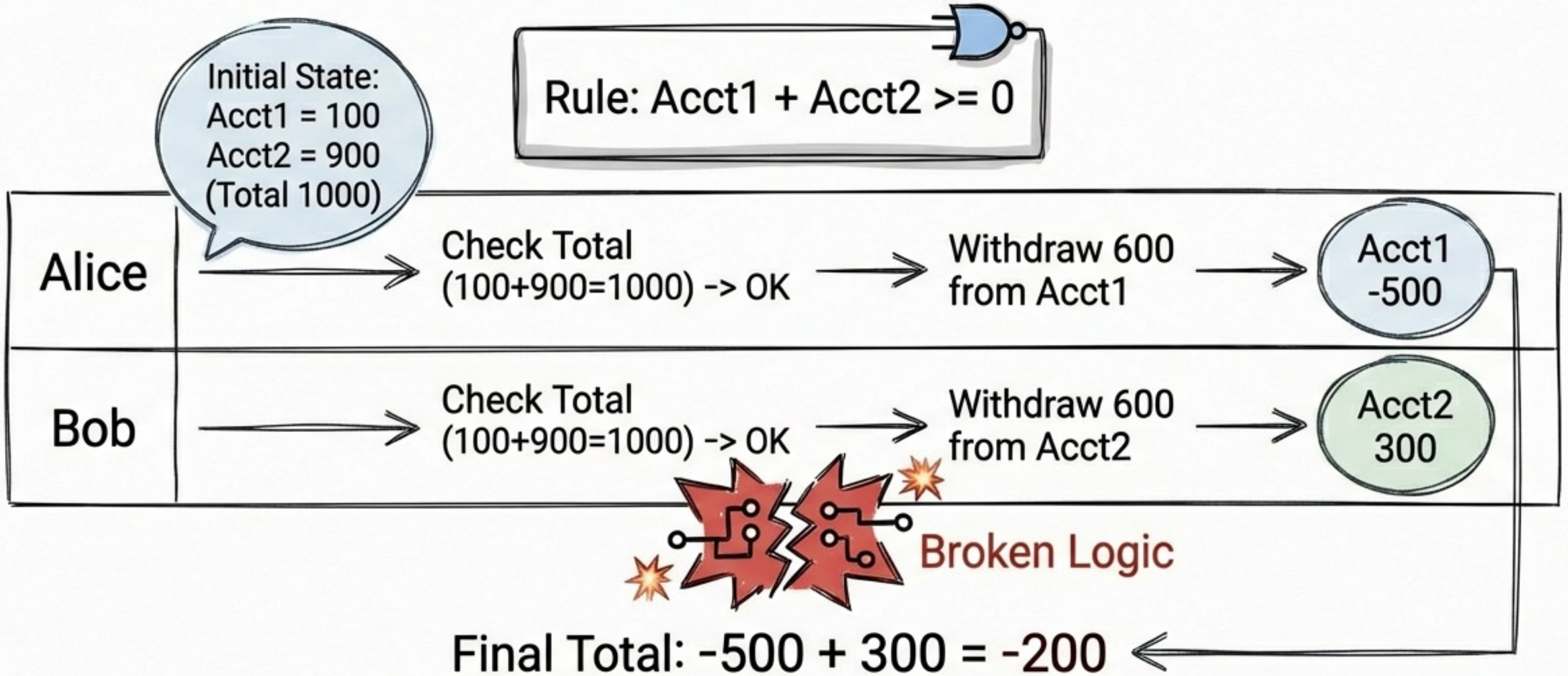


Phantom Read Allowed?	
SQL Standard	YES
PostgreSQL	NO

- 표준보다 더 엄격합니다. 팬텀 리드(Phantom Read)도 방지합니다.
- 트랜잭션 시작 시점의 스냅숏을 끝까지 유지합니다.

NotebookLM

숨겨진 함정: 쓰기 왜곡 (Write Skew)



반복 읽기(Repeatable Read)에서도 발생합니다.
데이터는 일관적(Snapshot)이지만, 비즈니스 로직이 깨집니다.

NotebookLM

최후의 수단: 직렬화 가능 (Serializable)

JetBrains Mono

```
ERROR: could not serialize access due to concurrent update
HINT:  The transaction might succeed if retried.
```

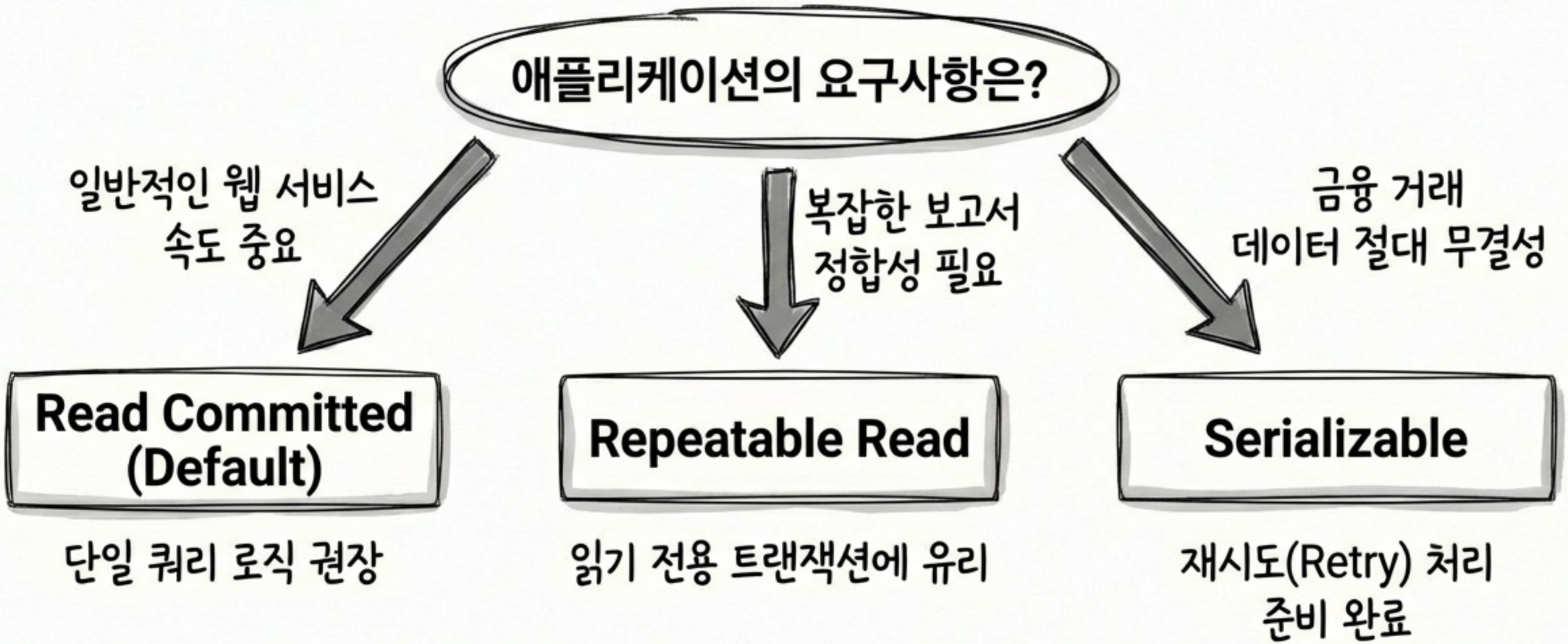


- 가장 엄격한 격리 수준. 모든 이상 현상(쓰기 왜곡 포함) 방지.
- 논리적으로 트랜잭션을 순차 실행하는 것과 동일한 결과 보장.

- 비용:: 충돌 시 트랜잭션이 중단됩니다. 재시도(Retry) 로직 필수.

NotebookLM

어떤 격리 수준을 선택해야 할까요?



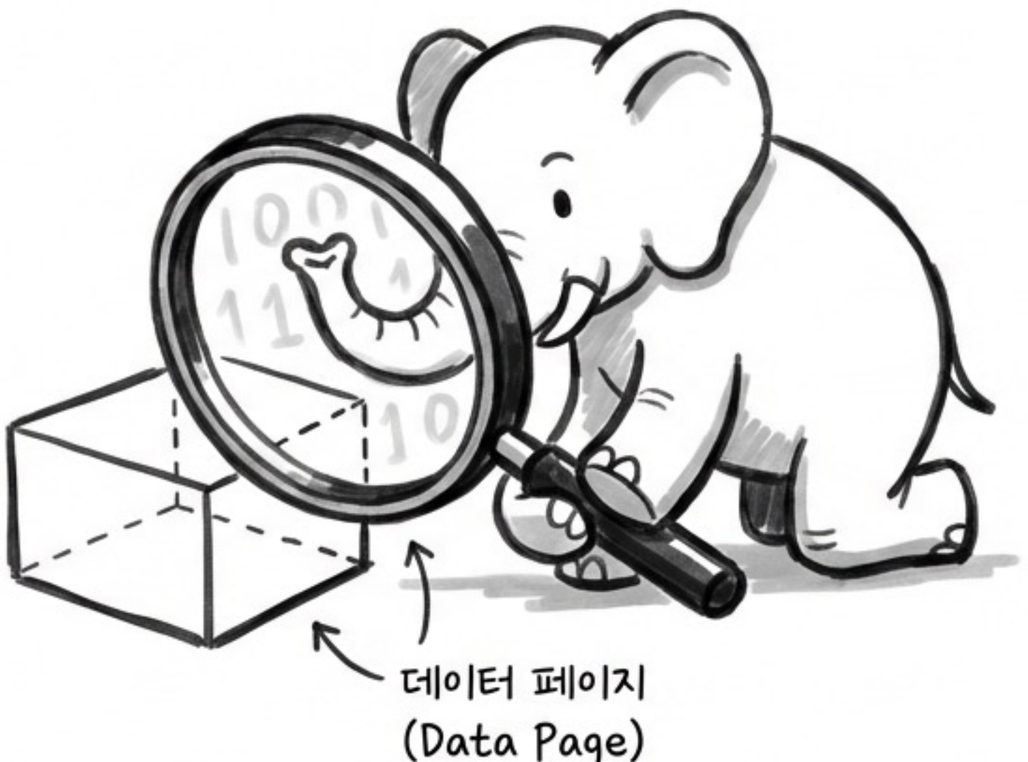
대부분은 Read Committed로 충분하지만,
비즈니스 로직에 따라 상위 레벨을 선택하세요.

NotebookLM

3. Pages and Tuples

PostgreSQL 내부 구조: 페이지와 튜플

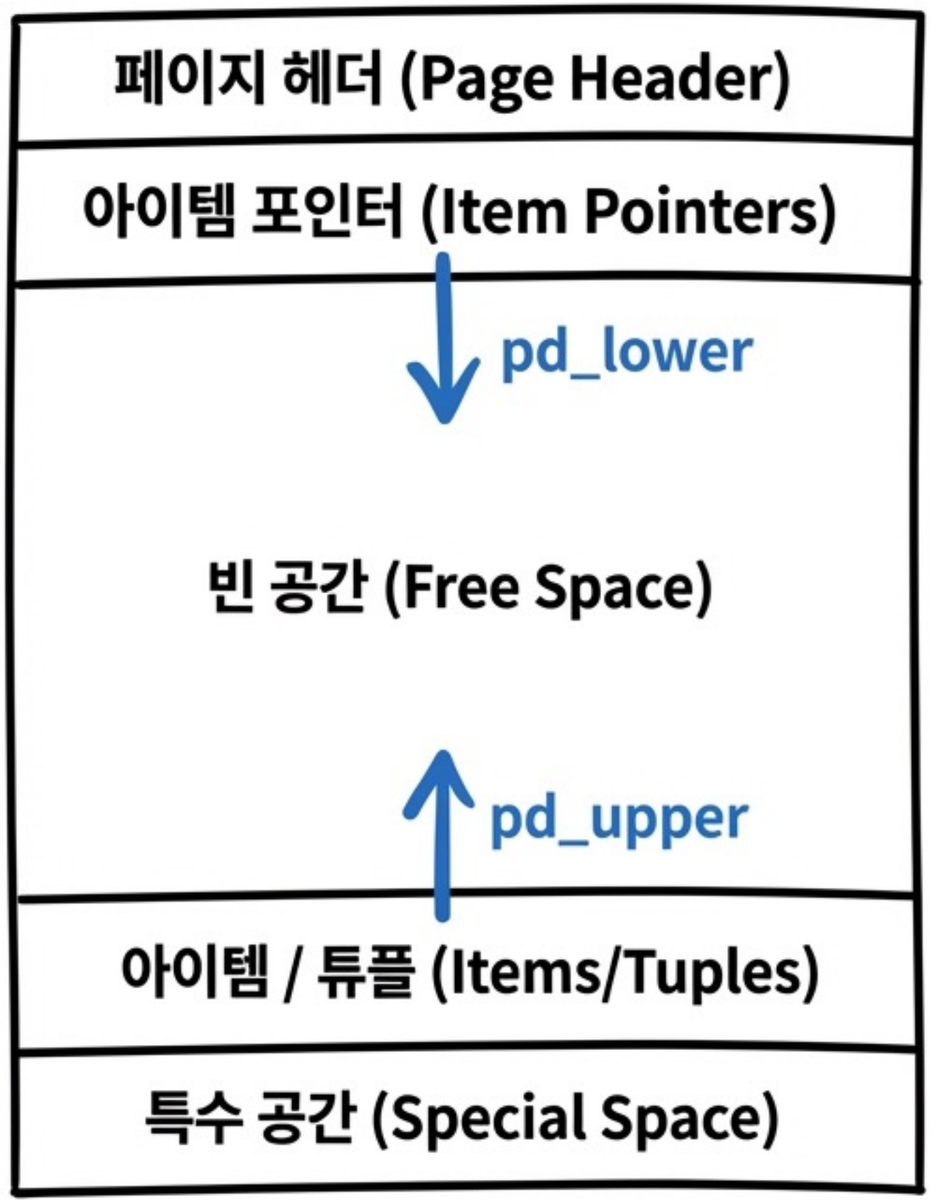
SQL 쿼리 너머의 물리적 데이터 저장 구조



3장: 페이지와 튜플

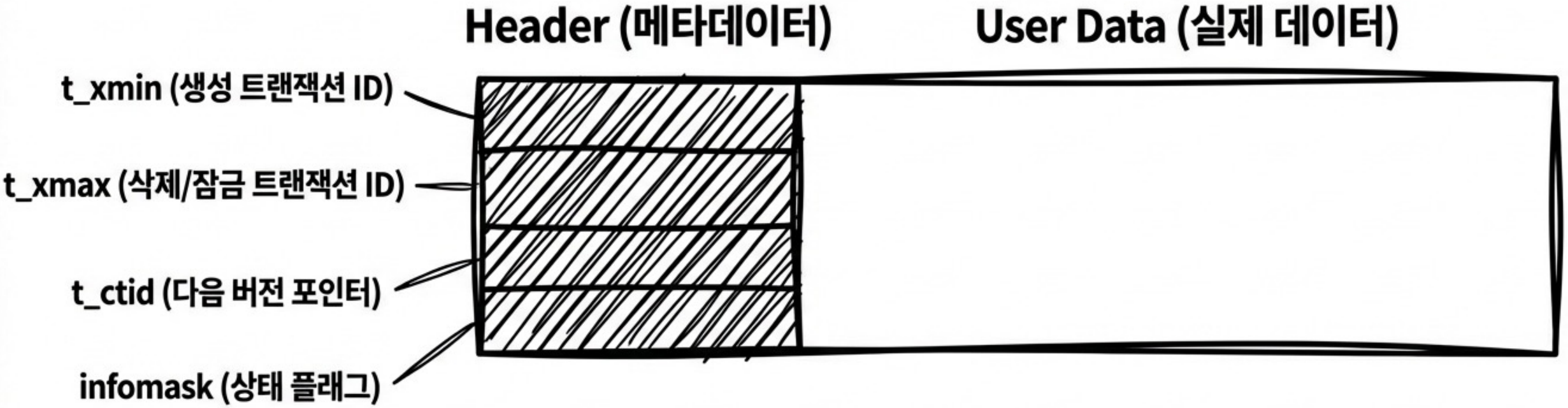
우리는 오늘 '구문'이 아닌 '메커니즘'을 살펴봅니다.

NotebookLM



- 8KB의 세상: 모든 행(Row)은 이 페이지 안에 존재합니다.
- 효율적 공간 관리: 포인터와 데이터가 양 끝끝에서 자라나며 단편화를 방지합니다.

NotebookLM



오버헤드 (Overhead): 튜플당 최소 23바이트가 필요합니다.
데이터보다 메타데이터가 더 클 수도 있습니다.

의미: 튜플은 단순한 데이터가 아니라, 버전 관리(MVCC)를 위한 독립된 객체입니다.

NotebookLM

INSERT INTO t(s) VALUES ('FOO');

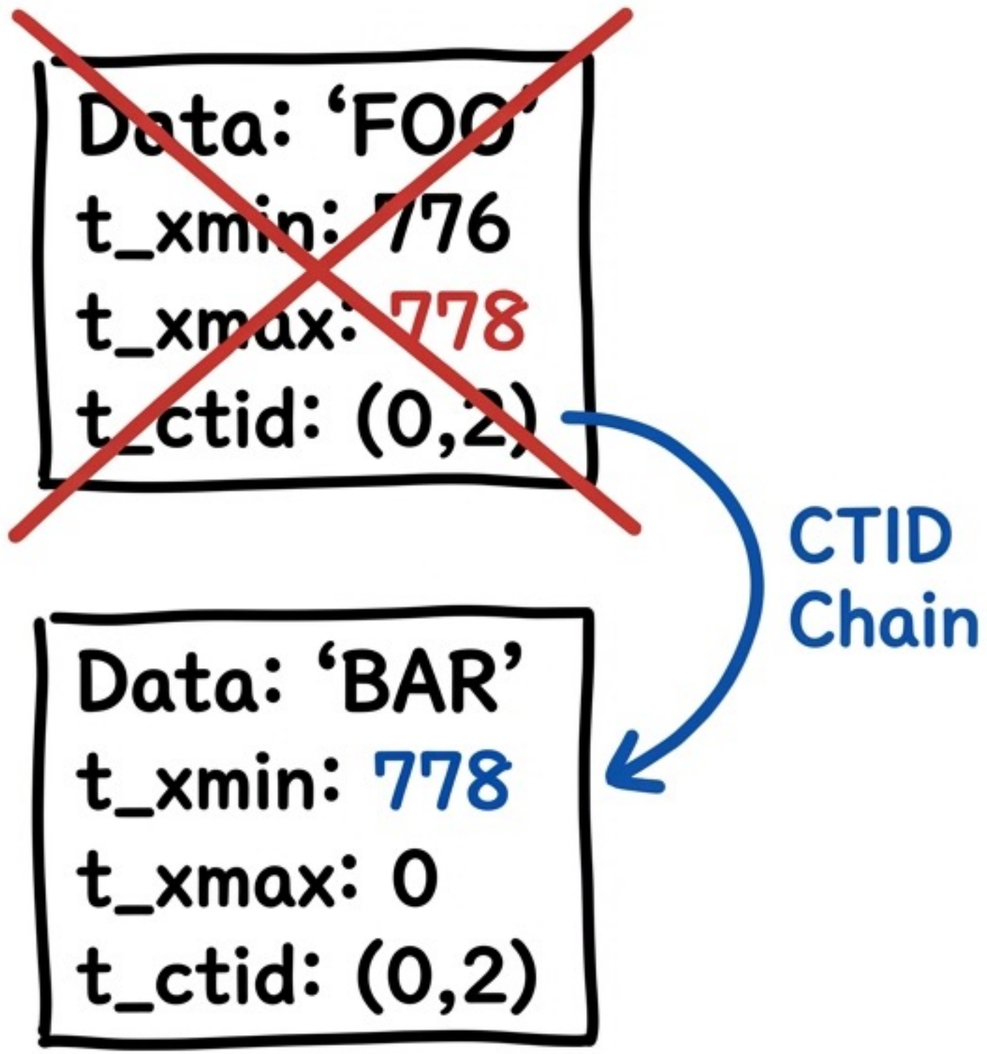


Data: 'FOO'
t_xmin: 776
t_xmax: 0
t_ctid: (0,1)

- 탄생 (Birth): 트랜잭션 776이 이 행을 생성했습니다.
- 격리성: 다른 트랜잭션에게는 아직 이 행이 보이지 않습니다.
- xmax = 0: 현재 살아있는 최신 버전입니다.

NotebookLM

UPDATE t SET s = 'BAR'; (TxID: 778)

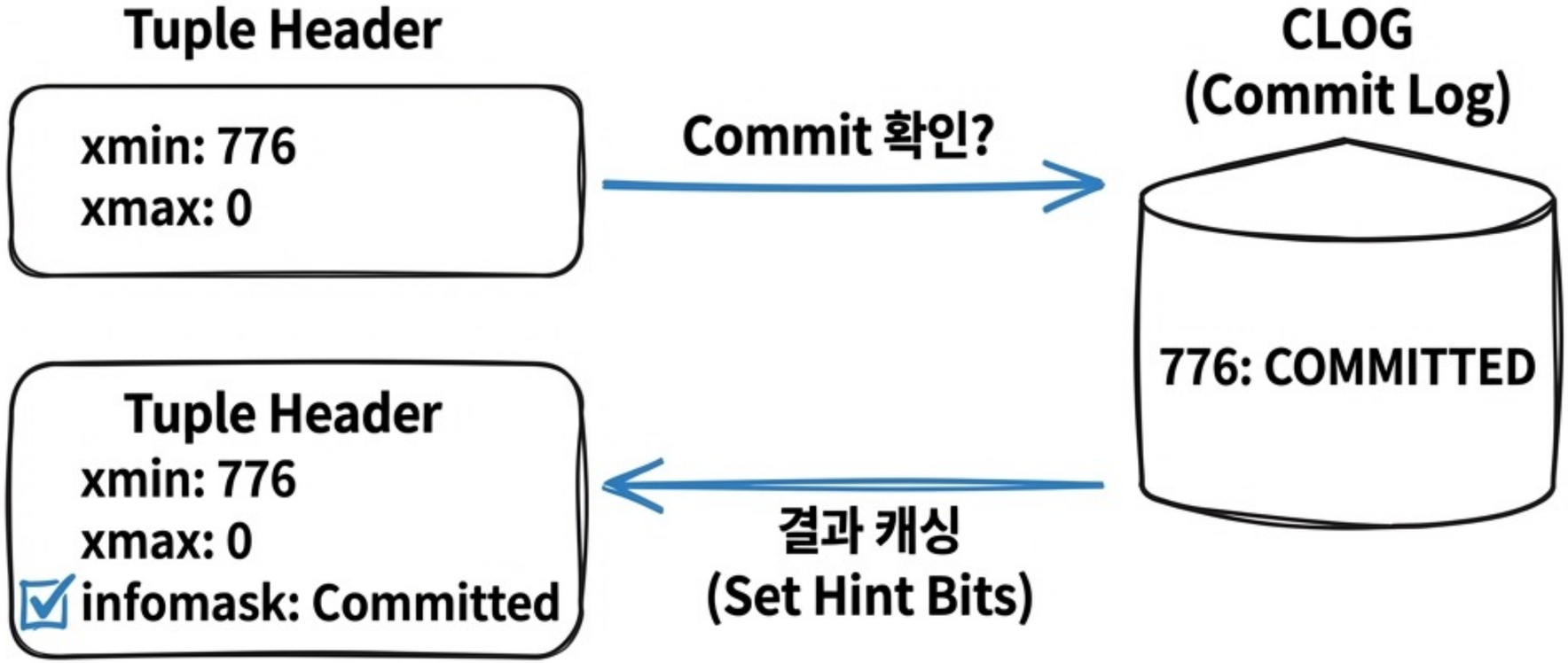


환상 (The Illusion): 데이터는 제자리에서 수정되지 않습니다.

Update = Delete + Insert

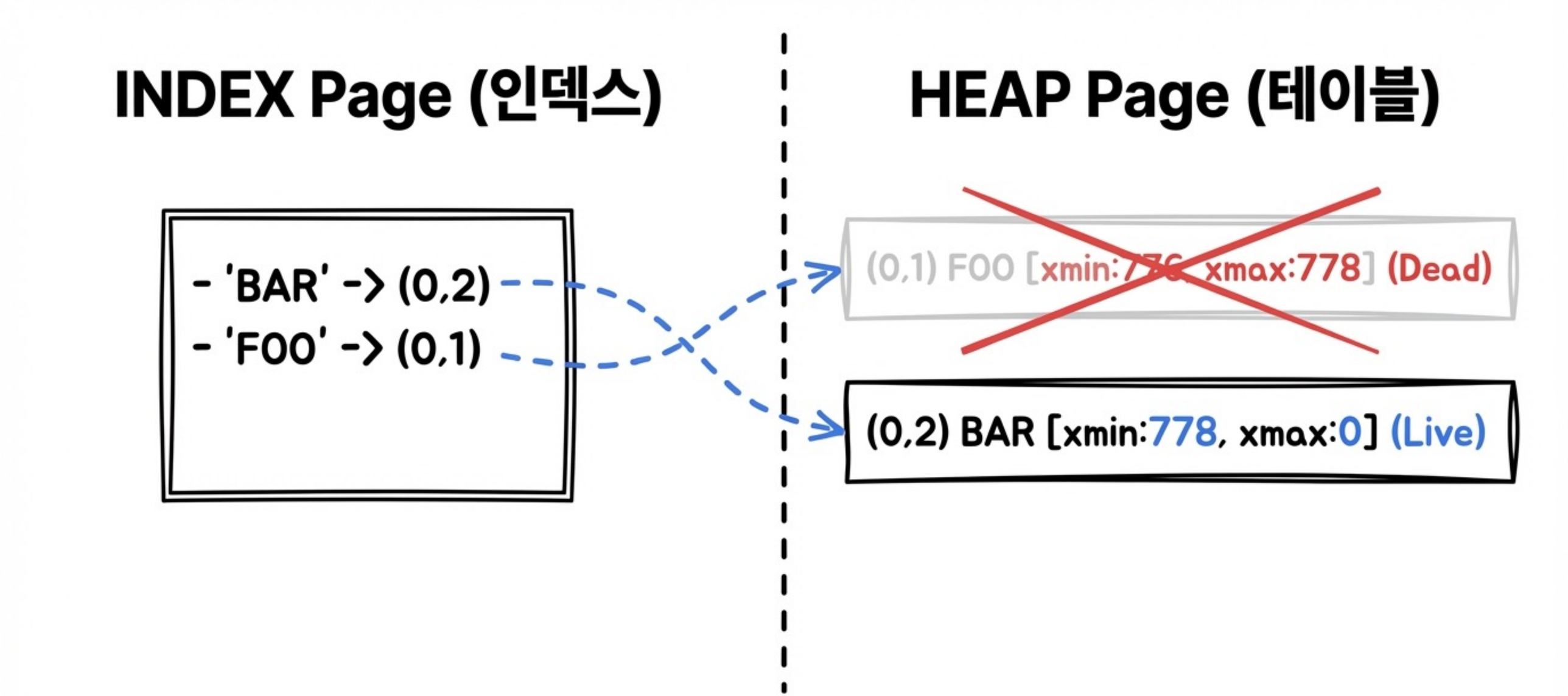
1. 이전 버전(FOO)을 트랜잭션 778이 '삭제' 표시.
2. 새로운 버전(BAR)을 트랜잭션 778이 '생성'.

NotebookLM



- 비용 절감: 매번 CLOG를 확인하는 것은 비쌉니다.
- 힌트 비트: 한 번 확인된 상태는 튜플에 '캐시'되어, 다음 읽기 작업의 속도를 높입니다.
- 읽기 작업(SELECT)도 데이터를 '수정'할 수 있습니다 (힌트 비트 설정 때문).

NotebookLM



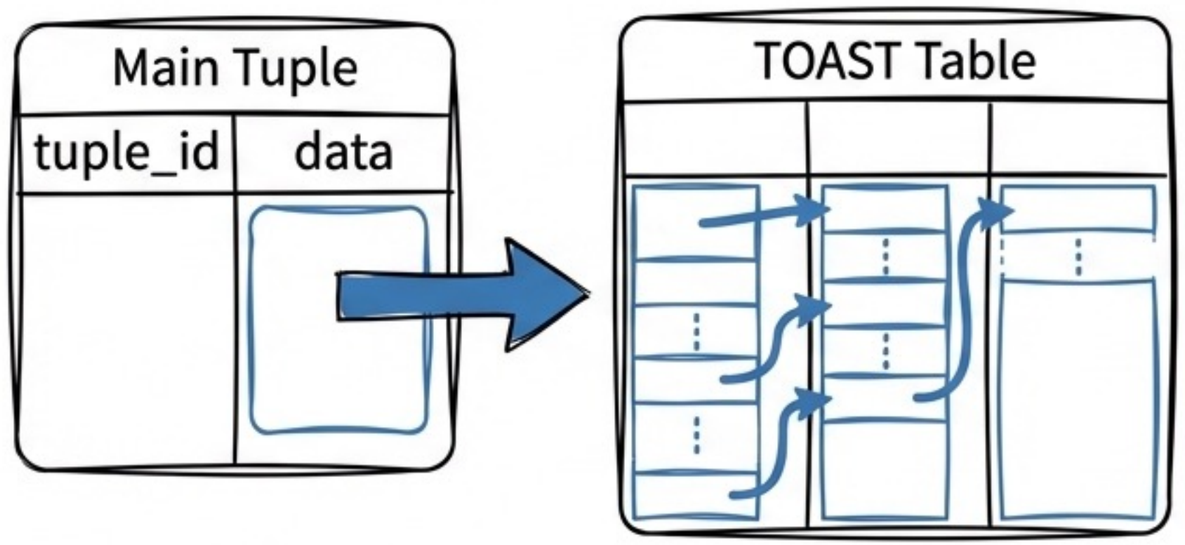
인덱스는 단순합니다: MVCC 정보가 없습니다. 위치만 안내합니다.

힙(Heap)이 가시성을 판단합니다: '이 버전이 당신에게 보이는가?'

NotebookLM

Outliers (예외 처리)

TOAST (The Oversized Attribute Storage Technique)



페이지 크기(8KB)를 넘는 데이터는
별도의 공간으로 잘라서 저장됩니다.

Virtual XID (가상 트랜잭션)

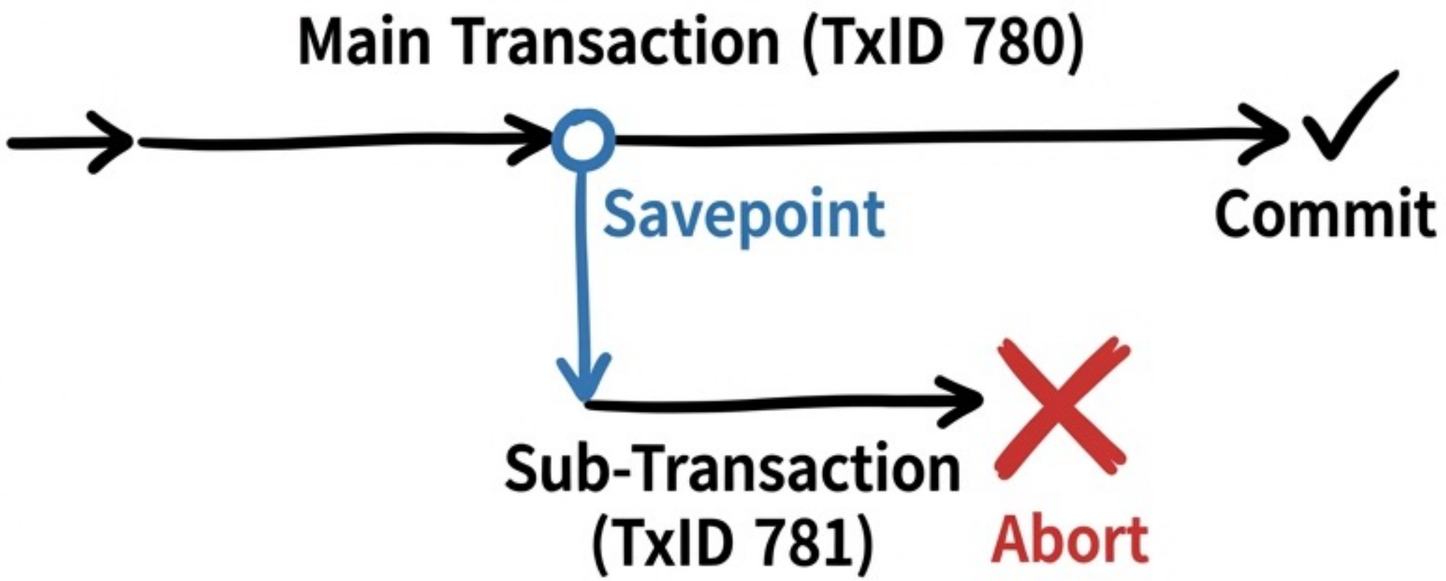


읽기 전용(Read-Only) 트랜잭션은
영구 ID를 소모하지 않습니다.

PostgreSQL은 다양한 워크로드(대용량 데이터, 빈번한 읽기)에 적응합니다.

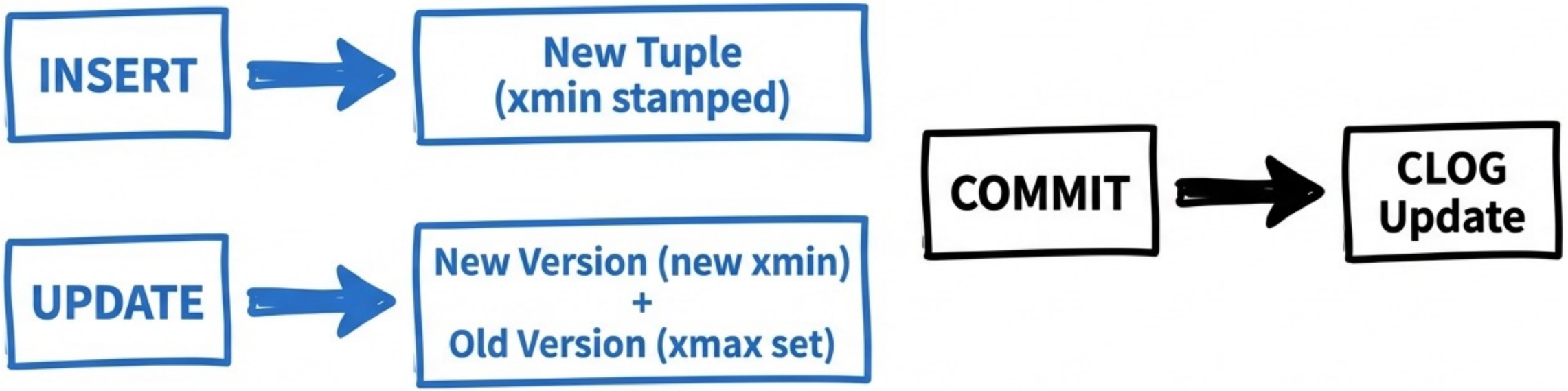
NotebookLM

Savepoints & Rollback (저장점과 롤백)



- 저장점 (Savepoints): 트랜잭션 내부의 하위 트랜잭션.
- 롤백의 진실: 데이터가 지워지는 것이 아닙니다.
- CLOG에 '중단됨(Aborted)'으로 표시될 뿐, 데이터는 VACUUM 전까지 남습니다.

NotebookLM



1. **Append-Only 모델:** 덮어쓰지 않고, 새로운 역사를 씁니다.

2. **MVCC의 핵심:** xmin과 xmax를 통한 가시성 제어.

3. **성능의 열쇠:** 이 구조를 이해해야 VACUUM과 튜닝을 이해할 수 있습니다.

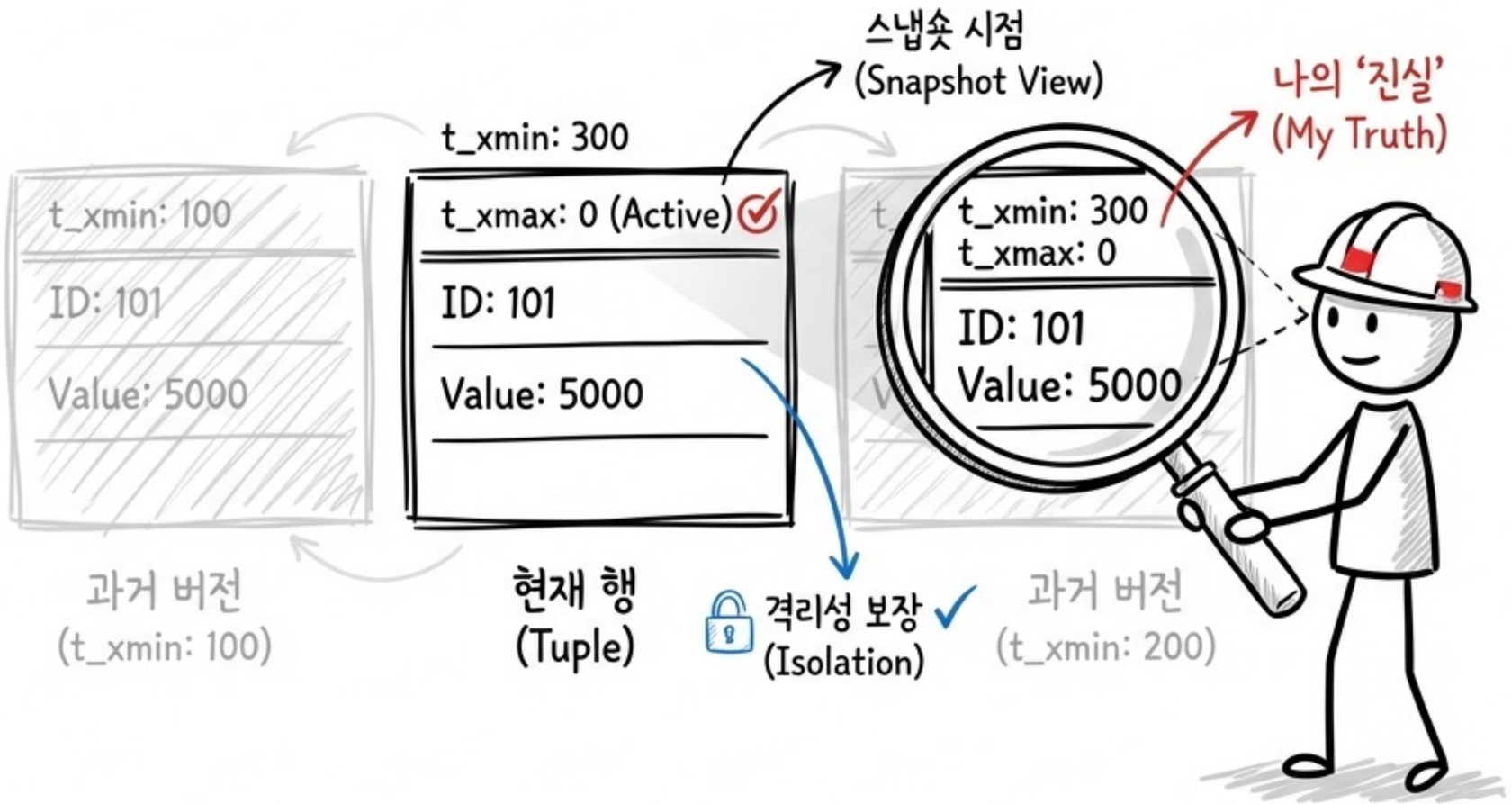
"데이터베이스는 블랙박스가 아닙니다. 살아있는 유기적 시스템입니다."

NotebookLM

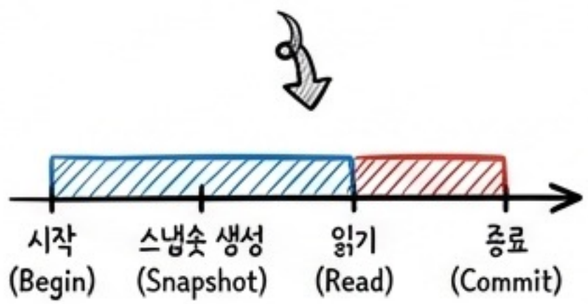
•4. Snapshots

PostgreSQL의 시간 여행: 스냅숏의 비밀

데이터의 일관성과 격리성 보장 원리



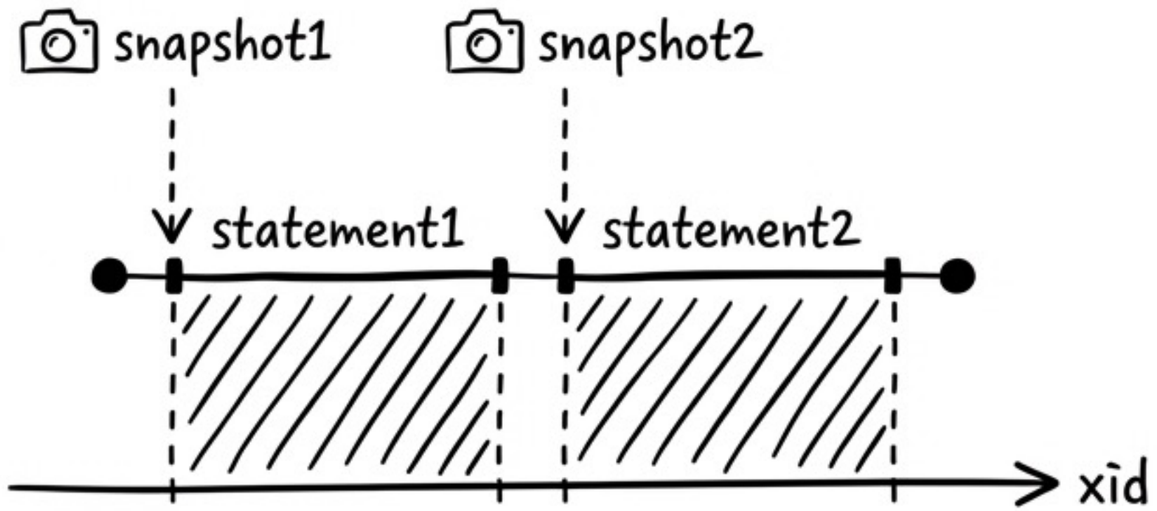
- 데이터는 정적이지 않습니다.
- 동일한 행(Row)에도 여러 버전이 존재합니다.
- PostgreSQL은 어떻게 당신에게 맞는 '진실'을 보여줄까요?



NotebookLM

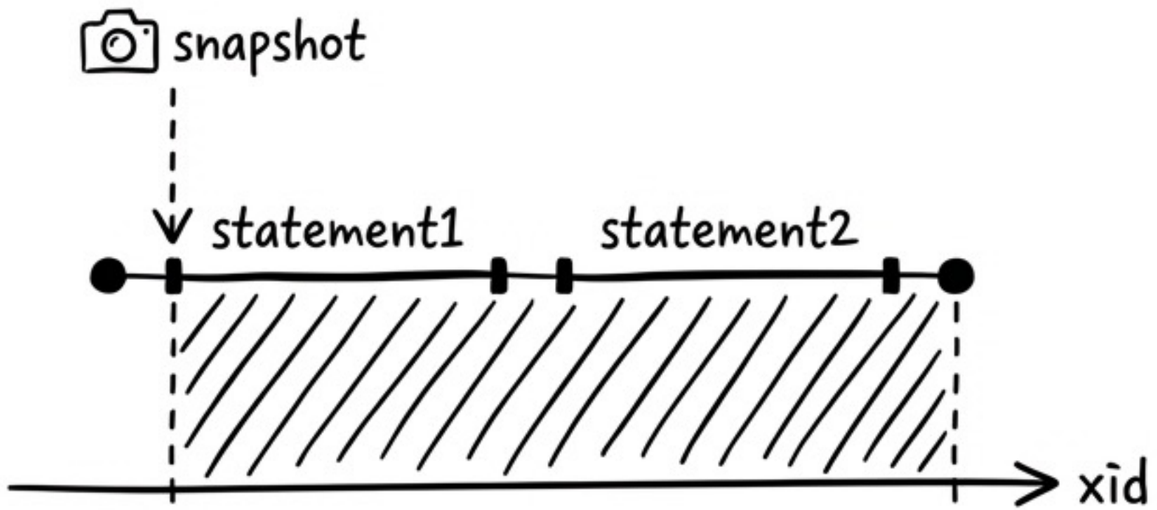
스냅숏의 촬영 시점: 격리 수준

Scenario A: 커밋된 읽기 (Read Committed)



문장이 시작될 때마다 촬영

Scenario B: 반복 읽기(Repeatable Read) / 직렬화(Serializable)

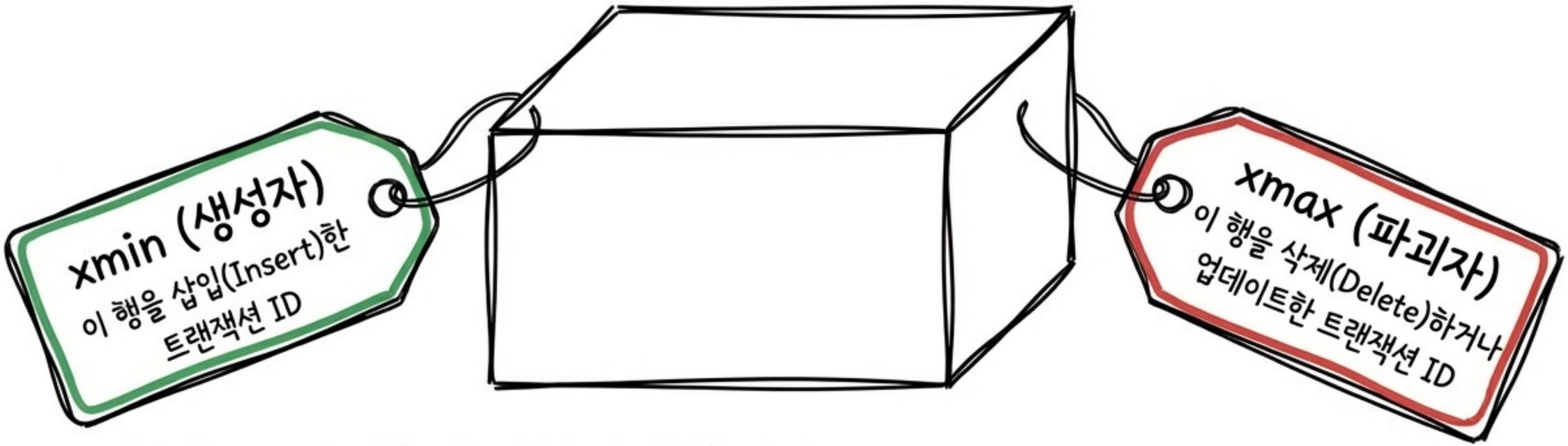


트랜잭션 시작 시 1회 촬영

격리 수준은 스냅숏의 생명주기를 결정합니다.

NotebookLM

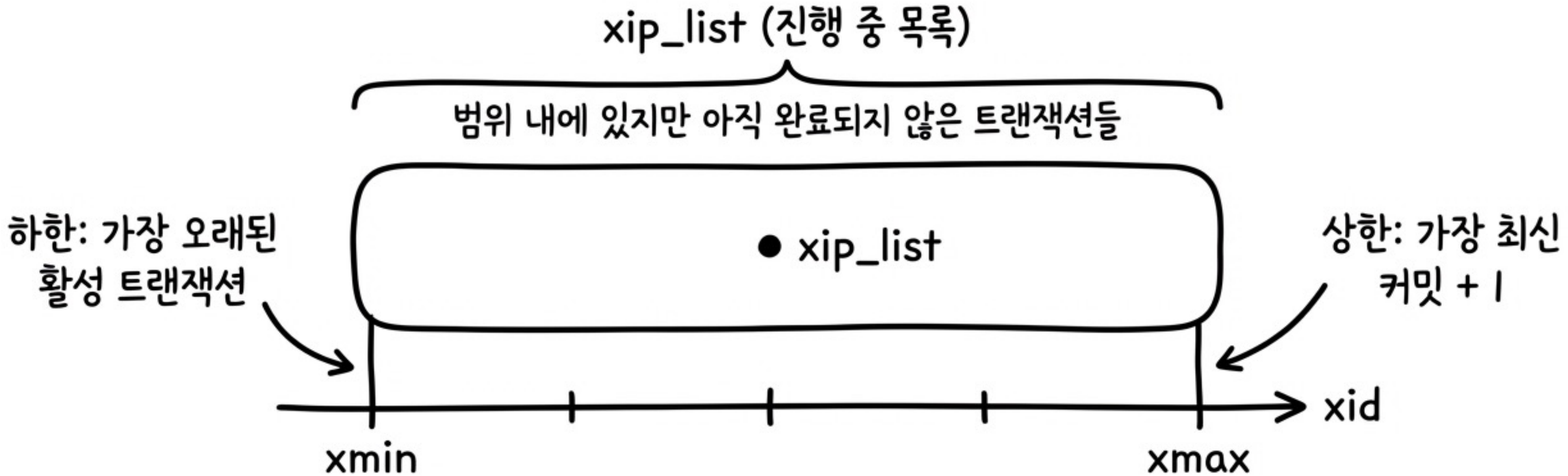
튜플의 생명주기 태그: xmin과 xmax



- 스냅숏은 데이터를 복사하지 않습니다.
- 이 숫자들을 보고 가시성(Visibility)을 판단합니다.
- 규칙: xmin과 xmax 간격이 겹치지 않으므로, 각 행은 스냅숏에서 한 버전으로만 나타납니다.

NotebookLM

스냅숏의 해부학

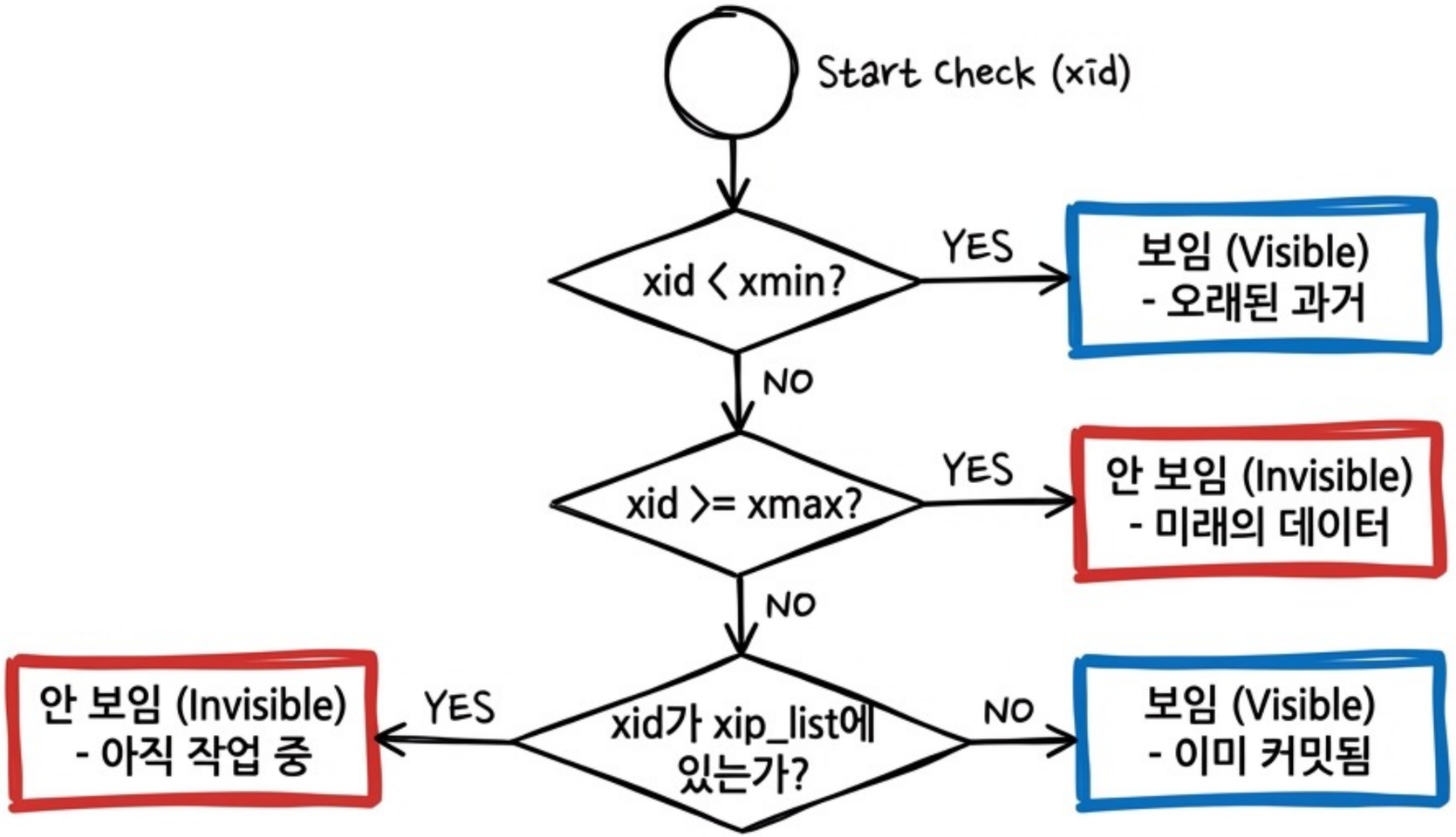


스냅숏은 xmin부터 xmax 사이의 직사각형 필터입니다.

시스템은 트랜잭션의 커밋/완료 시간을 별도로 기록하지 않습니다.
오직 현재 상태(활성/완료)만 알 수 있습니다.

NotebookLM

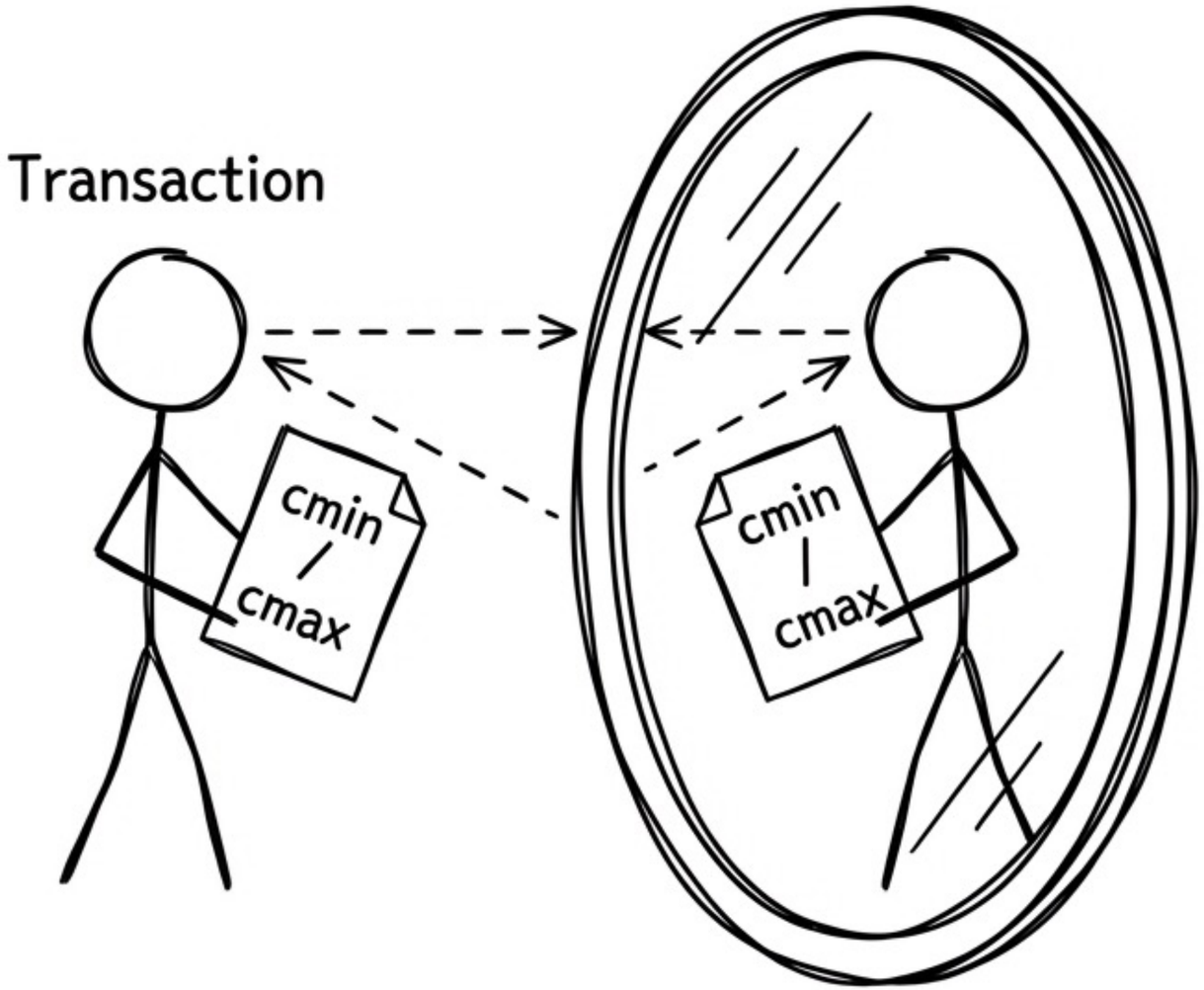
가시성 판단 알고리즘



이 규칙을 통해 과거 시점의 일관된 데이터를 복원합니다.

NotebookLM

예외: 내 변경 사항은 보인다



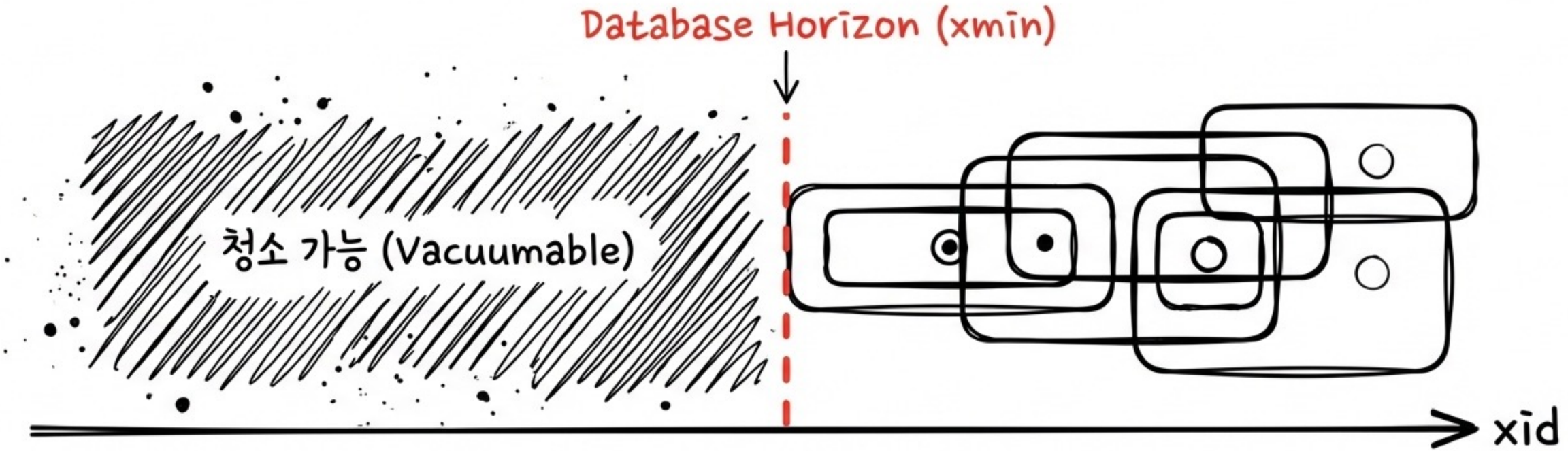
트랜잭션은 자신이 만든 커밋되지 않은 변경 사항을 볼 수 있습니다.

명령 ID (Command ID: cmin/cmax): 트랜잭션 내부의 작업 순서 번호.

예: 커서(Cursor)를 연 시점 이후에 자신이 추가한 행은 커서에 보이지 않아야 합니다.

NotebookLM

데이터베이스의 지평선과 베큠



- 가장 오래된 xmin이 데이터베이스의 지평선(Horizon)을 결정합니다.
- 문제점: 장기 실행 트랜잭션이 있으면 지평선이 이동하지 않습니다.
- 결과: 오래된 튜플을 베큠(Vacuum)이 정리할 수 없어 테이블이 비대해집니다(Bloat).

NotebookLM

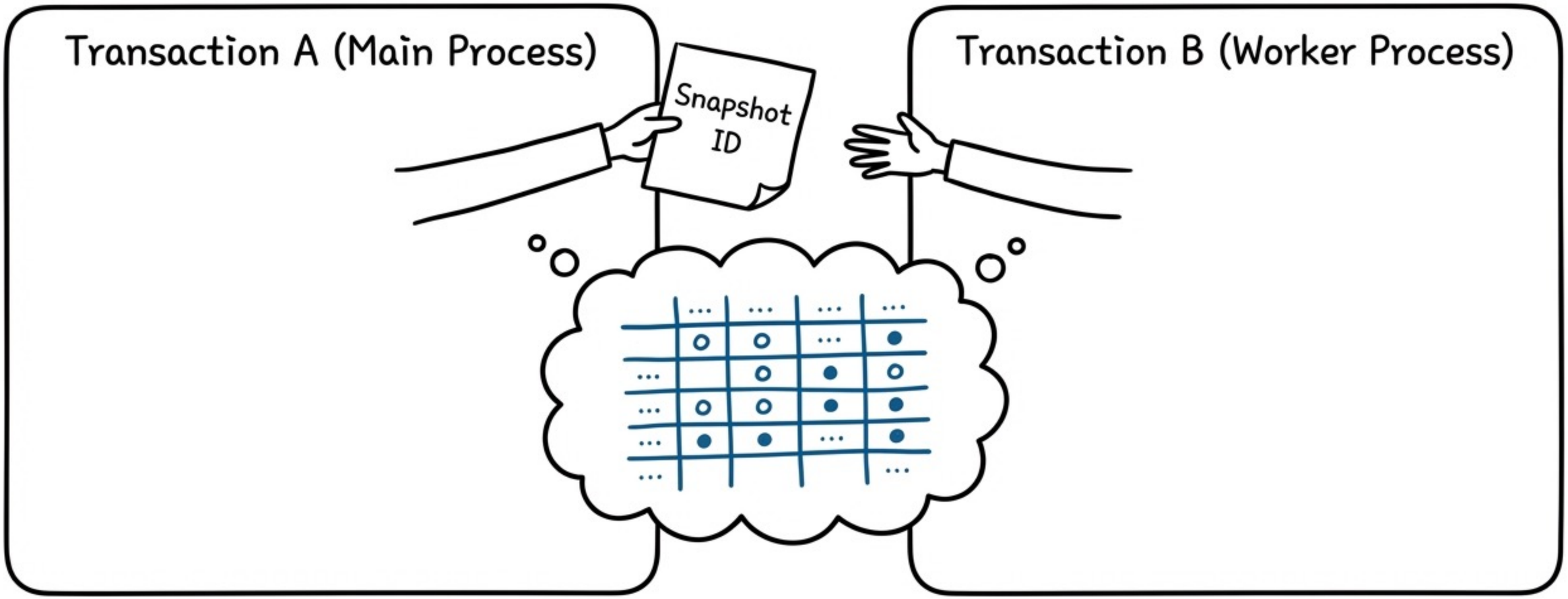
시스템 카탈로그의 특수성



- 시스템 카탈로그(메타데이터)는 일반적인 스냅숏 규칙을 따르지 않습니다.
- 이유 : 항상 최신 상태(Most Recent)를 봐야 무결성 제약 조건이나 테이블 정의 변경을 놓치지 않습니다.
- 예 : ALTER TABLE 후 INSERT 시, 트랜잭션은 즉시 변경된 제약 조건을 인식해야 합니다.

NotebookLM

스냅숏 공유하기



- 함수: pg_export_snapshot() → ID 반환
- 적용: SET TRANSACTION SNAPSHOT 'ID'
- 용도: pg_dump의 병렬 백업. 여러 프로세스가 완벽하게 동일한 시점의 데이터를 백업할 수 있게 해줍니다.

NotebookLM

핵심 요약 (Key Takeaways)

1. 필터(Filter), 복사 아님

스냅숏은 데이터 복사가 아니라,
xmin/xmax를 거르는 필터입니다.

2. 타이밍(Timing)

격리 수준(Isolation Level)에
따라 필터를 만드는 시점이
달라집니다.

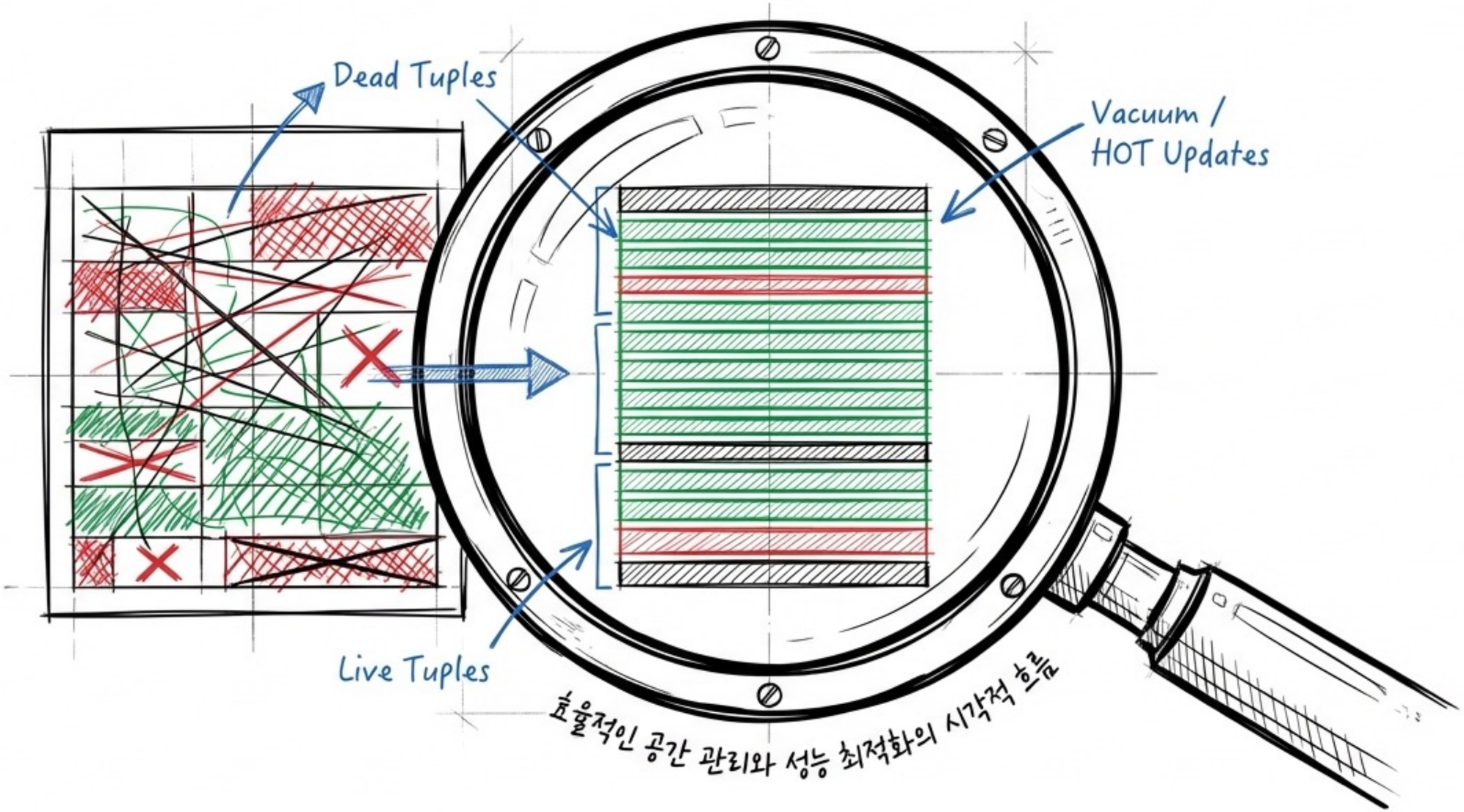
3. 책임(Responsibility)

오래된 스냅숏을 붙들고 있으면
베큠(Vacuum)을 방해하여
시스템 성능을 저하시킵니다.

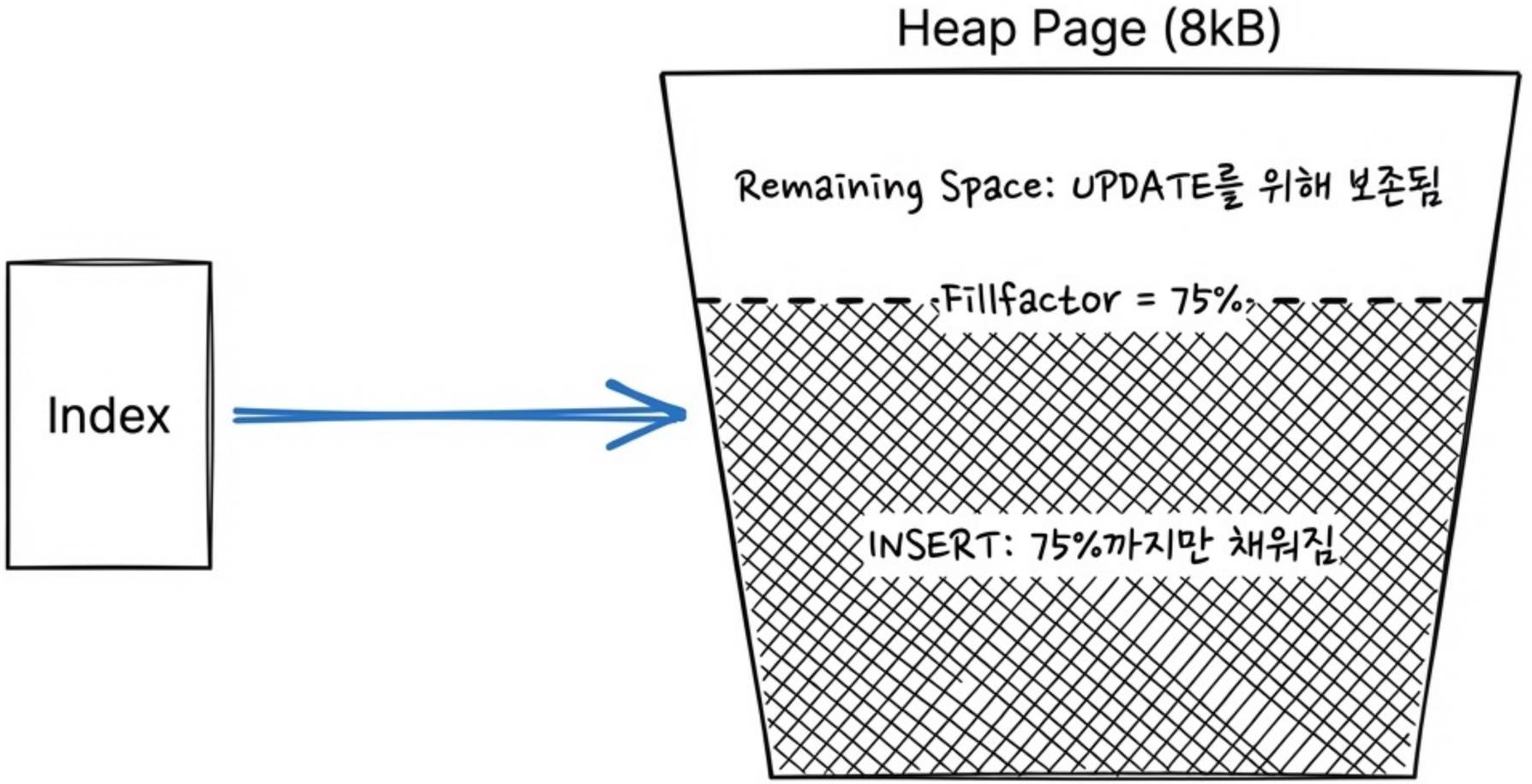
NotebookLM

- **5. Page Pruning and HOT Updates**

PostgreSQL: 페이지 정리와 HOT 수정



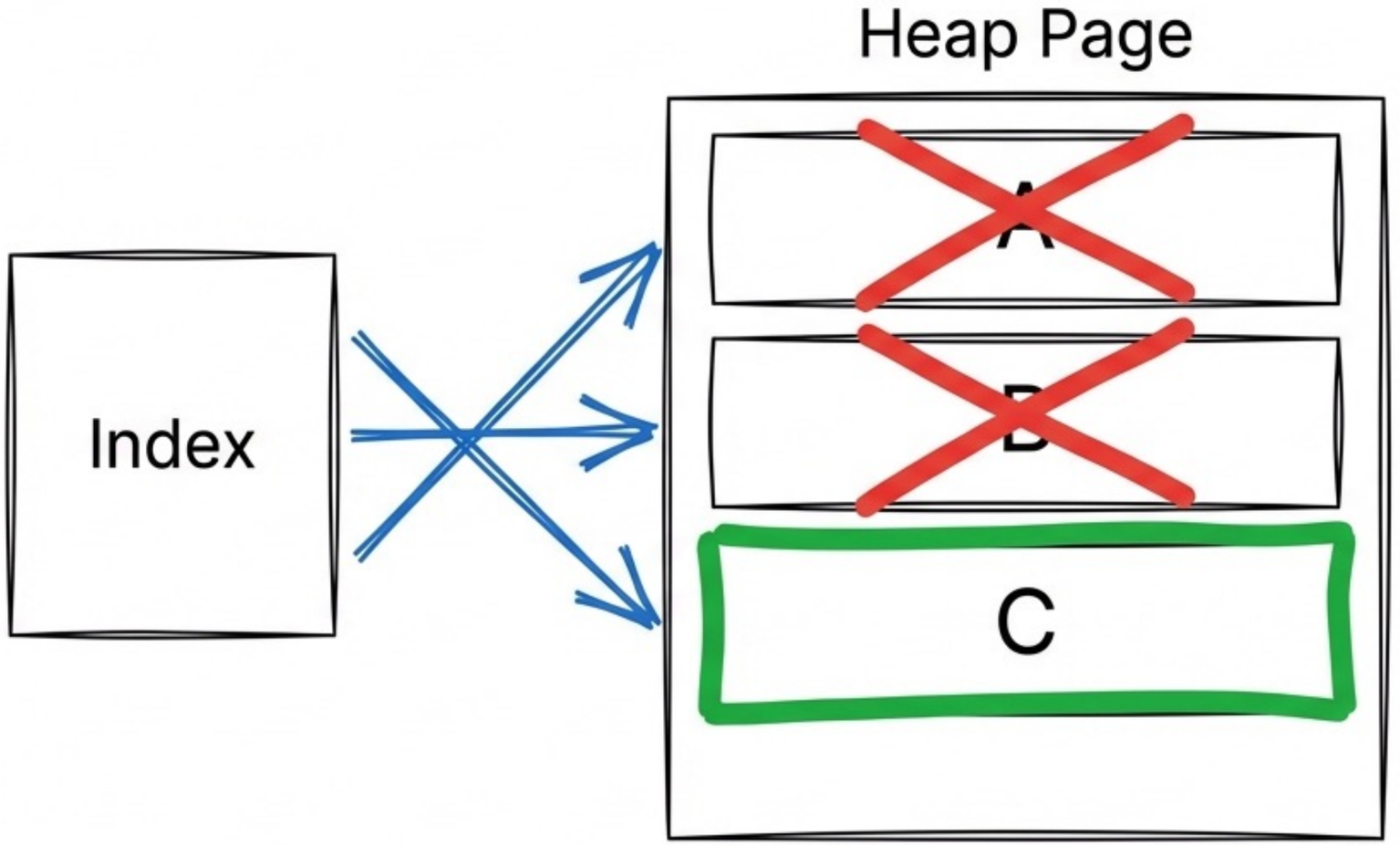
NotebookLM



공간이 예약되지 않으면 UPDATE 시 새로운 페이지가 필요합니다.

NotebookLM

Architectural Whiteboard

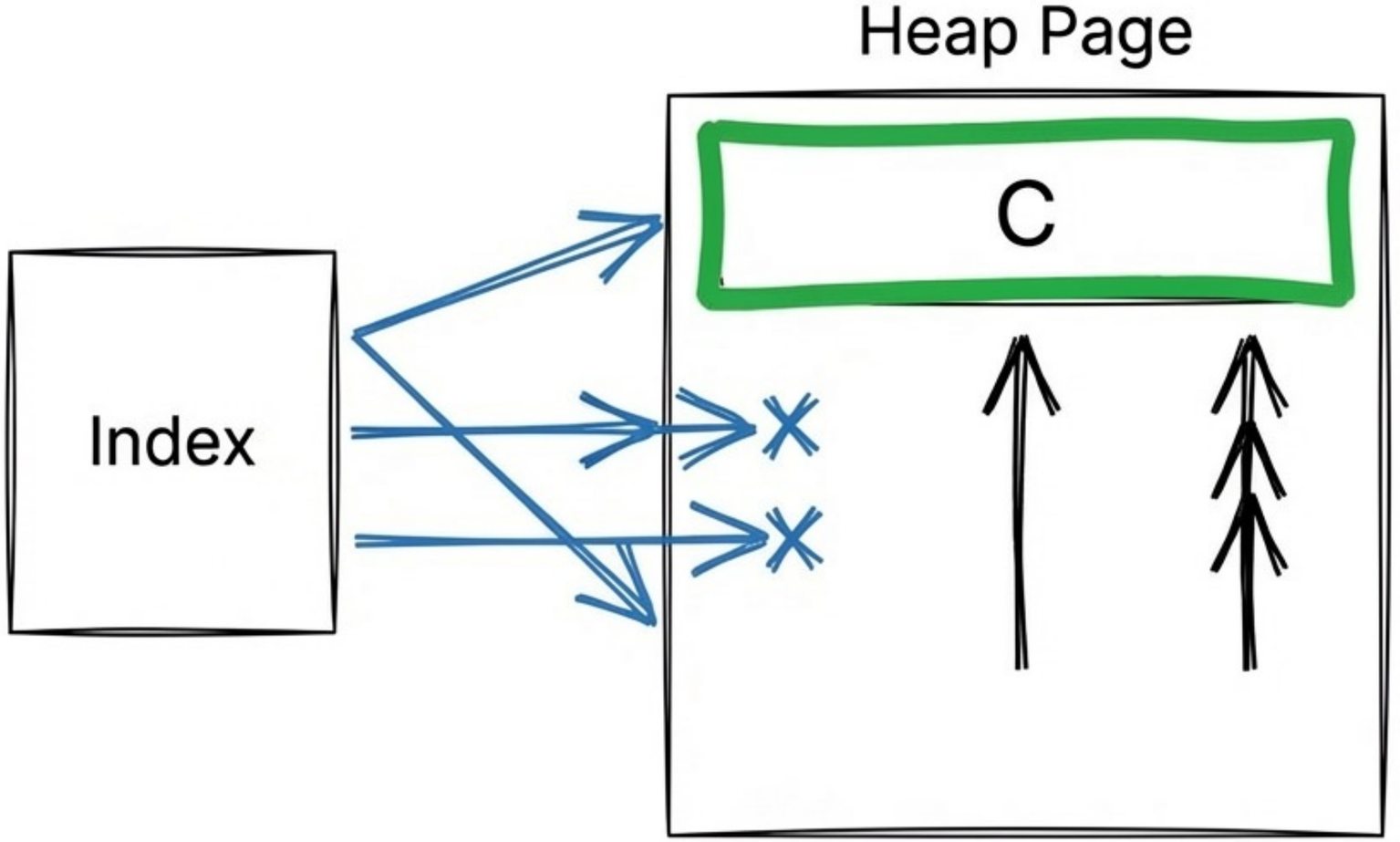


UPDATE 동작과 문제점

- UPDATE 동작: 기존 행은 보존(Dead), 새로운 행 추가
- 문제점: 인덱스가 모든 버전(A, B, C)을 가리킴
- 결과: 빠른 공간 고갈

NotebookLM

Architectural Whiteboard

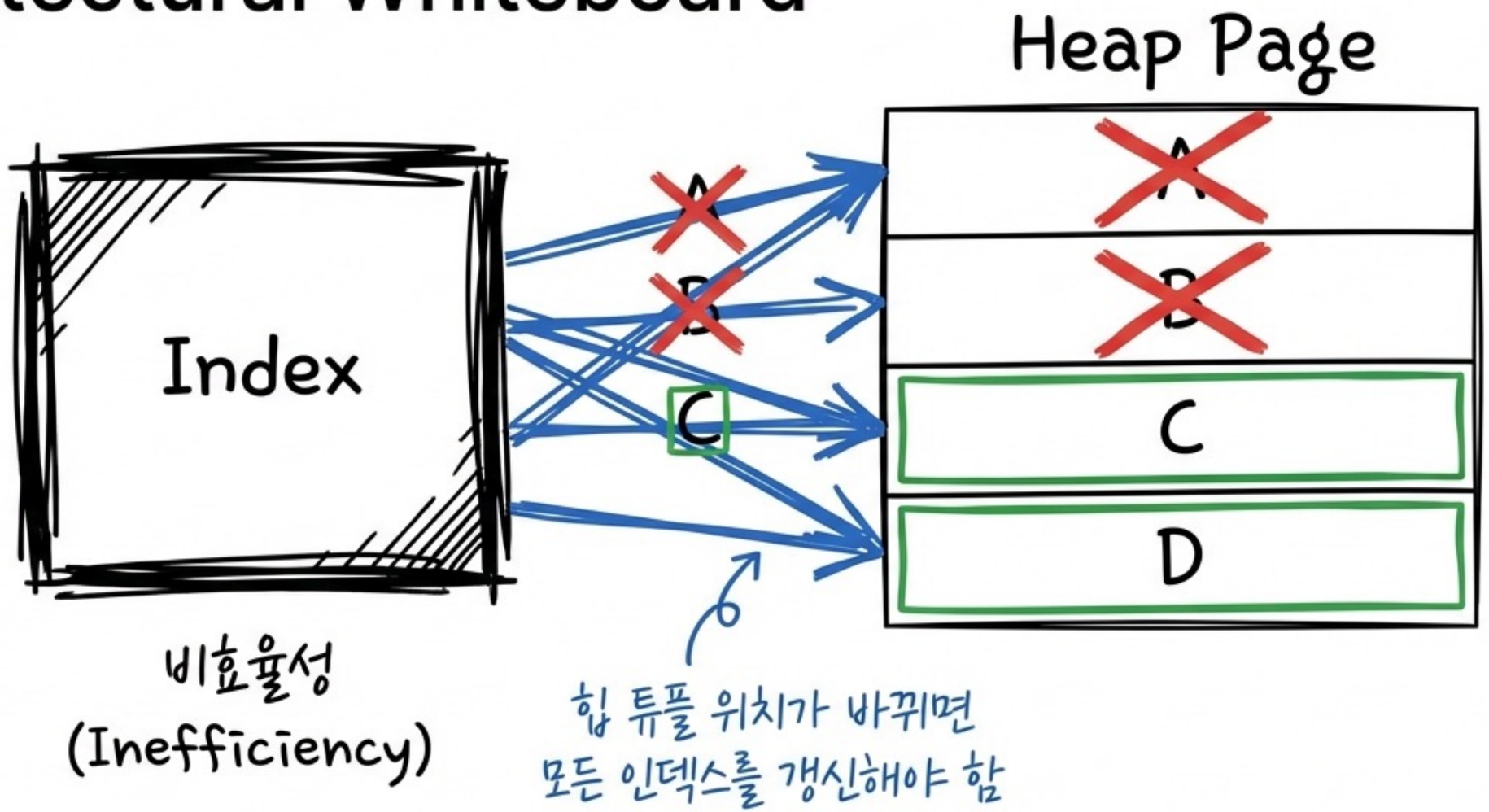


페이지 정리 (Page Pruning)

- 자동 실행: SELECT나 UPDATE 수행 시
- 동작: 스냅숏에서 볼 수 없는 튜플 제거
- 공간 회수: 빈 공간을 연속된 청크로 집약

NotebookLM

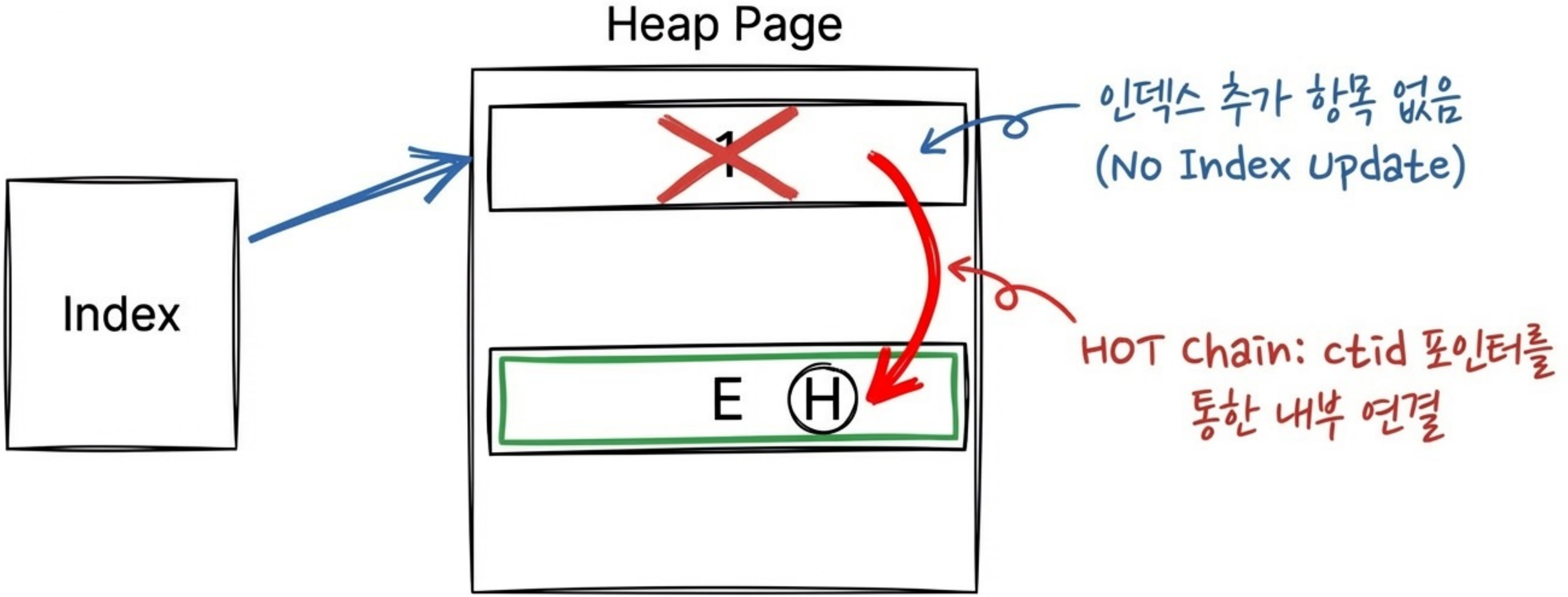
Architectural Whiteboard



비용: 쓰기 오버헤드(I/O) 증가

NotebookLM

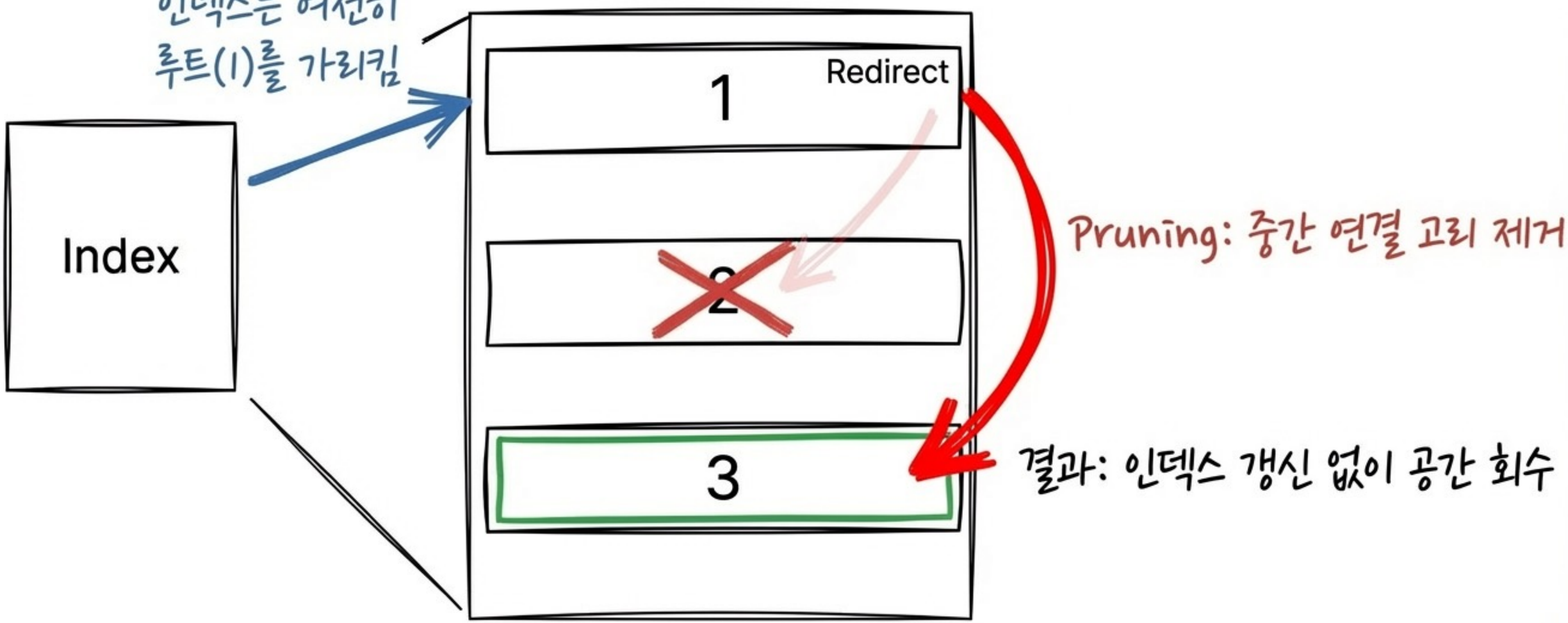
HOT (Heap-Only Tuple)



조건: 인덱스 컬럼이 변경되지 않은 경우

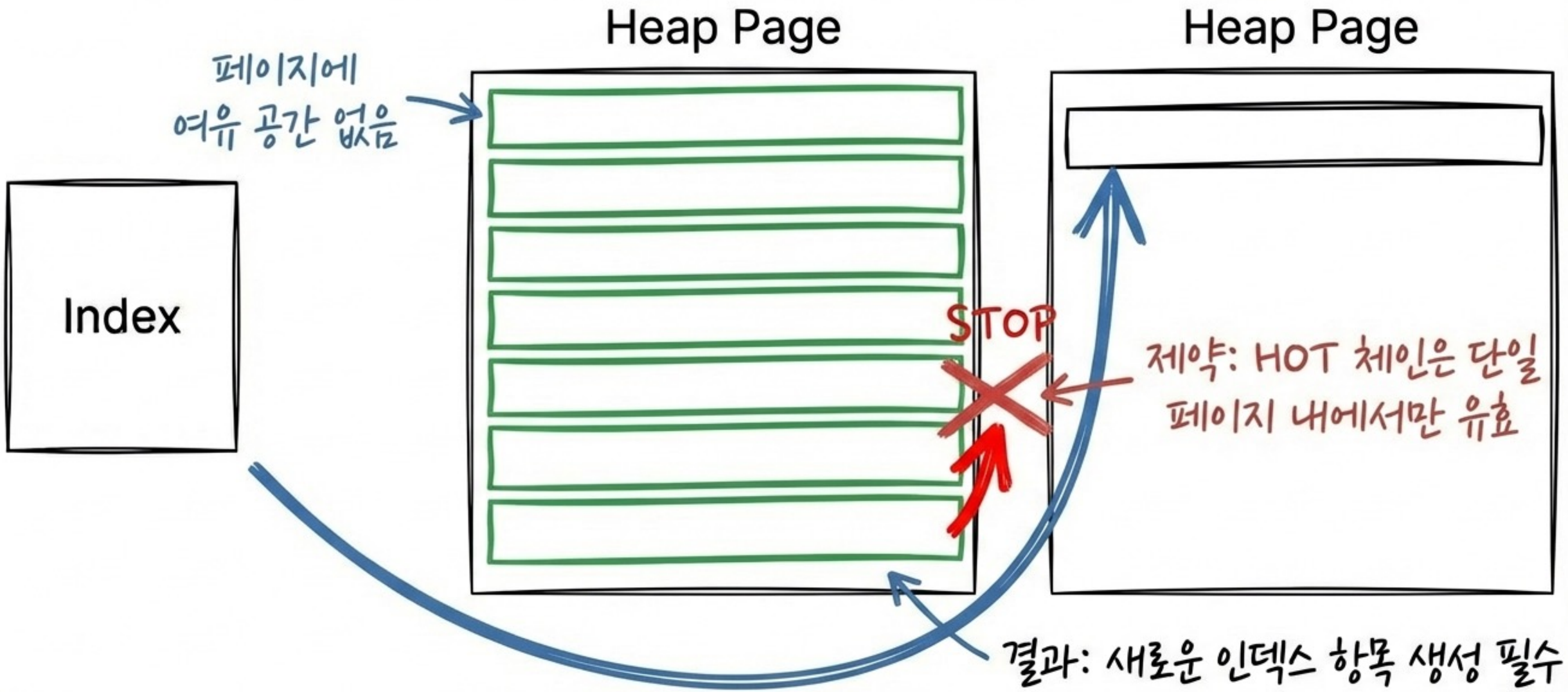
NotebookLM

이중 주소 지정 (Dual Addressing)



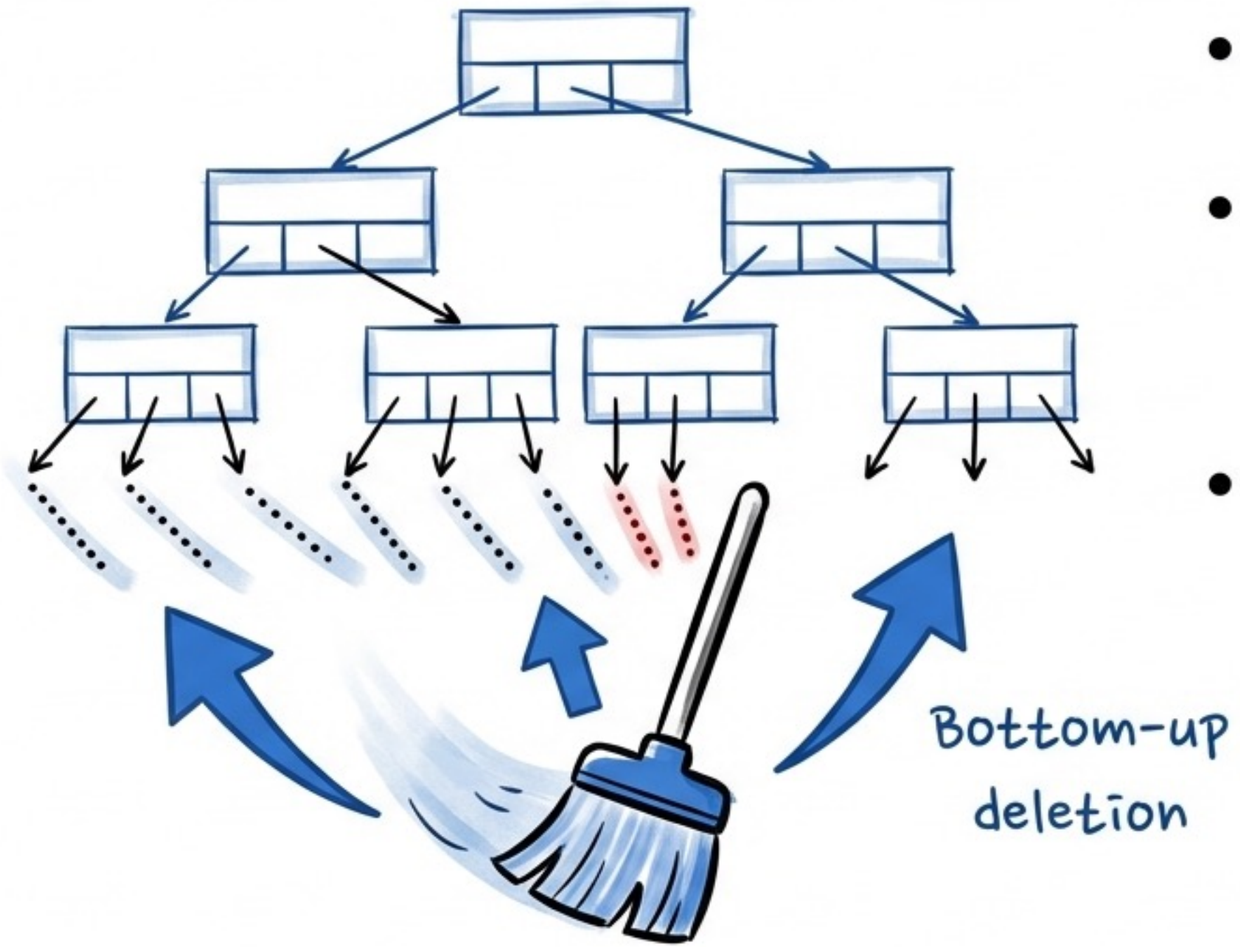
NotebookLM

HOT 사슬 분할 (Chain Splitting)



NotebookLM

인덱스 페이지 정리 (Index Page Pruning)



- 방식: Bottom-up deletion
- 시점: 페이지 분할(Split)이 예상될 때 수행
- 효과: 인덱스 팽창(Bloat) 방지

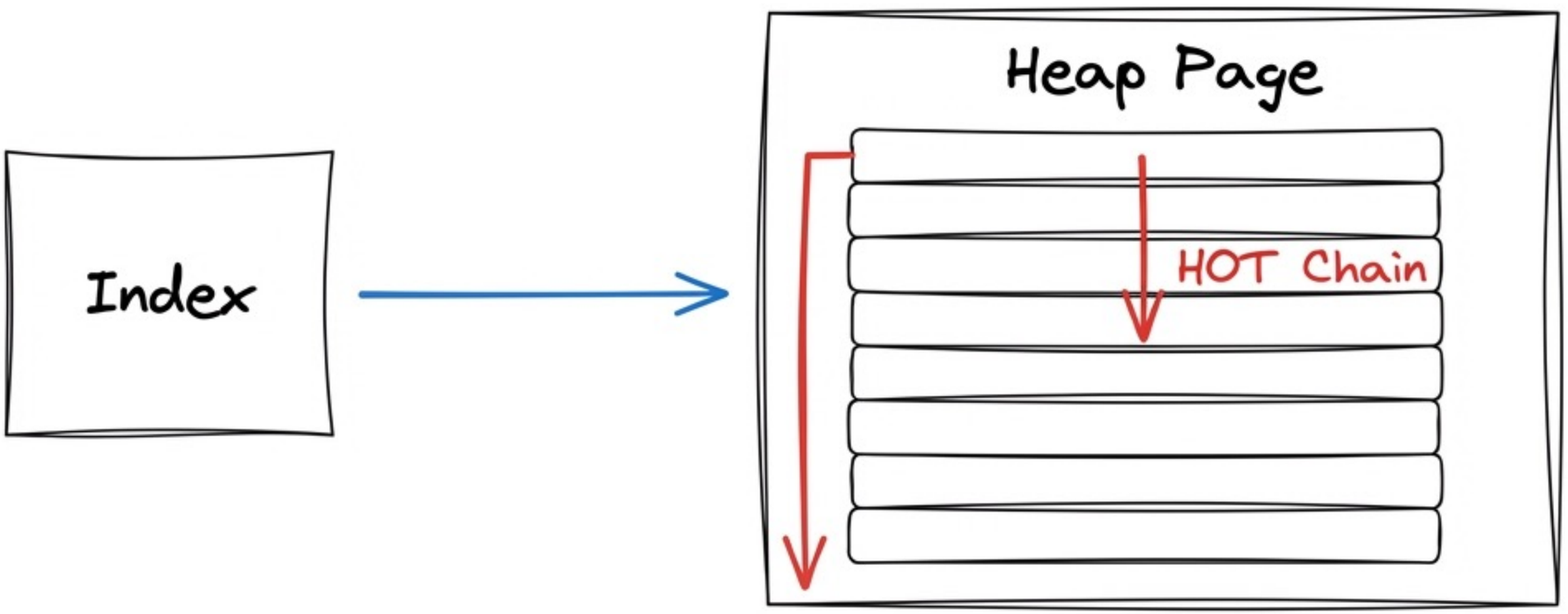
NotebookLM

요약 (Summary)

✅ 1. 쓰기 성능 향상: 인덱스 업데이트 횟수 감소

✅ 2. 자동 공간 관리: 별도의 Vacuum 없이 공간 재사용

✅ 3. 설정: 갱신이 잦은 테이블은 fillfactor 낮추기



NotebookLM

- **6. Vacuum and Autovacuum**



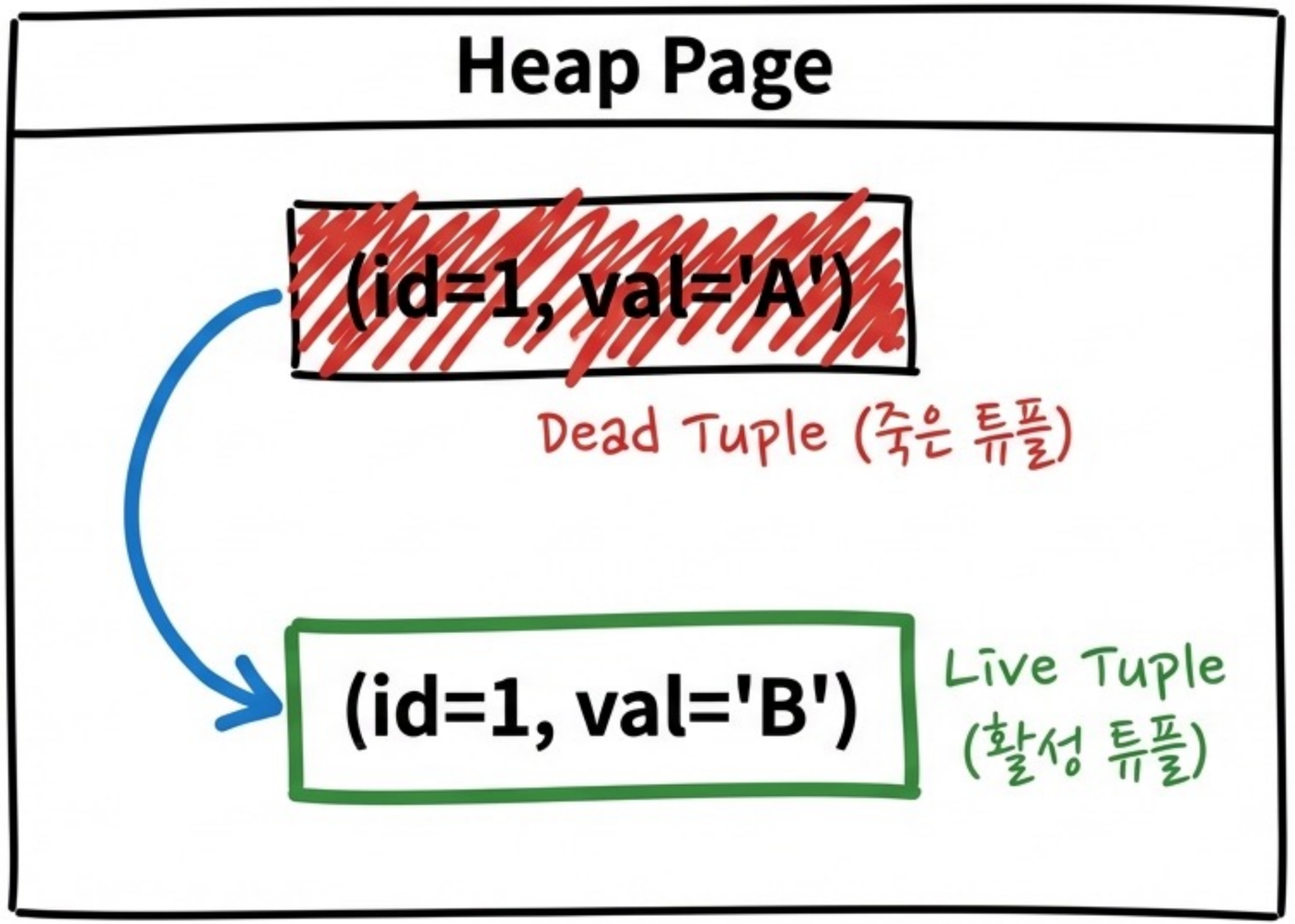
PostgreSQL Vacuum과 Autovacuum

MVCC 아키텍처와 데이터베이스 위생 관리의 핵심

데이터 수명 주기와 성능 최적화

NotebookLM

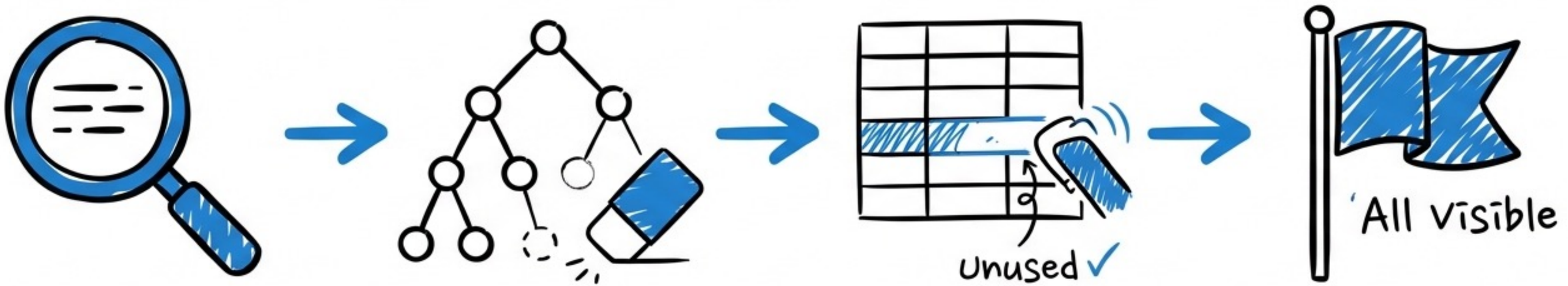
문제의 본질: 죽은 튜플 (Dead Tuples)



- PostgreSQL에서 UPDATE는 사실상 INSERT + DELETE입니다.
- 이전 버전(Ghost)은 스스로 사라지지 않고 공간을 차지합니다.
- 이것이 쌓이면 Bloat(팽창)가 발생하여 성능이 저하됩니다.

NotebookLM

청소의 매커니즘: 힙(Heap)과 인덱스(Index)의 춤



1. 스캔 (Scan)

죽은 튜플 식별

2. 인덱스 정리 (Index Vacuum)

포인터 제거

3. 힙 정리 (Heap Vacuum)

공간 재활용 (Unused)

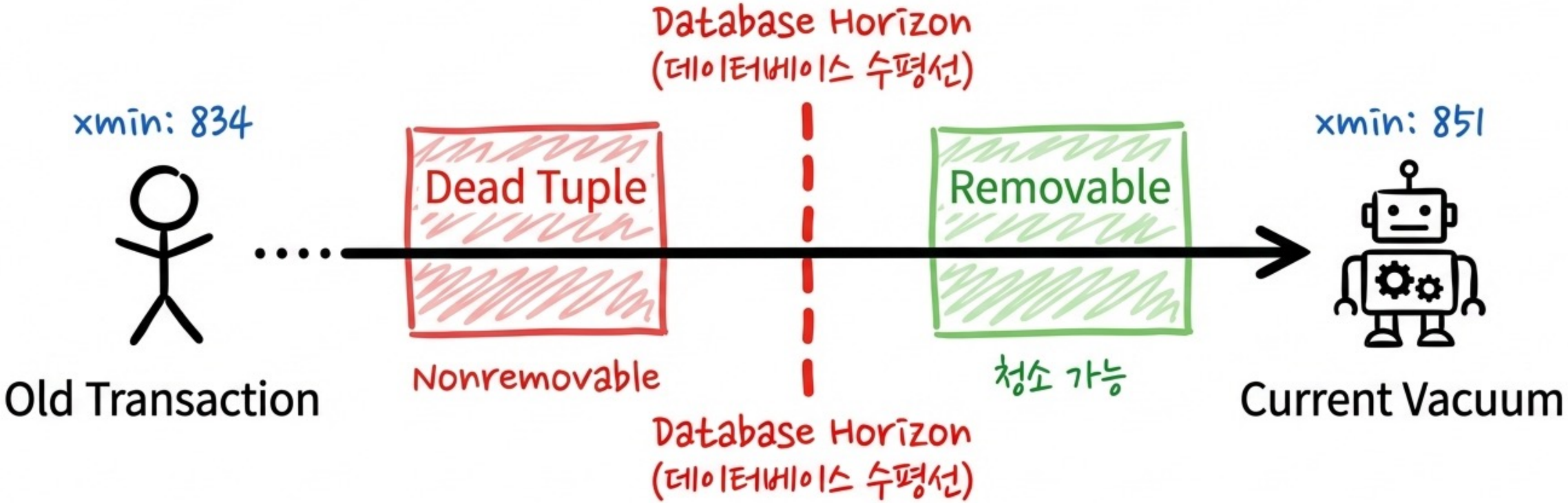
4. 가시성 맵 (Visibility Map)

'All Visible' 플래그 갱신

Note: 인덱스 정리가 선행되어야 힙 공간을 재활용할 수 있습니다.

NotebookLM

삭제할 수 없는 데이터: 데이터베이스 수평선 (Horizon)



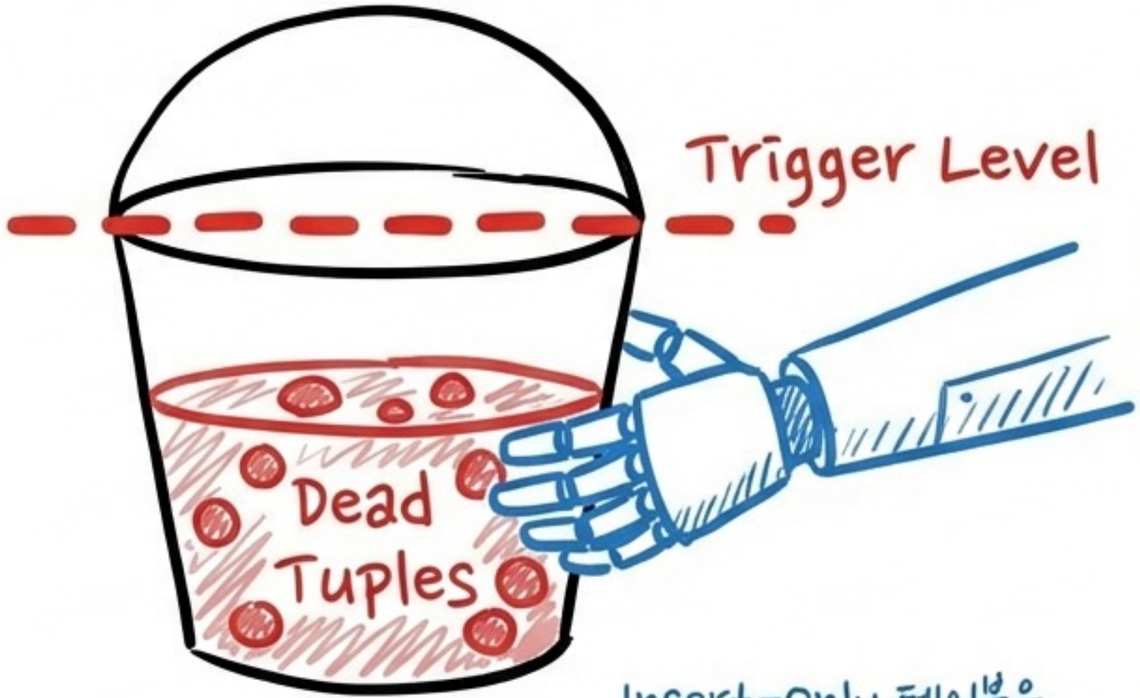
- xmin (트랜잭션 ID): 가장 오래된 활성 트랜잭션이 보고 있는 시점.
- 이 시점보다 최신인 '죽은 튜플'은 삭제할 수 없습니다 (Nonremovable).
- 오래 실행되는 트랜잭션(Long-running Transaction)은 청소를 막는 주범입니다.

NotebookLM

Autovacuum: 로봇은 언제 깨어나는가?

Trigger = Threshold + (Scale Factor × Total Rows)

Threshold = 50
Scale Factor = 0.2 (20%)

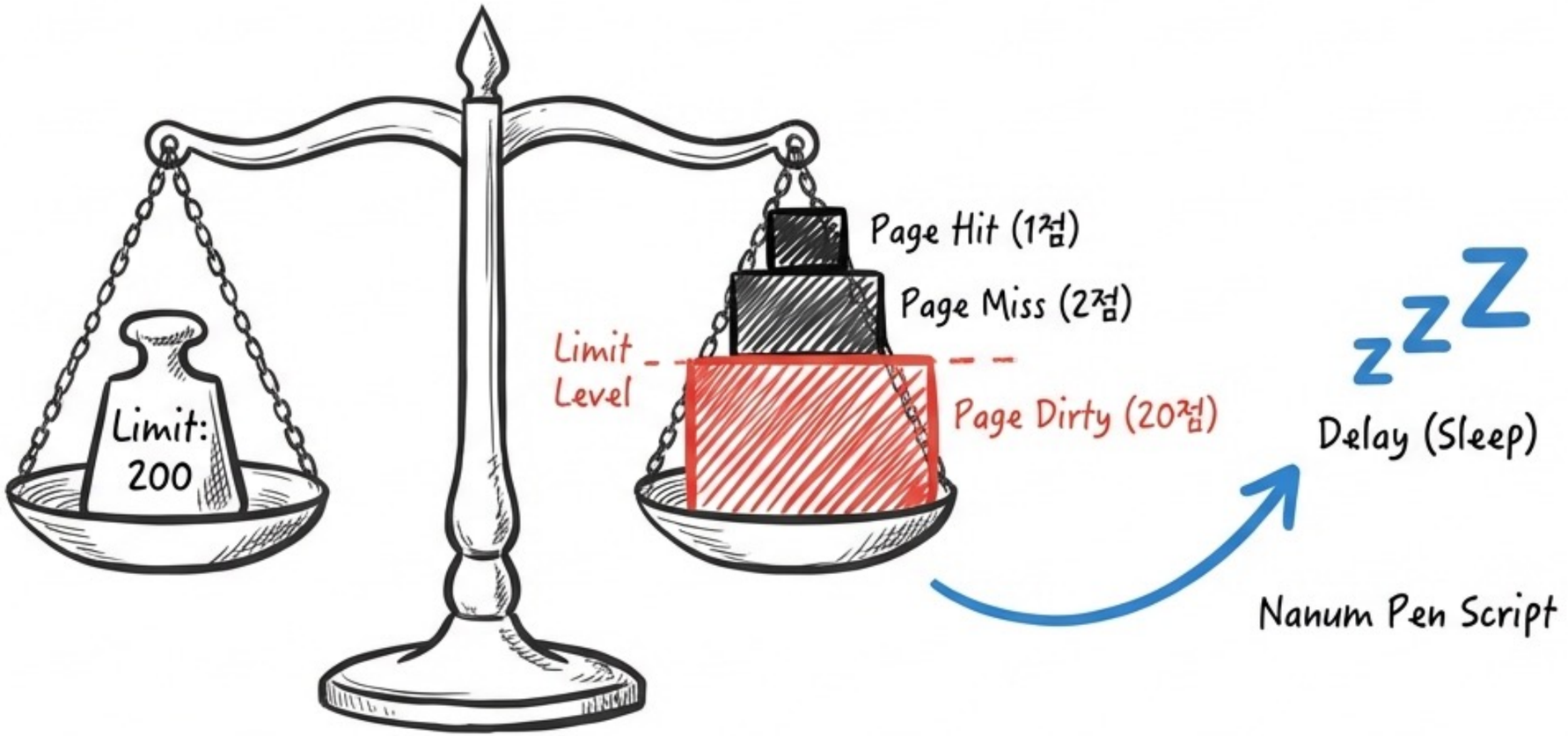


Insert-only 테이블은 autovacuum_vacuum_insert_threshold로 별도 관리.

- Threshold (기본값 50): 최소 발생 건수.
- Scale Factor (기본값 0.2): 테이블 크기의 20%가 변동되면 실행.
- 대용량 테이블에서는 20%가 너무 크므로, 이 계수를 줄여야 합니다.

NotebookLM

Noto Sans KR: 부하 관리: 비용(Cost) 기반 스로틀링



- 점수 합계가 Limit에 도달하면 프로세스는 Delay (기본값 2ms) 만큼 잠듭니다.
- 시스템 부하를 줄이기 위해 천천히 청소하는 원리입니다.

NotebookLM

모니터링: 청소 상태 확인

pg_stat_progress_vacuum

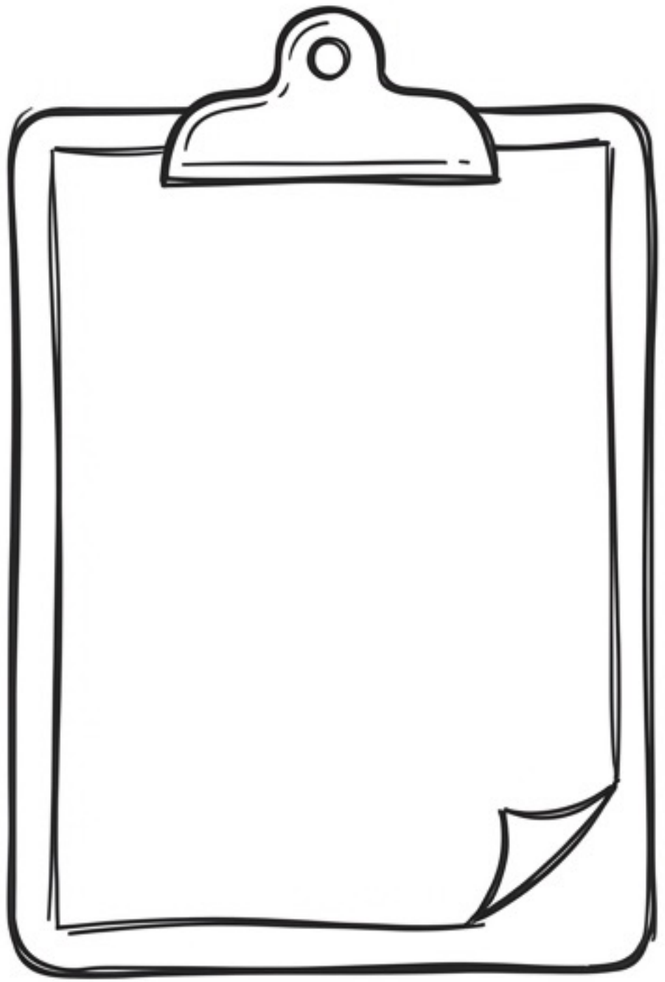
phase	heap_blks_vacuumed	max_dead_tuples
vacuuming indexes	1024	500000

```
automatic vacuum of table 'public.vac'
index scans: 3
removed 174522 dead item identifiers
```

- pg_stat_progress_vacuum: 실시간 진행 상황 파악.
- Logs: log_autovacuum_min_duration = 0 설정 시 상세 로그 기록.
- 인덱스 스캔 횟수가 많거나 maintenance_work_mem이 부족하면 성능이 저하됩니다.

NotebookLM

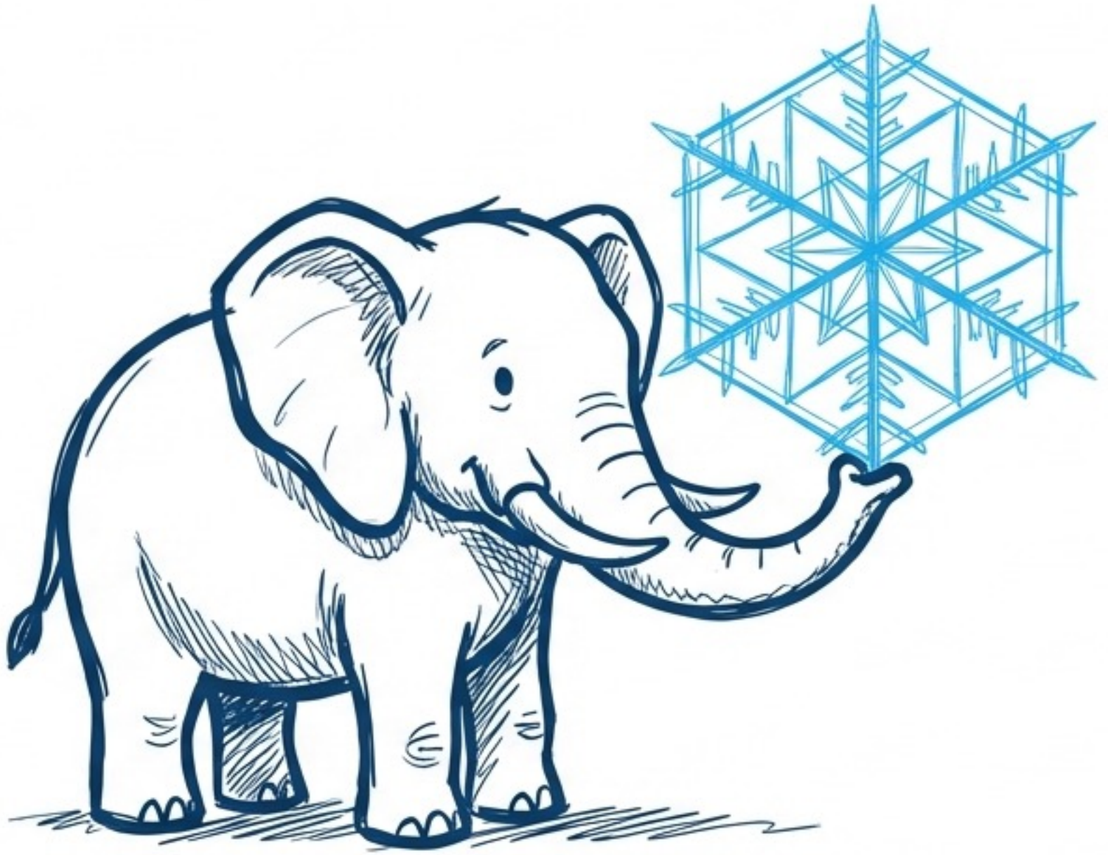
핵심 요약 및 모범 사례 (Best Practices)



- **Autovacuum을 끄지 마십시오**
  수동 관리는 거의 불가능합니다.
- **Scale Factor 튜닝**
  대용량 테이블은 20%(0.2)보다 낮게 설정하십시오.
- **모니터링**
  pg_stat_progress_vacuum 뷰를 주기적으로 확인하십시오.
- **트랜잭션 관리**
  xmin을 붙잡고 있는 오래된 트랜잭션을 주의하십시오.
- **리소스 할당**
  maintenance_work_mem을 충분히 할당하여 인덱스 스캔 횟수를 줄이십시오.

NotebookLM

- **7. Freezing**

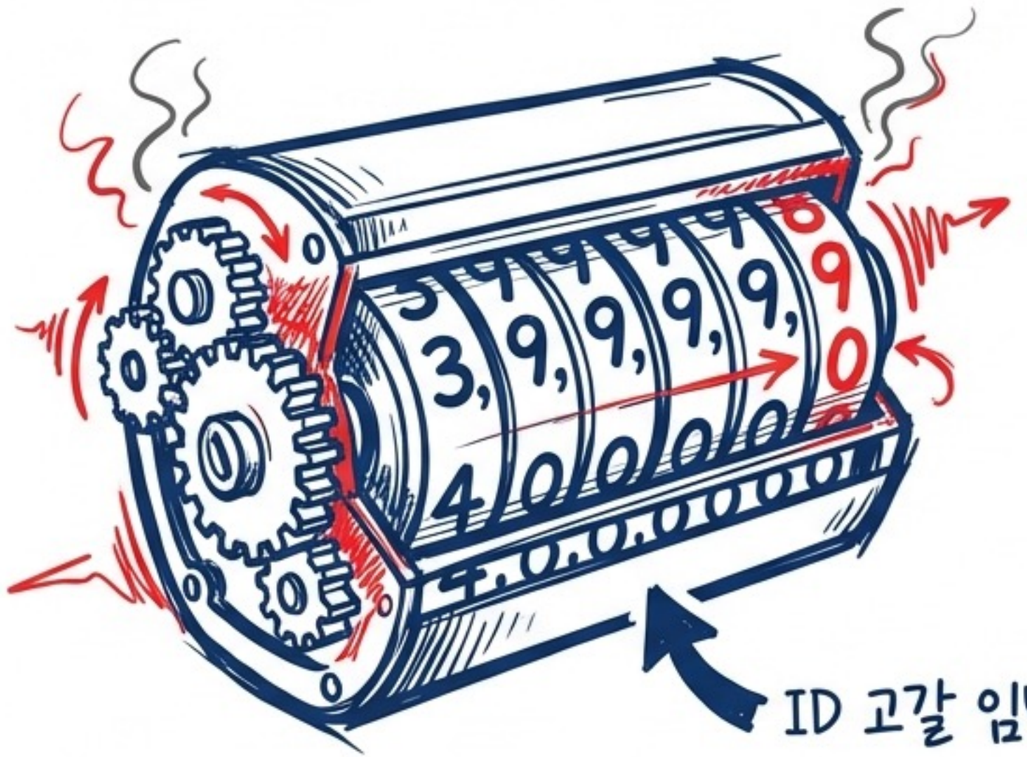


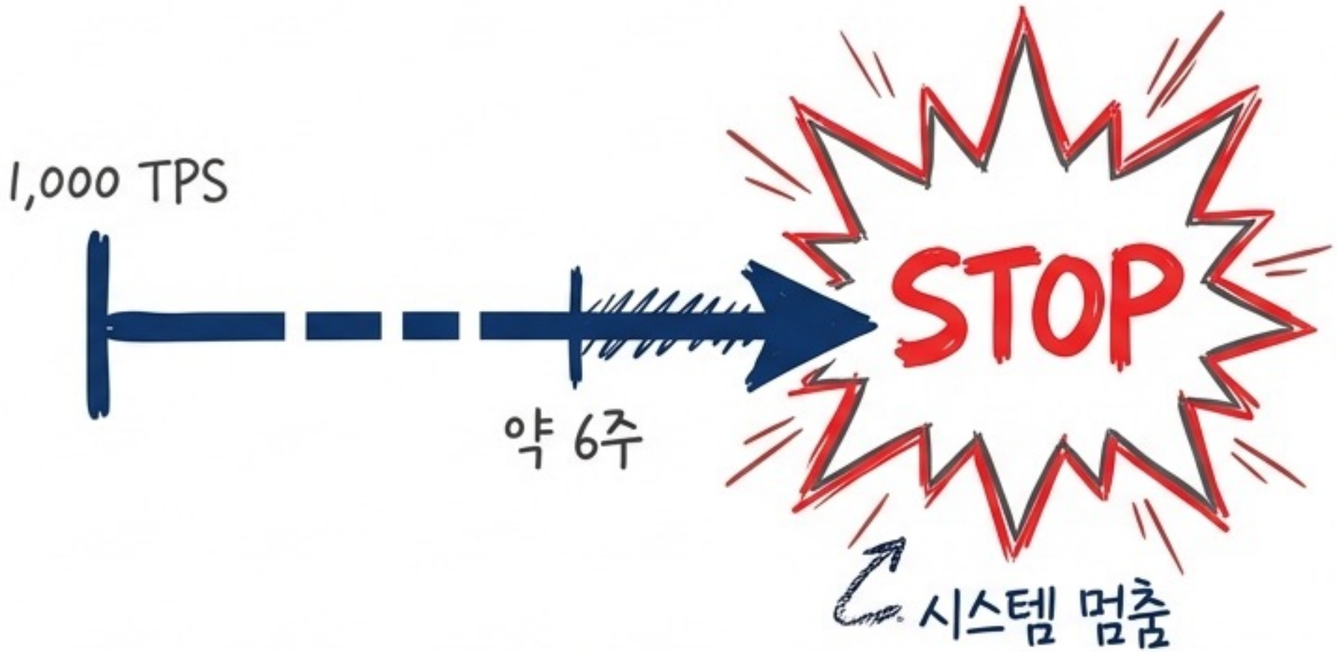
PostgreSQL 프리징과 트랜잭션 ID 랩어라운드

32비트의 한계와 데이터 영구 보존 전략

NotebookLM

32비트 트랜잭션 ID의 한계

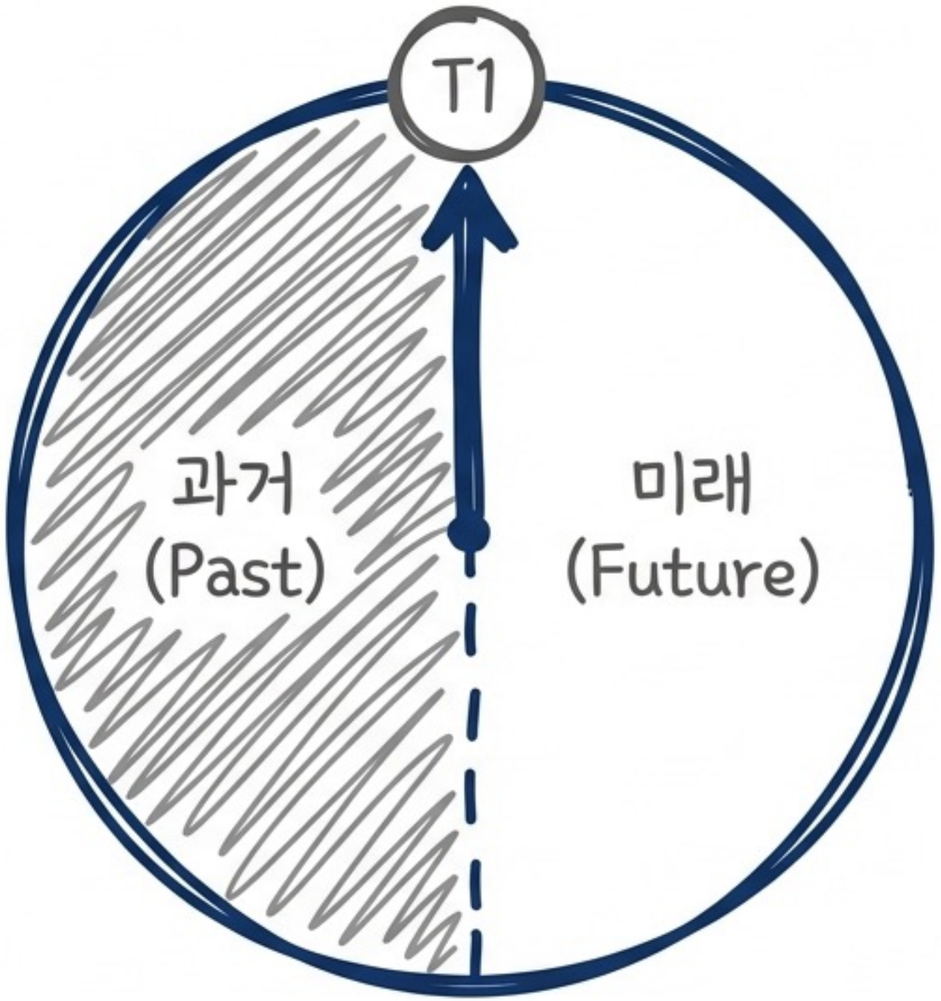




- 총 40억 개의 유한한 ID
- 초당 1,000 트랜잭션 발생 시, 약 6주 후 ID 고갈
- ID가 소진되면 데이터베이스 멈춤

NotebookLM

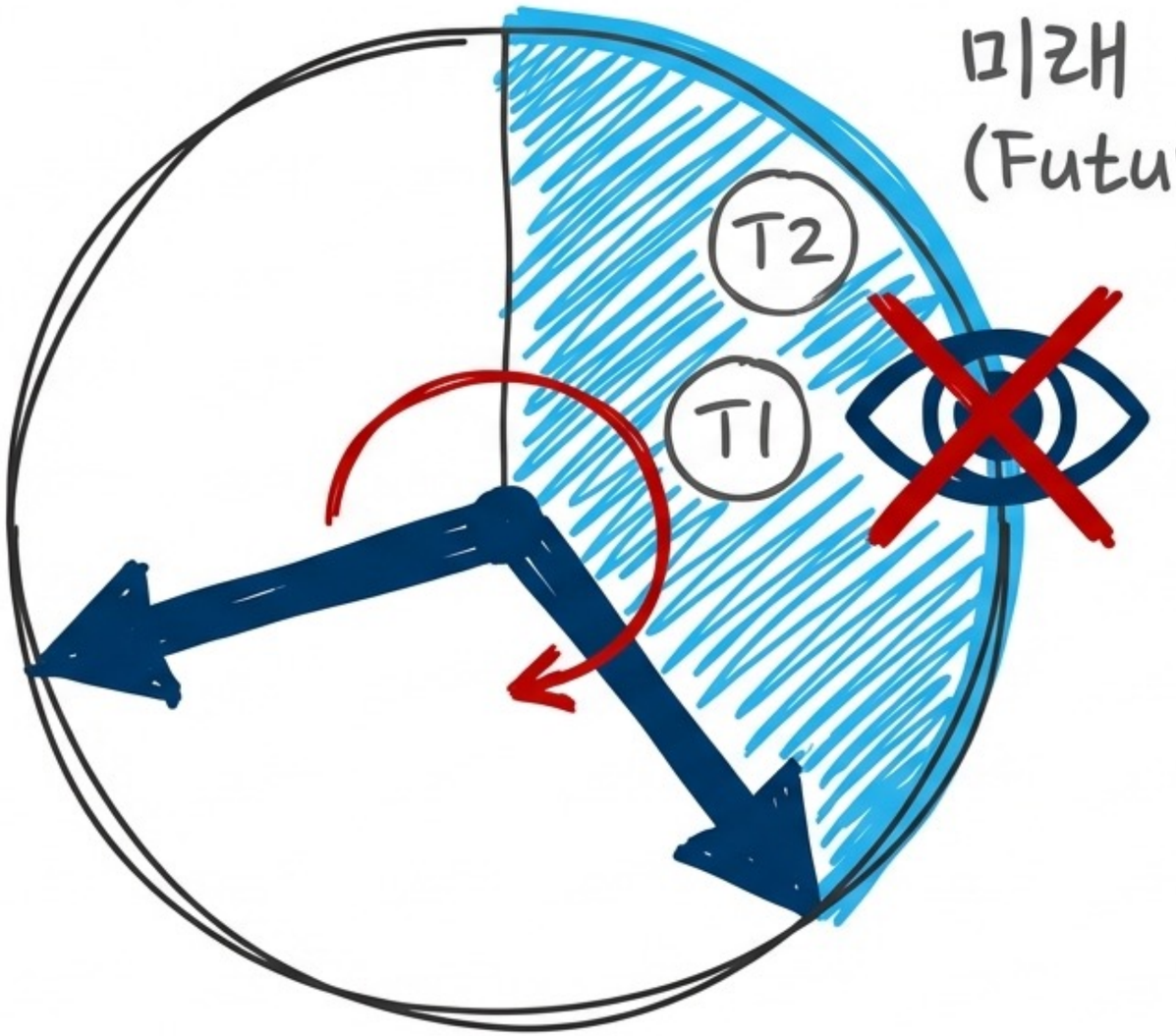
Noto Sans KR
트랜잭션 나이 비교: 원형 연산



- Nanum Pen Script
- ID의 크기보다는 '나이(Age)'가 중요
- 현재 ID 기준: 반시계 방향 = 과거, 시계 방향 = 미래
- 32비트 산술로 차이를 계산하여 순서 파악

NotebookLM

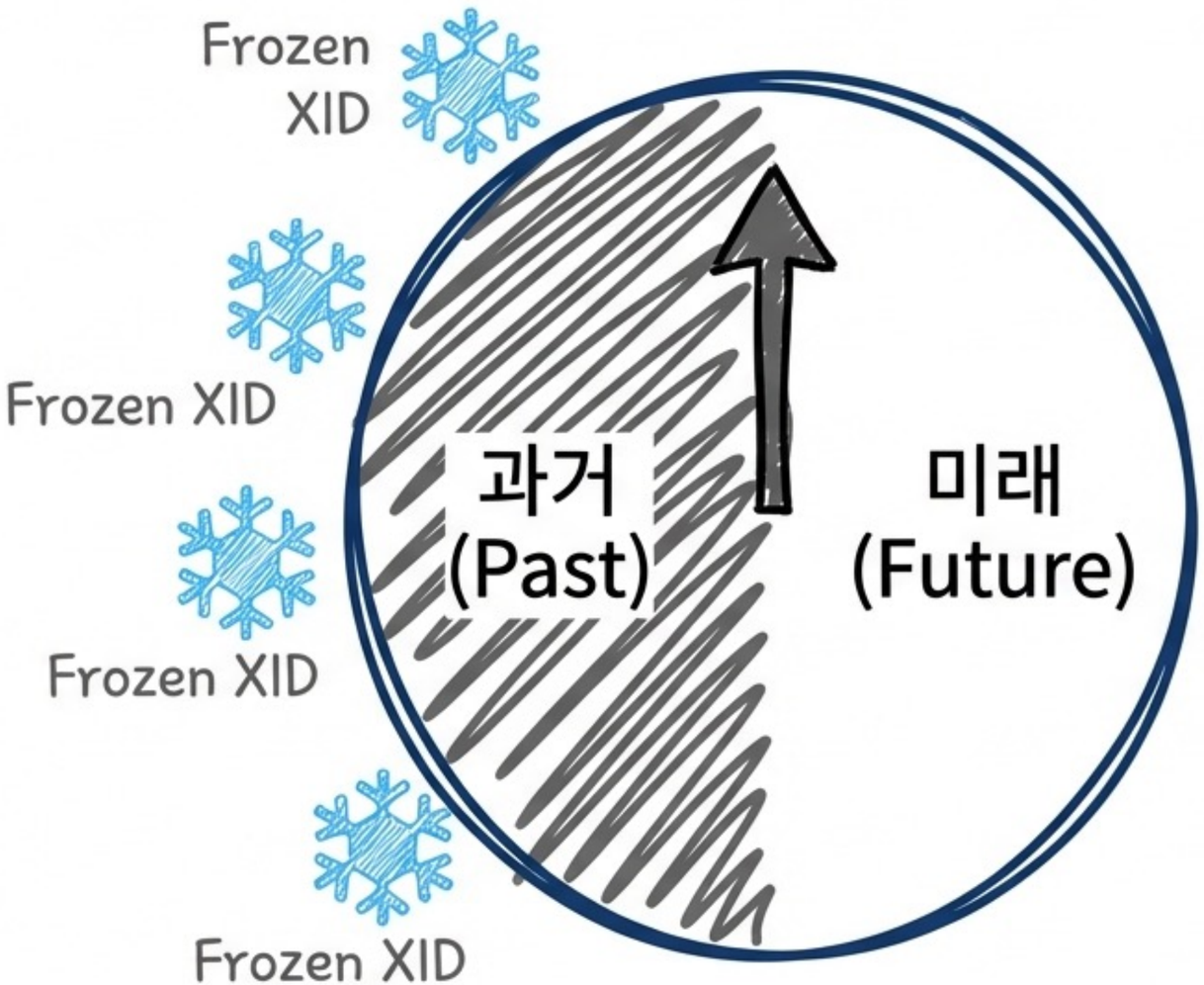
랩어라운드(Wraparound)와 데이터 소실



- ID가 한 바퀴를 돌아버리면 '과거'가 '미래'로 인식됨
- 과거 데이터가 보이지 않게 되는 'Invisible Data' 현상
- 치명적인 데이터 정합성 오류 발생

NotebookLM

해결책: 튜플 프리징 (Tuple Freezing)

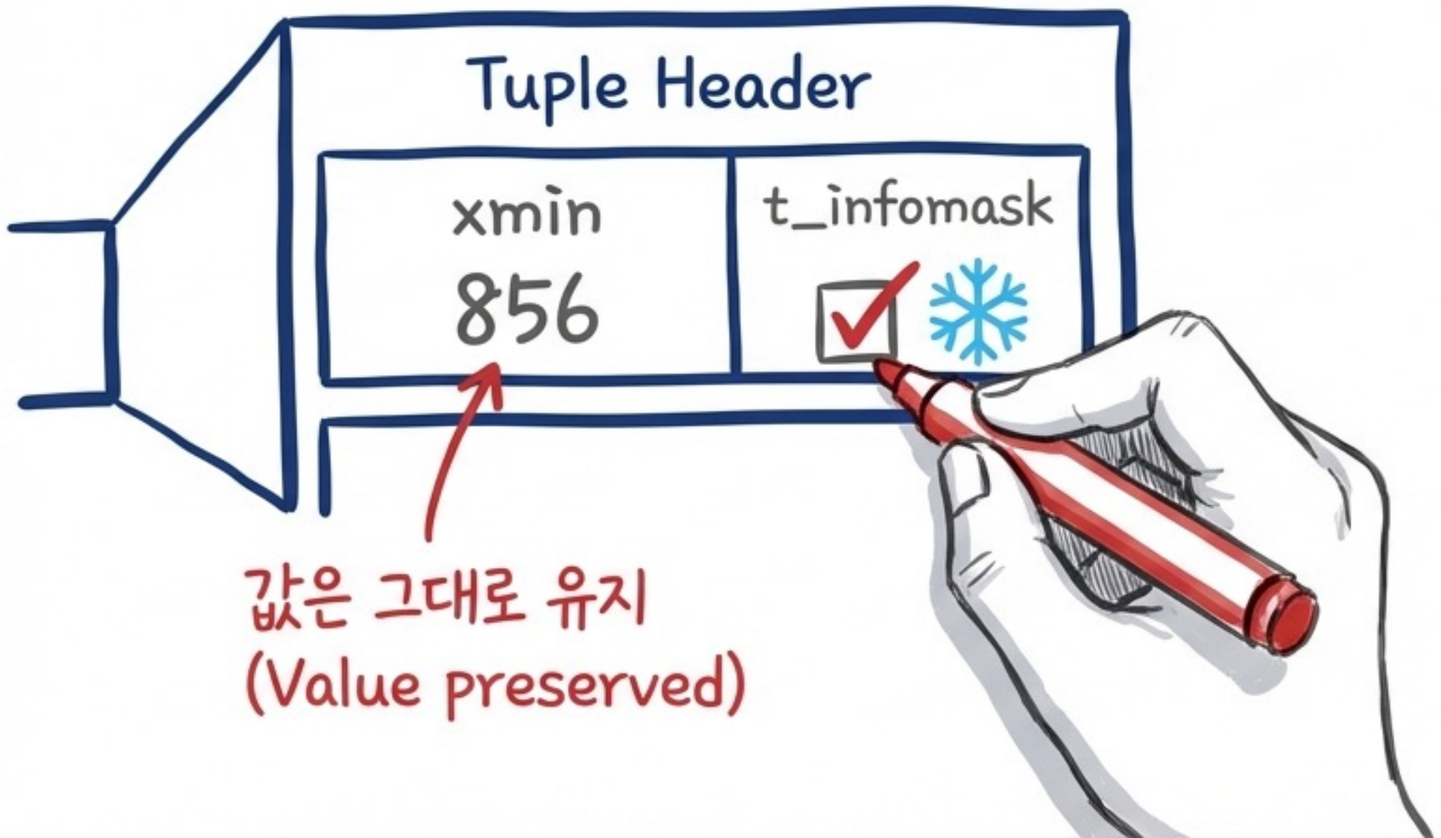


- 오래된 ID를
  '음의 무한대(Negative Infinity)'로
  취급
- 원형 경쟁에서 제외시켜
  영구적인 '과거' 상태로 고정
- 모든 트랜잭션에서
  무조건 보이는 상태가 됨

NotebookLM

프리징의 내부 동작

- 과거(9.4 이전): FrozenTransactionId(값 2)로 덮어쓰기
- 현재: xmin 값은 유지(디버깅 용도), 힌트 비트(Hint Bit) 설정
- 데이터 페이지의 비트 플래그만 변경하여 효율성 증대



NotebookLM

Autovacuum의 단계별 프리징 전략

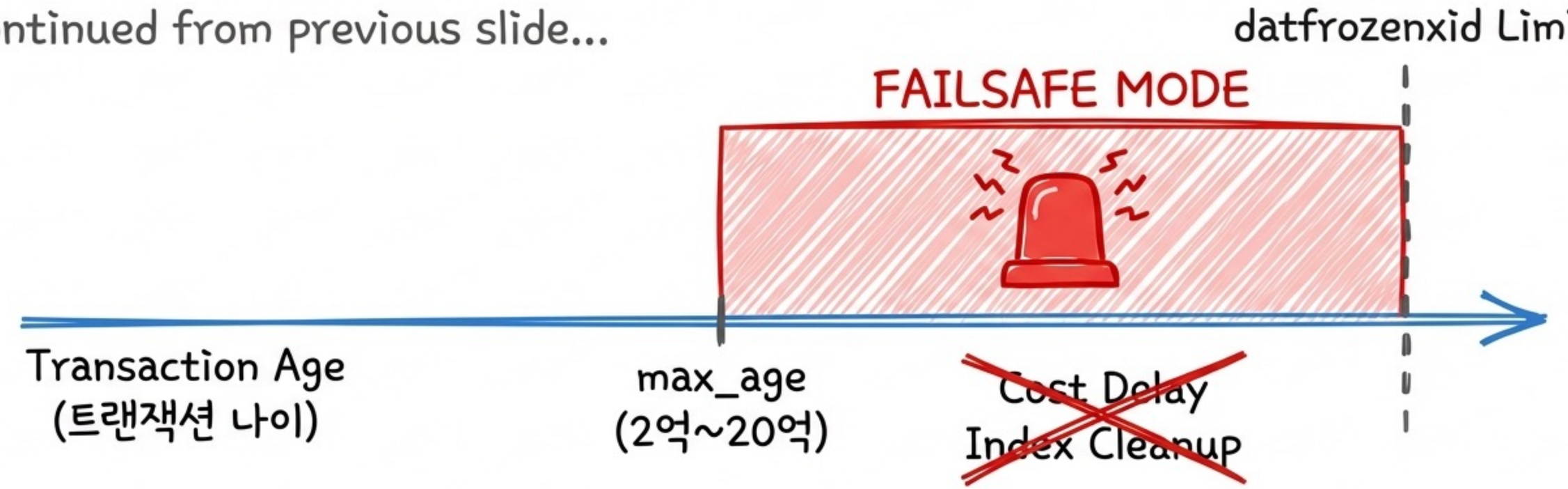


- 최소 나이(Min Age): 불필요한 프리징 방지, Dirty 페이지 위주 작업
- 공격적 나이(Table Age): relfrozenxid 갱신을 위해 테이블 전체 강제 스캔
- 가시성 맵(Visibility Map)을 활용해 이미 처리된 페이지는 스킵

NotebookLM

최후의 방어선: 강제 프리징과 Failsafe

Continued from previous slide...



- Max Age 도달 시: Autovacuum 강제 실행 (사용자 설정 무시)
- Failsafe 모드: vacuum_cost_delay 무시 (성능보다 데이터 보존 우선)
- 인덱스 정리를 생략하고 힙(Heap) 프리징에만 집중

NotebookLM

수동 최적화: COPY ... WITH FREEZE

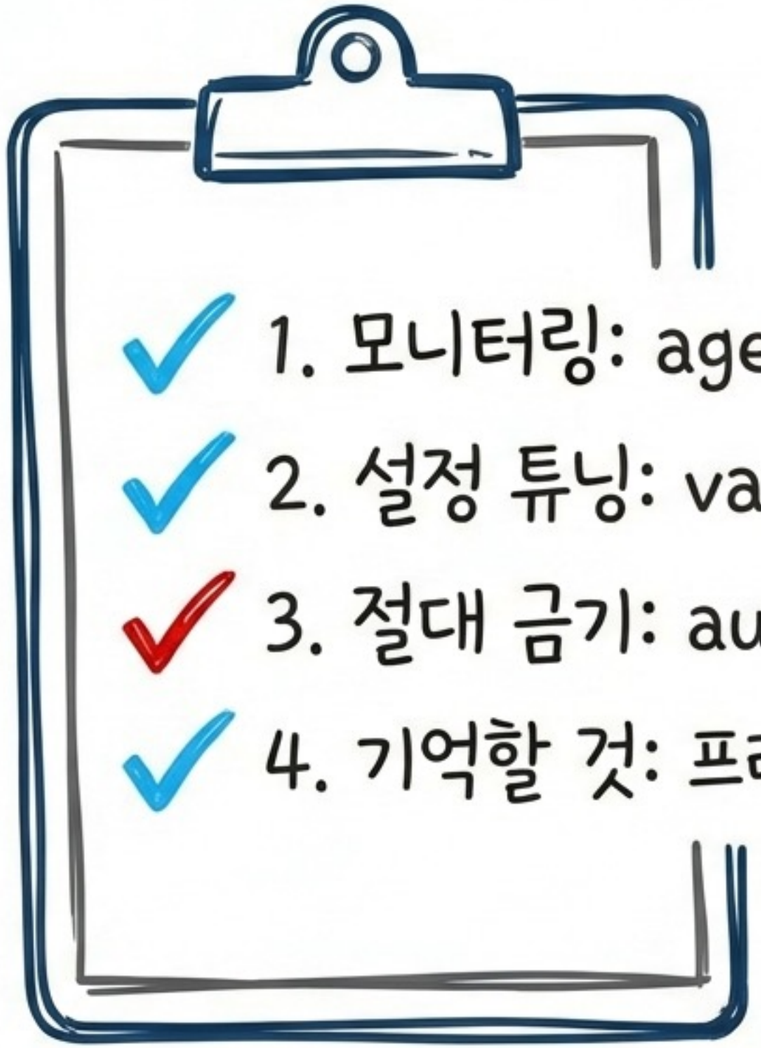


COPY ... WITH FREEZE

- 초기 적재(Bulk Load): 입력과 동시에 프리징 완료
- 추후 Vacuum의 부하를 사전에 제거
- 동일 트랜잭션 내 생성/삭제된 테이블에만 사용 가능

NotebookLM

핵심 요약 및 운영 수칙



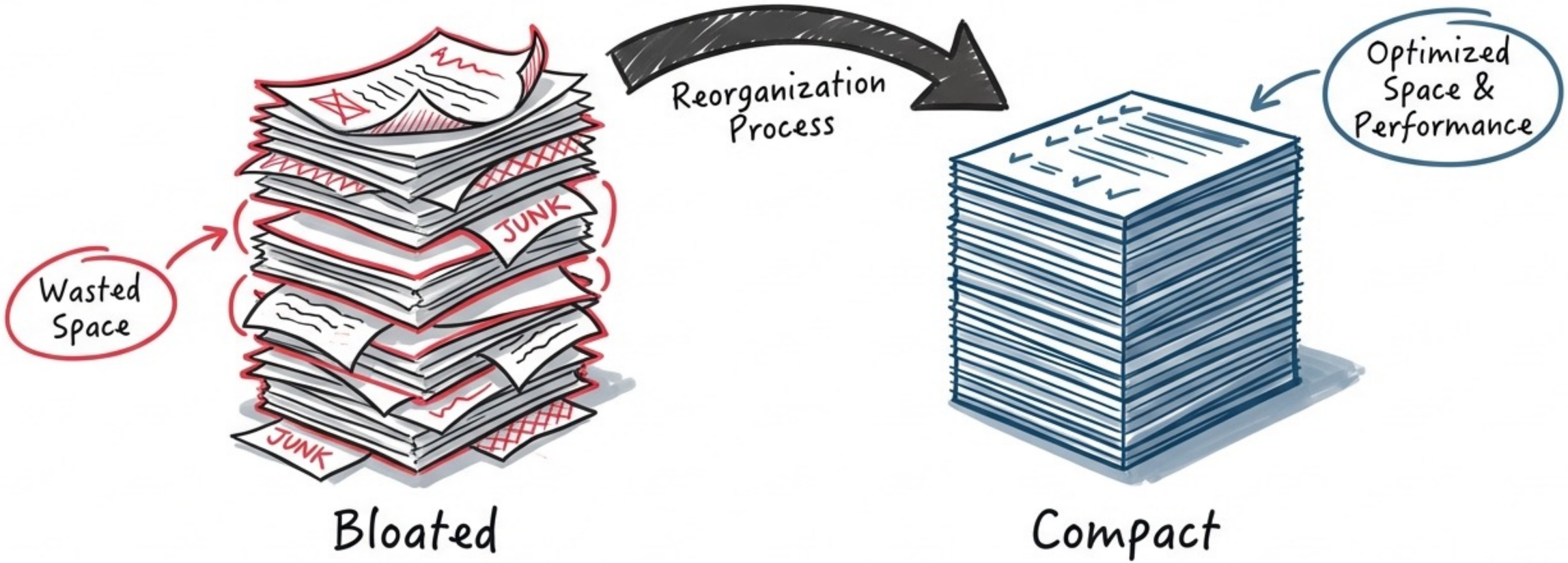
- 1. 모니터링: age(datfrozenxid) 값 주기적 확인
- 2. 설정 튜닝: vacuum_freeze_table_age 최적화
- 3. 절대 금기: autovacuum 끄지 말기
- 4. 기억할 것: 프리징은 시간을 멈춰 데이터를 영원히 보존하는 기술

NotebookLM

- **8. Rebuilding Tables and Indexes**

8장: 테이블과 인덱스 재구성

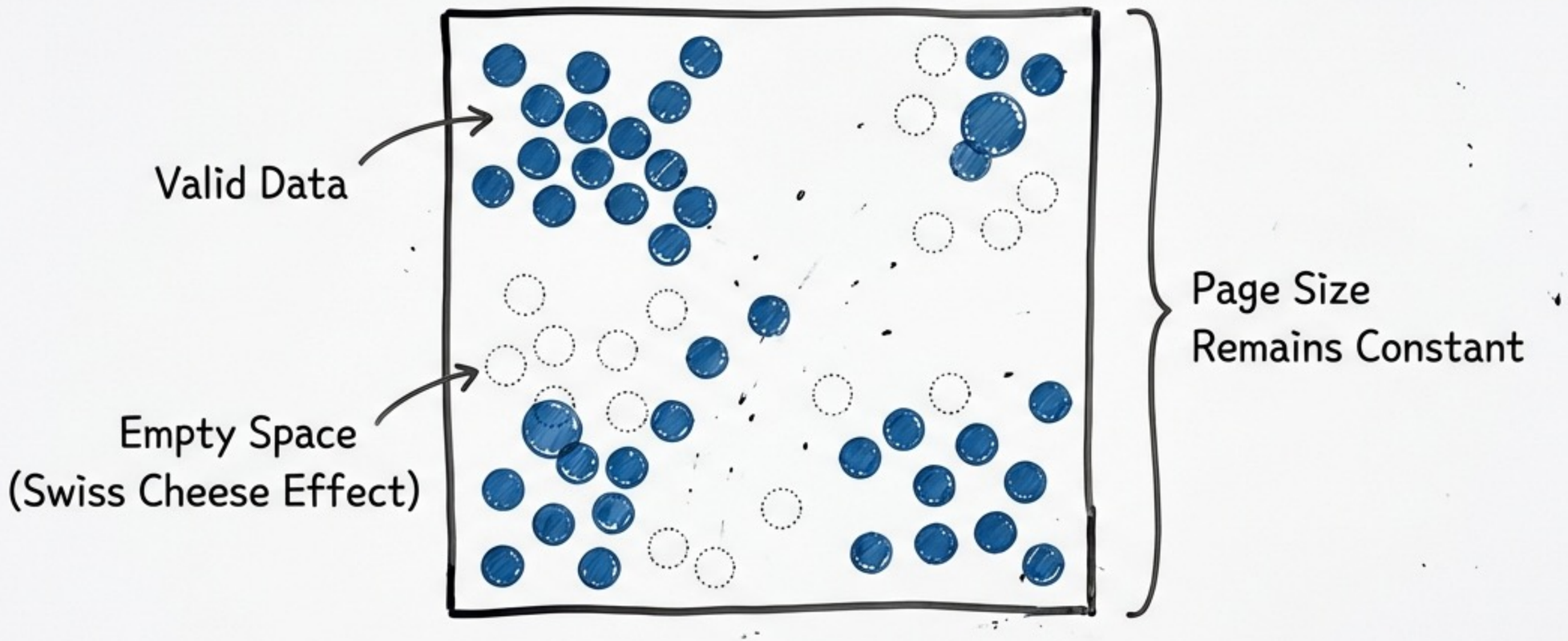
공간 회수와 성능 최적화를 위한 아키텍처 가이드



Based on PostgreSQL Internal

NotebookLM

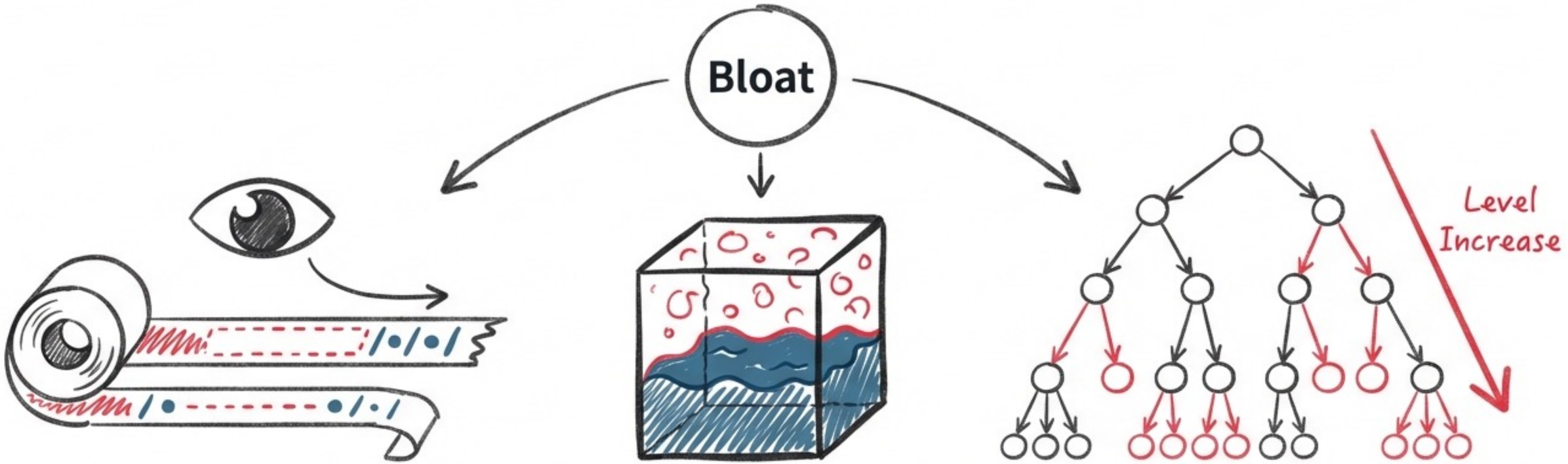
왜 VACUUM만으로는 부족한가?



- Page Pruning: 페이지 내부의 공간만 정리함
- No Size Reduction: 파일 끝(End of File)이 아니면 OS에 반환 안 됨
- Swiss Cheese Effect: 데이터 파일 내부에 '구멍'만 남음

NotebookLM

비대해진 테이블의 대가



Disk I/O

전체 스캔 속도 저하

Memory

캐시 효율 감소 (밀도 저하)

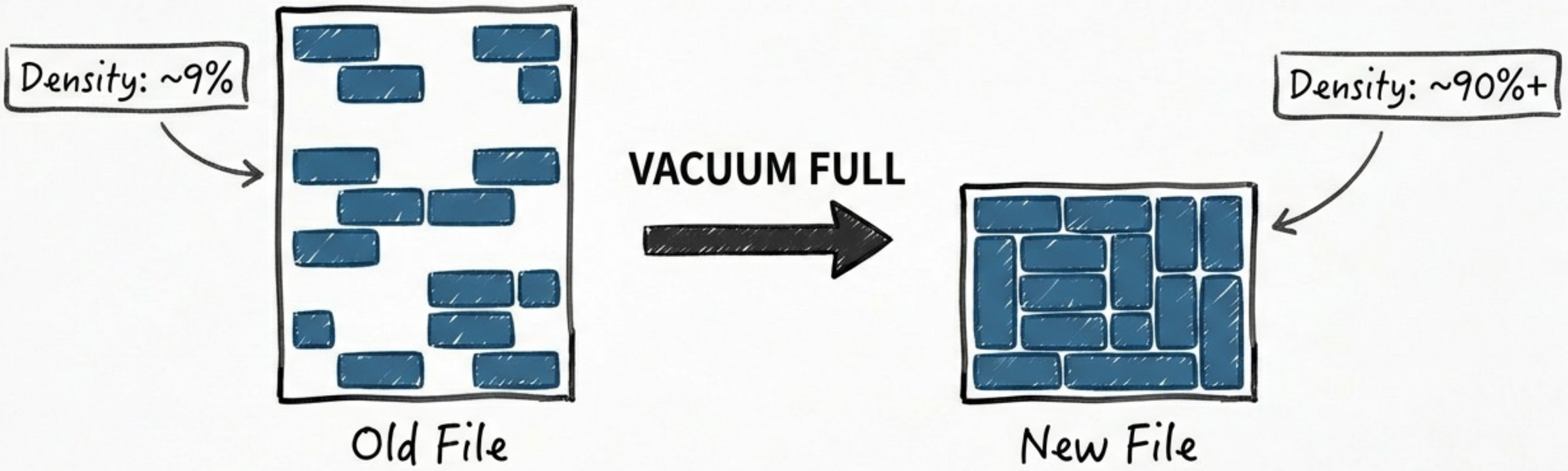
Index

레벨 증가 = 조회 속도 저하

디스크 낭비뿐만 아니라 I/O와 CPU 효율성까지 저하시킴.

NotebookLM

VACUUM FULL: 완벽한 재건축



- Process: 새로운 파일을 생성하여 데이터를 밀집(Packed) 이동
- Result: 테이블과 인덱스 완전 재구축
- Note: fillfactor 설정에 따라 밀집도 결정

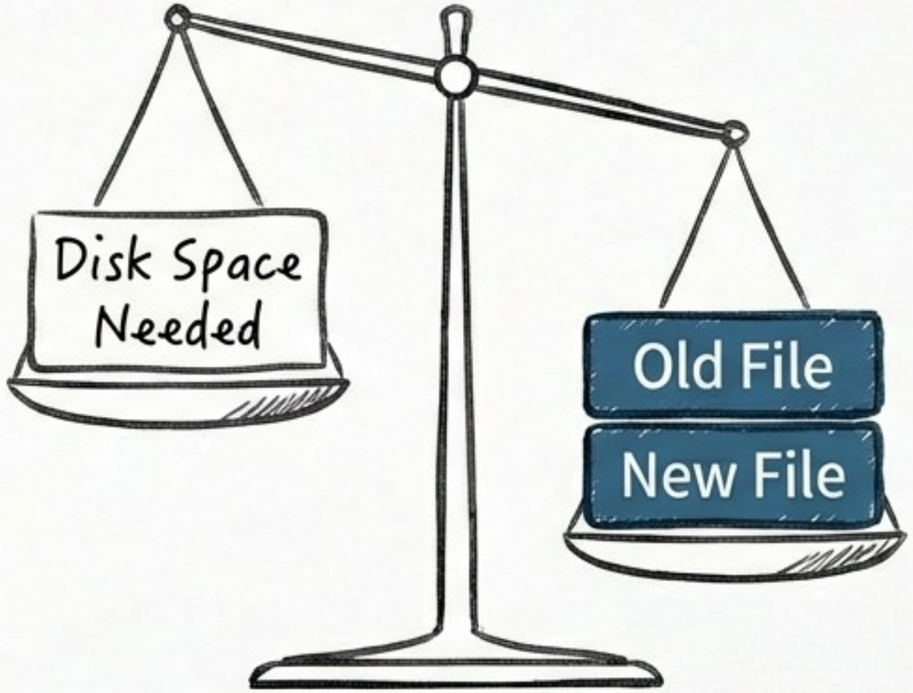
NotebookLM

주의: 공사 중 출입 금지

Access Exclusive Lock



읽기/쓰기 접근 완전 차단



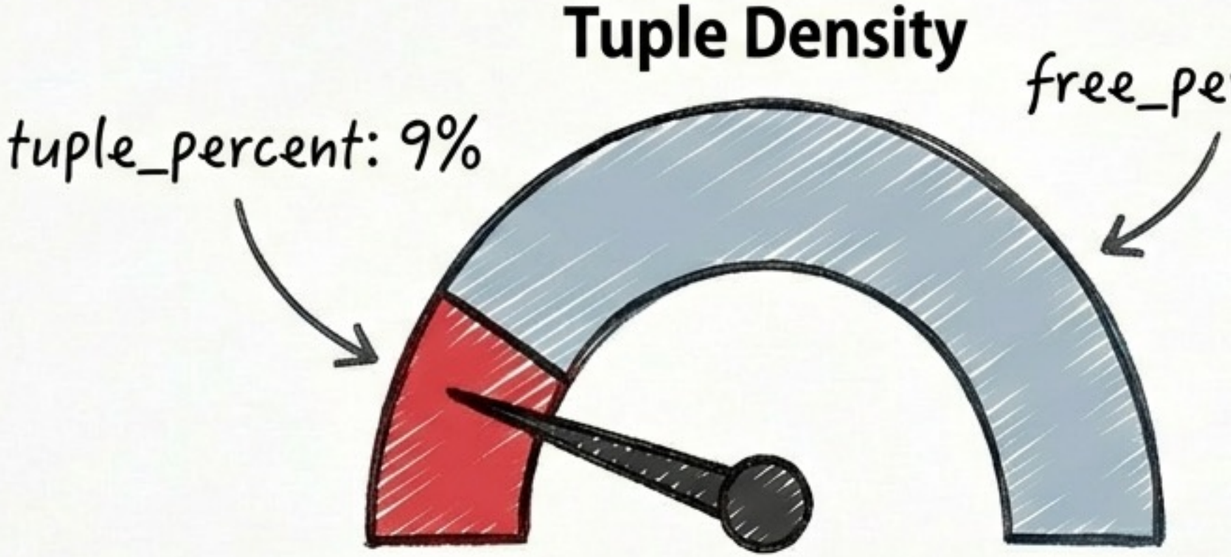
작업 도중 두 배의 디스크 공간 필요

운영 중인 서비스(Production)에서는 다운타임 발생 주의

NotebookLM

데이터 밀도 확인하기

```
SELECT * FROM pgstattuple('table_name');
```

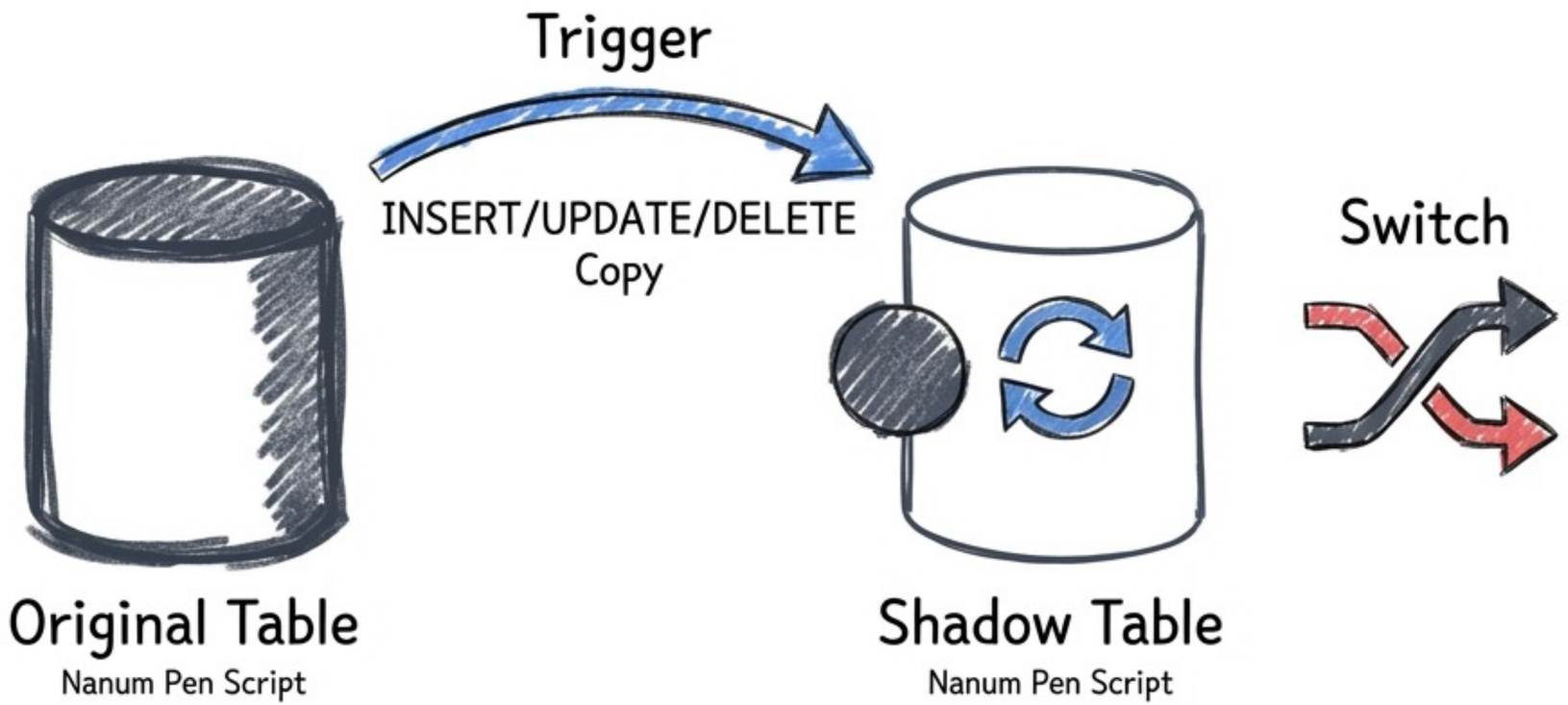


- tuple_percent: 유용한 데이터 비율 (낮으면 Bloat 심각)
- free_percent: 비어있는 공간 비율

Pro Tip: 큰 테이블은 pgstattuple_approx 함수 사용 권장

NotebookLM

중단 없는 재구성을 위해



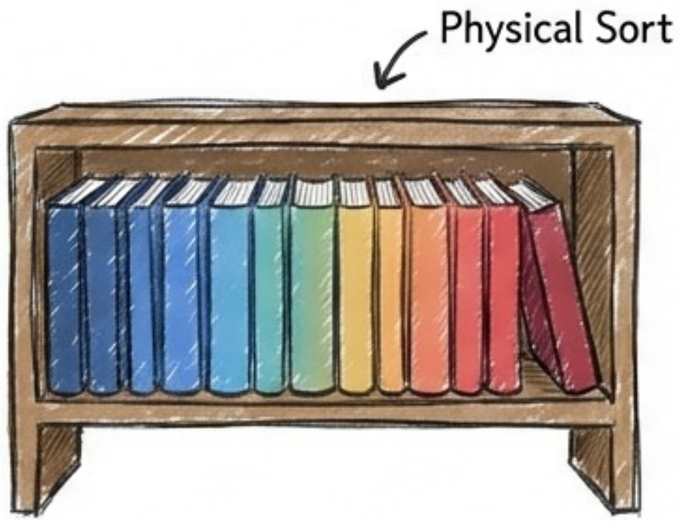
pg_repack: 트리거 + 섀도우 테이블 (짧은 잠금)

pgcompacttable: 데이터 앞쪽 이동 (잠금 최소화)

NotebookLM

그 외의 재구성 도구들

CLUSTER



Nanum Pen Script

인덱스 기준 물리적 정렬
(지속 안 됨)

REINDEX



Nanum Pen Script

인덱스만 새로 구축
(테이블 유지)

TRUNCATE



Nanum Pen Script

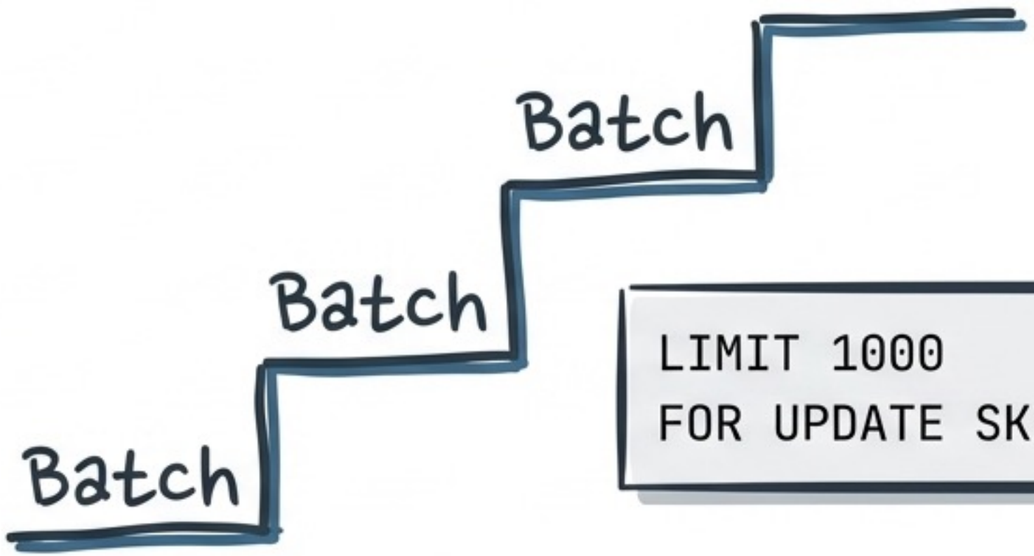
파일 삭제 후 재생성
(복구 불가, 매우 빠름)

NotebookLM

똑똑하게 데이터 수정하기



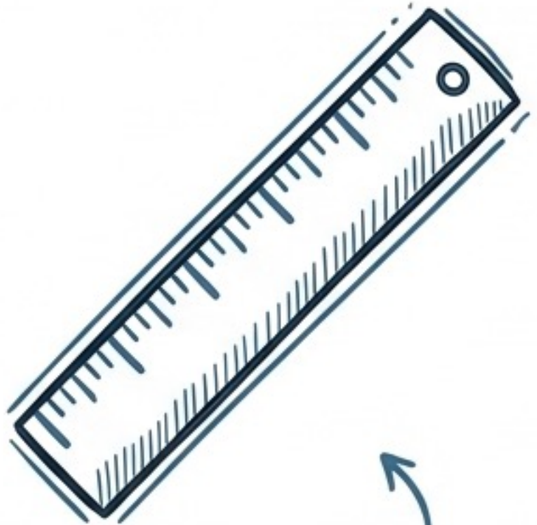
한 번의 트랜잭션 대신 배치를 나누어 처리하세요.
Autovacuum이 공간을 재사용할 시간을 줍니다.




```
LIMIT 1000
FOR UPDATE SKIP LOCKED;
```

NotebookLM

핵심 요약



진단 (Diagnosis)

pgstattuple로
정확한 밀도 확인



재건축 (Rebuild)

VACUUM FULL
(확실하지만 다운타임)



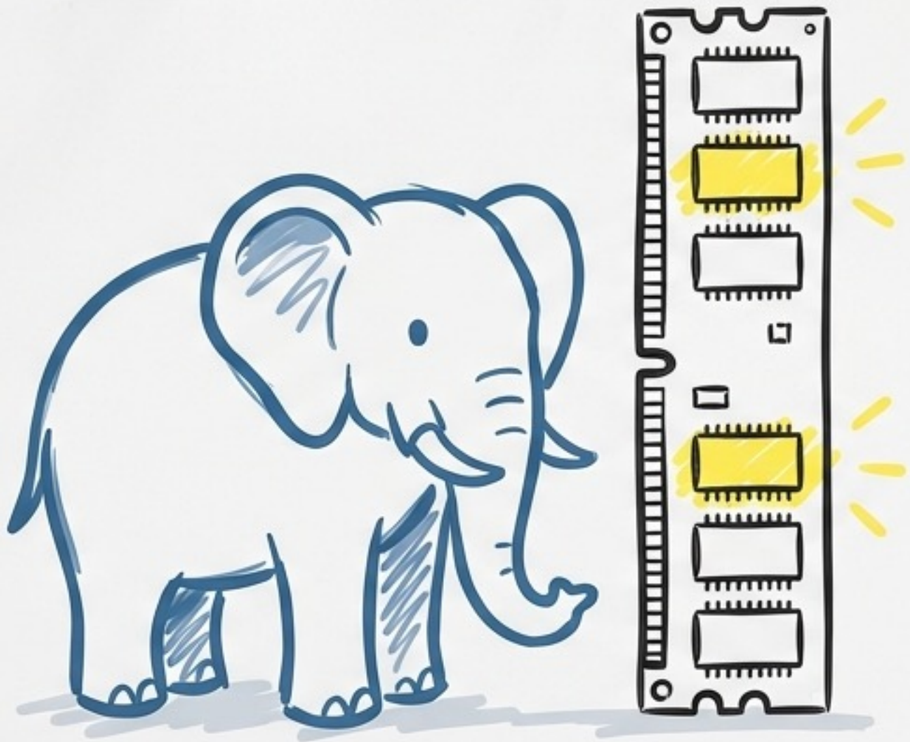
고가용성 (Uptime)

pg_repack 또는
배치 업데이트 활용

정기적인 관리와 정확한 도구 선택이 데이터베이스의 수명을 결정합니다.

NotebookLM

9. Buffer Cache

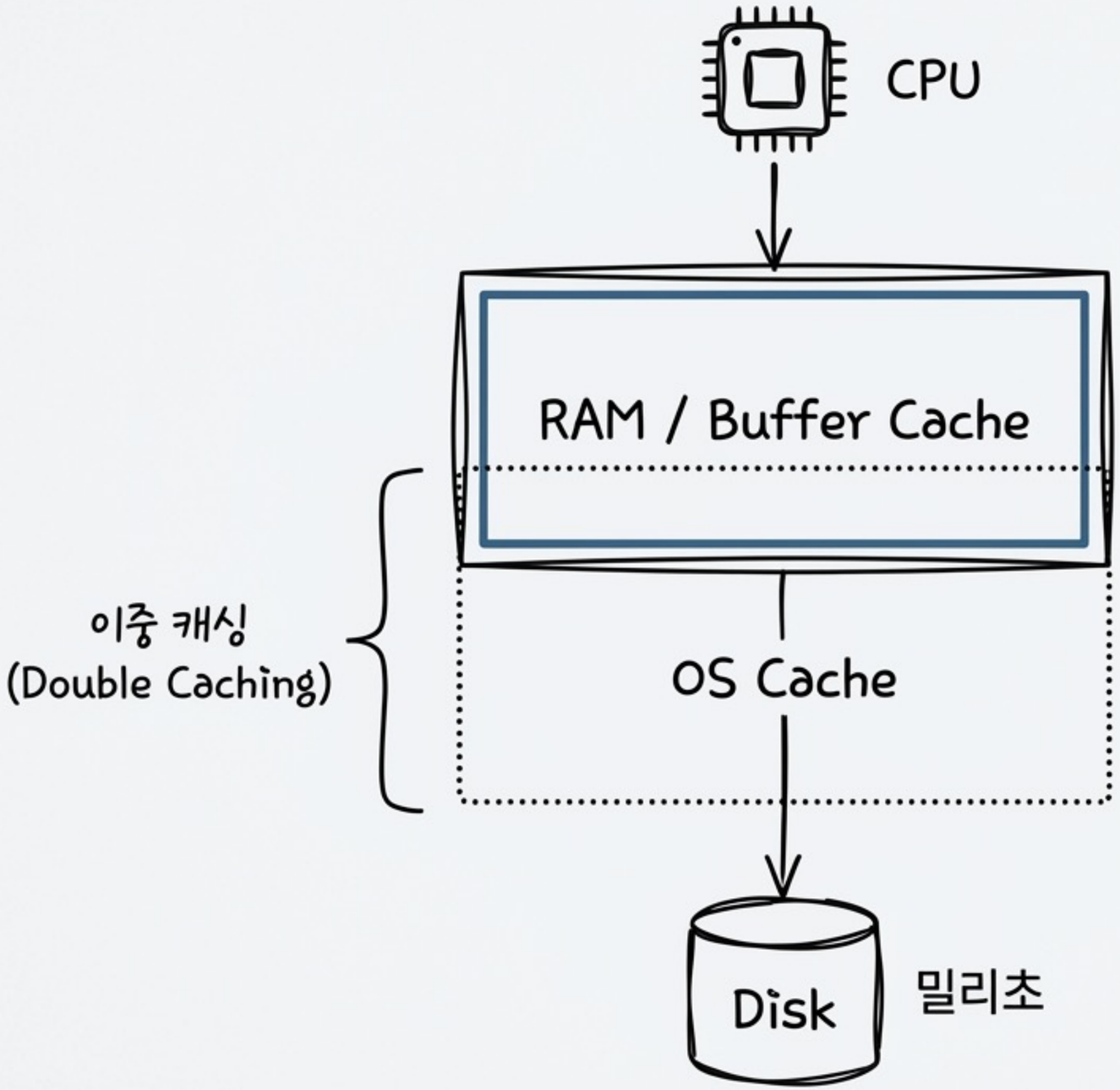


PostgreSQL 버퍼 캐시 완벽 가이드

PostgreSQL 버퍼 캐시 완벽 가이드

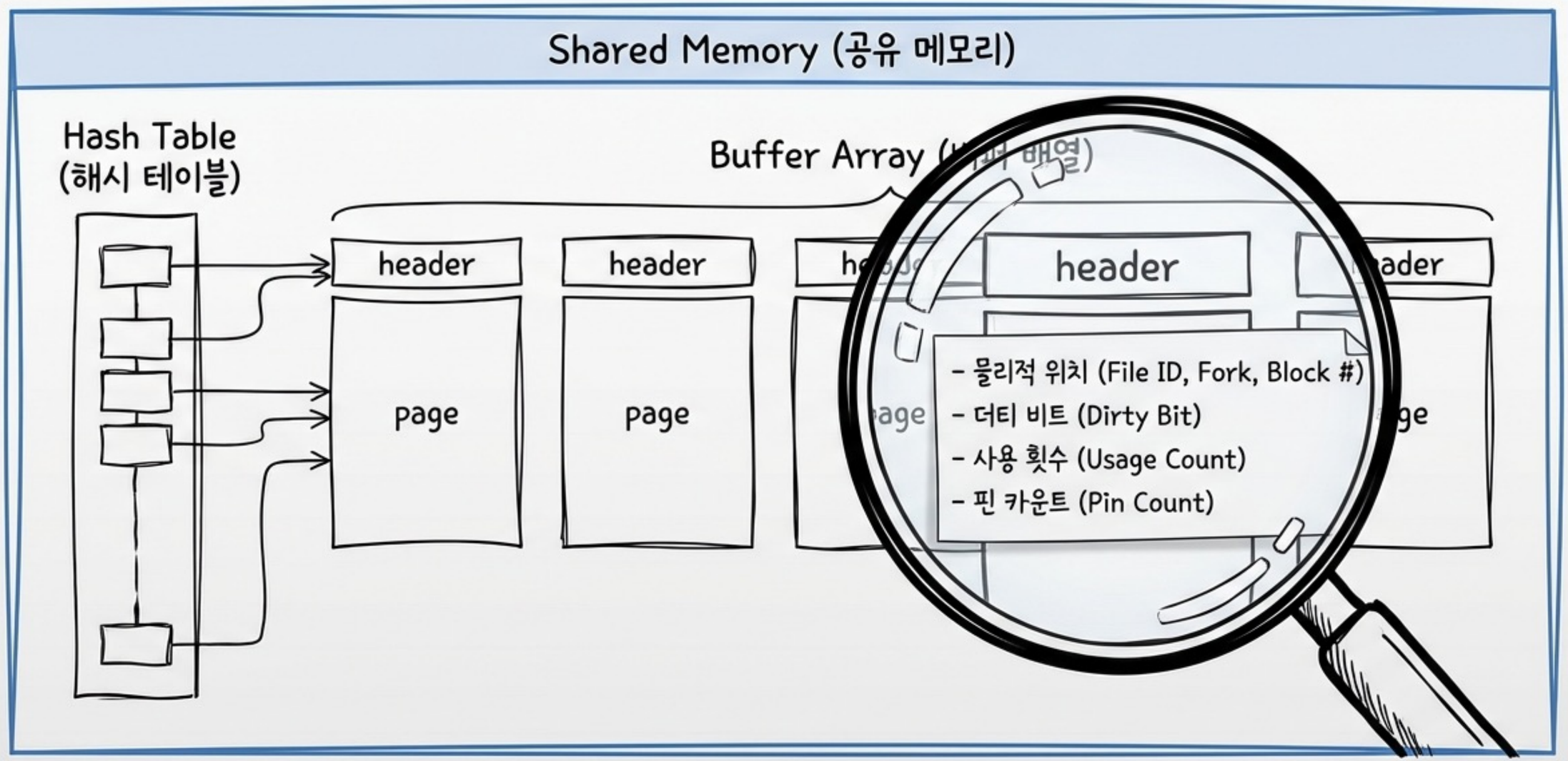
아키텍처부터 튜닝까지: 화이트보드 스타일 심층 분석

NotebookLM



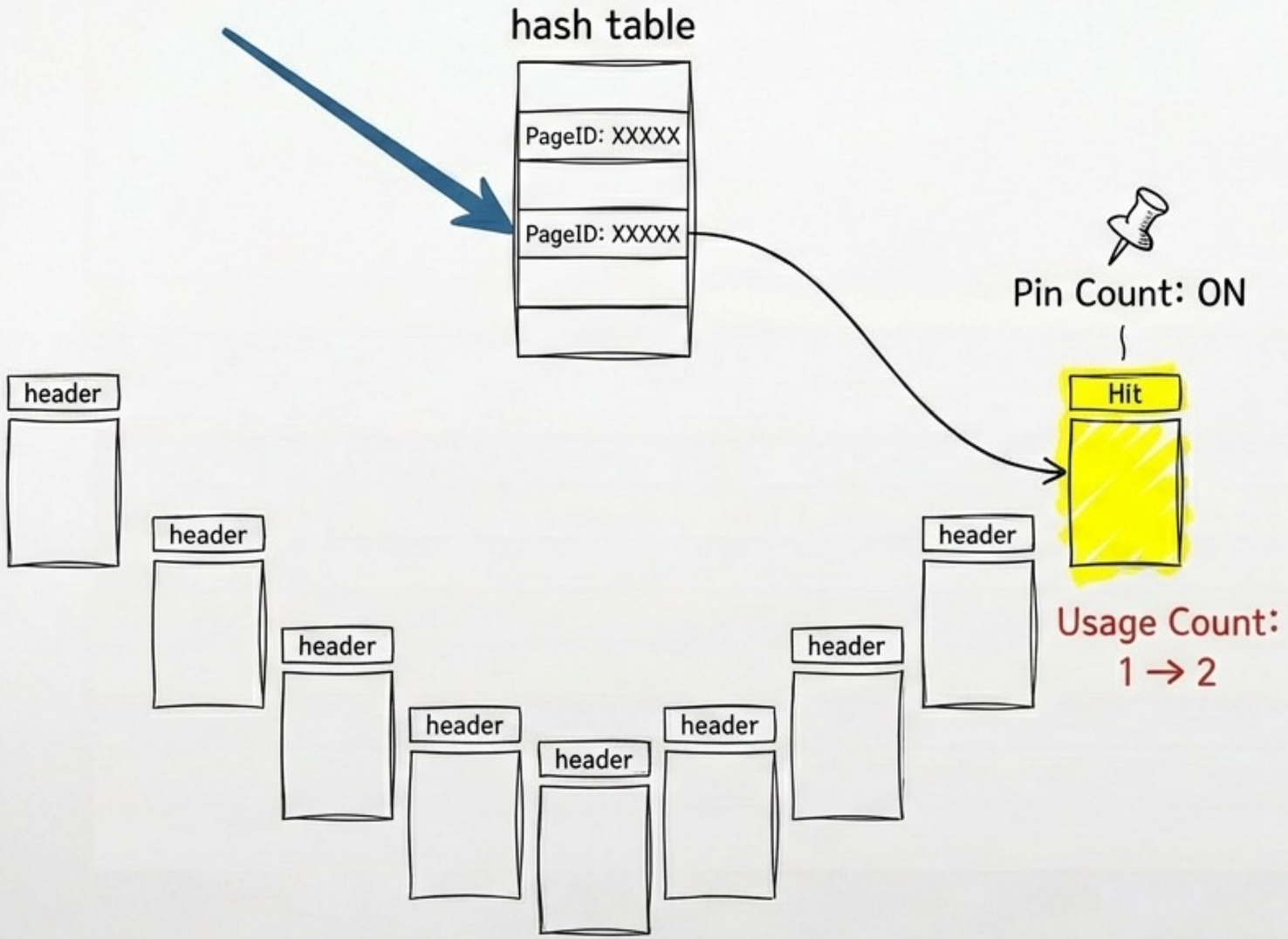
- 목적: 느린 디스크와 빠른 CPU 사이의 속도 차이 보완
- 특징: Buffered I/O 사용 (OS 캐시와 함께 작동)

NotebookLM



캐시는 단순한 저장소가 아닌, 정교한 파일 시스템입니다.

NotebookLM

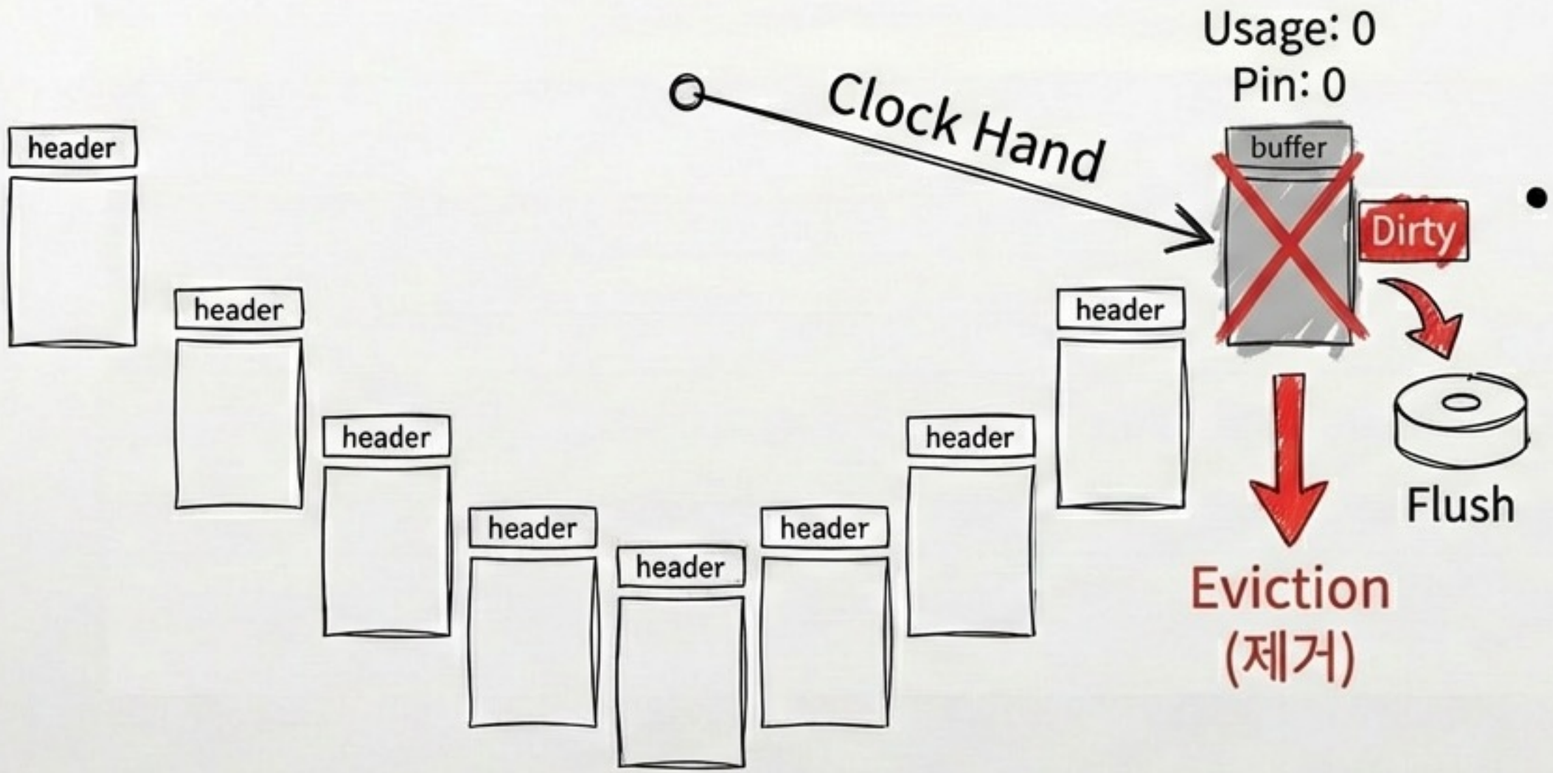


캐시 히트(Cache Hit): I/O 없이 데이터 접근

- 고정(Pinning): 사용 중인 페이지가 제거되지 않도록 보호
- 사용 횟수 증가: 자주 쓰는 데이터임을 표시

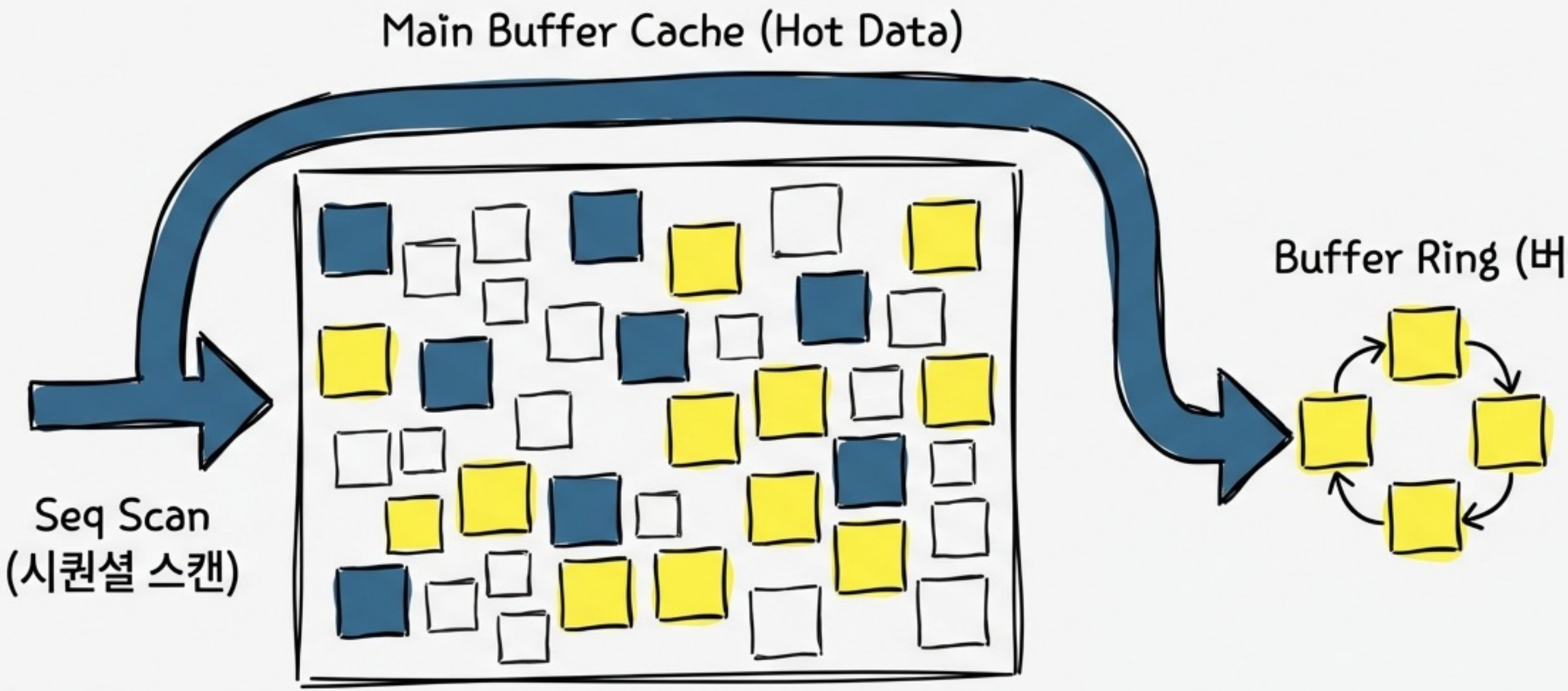
NotebookLM

클락 스윕 (Clock Sweep)



- 클락 스윕 (Clock Sweep): 사용 횟수를 확인하며 순회
- 제거 대상: 고정(Pin)되지 않고, 사용 횟수가 0인 버퍼

NotebookLM

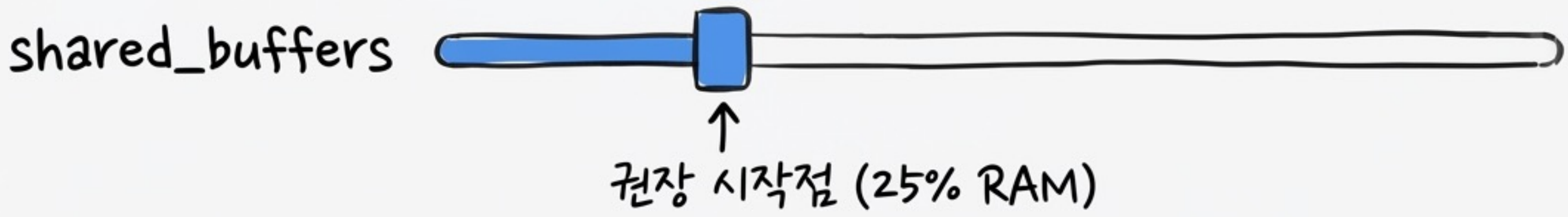


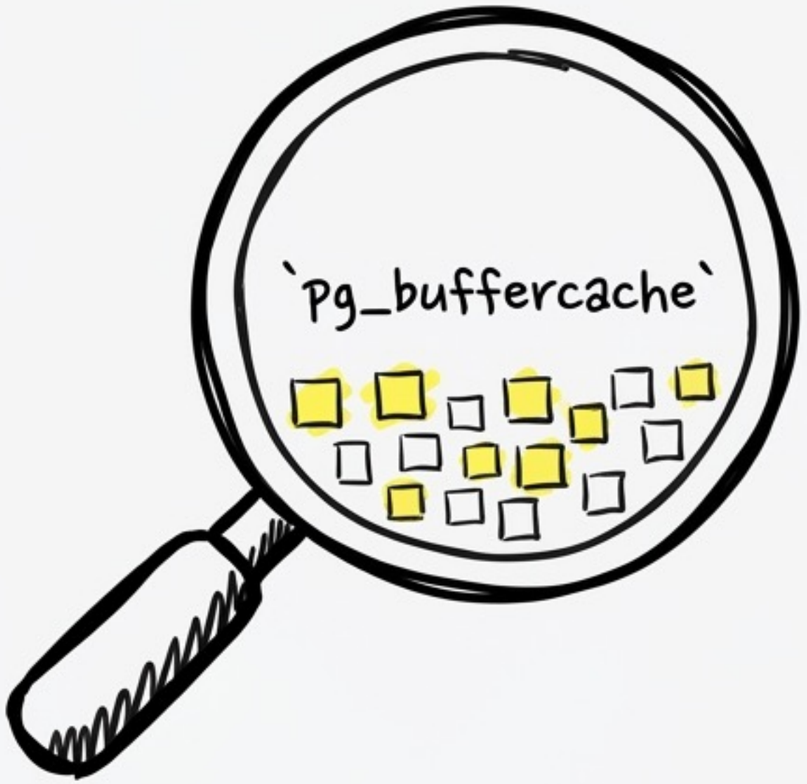
대량 읽기 전략: 256KB Ring size

대량 쓰기 전략: 16MB Ring size

Insight: 대량 작업(Bulk Operation)이 중요한 핫 데이터를 밀어내지 않도록 별도의 별도의 작은 링 버퍼를 사용합니다.

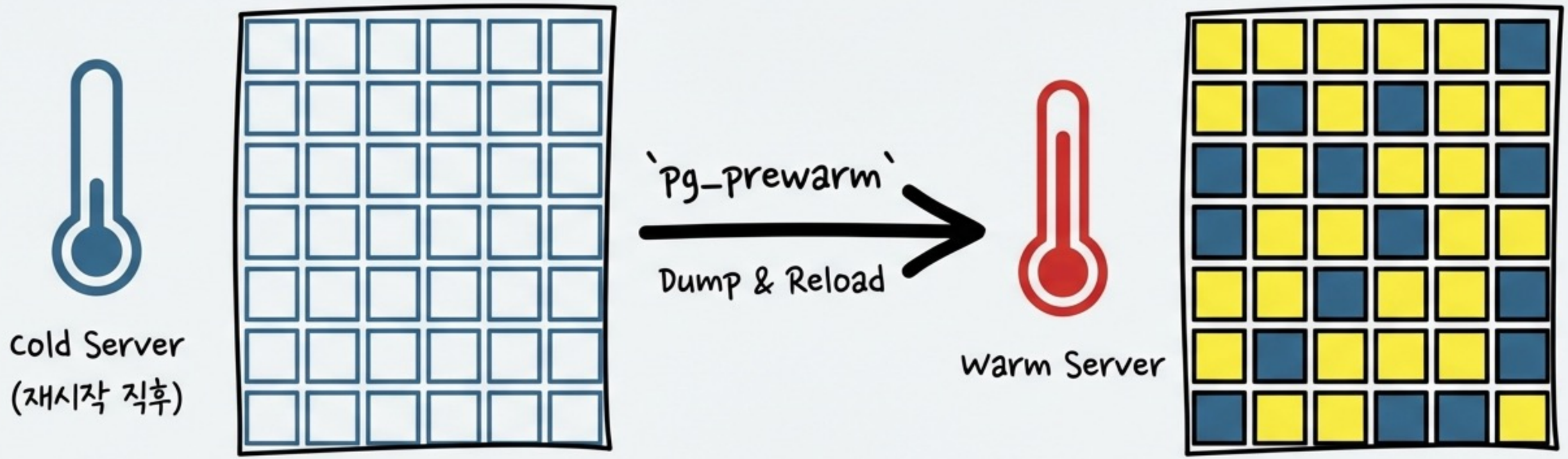
NotebookLM





- **크기 설정:** 일반적으로 RAM의 25% 권장
- **모니터링:** pg_buffercache 확장을 통해 버퍼 상태 실시간 확인
- **주의:** 너무 크면 OS 캐시 효율 저하 및 관리 비용(Double buffering) 증가

NotebookLM

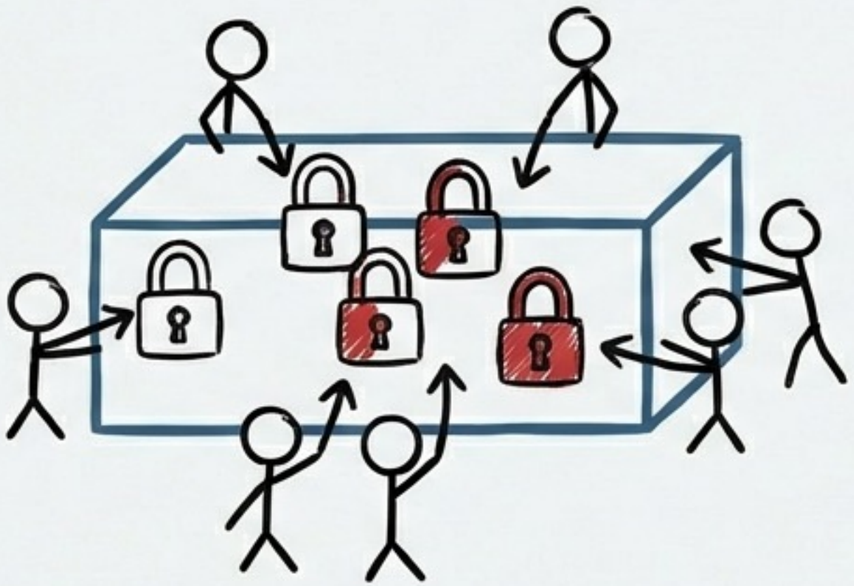


재시작 문제: 서버 재시작 시 캐시가 초기화됨

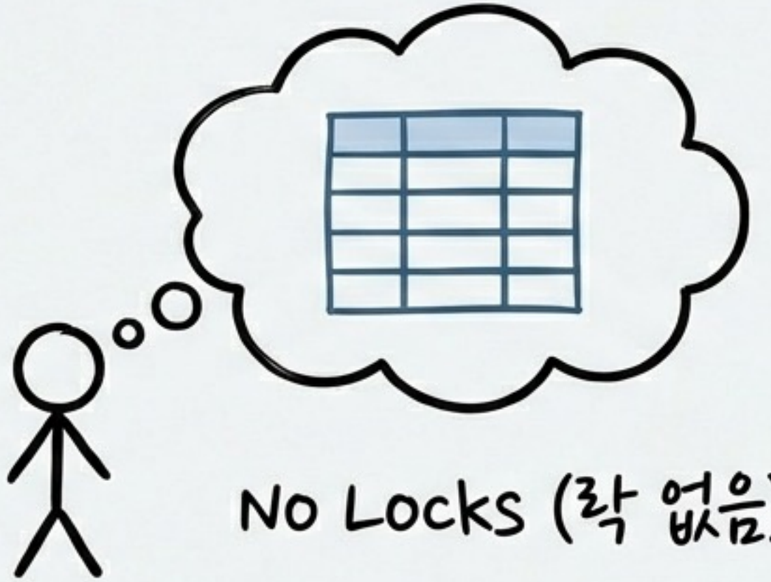
해결책: pg_prewarm 확장 기능
기능: 현재 캐시 상태를 디스크에 저장하고 재시작 시 자동 복원

NotebookLM

Shared Buffers (공유 버퍼)



Local Cache (로컬 캐시)



- 용도: 임시 테이블 (Temporary Tables) 전용
- 장점: 락(Lock)이 없어 빠르고 단순함
- 설정: temp_buffers (기본값 8MB)

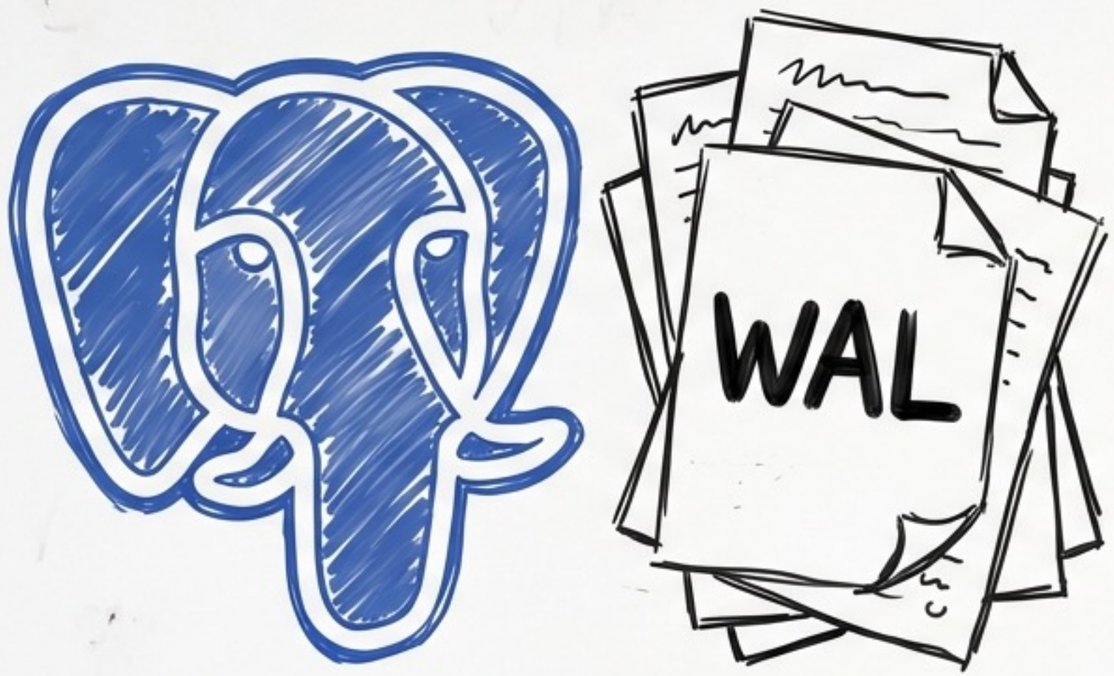
NotebookLM

- [x] 1. 구조 (Structure): 버퍼 배열 + 해시 테이블
- [x] 2. 동작 (Mechanism): Clock Sweep 알고리즘
- [x] 3. 보호 (Protection): 버퍼 링(Buffer Ring) 격리
- [x] 4. 관리 (Management): shared_buffers 및 pg_prewarm

Noto Sans KR 버퍼 캐시의 작동 원리를 이해하는 것이 PostgreSQL I/O 성능 최적화의 핵심입니다.

NotebookLM

- **10. Write-Ahead Log**



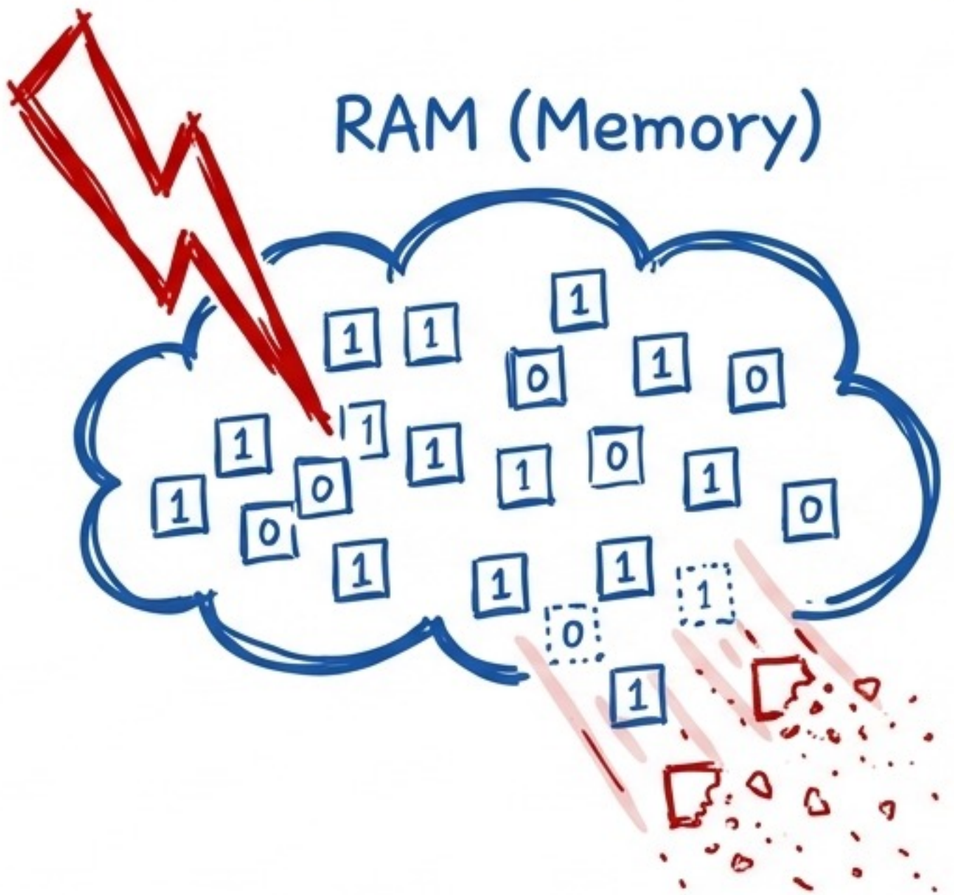
PostgreSQL 내부 구조: WAL과 복구

내구성과 성능의 균형을 맞추는 핵심 메커니즘

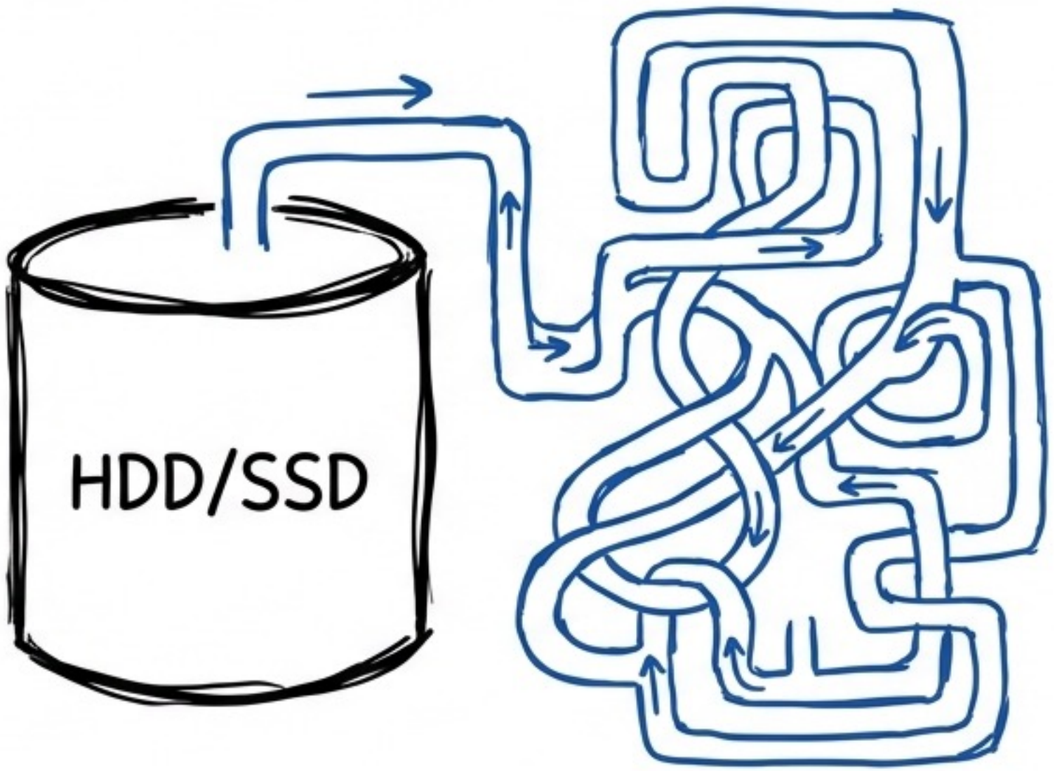
Whiteboard Series

NotebookLM

램(RAM)의 휘발성과 디스크의 딜레마



휘발성 (Volatile)

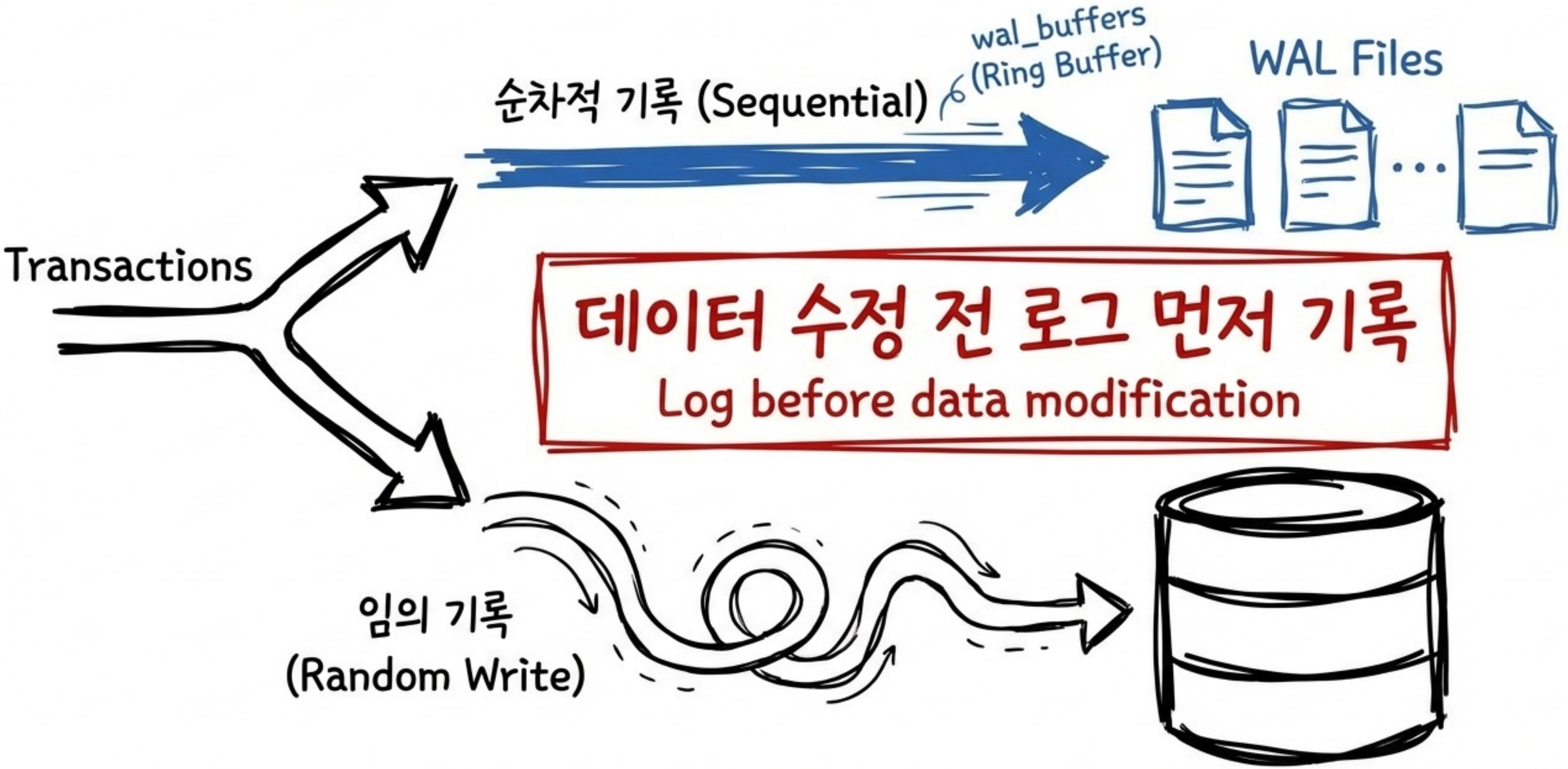


영구적이지만 느림 (Persistent but Slow)

핵심 목표: 성능을 유지하면서 데이터의 일관성(Consistency) 보장

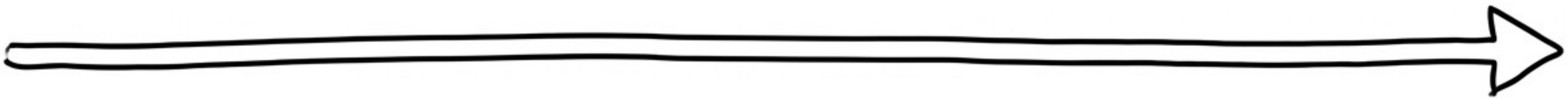
NotebookLM

해결책: 미리 쓰기 로그 (WAL - Write-Ahead Logging)



NotebookLM

LSN: 로그의 좌표 시스템 (Log Sequence Number)



timeline log sequence number

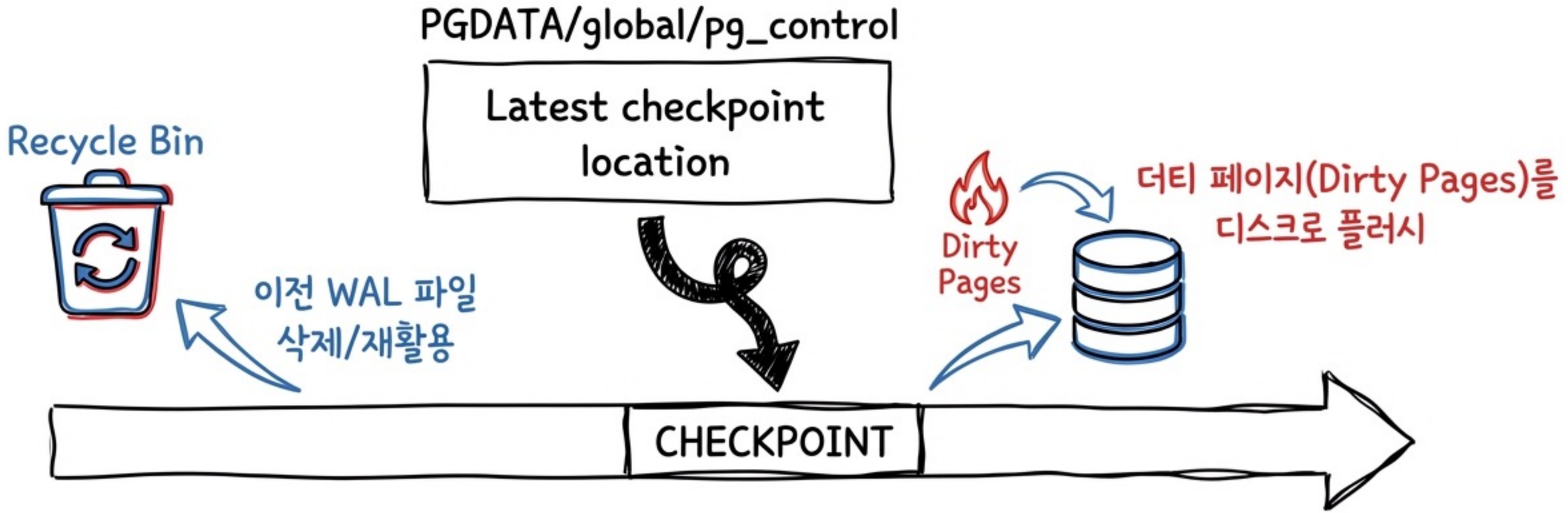
0/3DF708D8

pg_current_wal_lsn()

- pg_lsn: 64비트 바이트 오프셋
- 물리적 저장: 16MB 세그먼트 파일
- 파일명: 타임라인 ID + LSN 오프셋

NotebookLM

체크포인트 (Checkpoint): 복구 시간의 단축

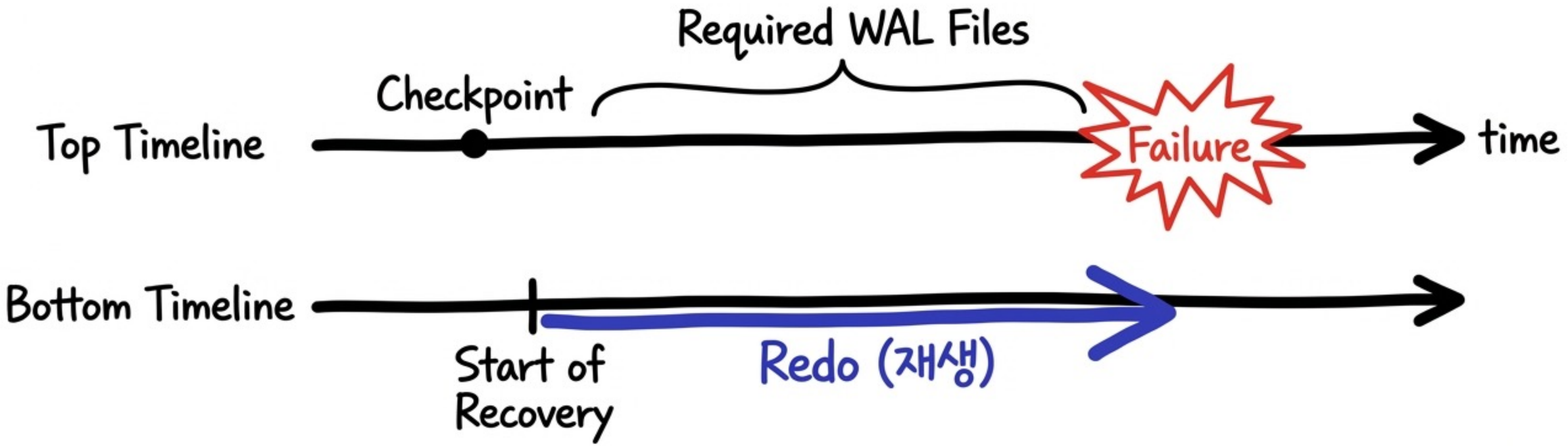


- 더티 페이지(Dirty Pages)를 디스크로 플러시
- pg_control 파일 업데이트
- 이전 WAL 파일 삭제/재활용

NotebookLM

복구 (Recovery): 사고 시점부터 다시 실행

- 자동 복구: 서버 시작 시 pg_control 확인
- Redo: 마지막 체크포인트부터 로그 재실행
- No Undo: PostgreSQL은 Undo 단계가 필요 없음



NotebookLM

Nanum Pen Script

백그라운드 프로세스 역할 분담

Backend



트랜잭션 ㅂ
&
WAL 기록

Checkpointer



주기적 대규모 플러시
(Flush)

Bgwriter



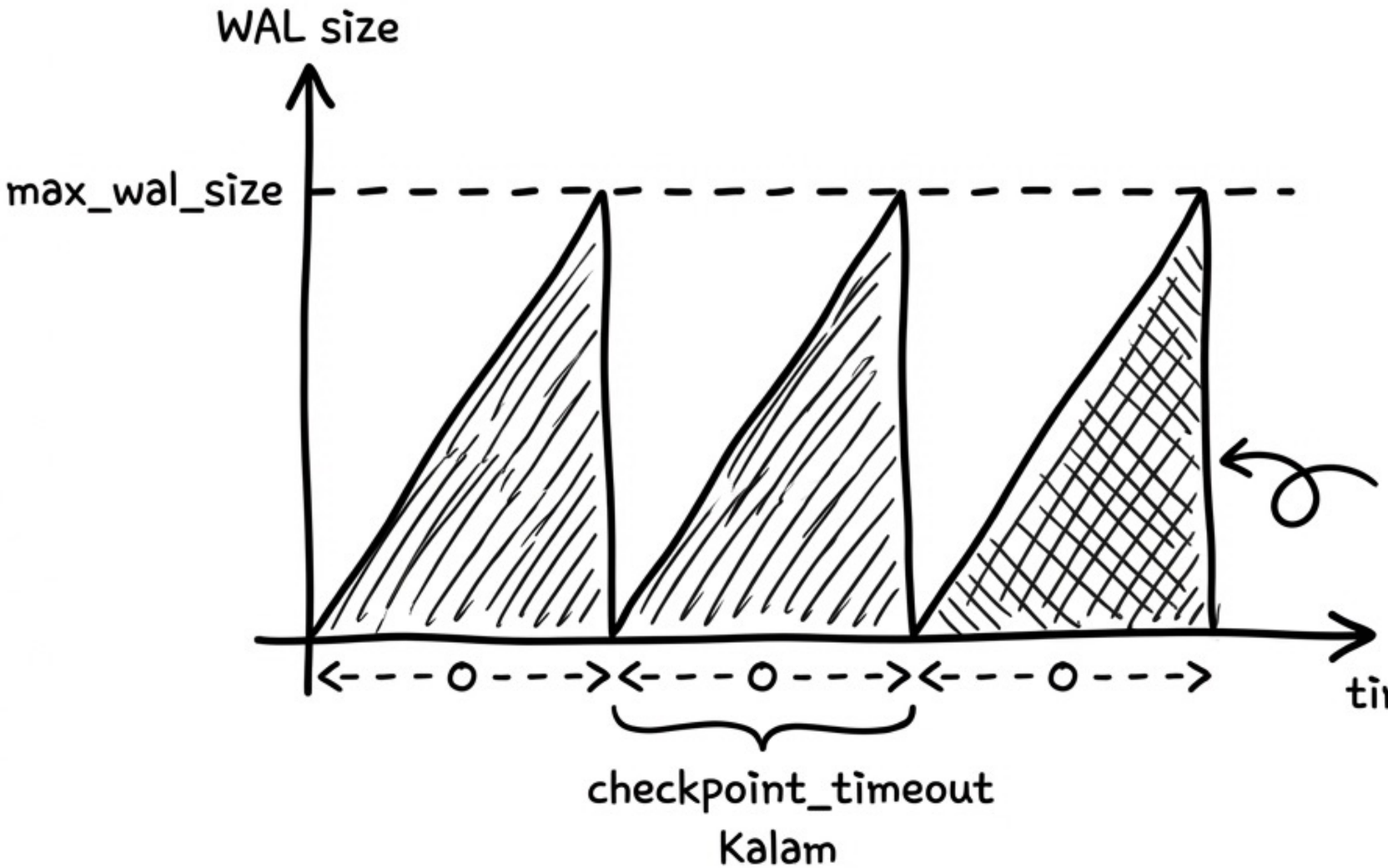
I/O 스파이크 방지
(미리 조금씩 기록)

NotebookLM

Nanum Pen Script

Permanent Marker

WAL 크기 관리와 튜닝 (Tuning WAL Size)



max_wal_size: Soft Limit
(강제 체크포인트 트리거)

checkpoint_completion_target:
I/O 분산 (기본값 0.9)

min_wal_size: 재활용을 위한 최소 공간

the size of WAL generated between the starts of two checkpoints Kalam

NotebookLM

핵심 요약 (Key Takeaways)

- 안전성: WAL은 데이터 수정 전에 기록되어 내구성(Durability)을 보장.
- 구조: LSN은 로그의 물리적 위치이자 복구의 기준점.
- 성능: 체크포인트는 복구 시간을 줄이고 로그 공간을 확보.
- 최적화: max_wal_size와 bgwriter 설정으로 I/O 부하 분산.

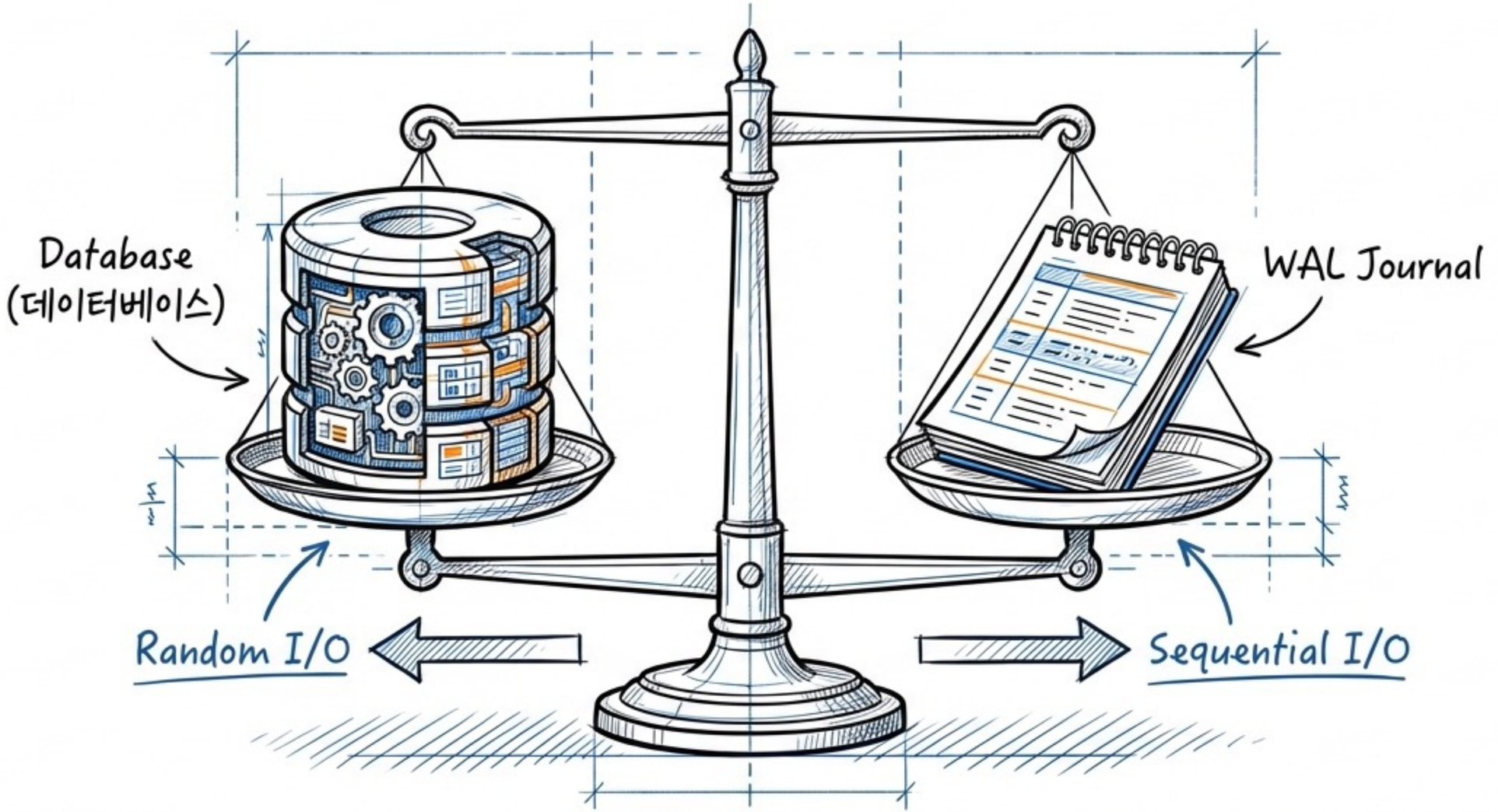


NotebookLM

- **11. WAL Modes**

PostgreSQL WAL 모드: 속도와 안정성의 균형

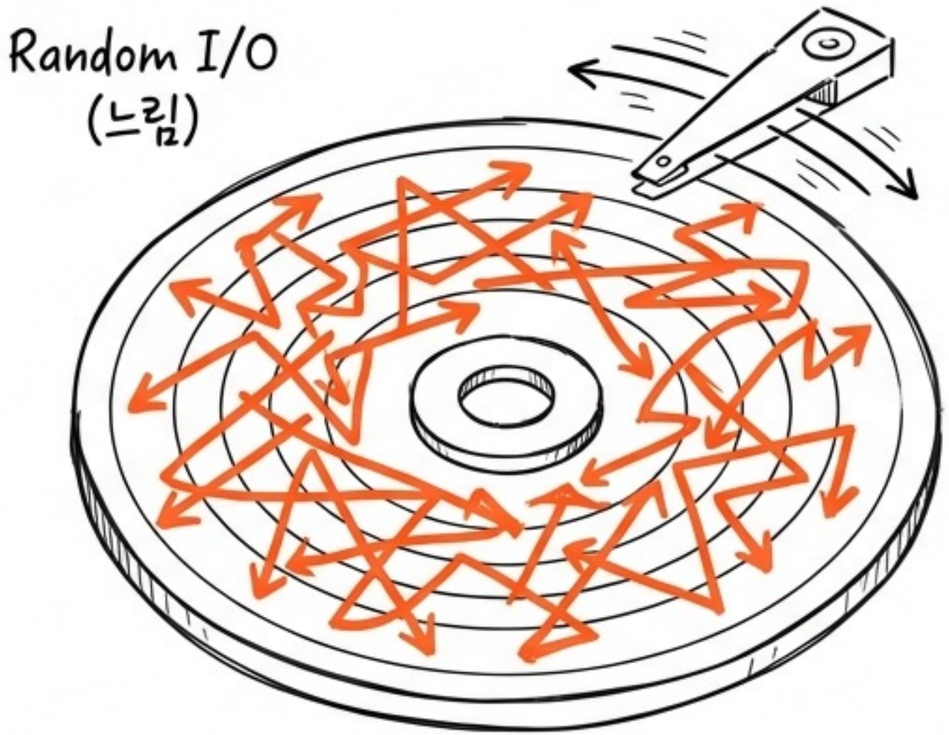
데이터 무결성을 지키며 성능을 최적화하는 핵심 메커니즘



NotebookLM

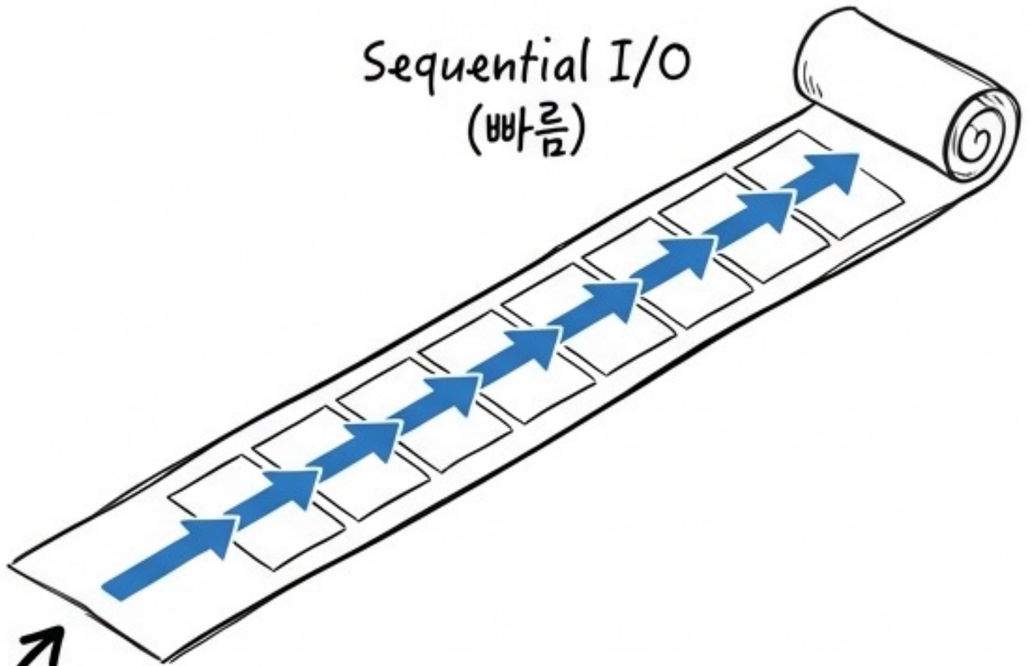
무작위 접근 vs. 순차적 기록

The Problem



데이터 파일 직접 쓰기 =
헤드 이동이 많은 Random I/O

The Solution

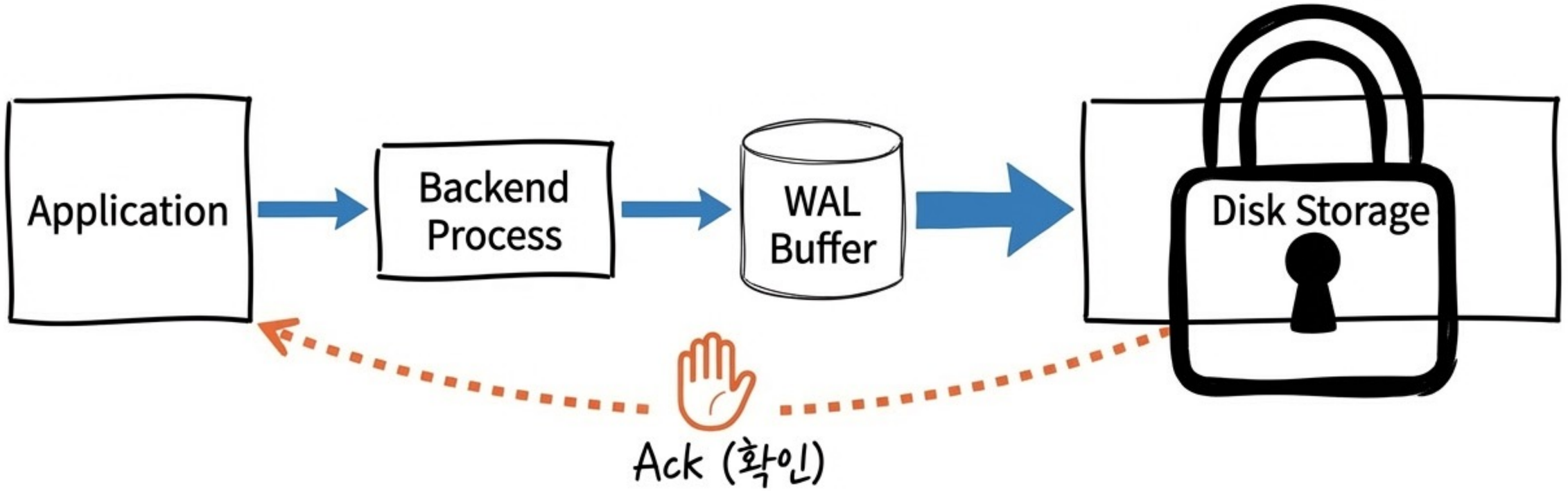


WAL 기록 =
순차적으로 추가하는 Sequential I/O

WAL: 물리적 디스크의 한계를 극복하는 버퍼

NotebookLM

동기 모드 (Synchronous Commit) - 기본값



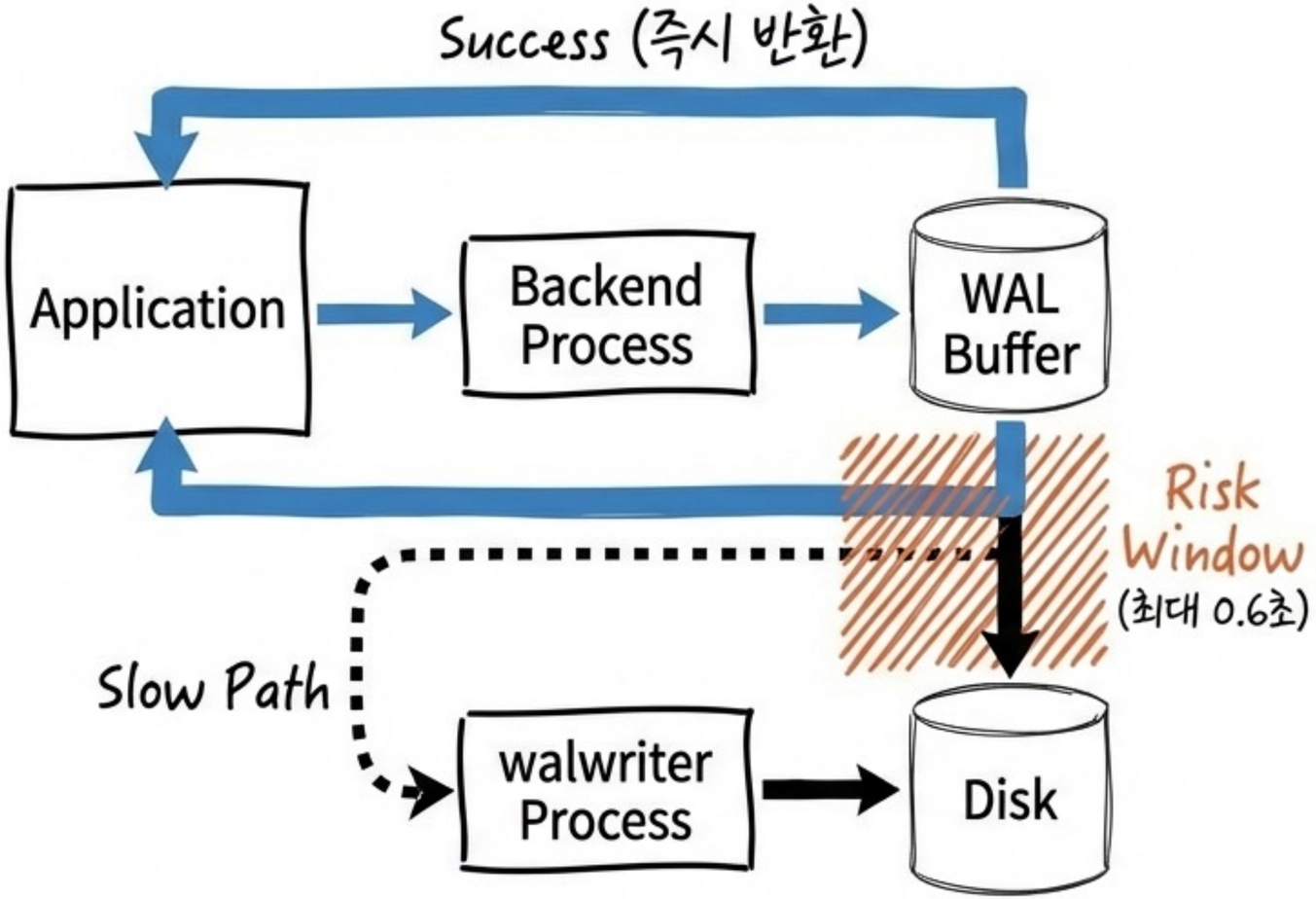
Commit means Commit: 트랜잭션 완료 = 디스크 기록 보장

안정성 최우선 (ACID Durability)

단점: 디스크 I/O 대기 시간(Latency) 발생

NotebookLM

비동기 모드 (Asynchronous Commit)

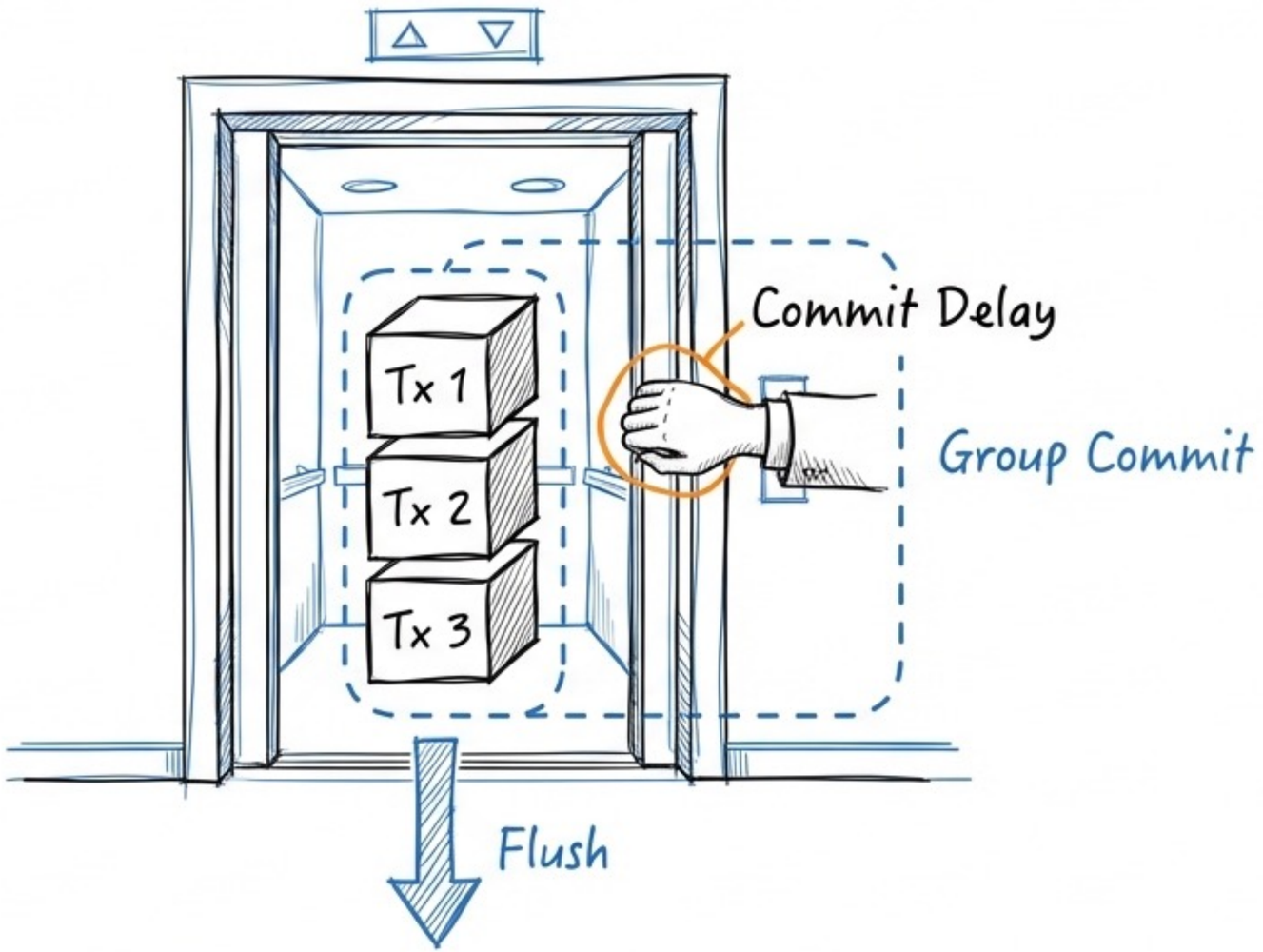


- synchronous_commit = off
- 속도 향상: 디스크 기록을 기다리지 않음
- 리스크: 충돌 시 walwriter 지연 시간만큼 데이터 손실
- 최대 손실: 3 × wal_writer_delay

NotebookLM

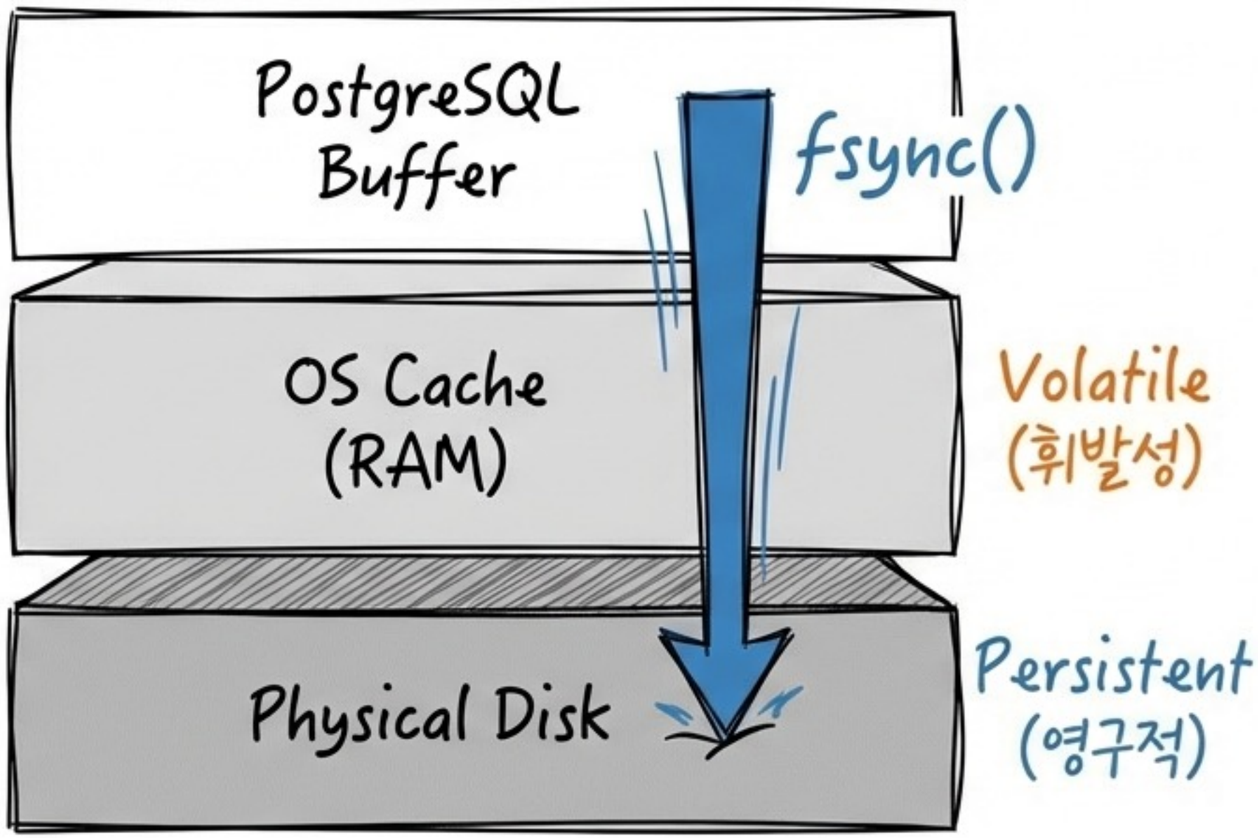
그룹 커밋 (Group Commit) 효과

- 엘리베이터 효과: 잠시 대기하여 여러 트랜잭션을 태움
- 한 번의 I/O Flush로 여러 작업을 처리
- 고부하 시스템에서 처리량(TPS) 급증



NotebookLM

장애 허용성: fsync의 역할

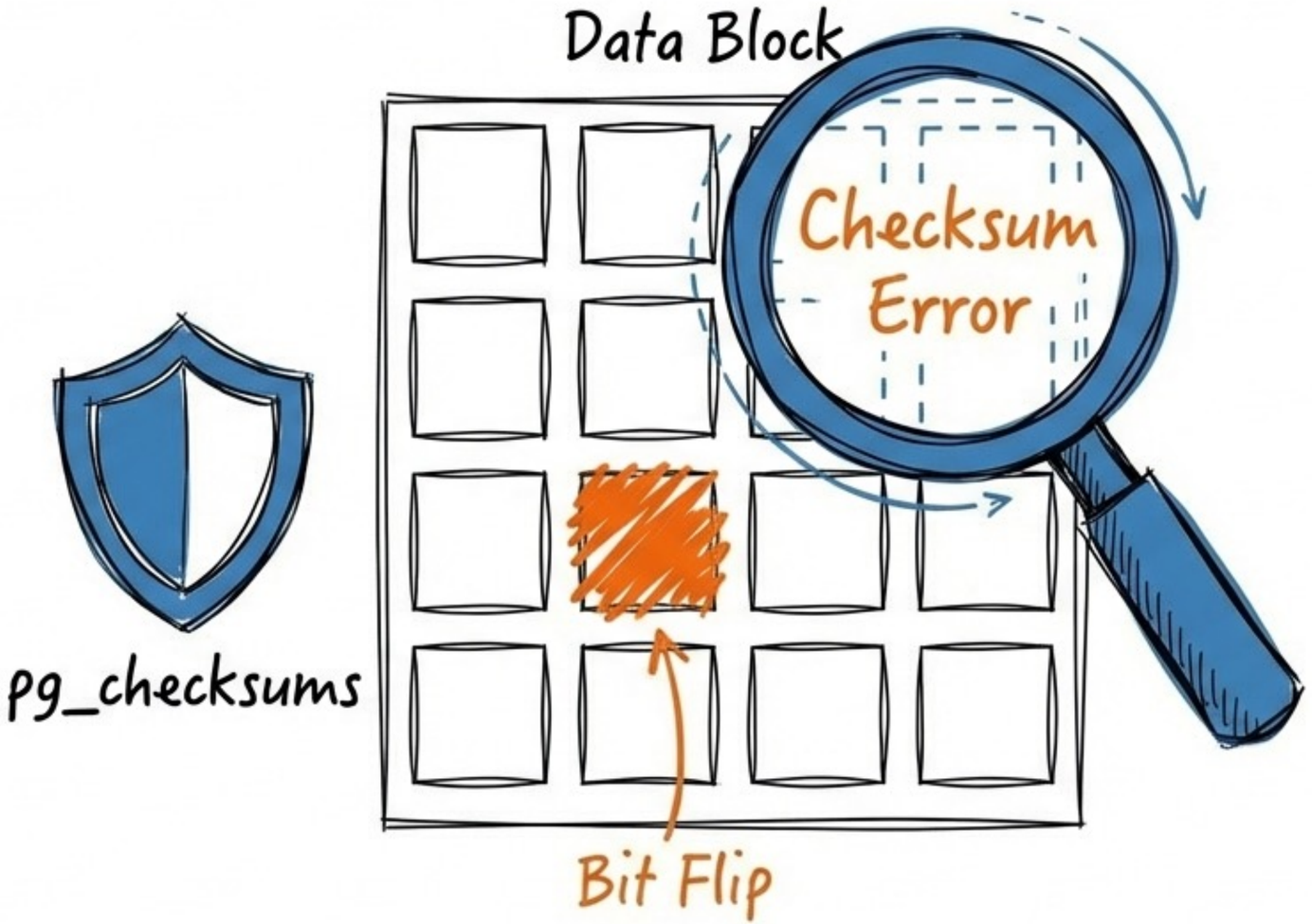


- OS 캐시에만 기록된 데이터는 전원 차단 시 증발함
- fsync: 데이터를 물리적 저장소까지 강제로 밀어넣는 명령
- fsync = off 설정은 데이터 망실 위험 매우 큼 (절대 주의)

NotebookLM

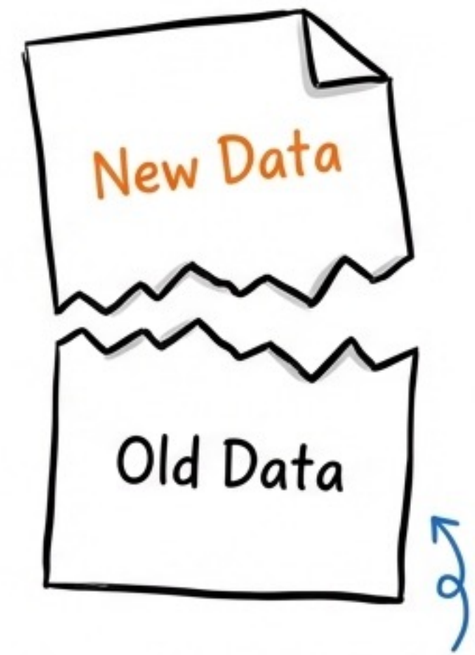
데이터 무결성 보호 (Checksums)

- 하드웨어 결함이나 비트 플립(Bit flip) 감지
- 성능 비용(CPU)이 들지만, 데이터 오염을 발견하는 유일한 수단
- 클러스터 초기화 시 활성화 권장



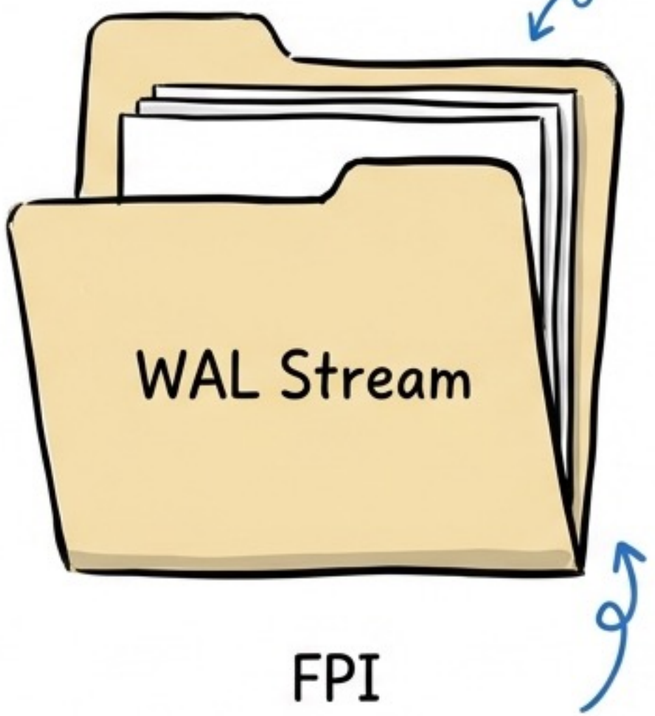
NotebookLM

부분 쓰기 방지와 FPI



Non-atomic Write
(비원자적 쓰기)

충돌 발생 시



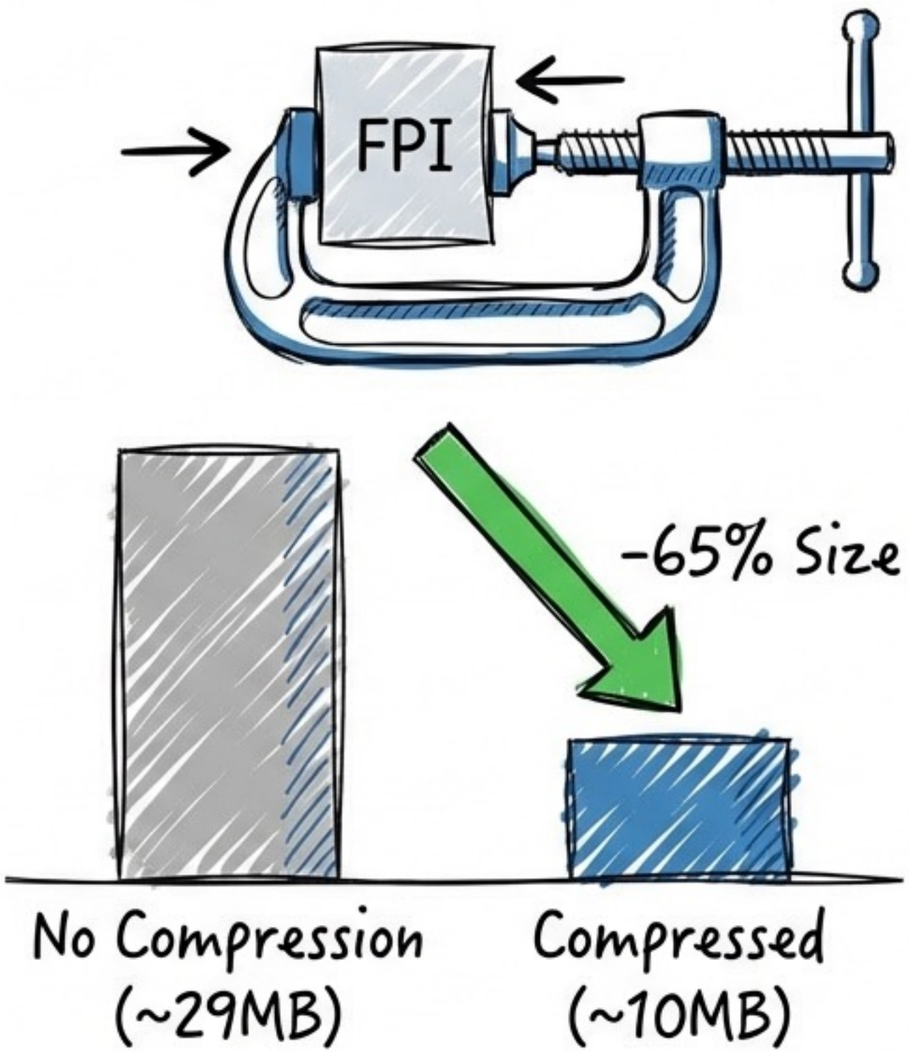
FPI
(Full Page Image)

- 체크포인트 이후 첫 변경 시, 페이지 전체 이미지(FPI)를 WAL에 기록
- 충돌 발생 시 FPI로 페이지를 복구한 후 변경사항 적용
- Trade-off: 안정성 확보 vs WAL 사이즈 증가

NotebookLM

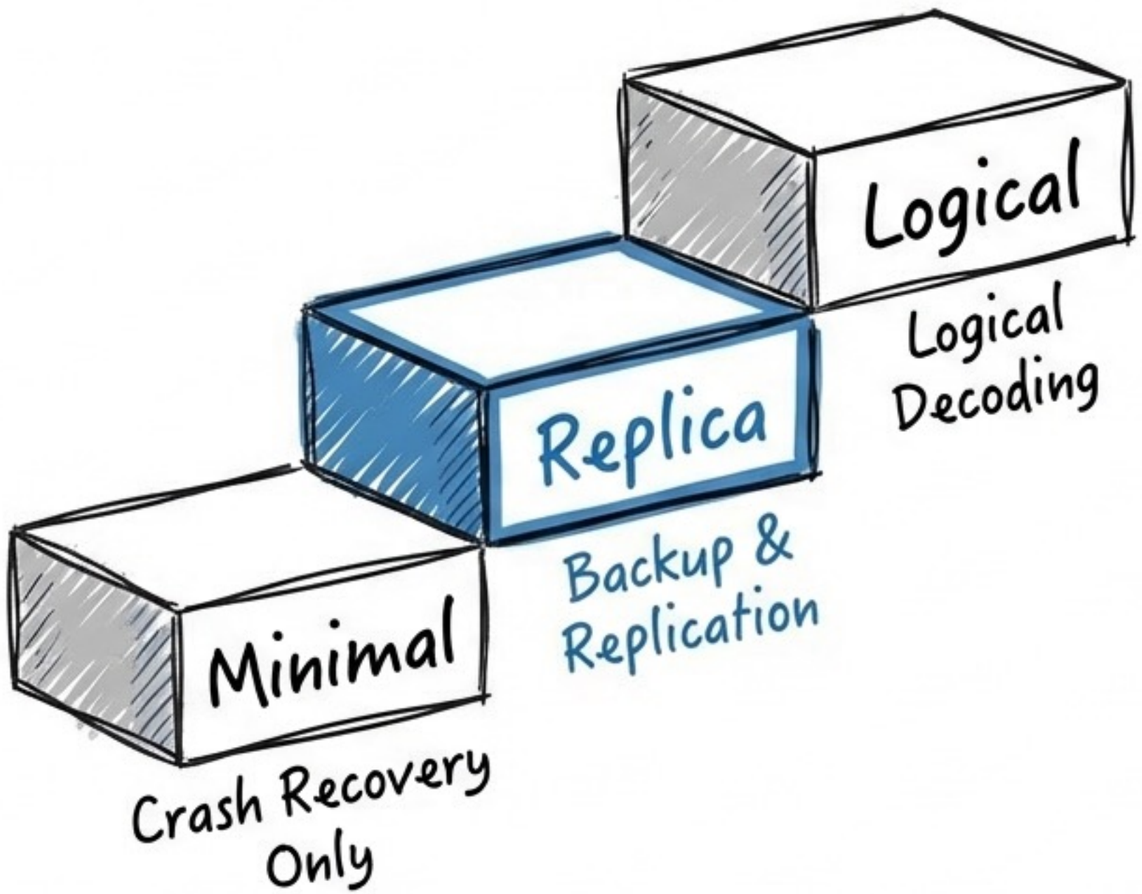
WAL 압축 (Compression)

- wal_compression = on
- CPU 사용량은 미미하게 증가, WAL 크기는 획기적으로 감소
- 디스크 공간 절약 및 복제 트래픽 감소 효과



NotebookLM

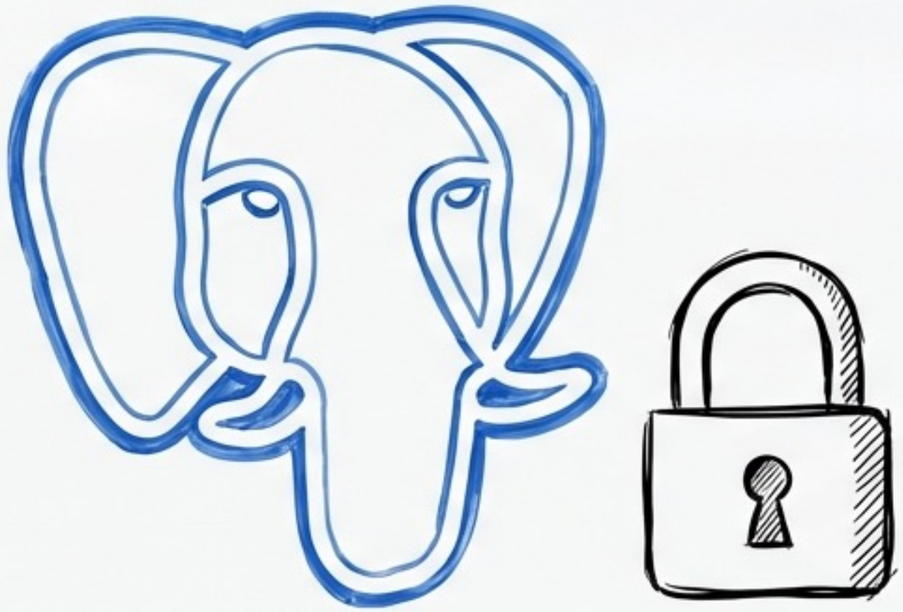
WAL 레벨 (WAL Levels)



- Logical: 외부 시스템 동기화를 위한 논리적 디코딩 정보 포함
- Replica (기본값): PITR(시점 복구) 및 스트리밍 복제 지원
- Minimal: 대량 작업 시 빠름, 아카이브 복구 불가능

NotebookLM

- **12. Relation-Level Locks**



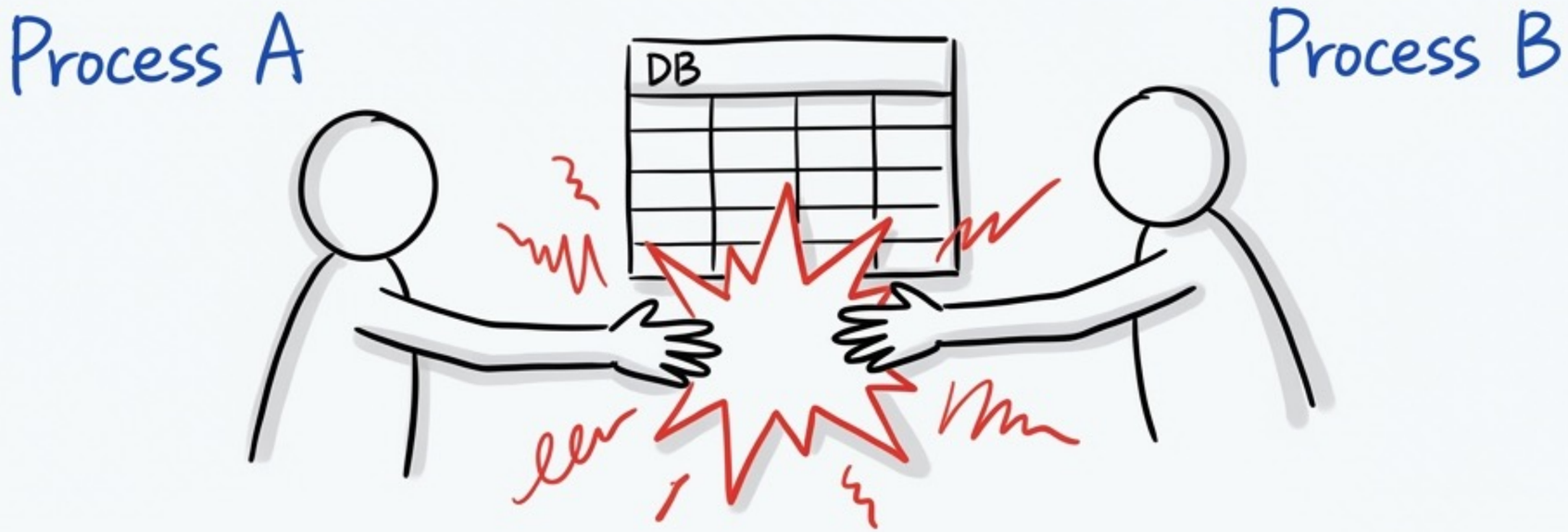
PostgreSQL 잠금 메커니즘

Chapter 12: 테이블 수준의 잠금

엔지니어링 딥다이브 세션

NotebookLM

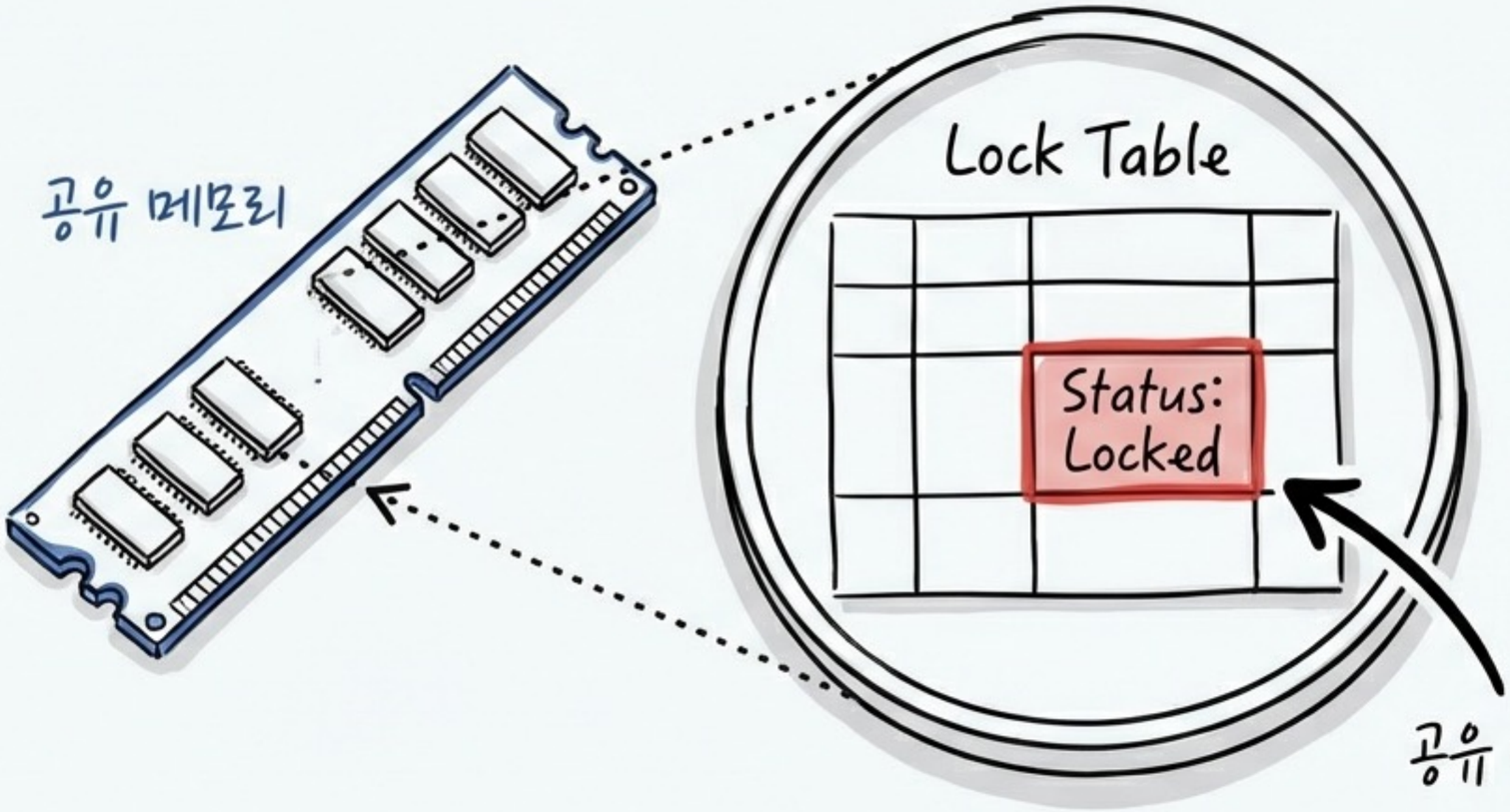
동시 액세스 제어



1. 동시 액세스: 여러 프로세스가 동시에 동일한 자원을 점유하려는 시도
2. 잠금이 없다면 데이터 무결성이 깨집니다
3. 잠금은 '신호등' 역할을 수행합니다

NotebookLM

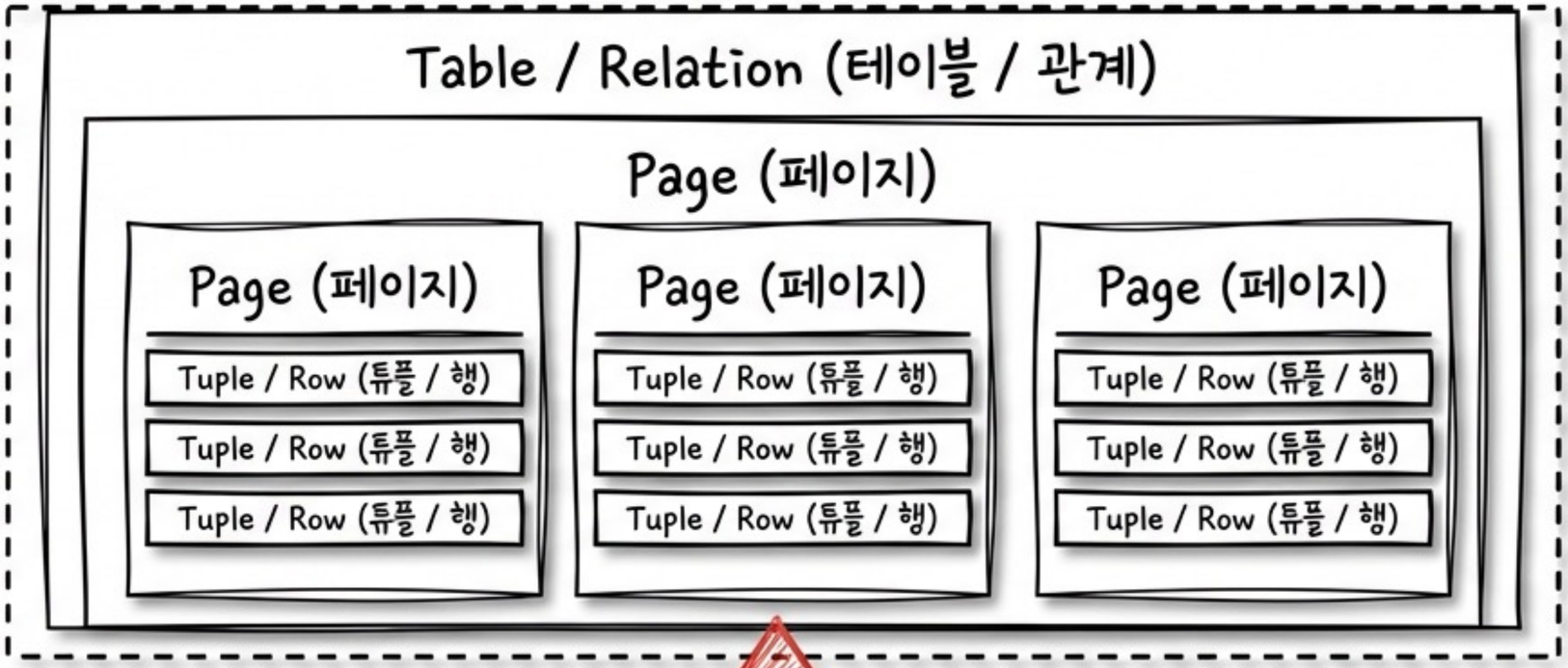
잠금의 물리적 실체



- 잠금은 단순히 상태를 정의하는 공유 메모리의 데이터입니다.
- 프로세스 번호(PID)나 획득 시간 정보를 포함합니다.
- 상태: 획득됨 / 획득되지 않음

NotebookLM

적정성 (Granularity): 테이블 vs 행



테이블 수준 (Coarse-grained)
관리가 쉽지만 동시성이 낮음.

행 수준 (Fine-grained)
동시성은 높지만 잠금 관리 비용이 증가.

잠금 승격 (Lock Escalation): 너무 많은 미세 잠금은 단일 잠금으로 대체될 수 있음.

NotebookLM

중량적 잠금 (Heavyweight Locks)



- Relation (관계 - 테이블, 인덱스)
- Page (페이지)
- Tuple (튜플)
- Transaction ID (트랜잭션 ID)

장기적 잠금: 트랜잭션이 끝날 때까지 유지됩니다.
가시성: pg_locks 뷰에서 확인 가능.
자동 관리: PostgreSQL이 자동으로 획득하고 해제합니다.

NotebookLM

트랜잭션 ID 잠금 (Transaction ID Lock)



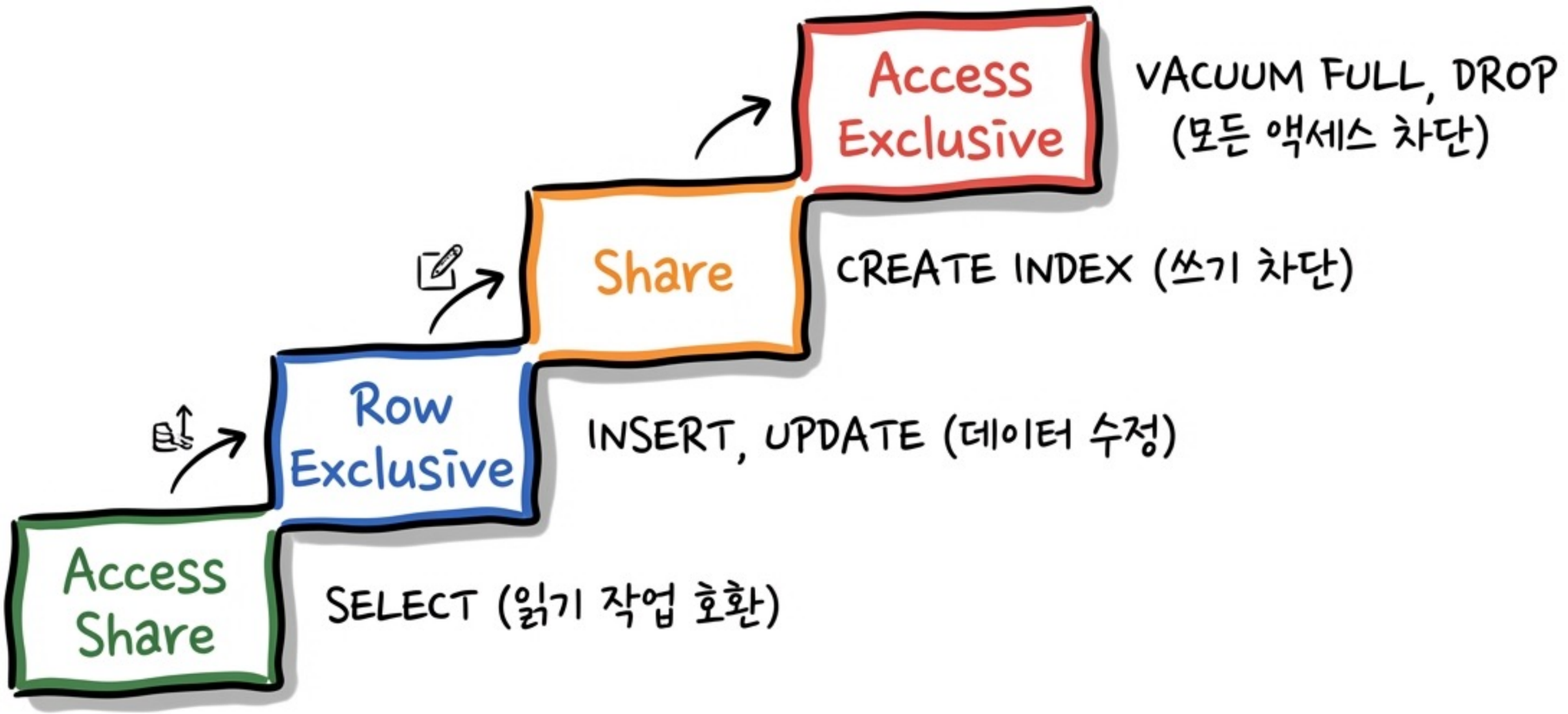
모든 트랜잭션은 자체 ID에 대해 배타적 잠금(Exclusive Lock)을 보유합니다.

다른 프로세스가 이 ID를 확인하려고 하면, 트랜잭션이 끝날 때까지 대기해야 합니다.

목적: 트랜잭션 완료 여부를 추적하기 위함.

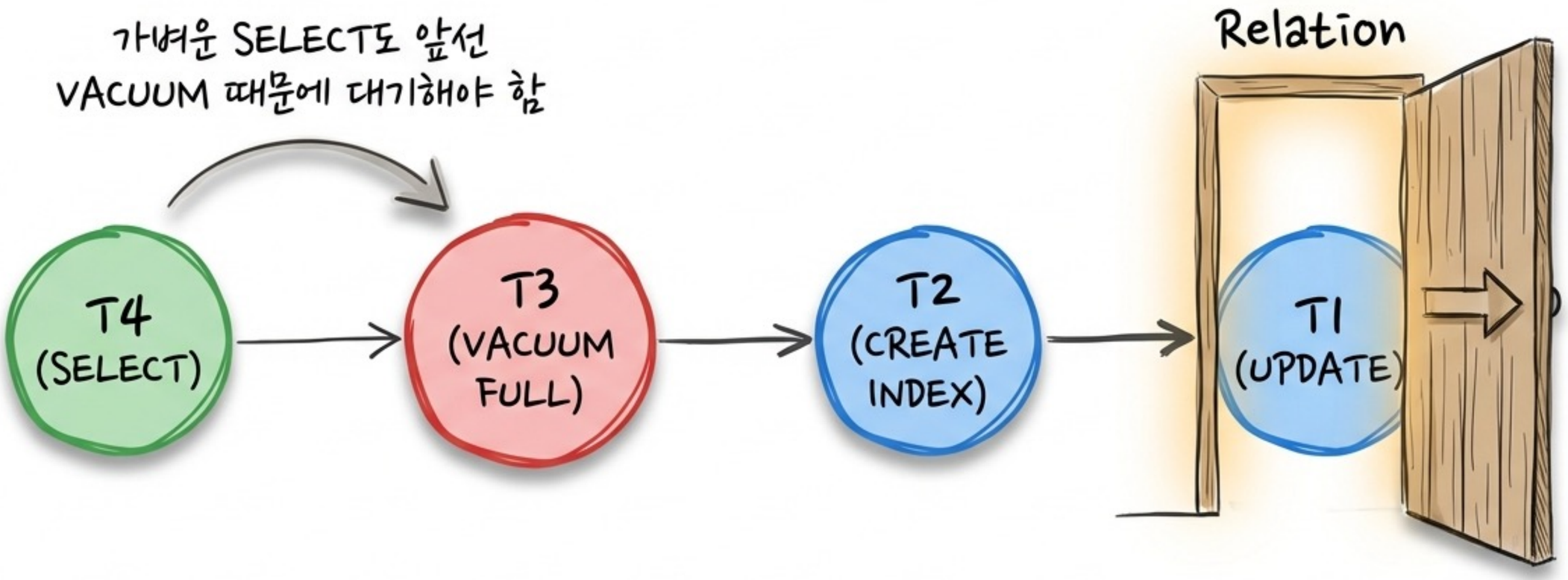
NotebookLM

8가지 잠금 모드와 호환성



NotebookLM

대기열 (Wait Queue): 공정한 줄서기



잠금은 선입선출(FIFO) 방식으로 대기열을 형성합니다.

NotebookLM

잠금 모니터링 및 해결

pg_blocking_pids(pid)

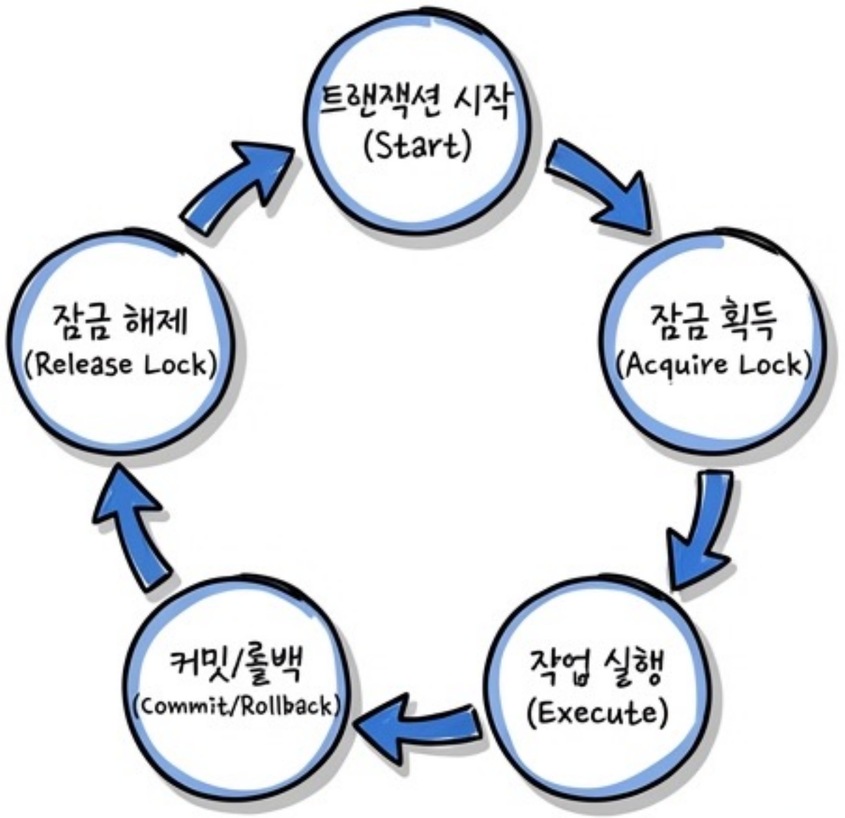
pg_stat_activity



- 대기 중인 프로세스는 CPU를 낭비하지 않고 수면(Sleep) 상태가 됩니다.
- 교착 상태(Deadlock) : 서로가 서로의 잠금을 기다리는 경우, PostgreSQL이 자동으로 감지하여 하나를 중단시킵니다.

NotebookLM

결론: 자동화와 이해



대부분의 잠금은 자동으로 관리됩니다. 하지만 잠금 경합을 줄이기 위해 메커니즘 이해는 필수적입니다. 트랜잭션을 짧게 유지하여 대기열을 최소화하십시오.

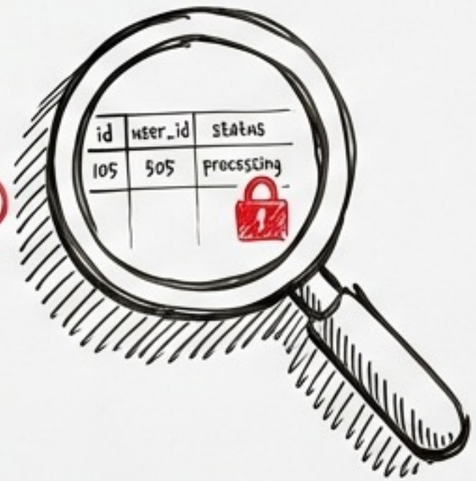
NotebookLM

- **13. Row-Level Locks**

PostgreSQL 행 수준 잠금 (Row-Level Locking)

13장: 내부 작동 원리와 잠금 설계

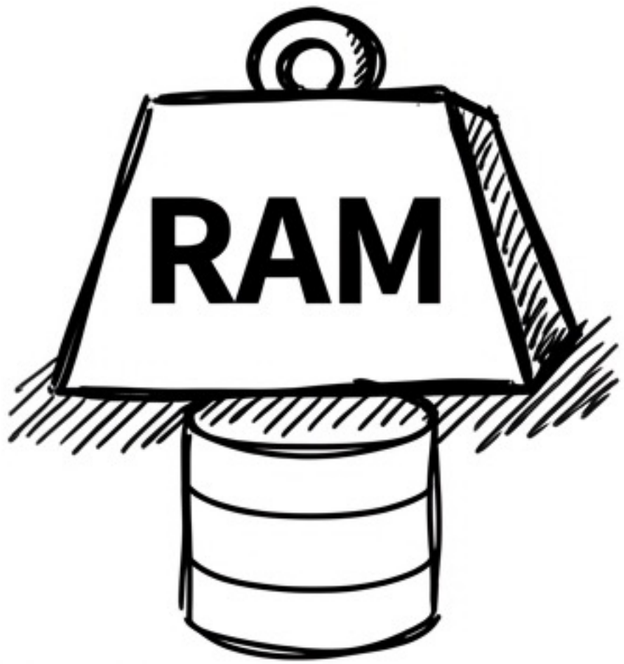
id	user_id	status	created_at	data
101	500	pending	2023-10-27	{'action': 'login'}
102	501	active	2023-10-28	{'action': 'purchase'}
105	505	processing	2023-10-29	{'action': 'update_profile'}



동시성 제어의 핵심 엔진을 해부합니다.

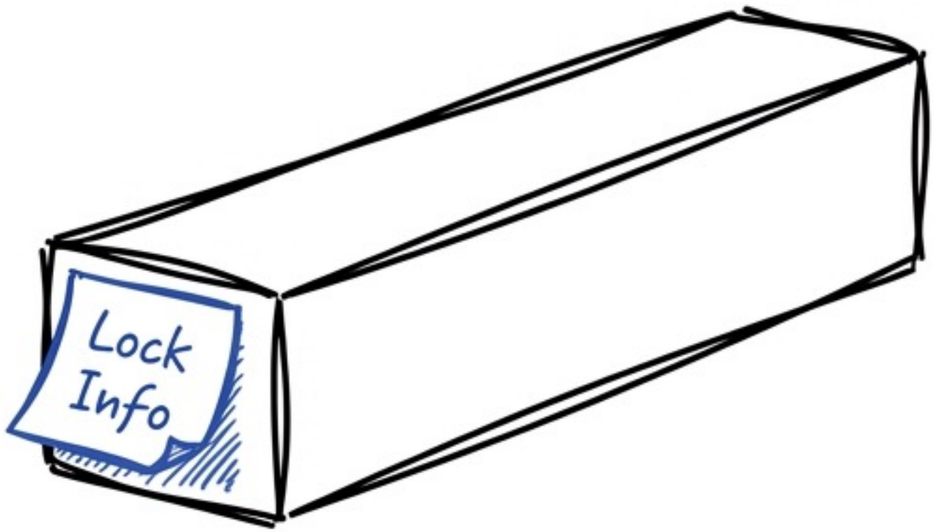
NotebookLM

설계 철학: RAM 대 튜플 헤더



중량 잠금 (Heavy Locks)

- 전통적 방식
- 잠금 정보를 공유 메모리(RAM)에 저장
- 동시성 처리에 한계

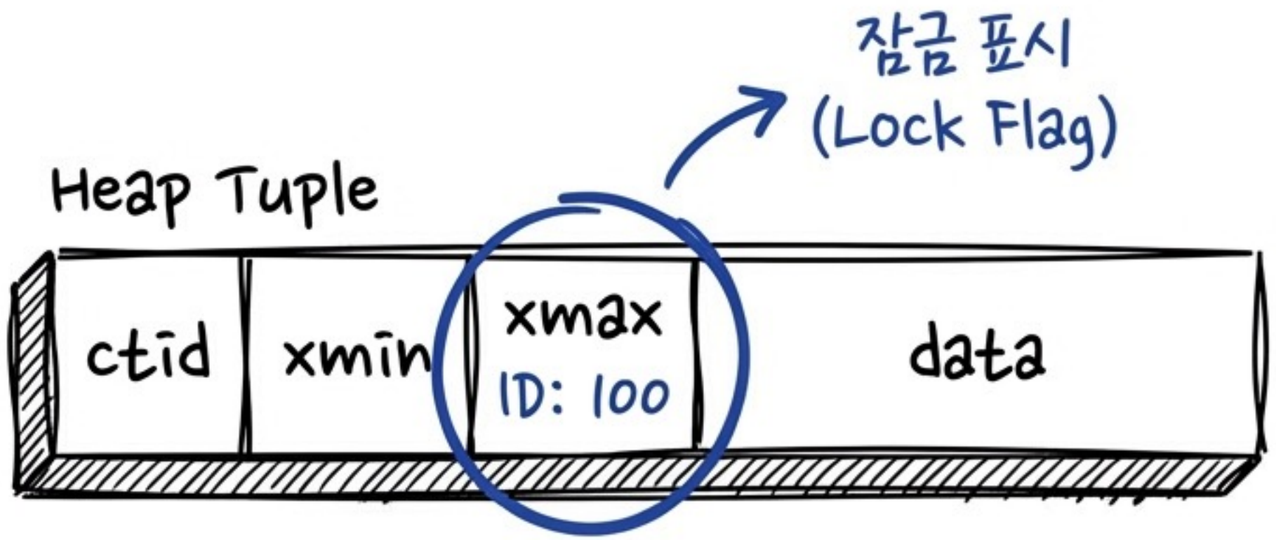


튜플 헤더 (Tuple Header)

- PostgreSQL 방식
- 잠금 정보를 데이터 행(Tuple) 자체에 기록
- 메모리 제한 없는 무제한 행 잠금

NotebookLM

잠금의 정체: `xmax` 필드



Noto Sans KR

메커니즘: 트랜잭션이 행을 수정할 때, 자신의 Transaction ID (XID)를 xmax 필드에 기록합니다.

규칙: xmax에 활성 트랜잭션 ID가 있다면 = 잠금 상태

튜플 헤더: 이 정보는 RAM이 아닌 디스크의 데이터 페이지에 존재합니다.

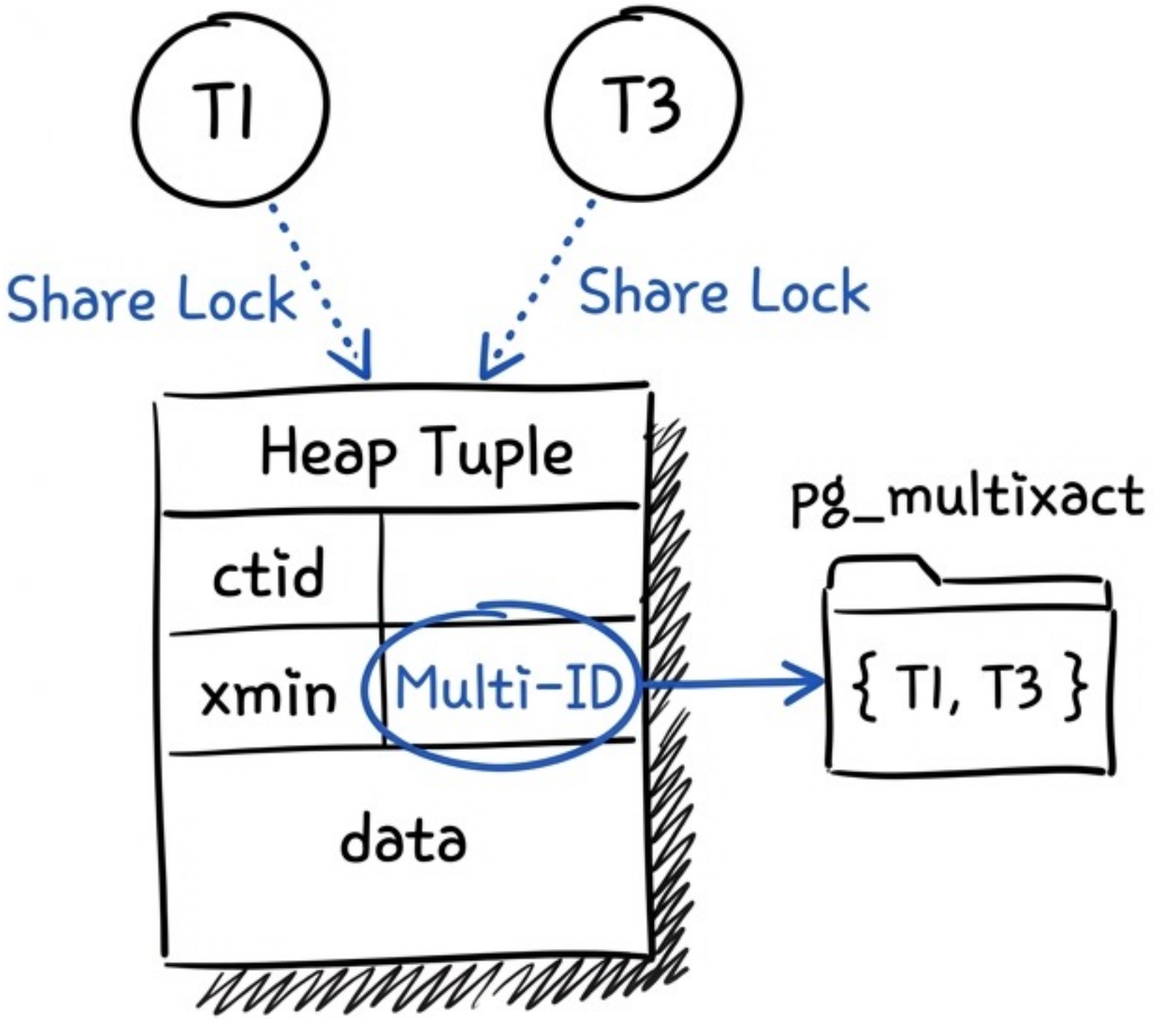
NotebookLM

잠금 모드와 호환성 매트릭스

	Key Share (키 공유)	Share (공유)	No Key Update (비 키 수정)	Update (수정)
Key Share (키 공유)	✓	✓	✓	✗
Share (공유)	✓	✓	✗	✗
No Key Update (비 키 수정)	✓	✗	✗	✗
Update (수정)	✗	✗	✗	✗

읽기는 읽기를 방해하지 않지만, 수정은 배타적입니다.

NotebookLM

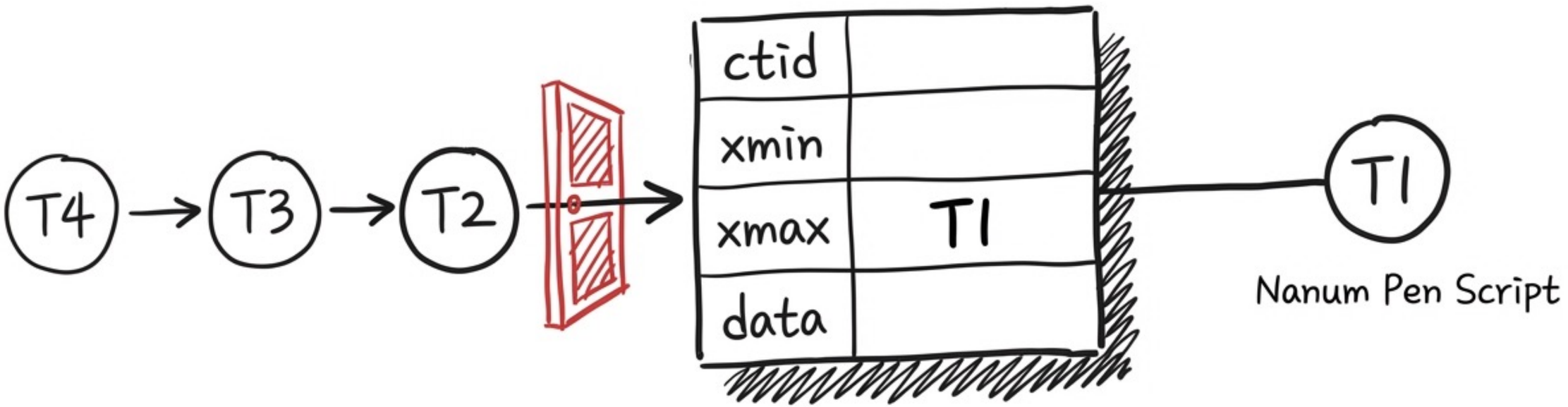


다중 트랜잭션 (MultiXact)

- **문제:** xmax 필드는 하나의 ID만 저장 가능.
- **해결:** 여러 트랜잭션이 공유 잠금을 요청하면, 새로운 '그룹 ID (MultiXact ID)'를 발급.
- **저장소:** 그룹 멤버 정보는 pg_multixact 디렉토리에 별도 저장.

NotebookLM

대기열 (Wait Queue)과 순서 보장



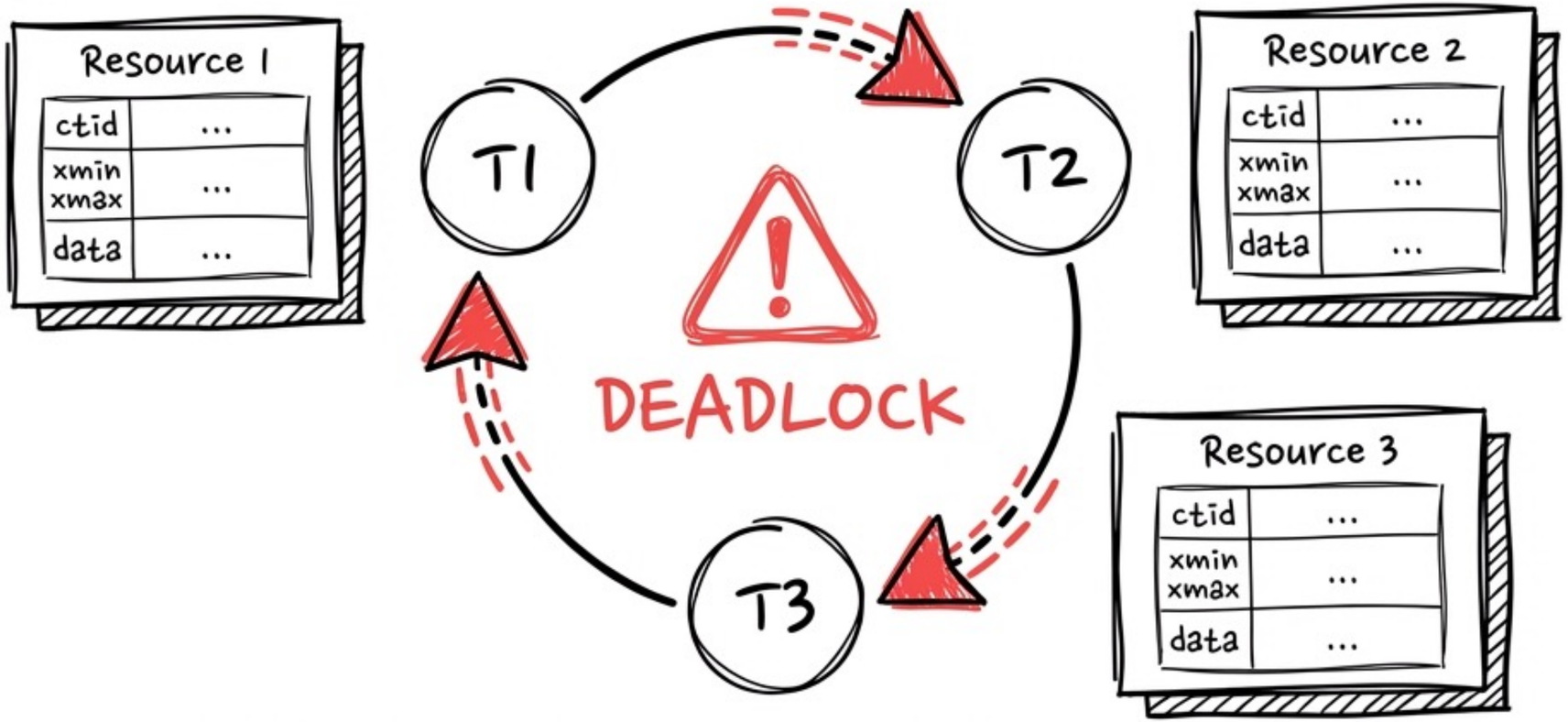
기아 현상(Starvation) 방지: 무작위 경쟁이 아닌, 줄을 서서 기다리는 구조.

튜플 잠금 (Tuple Lock): 대기 중인 트랜잭션 관리를 위해 메모리 상의 '중량 잠금'을 일시적으로 사용.

일시적으로 사용.

NotebookLM

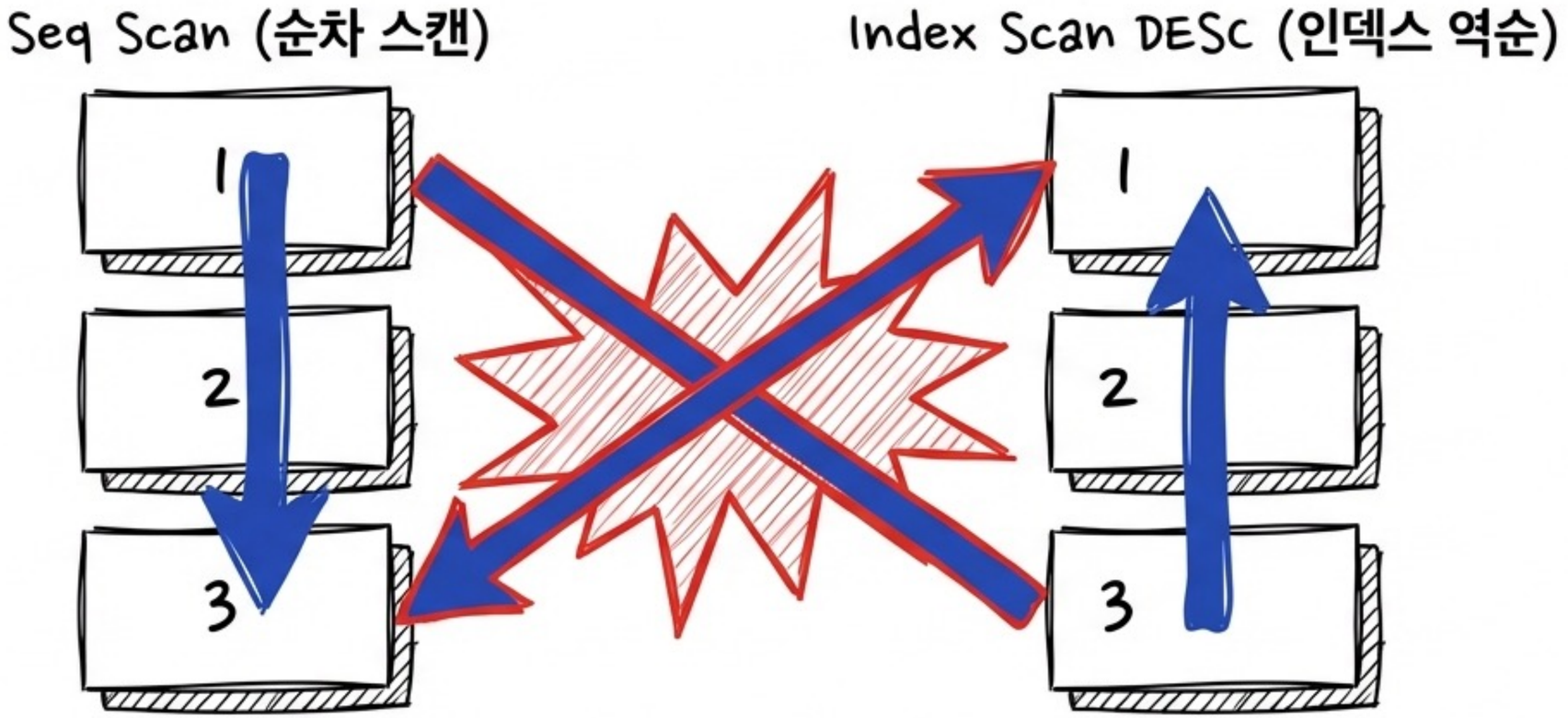
교착 상태 (Deadlock)



- 정의: 서로 상대방이 보유한 자원을 기다리는 순환 고리.
- 감지 (Detection): deadlock_timeout 후 대기 그래프 확인.
- 해결: 순환 고리를 끊기 위해 트랜잭션 중 하나를 강제 종료(Abort).

NotebookLM

숨겨진 위험: UPDATE 처리 순서



- 시나리오: 동일한 테이블을 수정하더라도 접근 순서가 다르면 교착 상태 발생.
- 예시: Seq Scan(정순) vs Index Scan(역순).
- 교훈: SQL은 집합 언어이지만, 실제 잠금은 행 단위로 순차적으로 발생합니다.

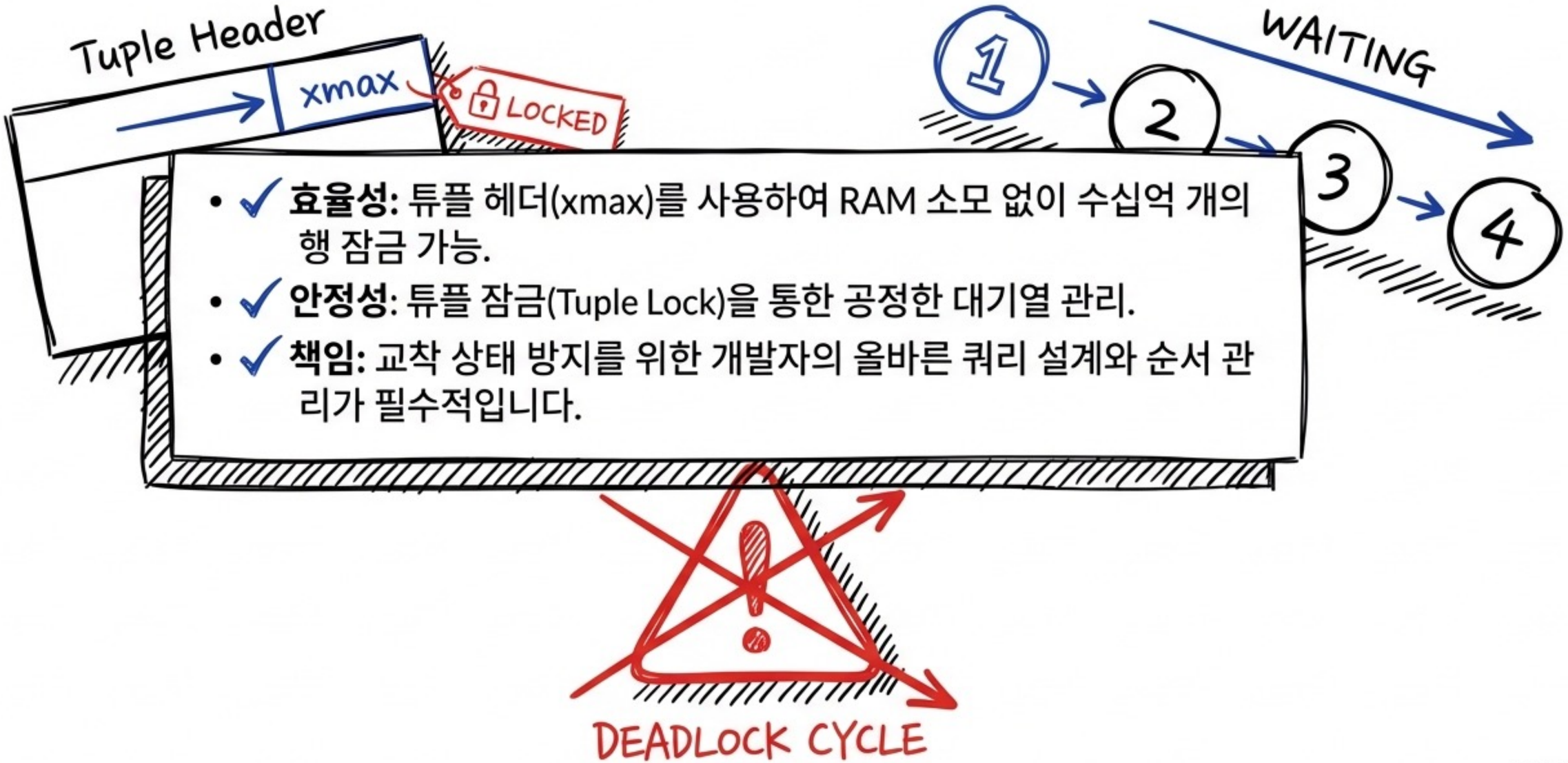
NotebookLM

해결 방안 및 모범 사례



NotebookLM

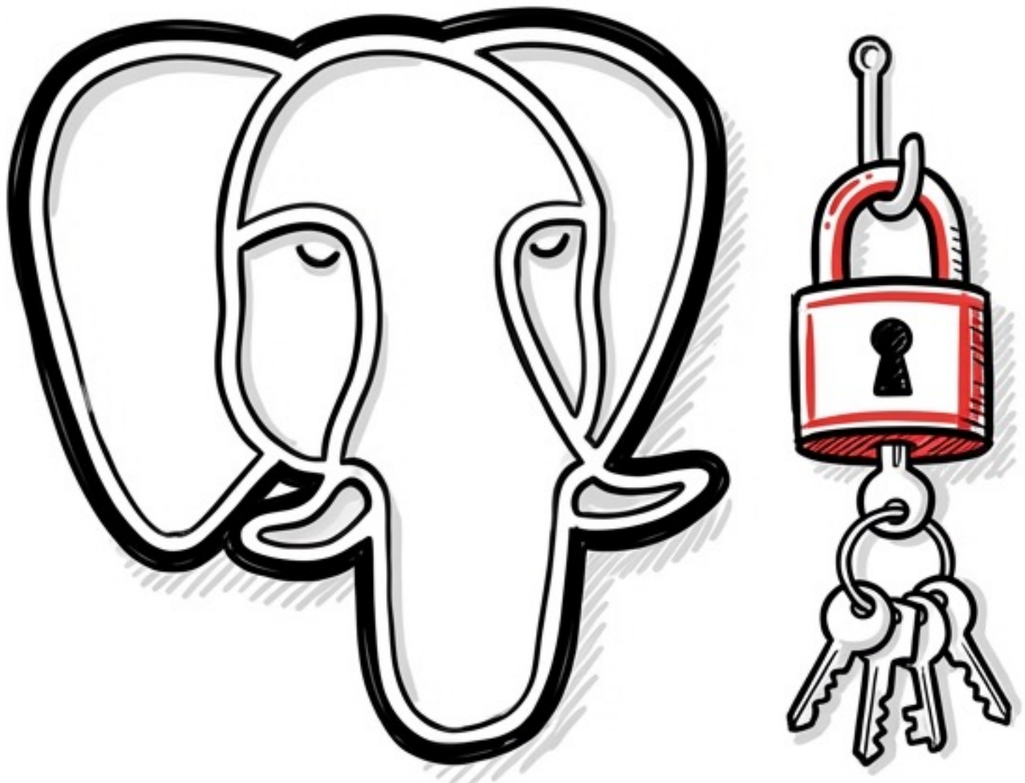
요약: 효율성과 안정성의 균형



- ✓ **효율성:** 튜플 헤더(xmax)를 사용하여 RAM 소모 없이 수십억 개의 행 잠금 가능.
- ✓ **안정성:** 튜플 잠금(Tuple Lock)을 통한 공정한 대기열 관리.
- ✓ **책임:** 교착 상태 방지를 위한 개발자의 올바른 쿼리 설계와 순서 관리가 필수적입니다.

NotebookLM

- **14. Miscellaneous Locks**

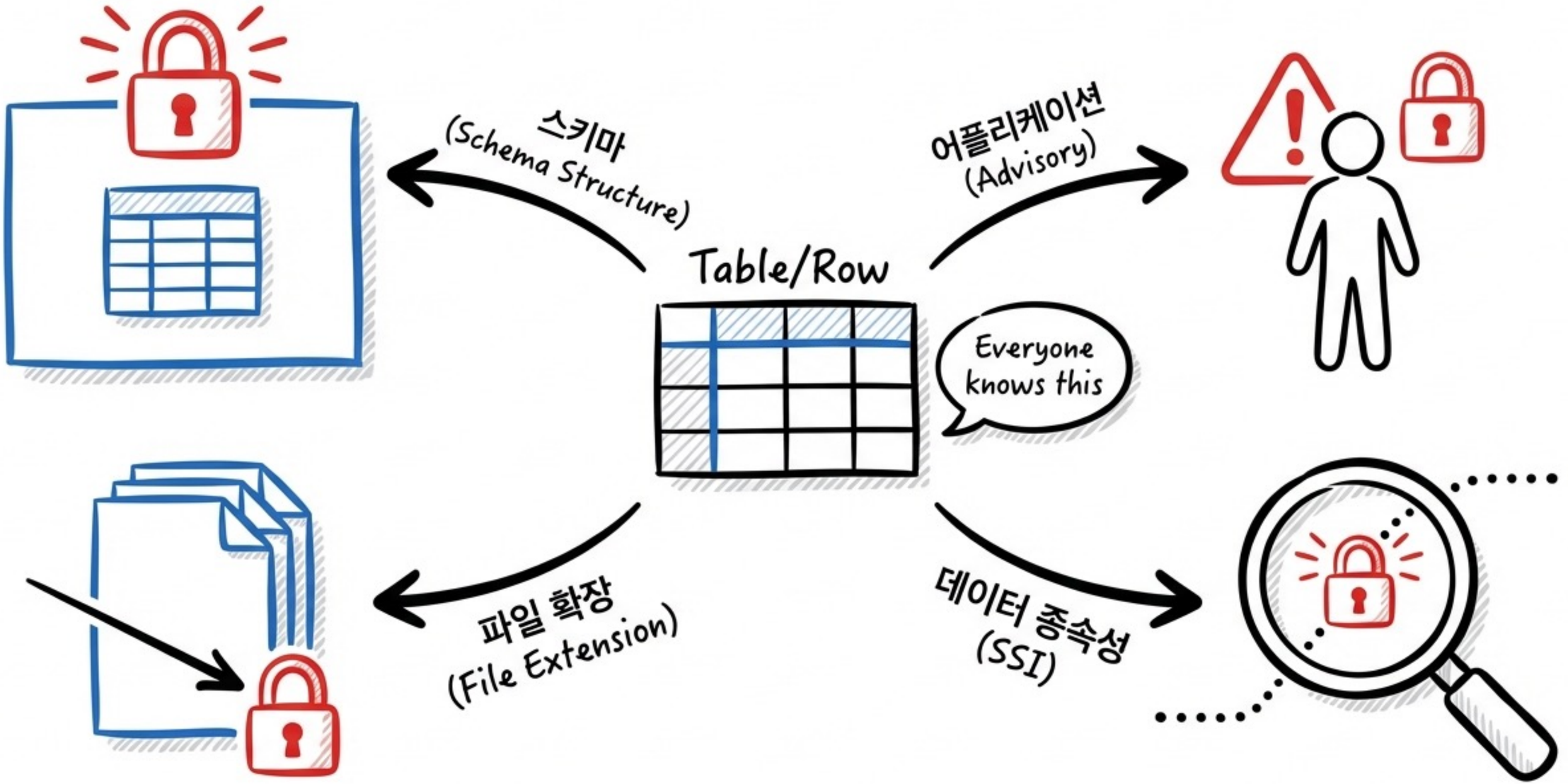


PostgreSQL: 그 외의 잠금들

14장: 객체, 확장, 페이지, 권고, 조건부 잠금

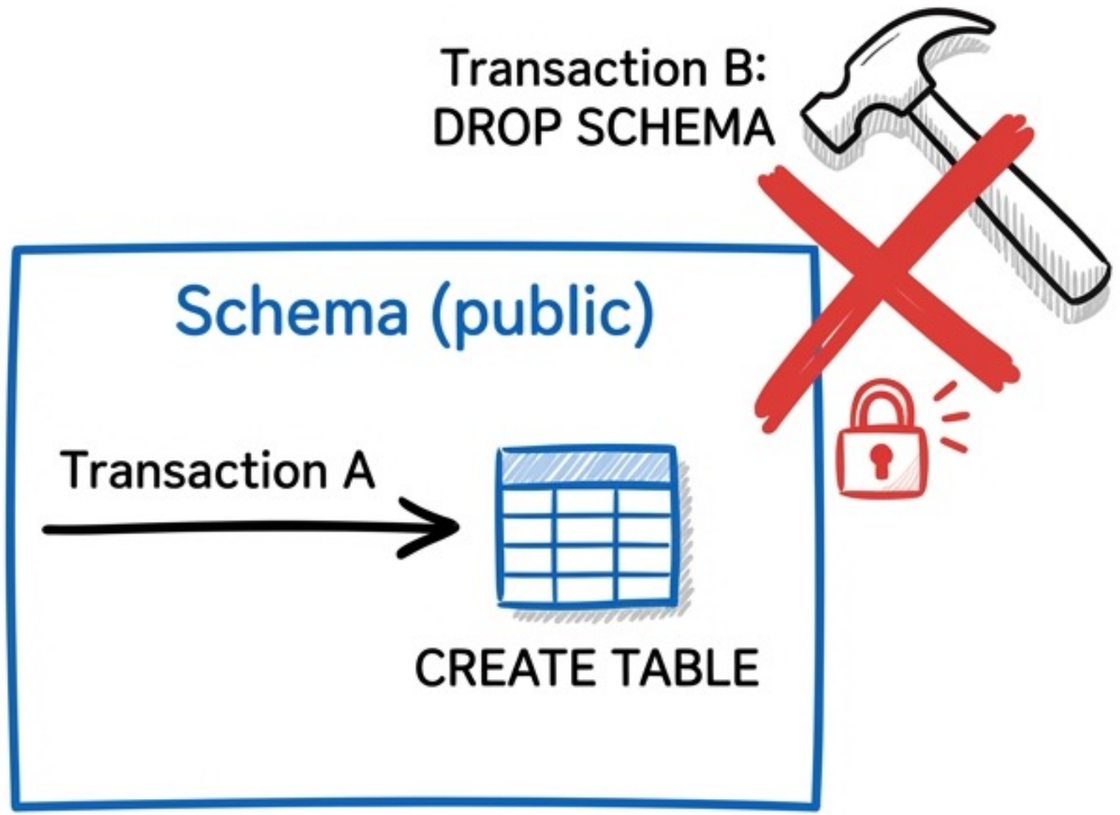
NotebookLM

데이터 그 너머의 잠금 (Locks Beyond Data)



NotebookLM

14.1 비객체 잠금 (Object Locks)

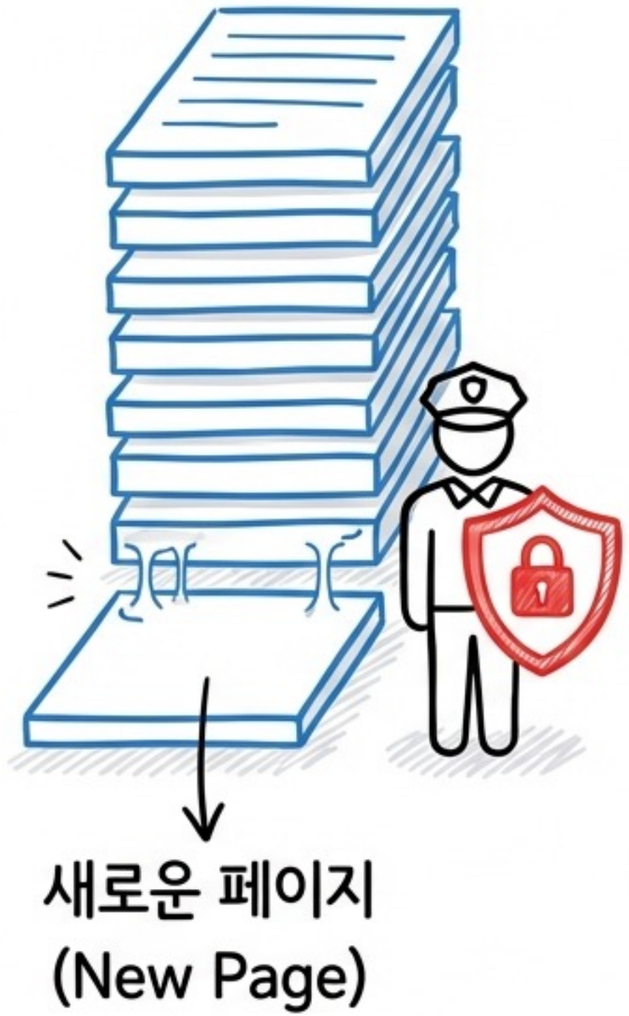


- 대상: 관계(Relation)가 아닌 리소스 (테이블스페이스, 스키마, 구독 등).
- 작동: pg_locks 테이블의 locktype = ‘object’.
- 예시: 테이블 생성 시 pg_namespace(public 스키마)를 잠가 삭제 방지.

```
SELECT * FROM pg_locks WHERE locktype = ‘object’
```

NotebookLM

14.2 관계 확장 잠금 (Relation Extension Locks)



- 발생: 물리적 파일 끝에 새로운 페이지를 추가할 때.
- 특징: 트랜잭션 종료를 기다리지 않고 즉시 해제. (데드락 감지 제외).
- 최적화: 힙(Heap) 파일은 경합을 줄이기 위해 한 번에 최대 512페이지씩 확장.

NotebookLM

14.3 페이지 잠금 (Page Locks)



- 대상: GIN 인덱스 전용.
- 목적: fastupdate 대기 목록을 주 인덱스로 이동할 때 메타페이지 보호.
- 특징: 작업 완료 후 즉시 해제 (트랜잭션 지속 X).

NotebookLM

14.4 조언적 잠금 (Advisory Locks)

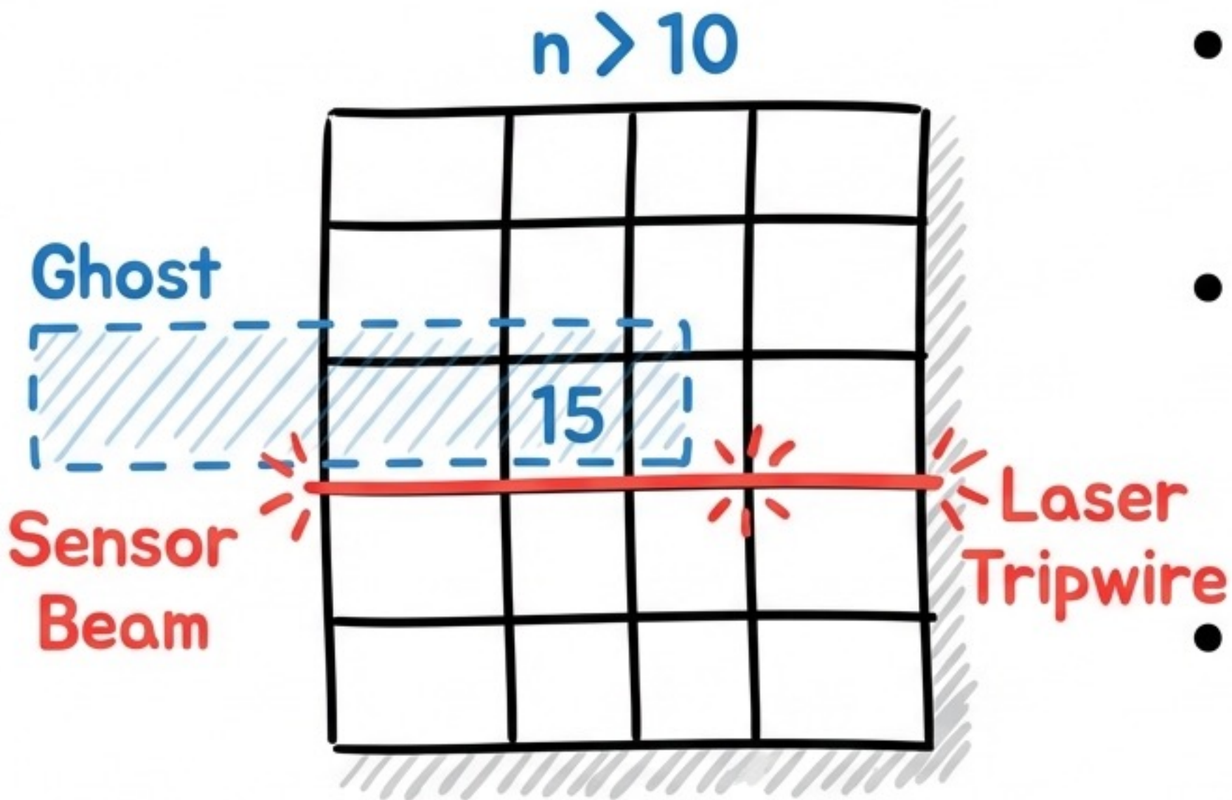


- 개념: 개발자가 직접 제어하는 잠금 (DB가 자동 획득 X).
- 용도: DB 객체가 아닌 논리적 리소스 동시성 제어.
- 사용법:
  - pg_advisory_lock(key): 획득
  - pg_advisory_unlock(key): 해제
- 범위: 세션 레벨 혹은 트랜잭션 레벨(xact) 선택 가능.

NotebookLM

14.5 조건부 잠금 (Predicate Locks) - SSI

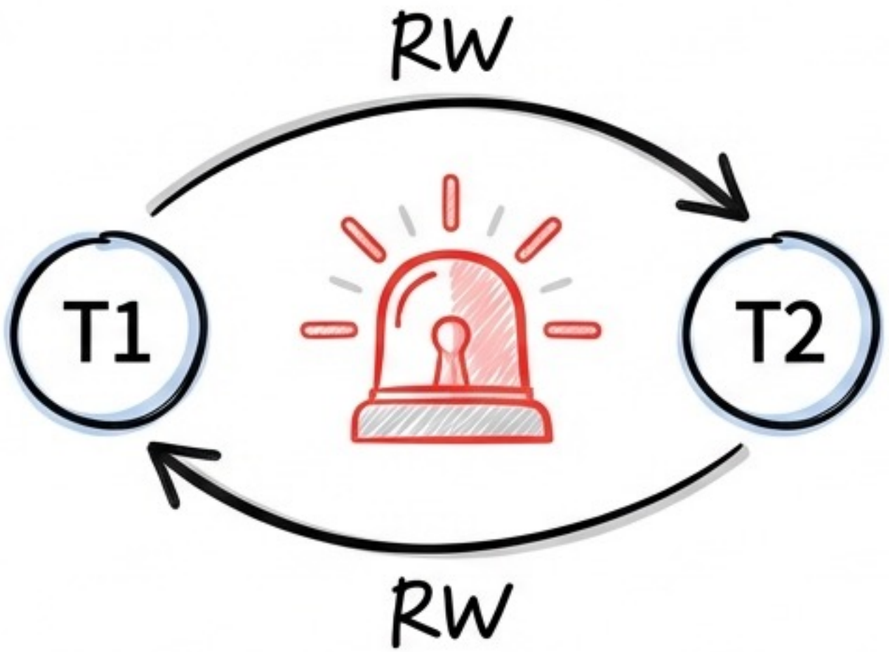
실제로 아무것도 잠그지 않는다.



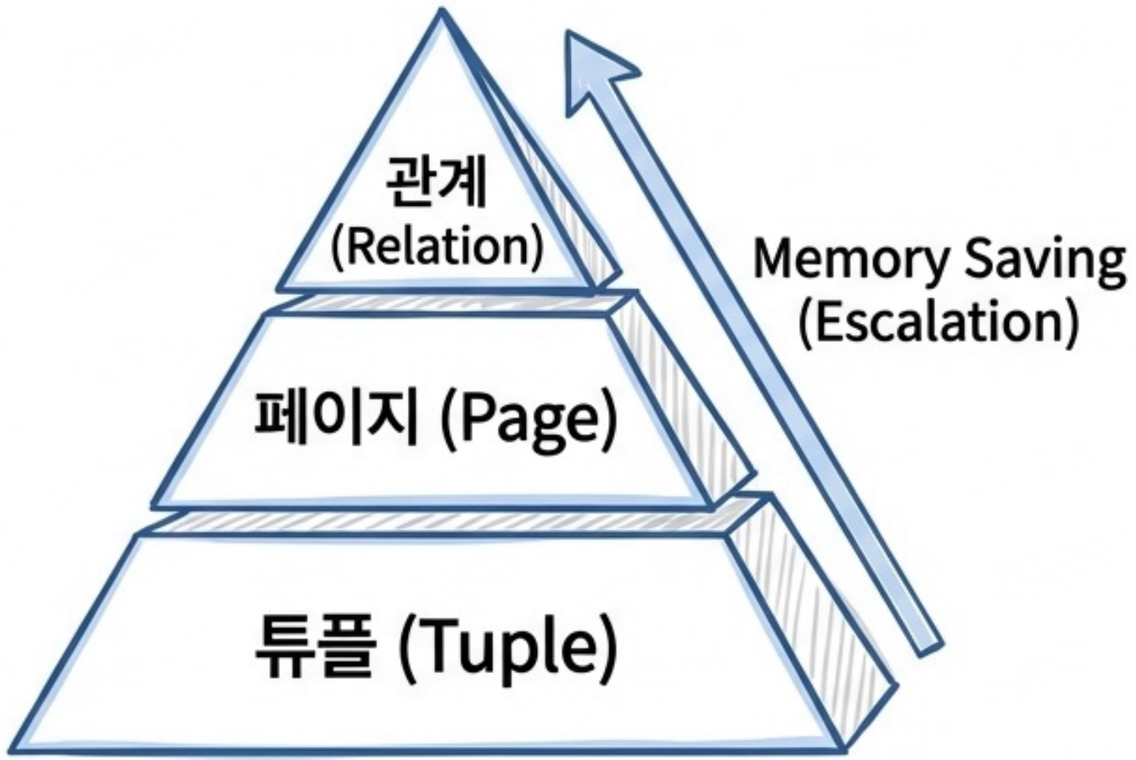
- 직렬화(Serializable) 격리 수준을 보장하기 위한 메커니즘.
- 유령 행(Phantom Read) 방지: 조건(n > 10)을 만족하는 미래의 데이터 삽입을 추적.
- SIReadLock 모드로 pg_locks에 표시됨.

NotebookLM

종속성 감지 및 에스컬레이션



RW 종속성 사이클 = 위험



위험한 종속성 사이클이 생기면 트랜잭션 중단.
메모리 관리: 잠금 수가 많아지면 상위 레벨로 격상하여 RAM 절약.

NotebookLM

핵심 요약 (Summary)

- ☑ 비객체 (Object): 스키마 및 구조 보호.
- ☑ 확장/페이지 (Extension/Page): 파일의 물리적 확장 및 GIN 인덱스 보호 (짧은 잠금).
- ☑ 조언적 (Advisory): 비즈니스 로직을 위한 수동 제어.
- ☑ 조건부 (Predicate): SSI를 위한 데이터 종속성 추적 (잠금 아님).



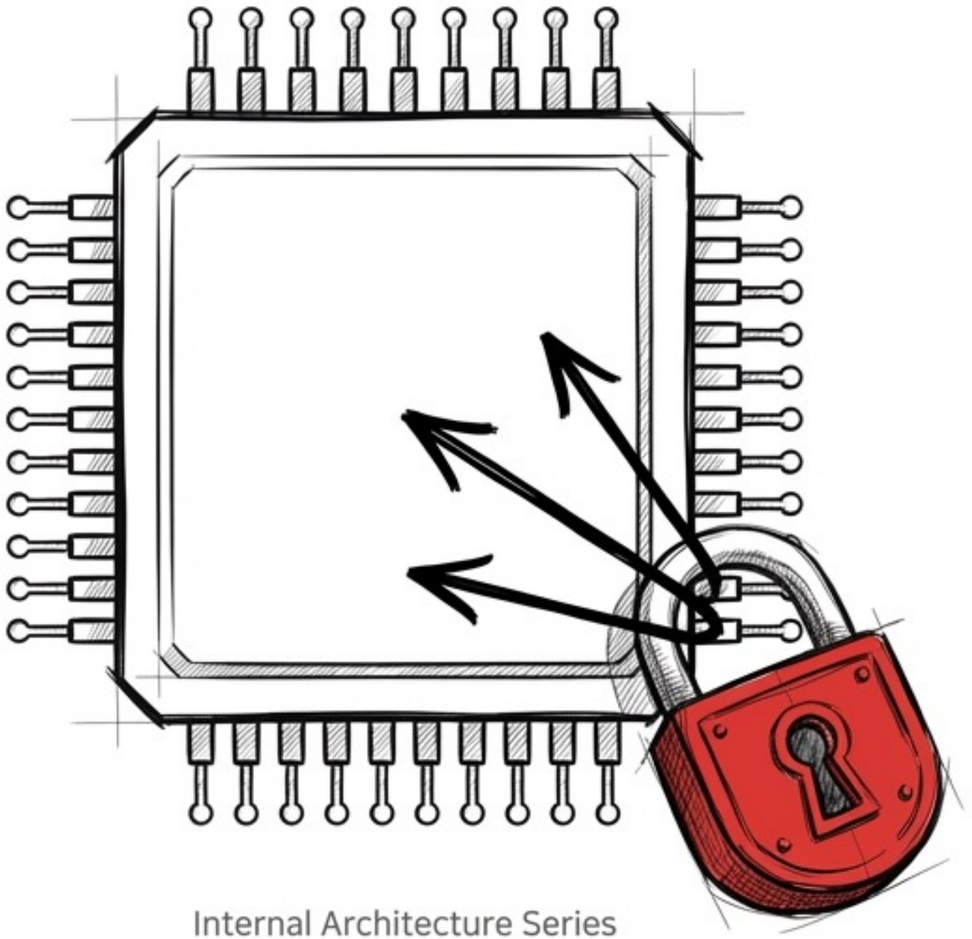
Tip: 읽기 전용 트랜잭션은 READ ONLY로 선언하여 조건부 잠금 오버헤드 감소.

NotebookLM

- **15. Locks on Memory Structures**

PostgreSQL 메모리 구조의 잠금

데이터 무결성을 지키는 보이지 않는 규칙들



Internal Architecture Series

NotebookLM

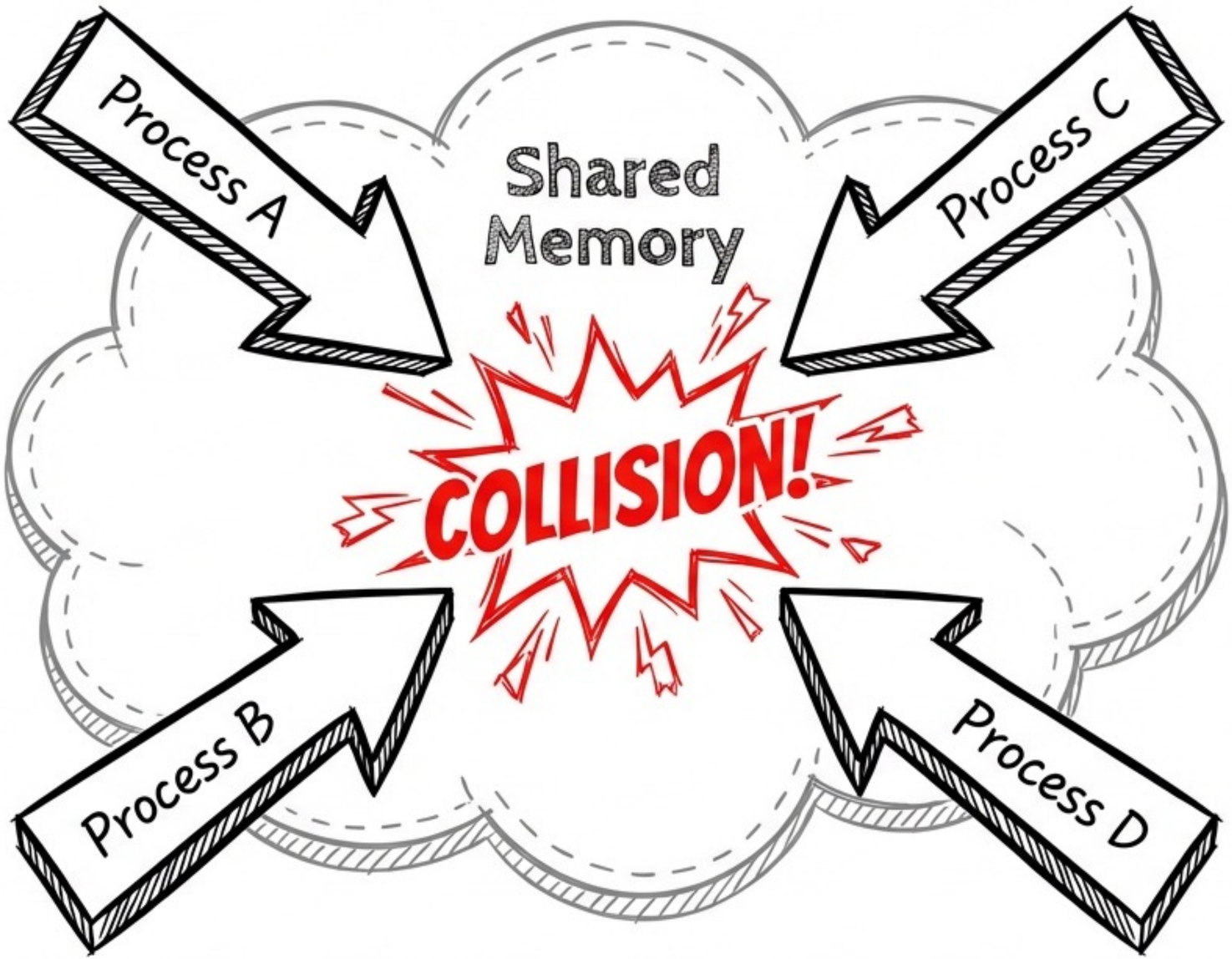
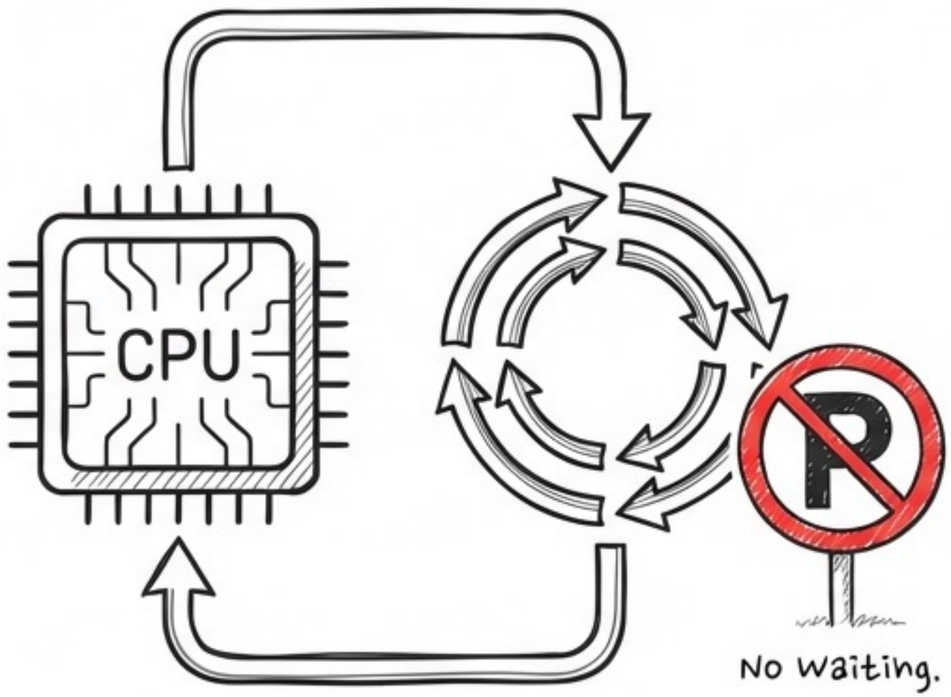


Figure 1.1: Data Collision Scenario

왜 잠금이 필요한가?

- **공유 메모리 보호:** 여러 프로세스가 동시에 동일한 메모리 구조에 접근합니다.
- **충돌 방지:** 동시 업데이트로 인한 데이터 손상(Corruption)을 막아야 합니다.
- **해결책:** 무거운 잠금(Heavy Lock) 대신 가볍고 비용이 적은 잠금 전략을 사용합니다.

NotebookLM



스핀잠금 (Spinlocks): 초고속 경비원

- 특징: 매우 짧은 시간 (CPU 사이클 이내) 보호.
- 작동 원리: 원자적 CPU 명령 (CAS: Compare-and-Swap).
- Busy-wait: 잠금 획득 실패 시, 멈추지 않고 반복 시도('Spinning').
- 주의: 교착상태(Deadlock) 감지 기능 없음. 오직 배타적 모만 지원.

NotebookLM

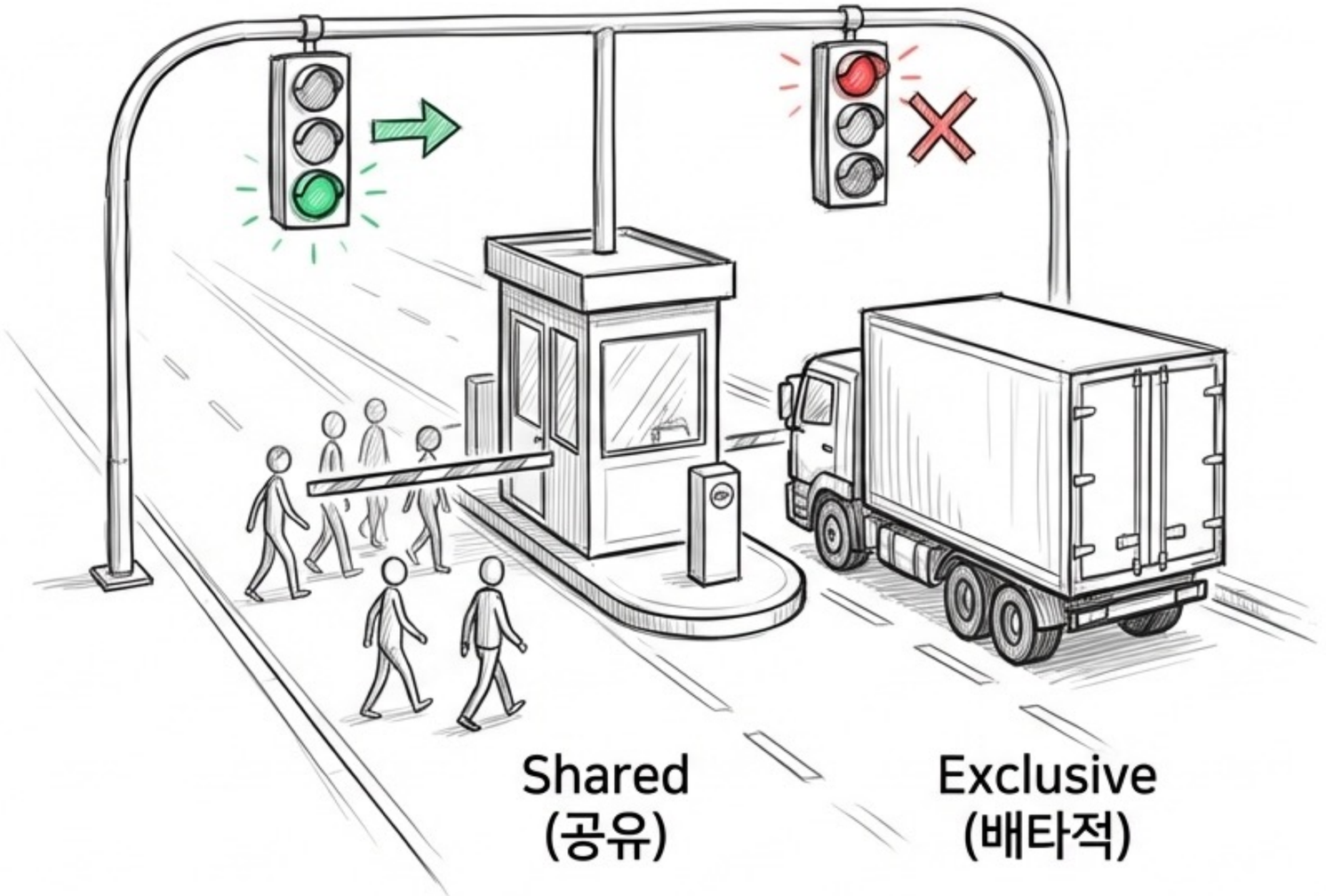


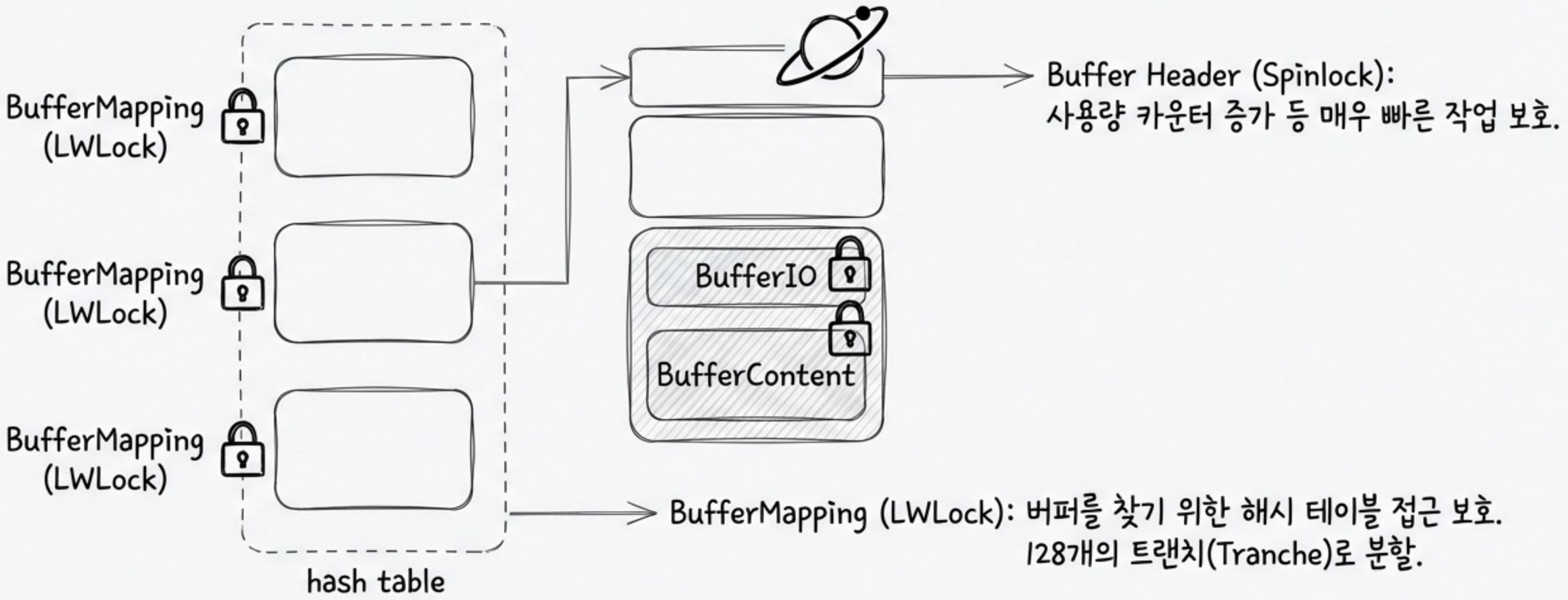
Figure 3.1: LWLocks Conceptual Model

경량 잠금 (LWLocks): 구조적 관리자

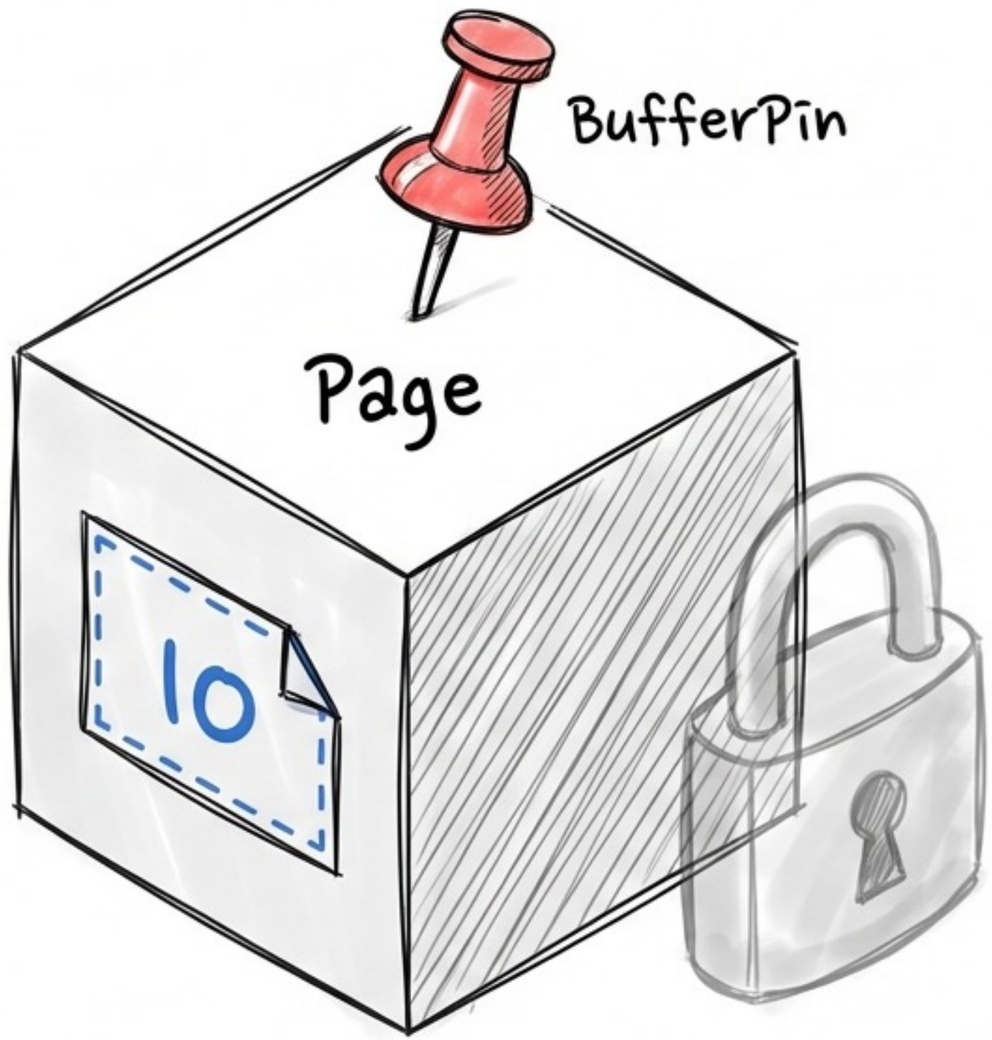
- **용도**: 해시 테이블, 포인터 목록 등 주요 데이터 구조 보호.
- **모드**: 공유 모드(읽기) vs 배타적 모드(수정).
- **특징**: 큐(Queue)가 없음. 경쟁 발생 시 무작위 획득(운이 작용함).
- **장점**: 스핀잠금과 달리 모니터링(계측) 가능.

NotebookLM

사례 연구 1: 버퍼 캐시 구조(Buffer Cache)



NotebookLM



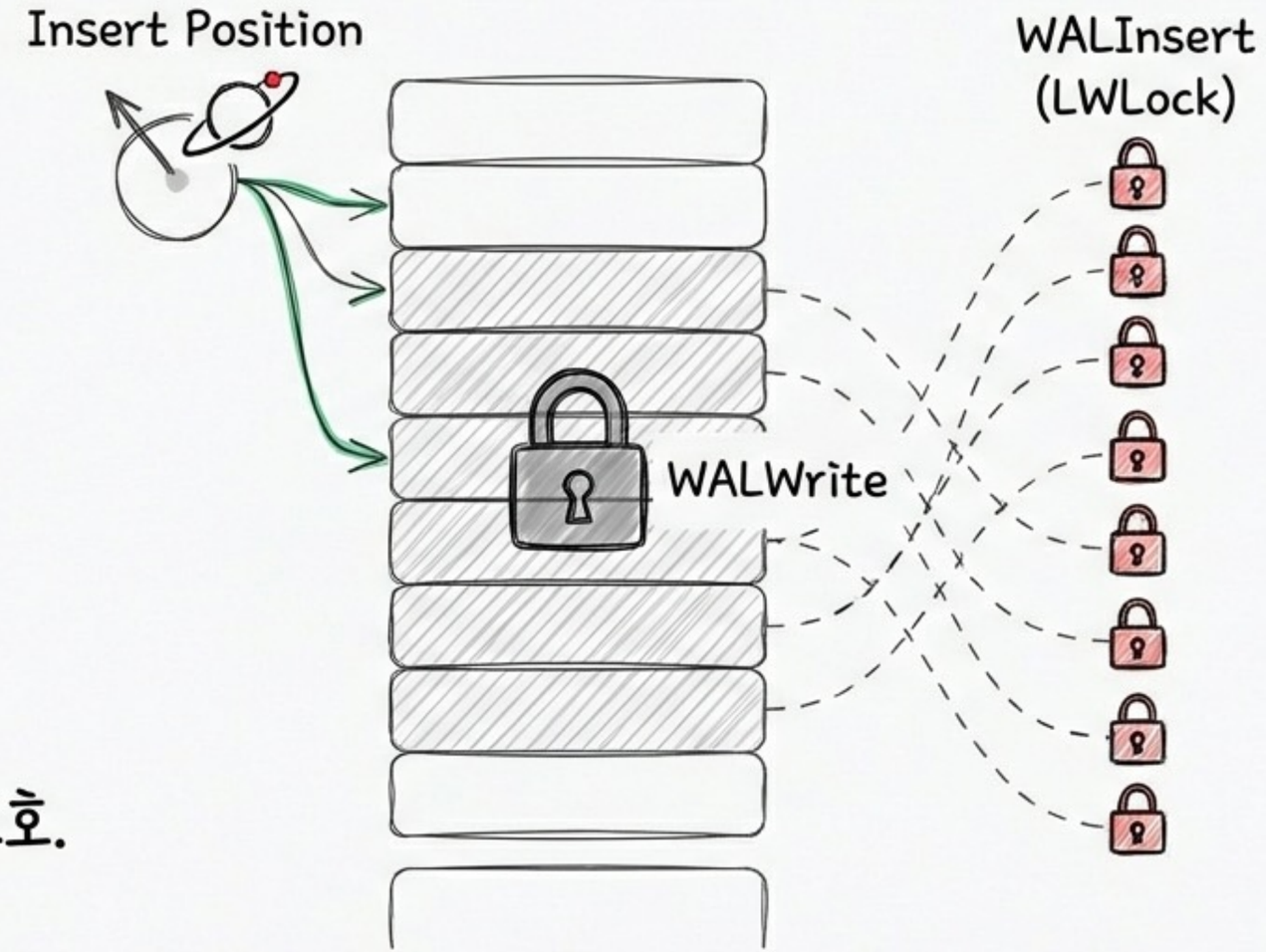
버퍼 읽기 및 I/O 작업

- BufferContent (LWLock): 페이지 내용을 읽거나(공유) 수정할 때(배타적) 획득.
- BufferIO (속성): 실제 잠금이 아닌 '상태 플래그'. 디스크 I/O 중일 때 설정.
- Buffer Strategy (Spinlock): 빈 버퍼나 Clock Hand 이동 시 사용.

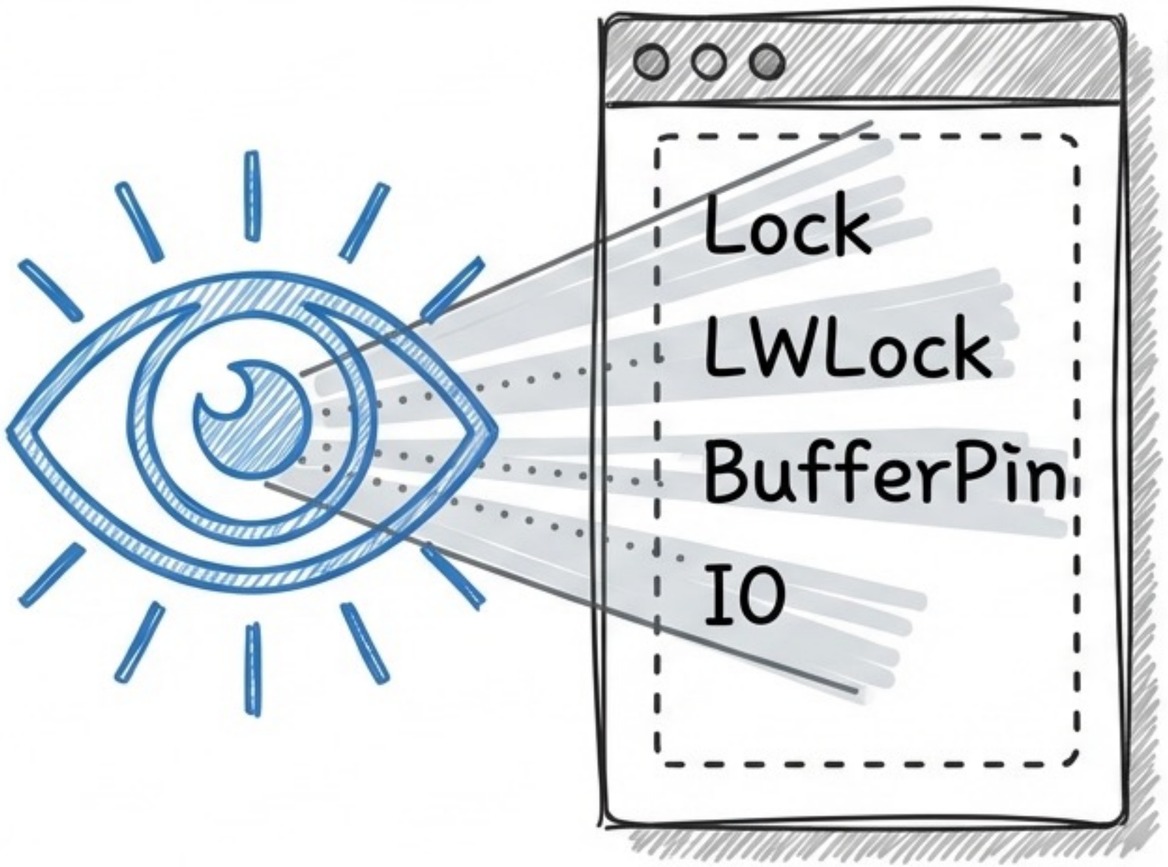
NotebookLM

사례 연구 2: WAL 버퍼 (WAL Buffers)

- WALBufMapping:
  단일 LWLock 사용.
- Insert Position (Spinlock):
  공간 예약은 엄격한 순서가 필요.
- WALInsert (LWLock):
  공간 예약 후 8개의 잠금 중 하나를
  획득하여 병렬 복사.
- WALWrite: 디스크 기록 시 배타적 보호.



NotebookLM



대기 모니터링: 현재 상태 진단

도구: pg_stat_activity 뷰

- Lock: 무거운 잠금 (테이블 레벨 등).
- LWLock: 경량 잠금 (메모리 구조).
- BufferPin: 버퍼 고정 대기.
- IO: 디스크 읽기/쓰기 대기.

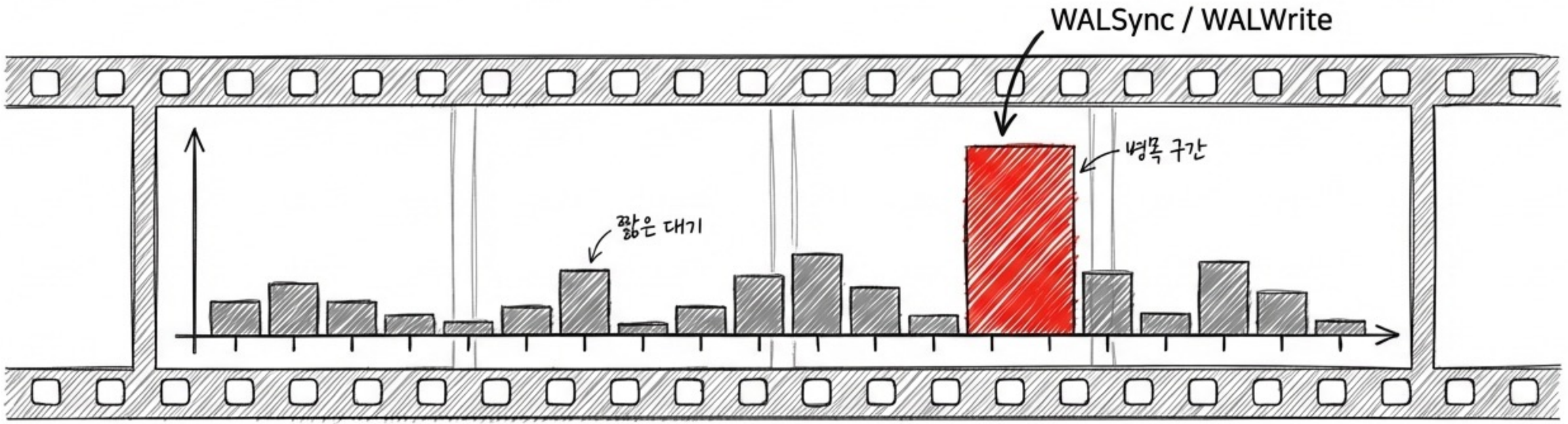
한계: 현재 순간(Snapshot)의 정보만 제공.

NotebookLM

이력 분석과 샘플링 (Sampling)

도구: pg_wait_sampling 확장 기능

- **필요성**: 짧은 대기(Short waits)는 순간 포착이 어려움.
- **활용**: 대기 프로파일(Profile)을 통해 병목 구간 식별.
- **분석 예시**: WALSync(디스크 플러시)나 WALWrite 같은 I/O 병목 발견.



NotebookLM



핵심 요약 (Key Takeaways)

- **계층 구조**: Spinlock(초고속) vs LWLock(구조적).
- **적용**: Buffer Cache와 WAL은 이들을 정교하게 조합하여 데이터 무결성 달성.
- **운영**: pg_stat_activity와 pg_wait_sampling을 통해 보이지 않는 대기를 시각화 하는 것이 필수.

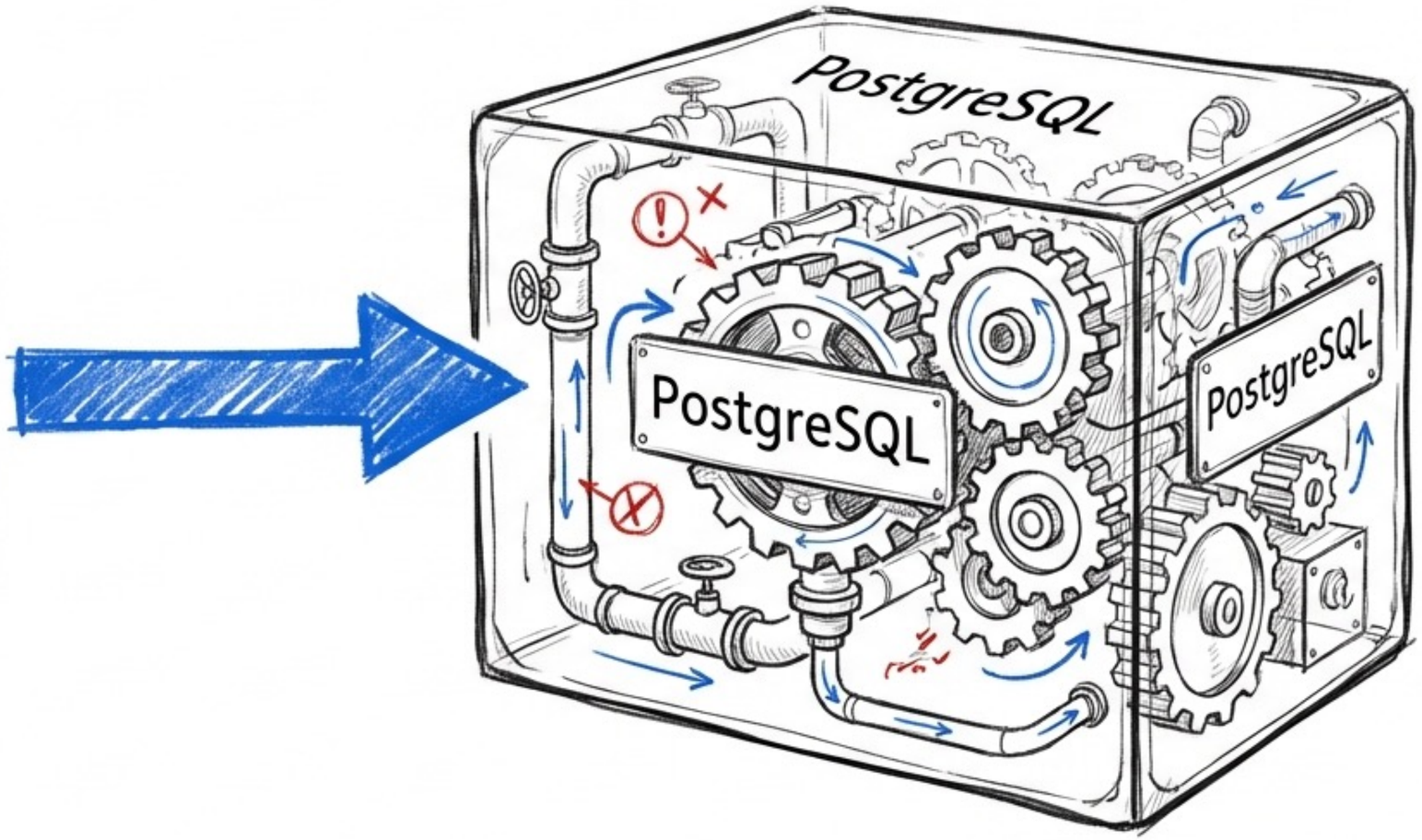
NotebookLM

- **16. Query Execution Stages**

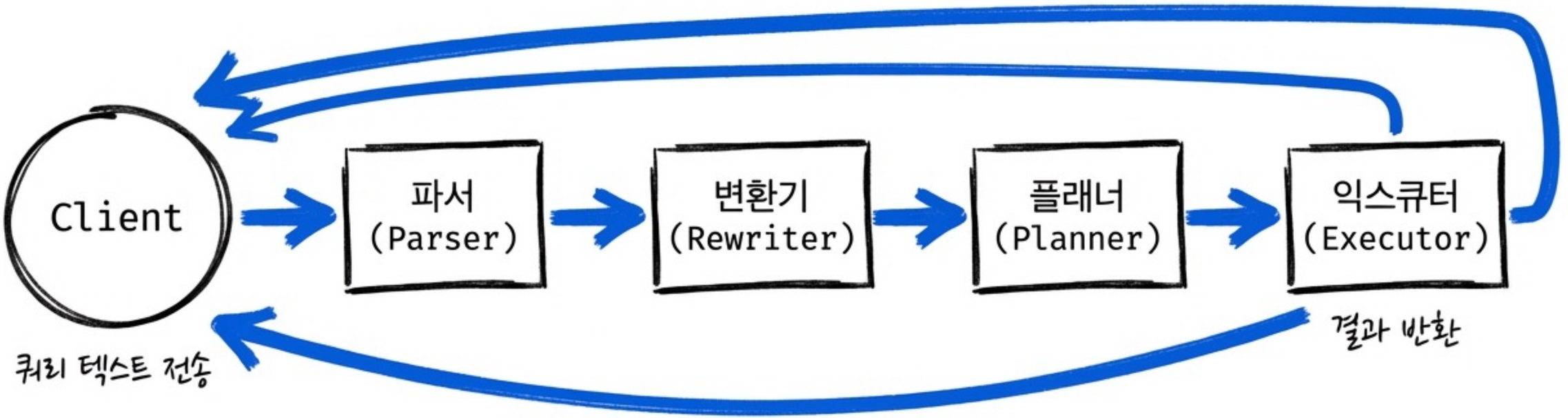
PostgreSQL 쿼리 실행 단계

쿼리의 생애주기 (The Life of a Query)

```
SELECT *
FROM flights;
```

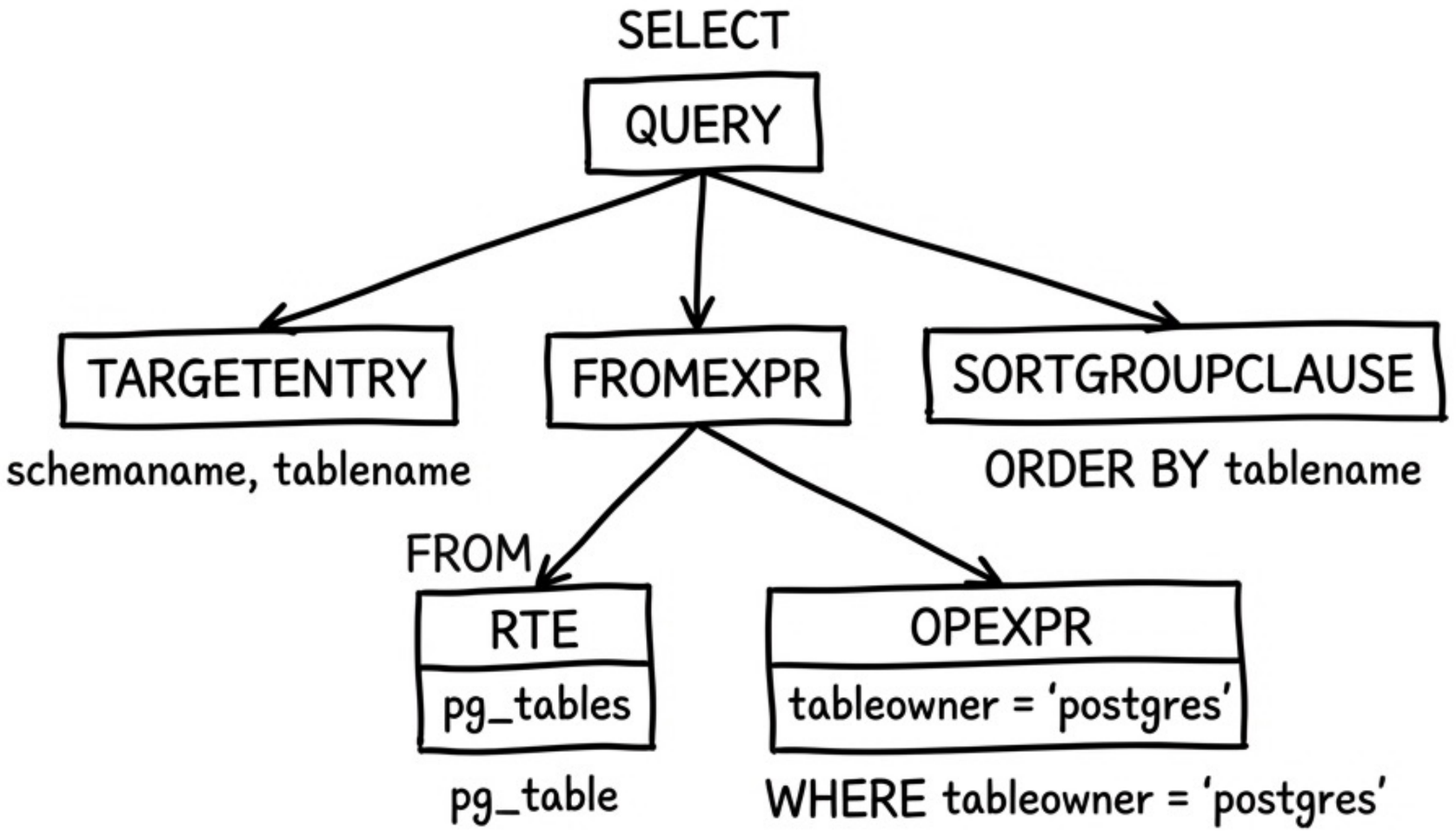


NotebookLM


Client
파서
(Parser)
변환기
(Rewriter)
플래너
(Planner)
익스큐터
(Executor)
쿼리 텍스트 전송
결과 반환

NotebookLM

1단계: 파싱 (Parsing) - 이해의 단계



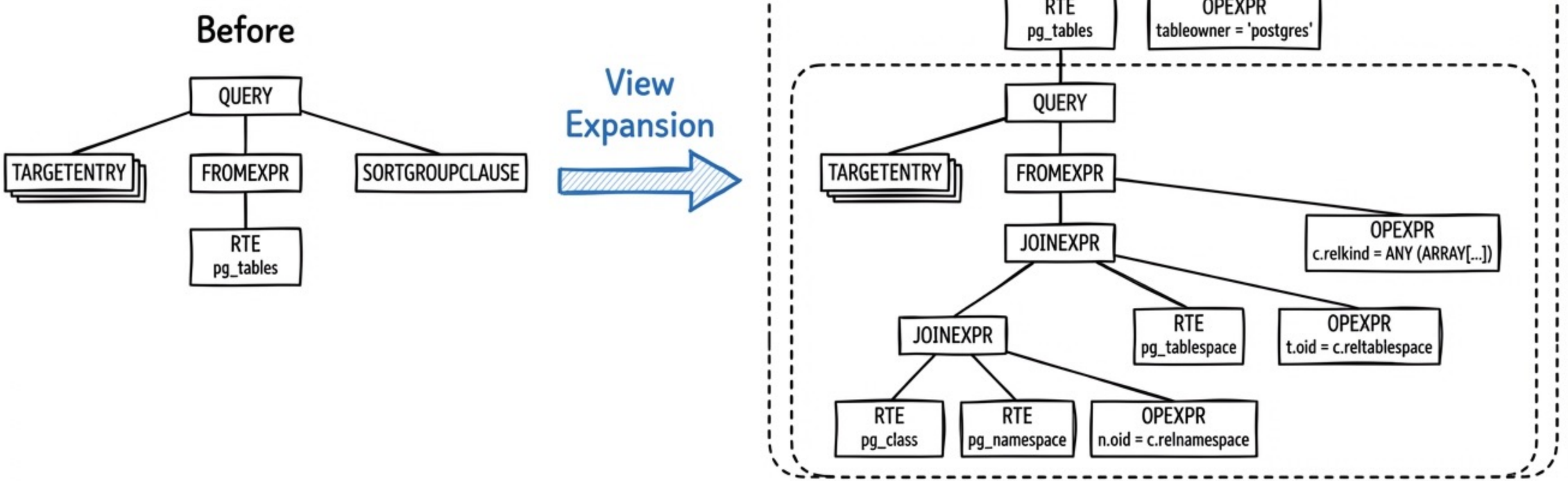
- 어휘 및 구문 분석
(Lexical & Syntax Analysis)

- 파싱 트리 (Parse Tree)

- 의미 분석 (Semantic Analysis):
System Catalog 확인

NotebookLM

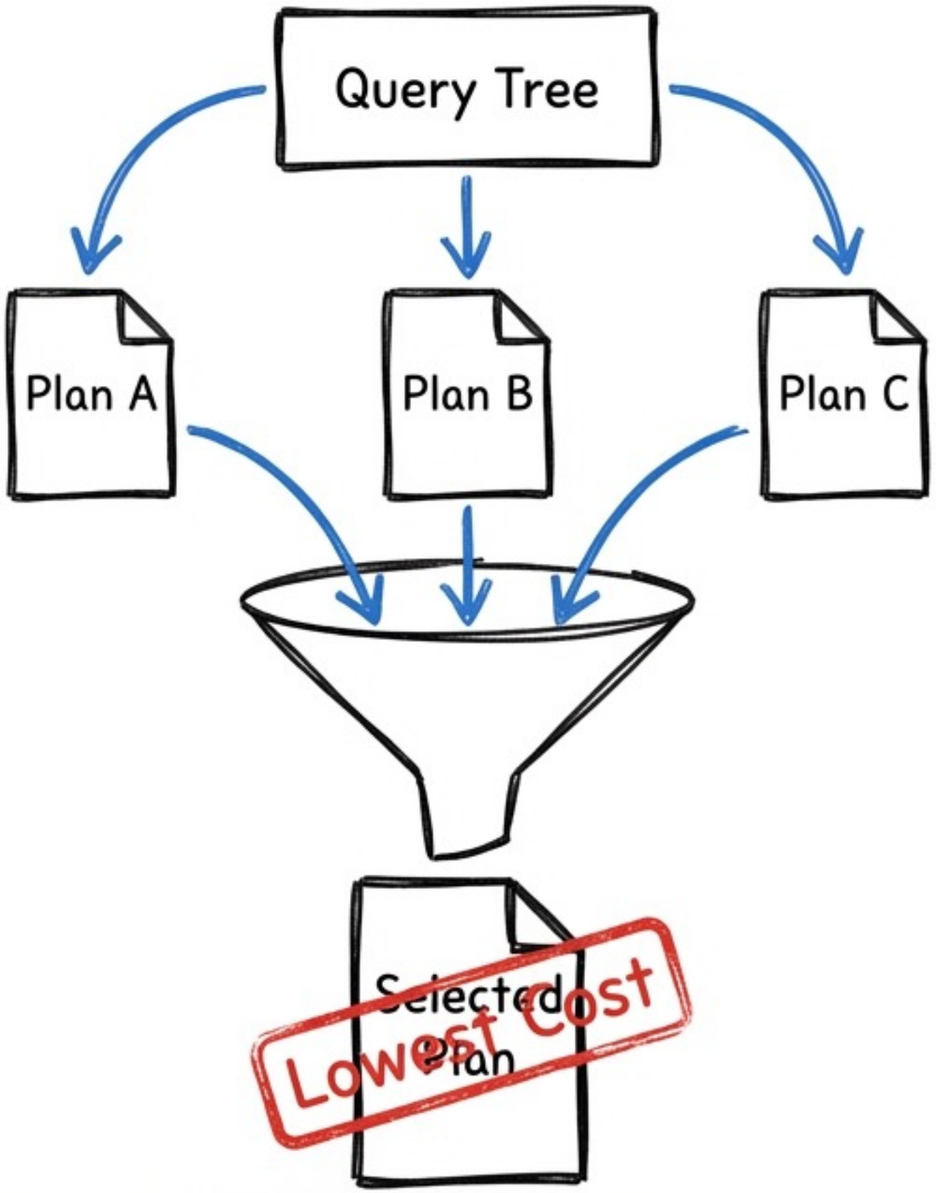
2단계: 변환 (Rewriter) - 룰 시스템



- 뷰(View) 확장: pg_tables 예시
- 룰 시스템 (Rule System)
- 행 수준 보안 (RLS) 적용

NotebookLM

3단계: 플래닝 (Planning) - 최적의 경로 탐색

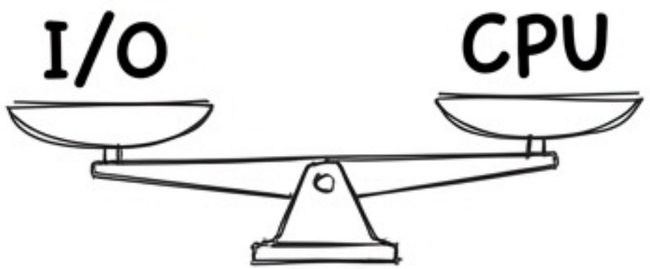


- SQL은 선언적 언어 (무엇을 vs 어떻게)
- 경로 탐색: Seq Scan vs Index Scan
- 조인 순서 및 알고리즘 결정
- 비용 기반 옵티마이저 (Cost-based Optimizer)

NotebookLM

플래너의 핵심: 비용 산정 (Cost Estimation)

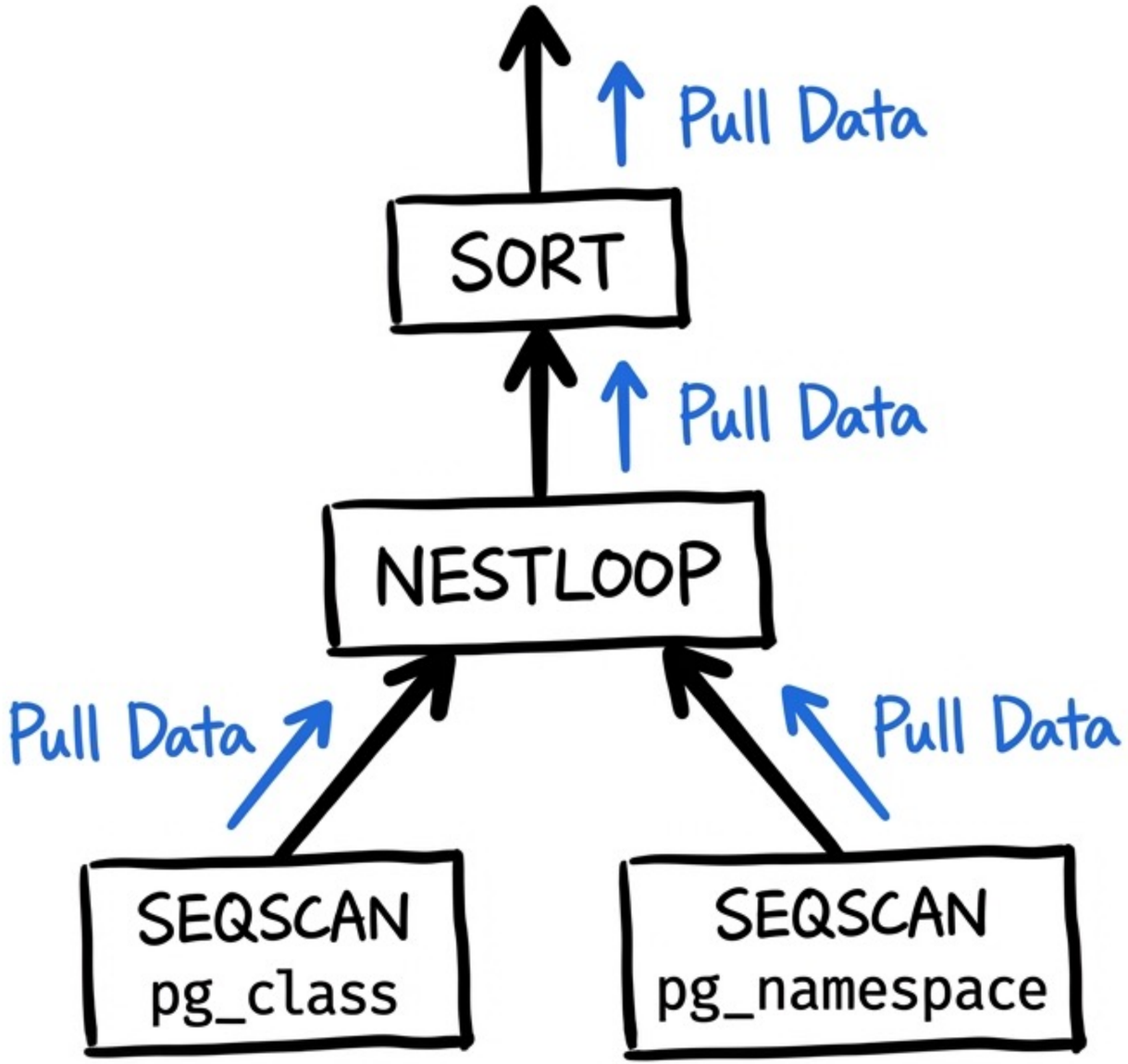
Cost = Startup Cost + Total Cost



- 비용 (Cost): 단위 없는 추정치
- 카디널리티 (Cardinality): 예상 행(Row) 수
- 선택도 (Selectivity): 필터 통과 비율 (0.0 ~ 1.0)
- 통계 정보 (Statistics): AutoVacuum이 수집

NotebookLM

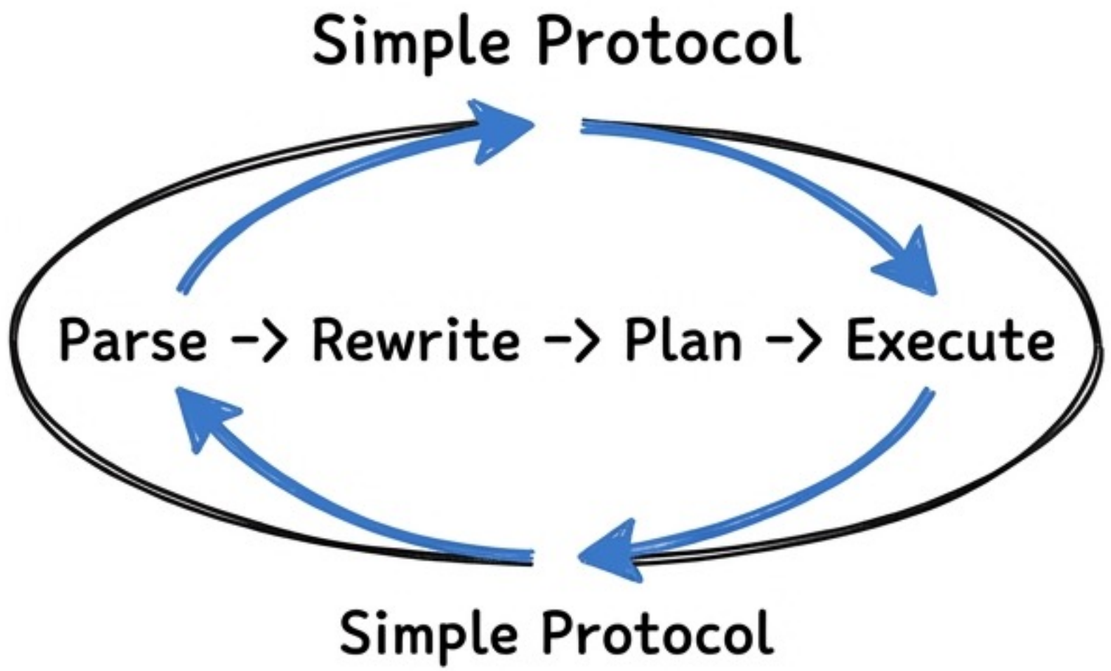
4단계: 실행 (Execution) - 파이프라인

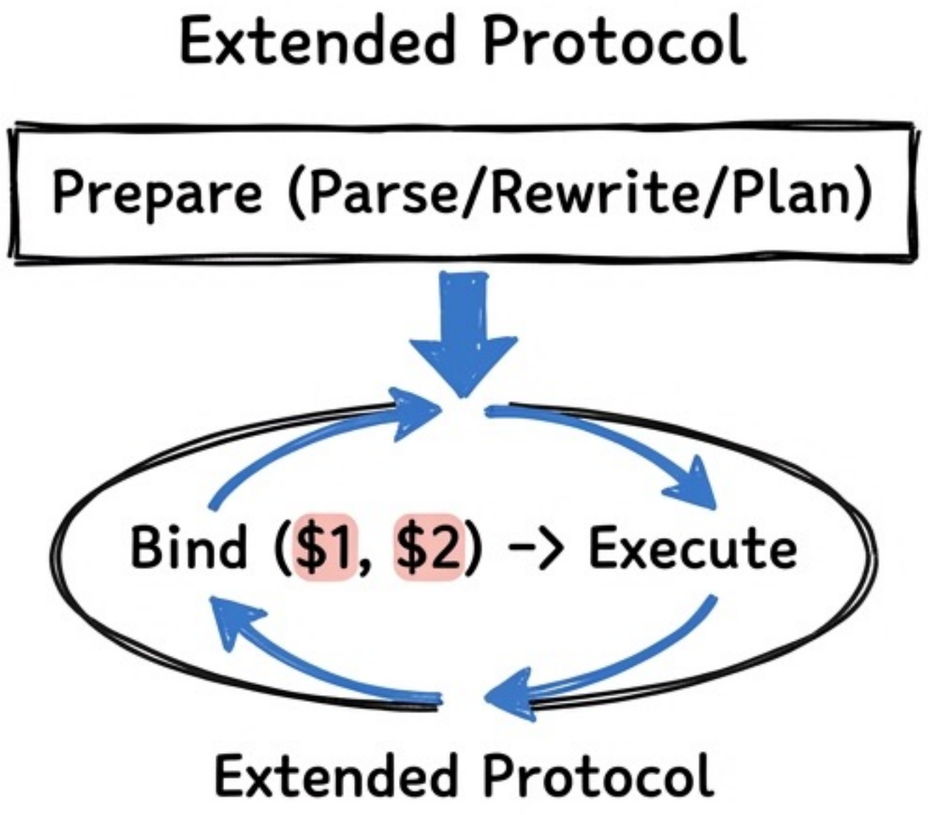


- 재귀적 호출 (Recursive Call)
- 포털 (Portal): 메모리 객체
- Work_mem: 정렬/해시 작업 공간
- 파이프라인 (Pipeline) 방식

NotebookLM

확장 쿼리 프로토콜 (Extended Query Protocol)

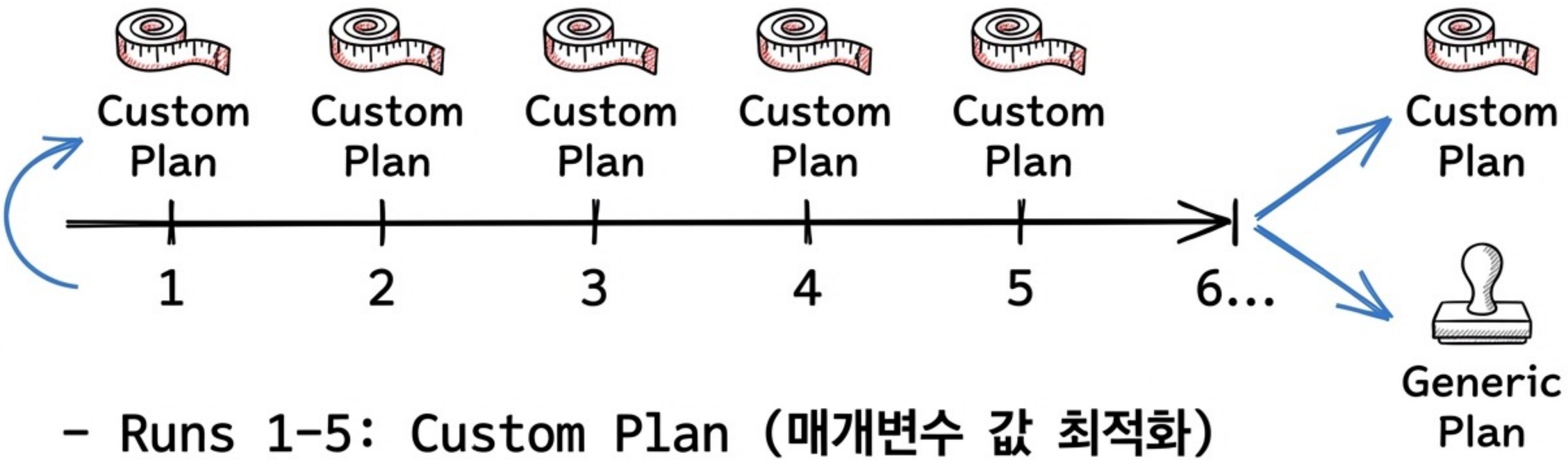




- Prepare: 파싱/계획 1회 수행
- Bind: 매개변수 바인딩 (SQL Injection 방지)
- Execute: 반복 실행 효율성

NotebookLM

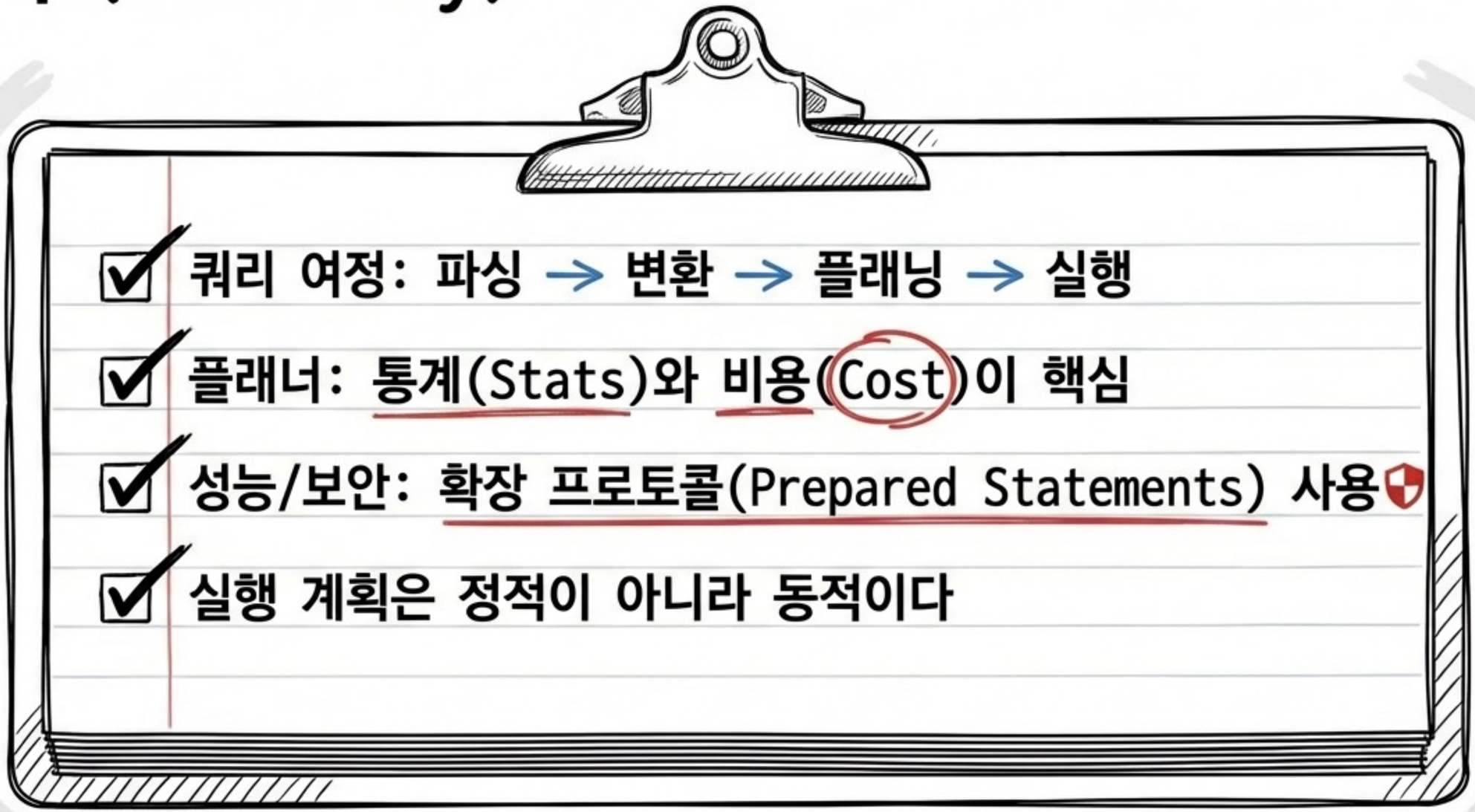
실행 계획 캐싱 (Plan Caching)



- Runs 1-5: Custom Plan (매개변수 값 최적화)
- Run 6+: Generic Plan 전환 고려
- 조건: Generic Plan 비용 < Custom Plan 평균 비용

NotebookLM

요약 (Summary)

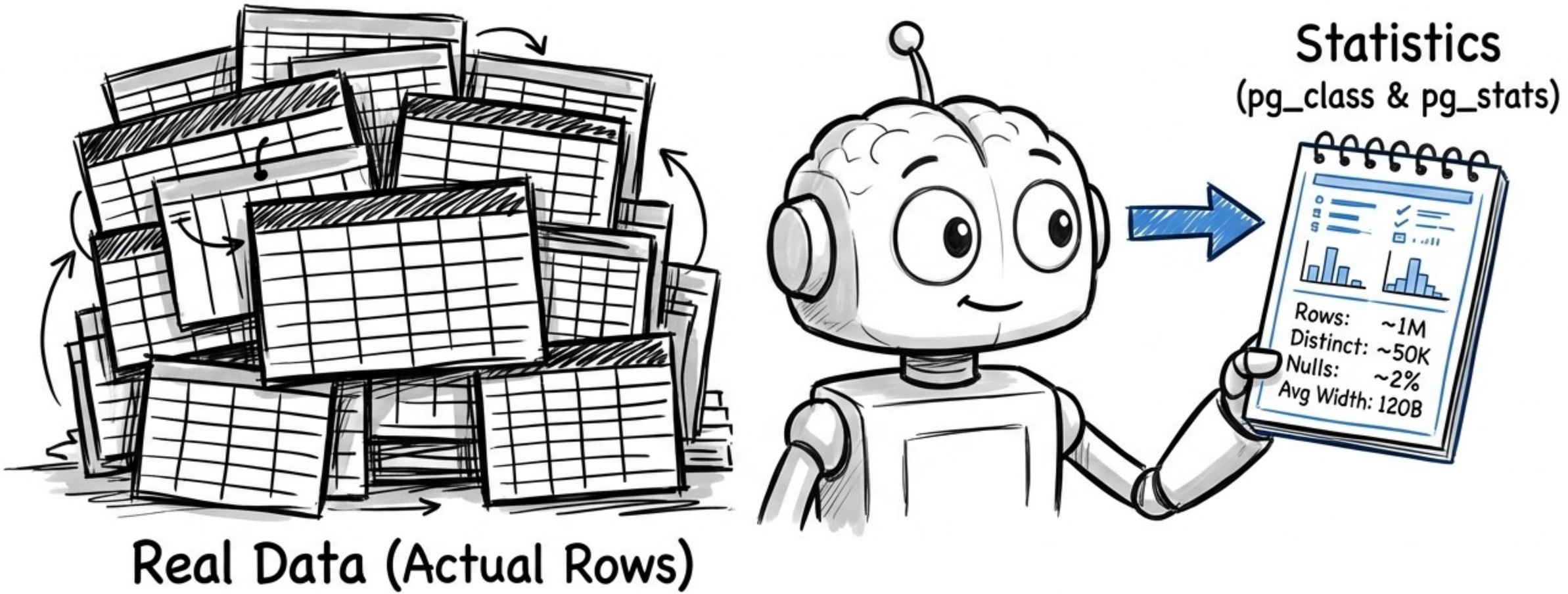


NotebookLM

17. Statistics

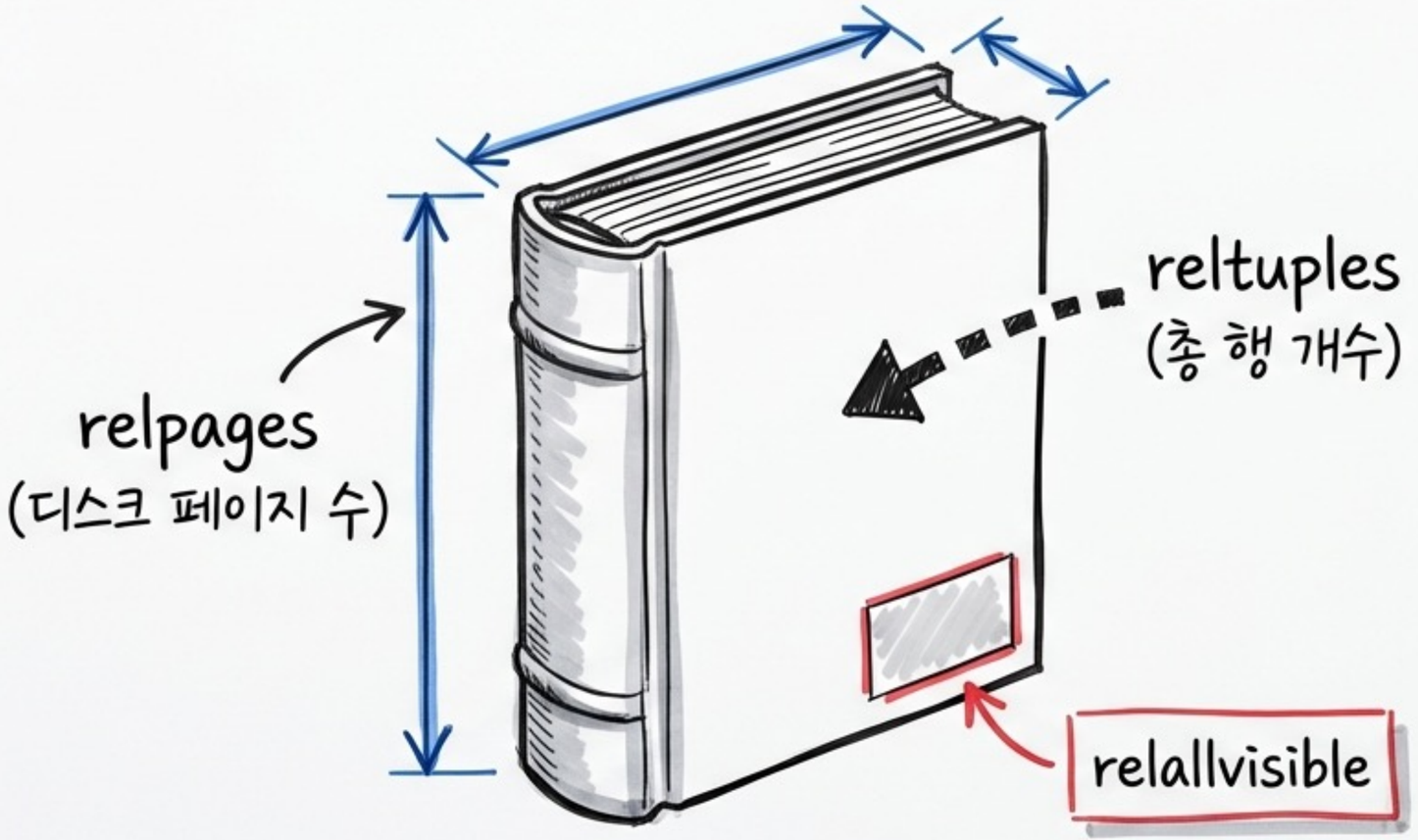
쿼리 최적화의 핵심은 '추정'입니다

- 플래너는 실제 데이터를 모두 읽지 않습니다. 시간이 너무 오래 걸리기 때문입니다.
- 대신 '통계(Statistics)'라는 요약된 정보를 참조하여 비용을 계산합니다.
- '지도는 영토가 아니다' — 통계는 현실의 근사치일 뿐입니다.



NotebookLM

가장 기초적인 통계: pg_class

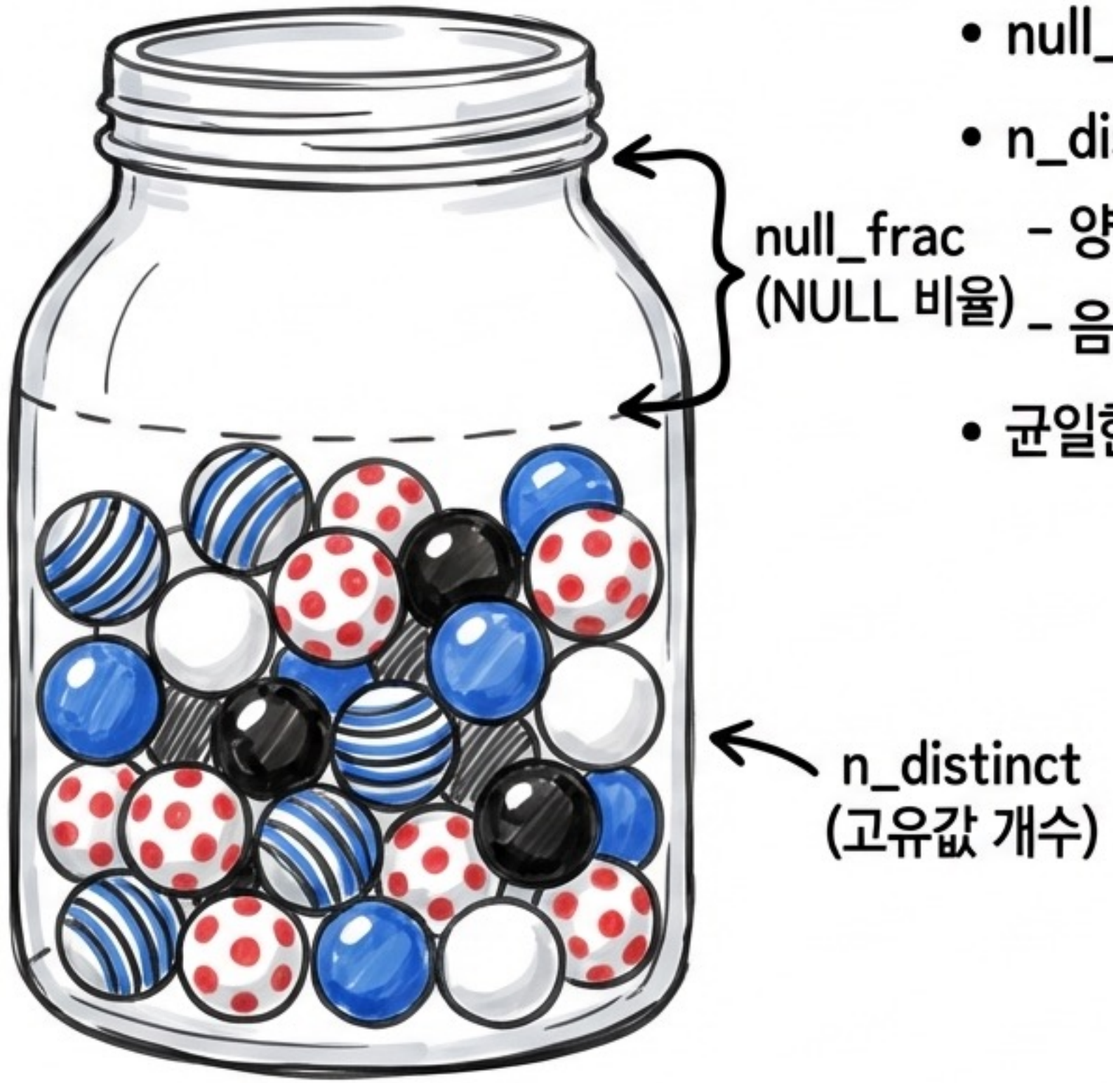


$$\text{Density} = \frac{\text{reltuples}}{\text{relpages}}$$

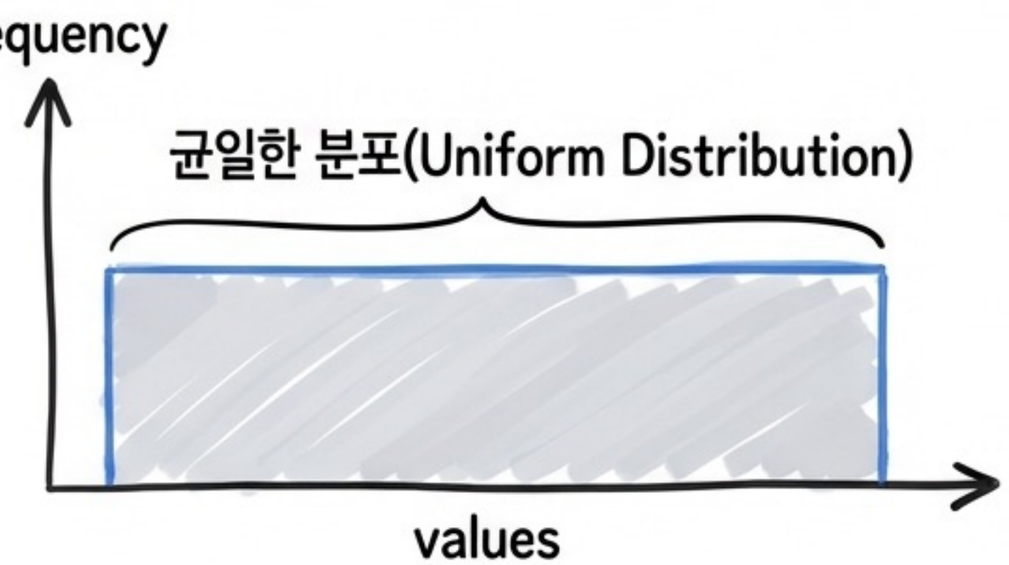
- reltuples: 테이블에 있는 행(Row)의 총 개수 추정치.
- relpages: 데이터가 차지하는 디스크 페이지 수.
- relallvisible: 모든 튜플이 트랜잭션에서 보이는 페이지 수.
- 이 값들은 VACUUM이나 ANALYZE 시점에 갱신됩니다.

NotebookLM

데이터의 불확실성: NULL과 고유값



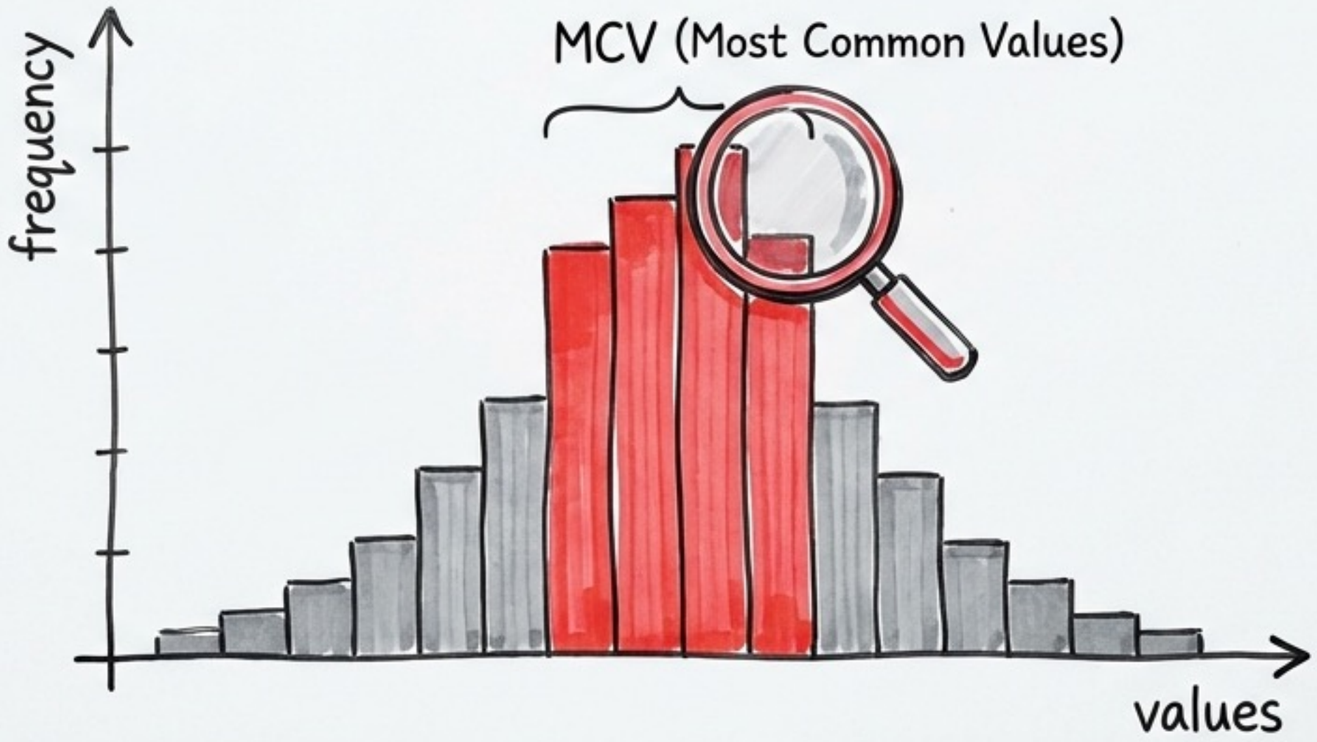
- null_frac: 컬럼 내 NULL 값의 비율.
- n_distinct: 고유한 값(Unique Value)의 개수.
  - 양수: 실제 고유값의 개수
  - 음수: 전체 행 수 대비 고유값의 비율 (예: -1 = All Unique)
- 균일한 분포(Uniform Distribution)를 가정할 때 주로 사용됩니다.



NotebookLM

데이터 분포가 치우쳤을 때: 최빈값(MCV)

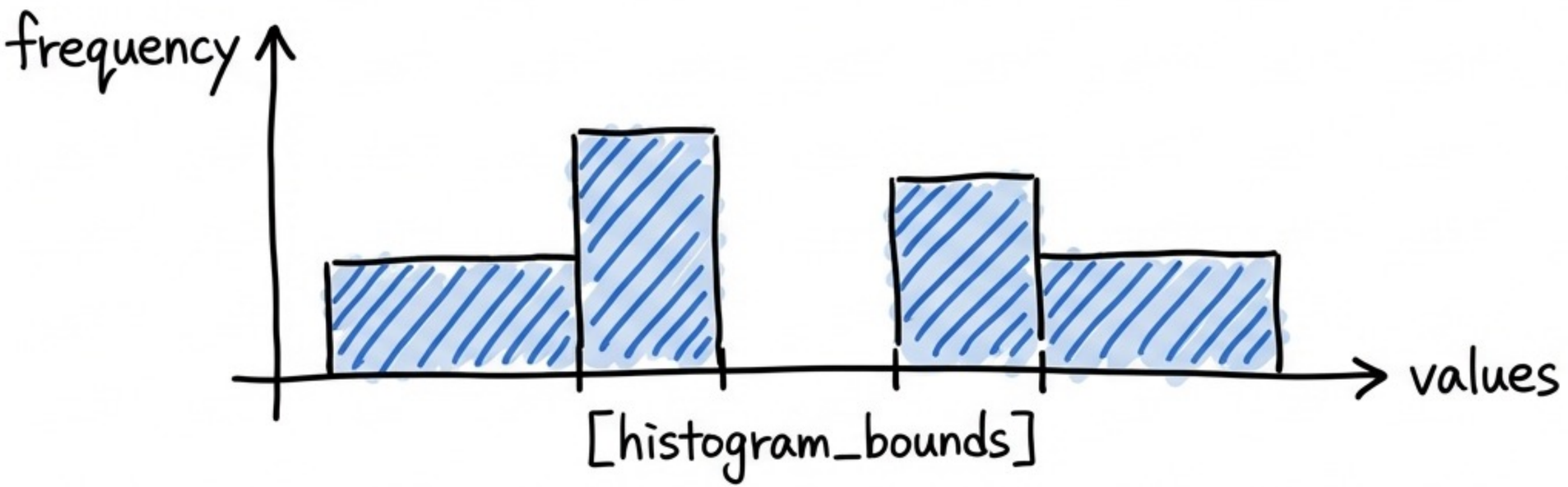
- MCV (Most Common Values): 데이터에서 가장 빈번하게 등장하는 값들의 리스트.
- MCF (Most Common Frequencies): 해당 값들이 등장하는 빈도.
- 플래너는 특정 값 조회 시 이 리스트를 가장 먼저 확인합니다.
- 분포가 불균형할 때 추정 정확도를 획기적으로 높여줍니다.



NotebookLM

나머지 데 데이터의 분포: 히스토그램

- MCV를 제외한 나머지 데이터는 버킷(Bucket)으로 나누어 관리합니다.
- Equi-depth 방식: 각 버킷은 동일한 비율의 데이터를 담도록 경계(histogram_bounds)가 설정됩니다.
- 범위 검색(>, <)의 선택도를 추정할 때 사용됩니다.

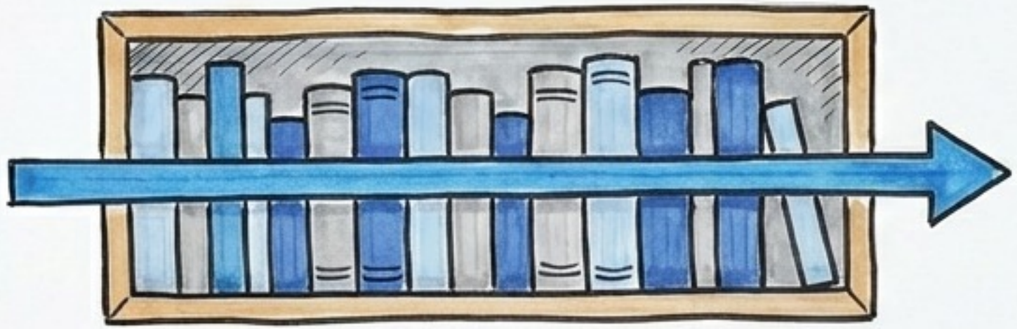


NotebookLM

물리적 저장 순서: 상관관계 (Correlation)

- Correlation (1.0 ~ -1.0): 논리적 순서와 물리적 디스크 저장 순서의 일치 정도.
- 1.0에 가까울수록: 인덱스 스캔 효율적 (순차적 I/O).
- 0에 가까울수록: 랜덤 I/O 발생 비용 증가 (Bitmap Scan이나 Seq Scan이 유리).

Correlation ≈ 1.0
(Fast Index Scan)



Correlation ≈ 0.0
(Random Access / Slow)



NotebookLM

컬럼 간의 관계: 다변량 통계 (Extended Statistics)

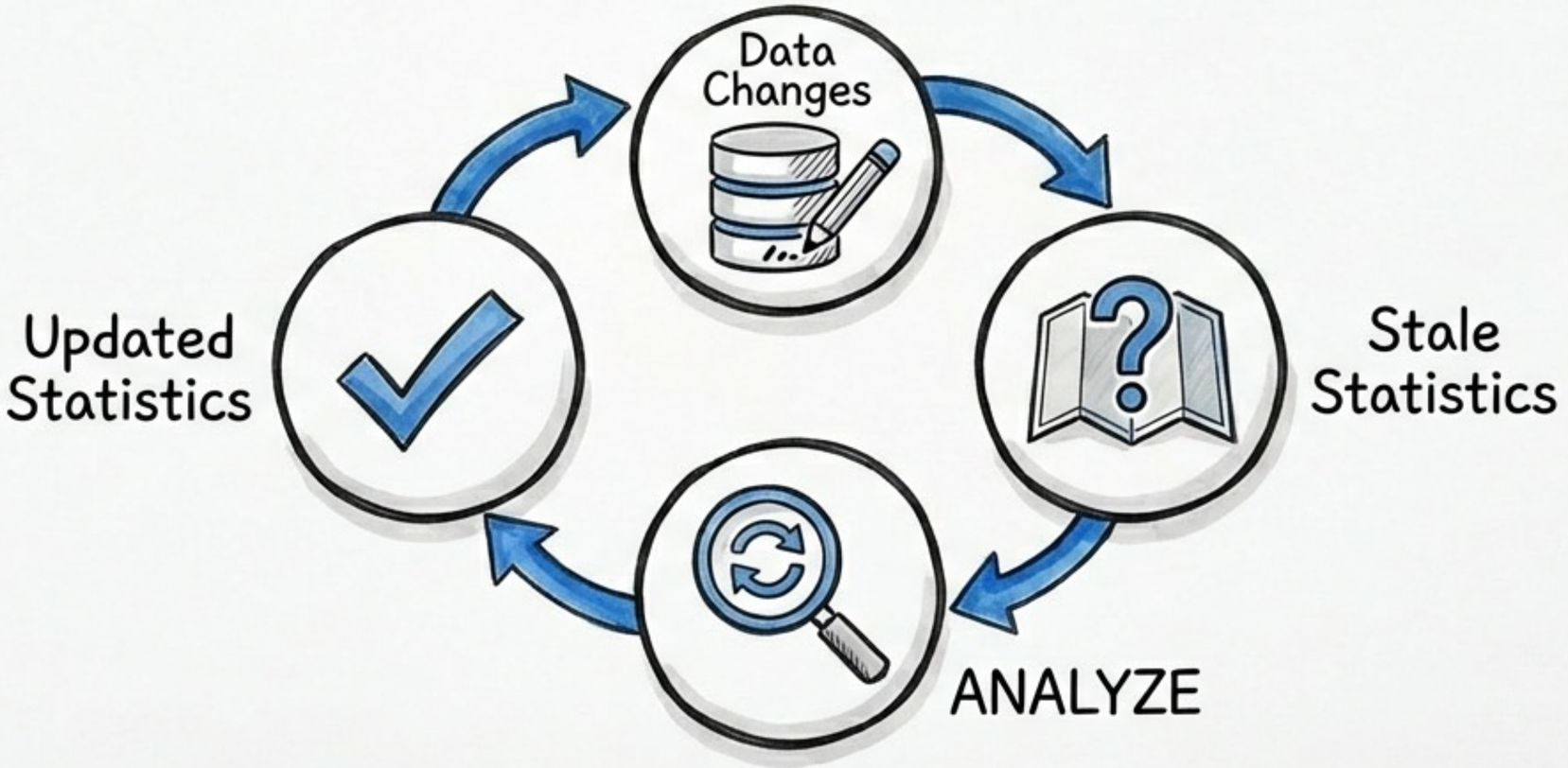
- 기본 통계는 컬럼들이 서로 독립적(Independent)이라고 가정합니다.
- 실제 데이터는 연관된 경우가 많습니다 (예: 항공편명 → 출발 공항).
- CREATE STATISTICS: 다변량 통계를 생성하여 조건절 결합 시 발생하는 과소평가(Underestimation) 오류를 방지합니다.



NotebookLM

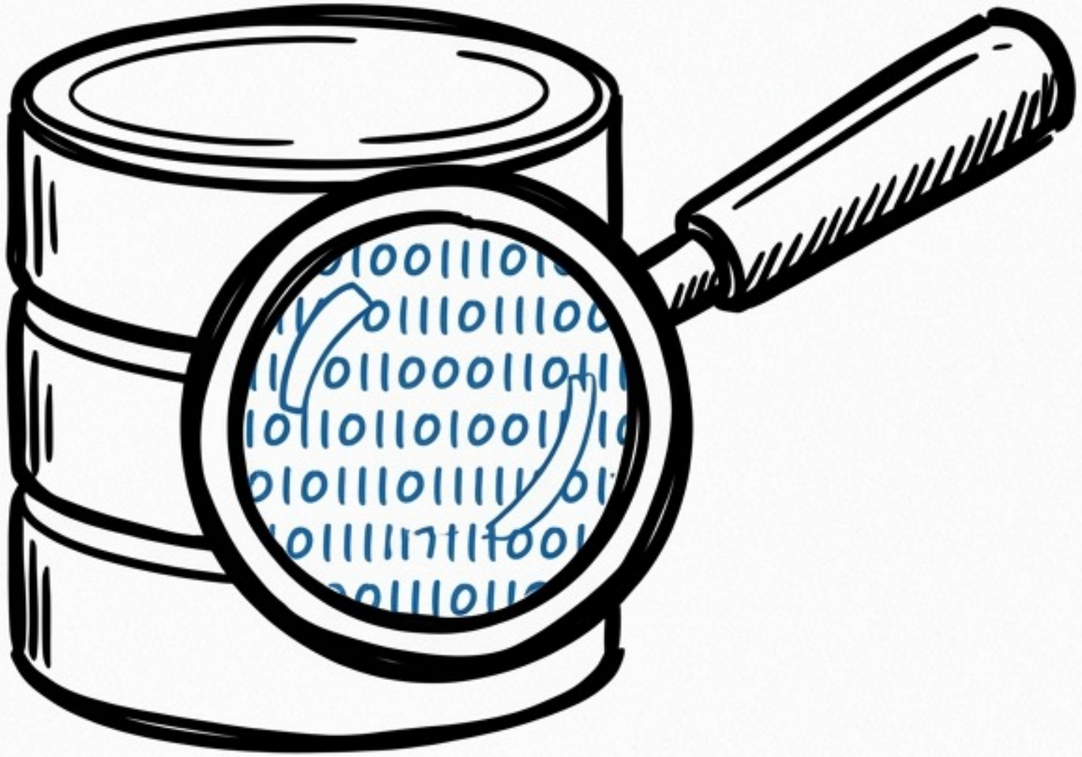
통계의 생명주기 관리

- ANALYZE: 통계 정보를 갱신하는 명령어 (Autovacuum이 자동 수행).
- default_statistics_target: 기본값 100. 값을 늘리면 샘플링(300 x target)이 증가하여 정밀도 향상.
- 결론: 정확한 통계가 최적의 성능을 결정합니다.



NotebookLM

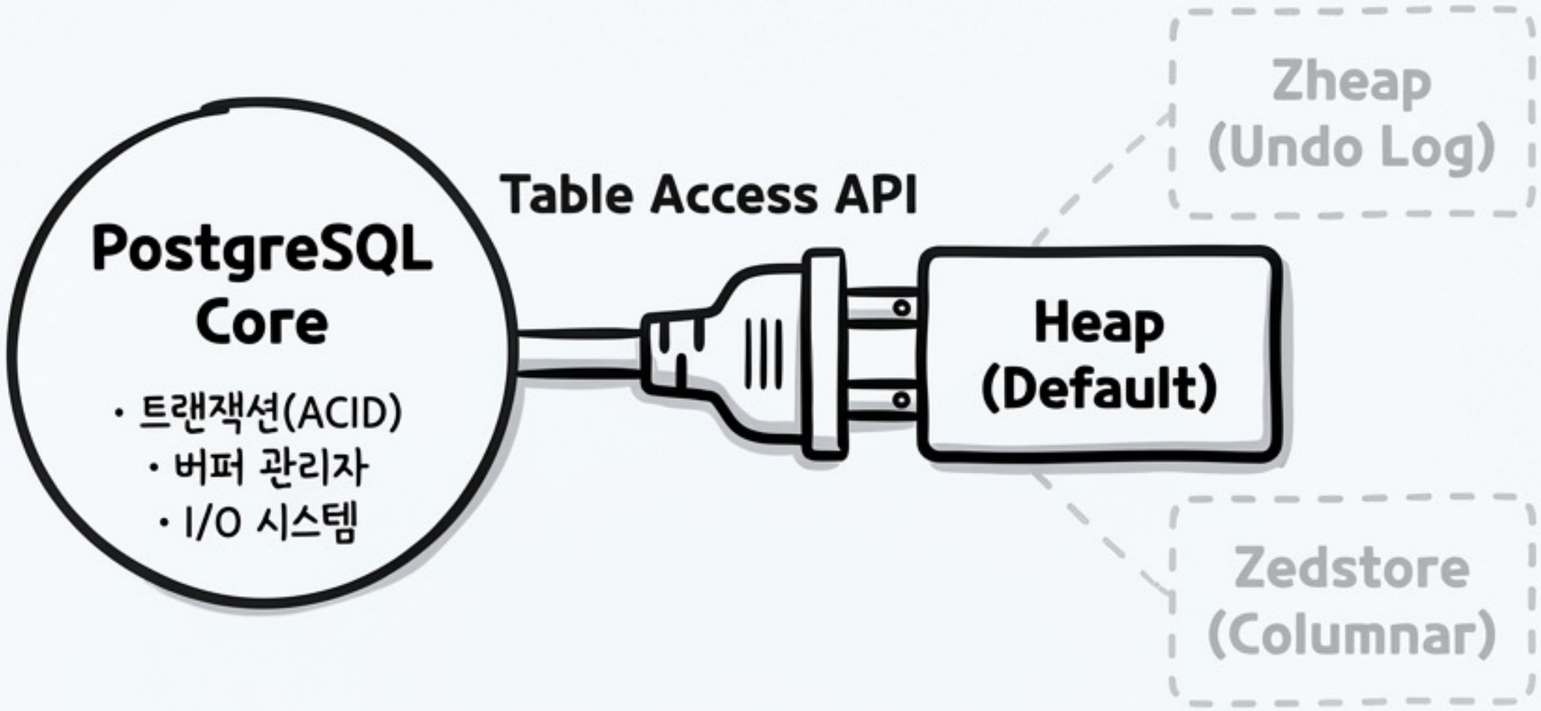
- **18. Table Access Methods**



PostgreSQL 테이블 액세스 방법

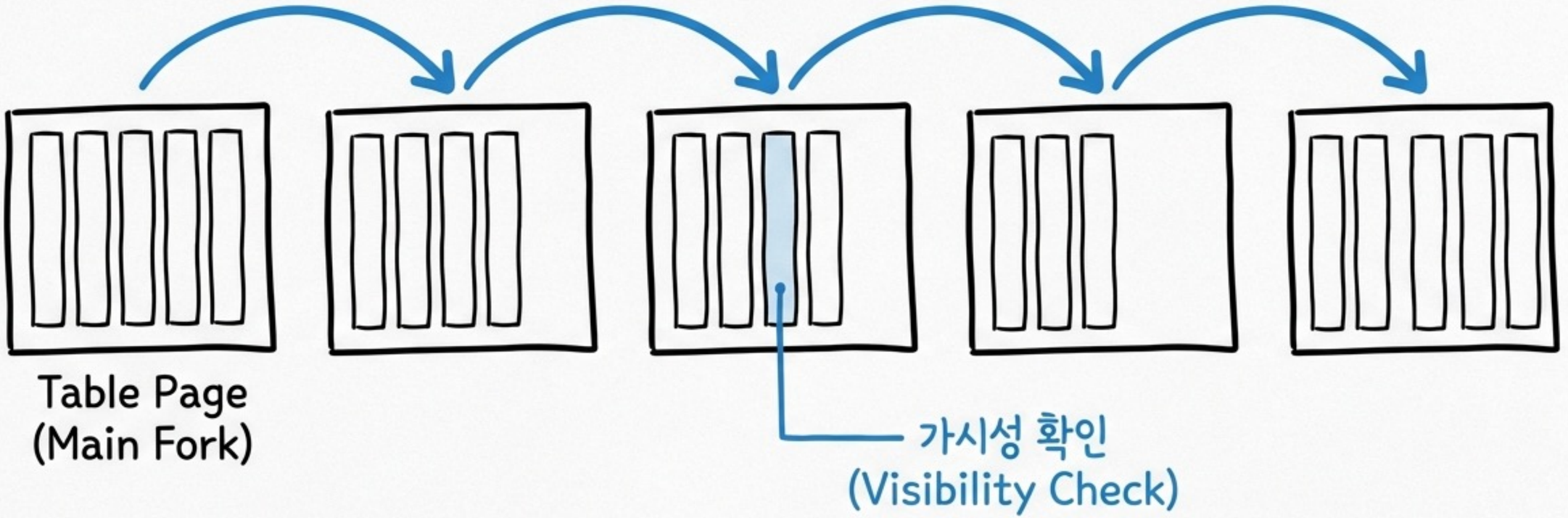
18장: 스토리지 엔진부터 병렬 스캔까지

NotebookLM



- 플러그형 아키텍처: 표준 인터페이스를 통해 다양한 엔진 교체 가능
- 기본 엔진 (Heap): 현재 표준이며, 대부분의 기능이 여기에 최적화됨
- 공유 구성요소: 트랜잭션, 버퍼 관리자 등은 엔진과 무관하게 코어에서 제공

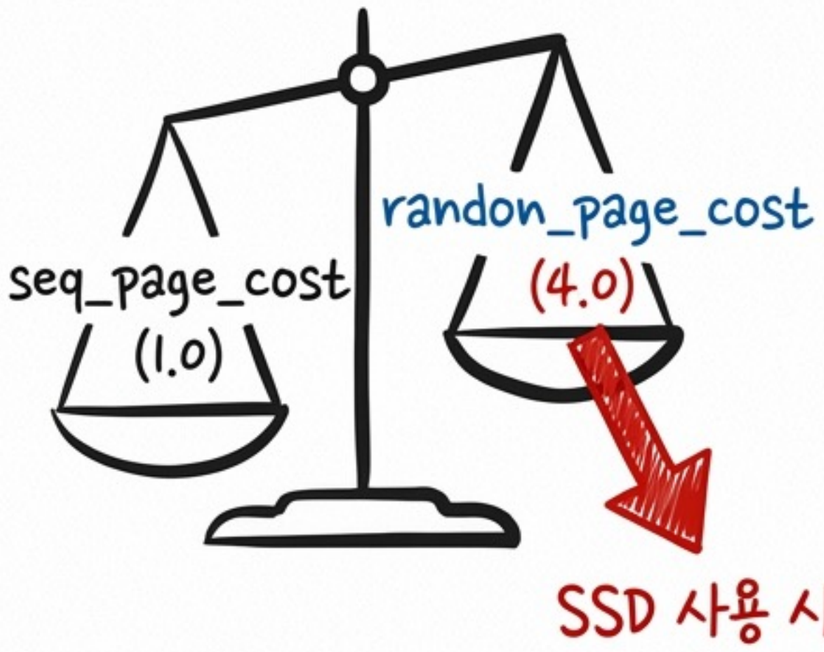
NotebookLM



**순차 스캔 (Sequential Scan)

- 테이블 파일(Main Fork)을 처음부터 끝까지 읽음
- 동기화된 스캔: 여러 프로세스가 버퍼 링(Buffer Ring)을 공유하여 I/O 절약
- 선택도(Selectivity)가 낮을 때, 즉 대량의 데이터를 읽을 때 가장 효율적

NotebookLM



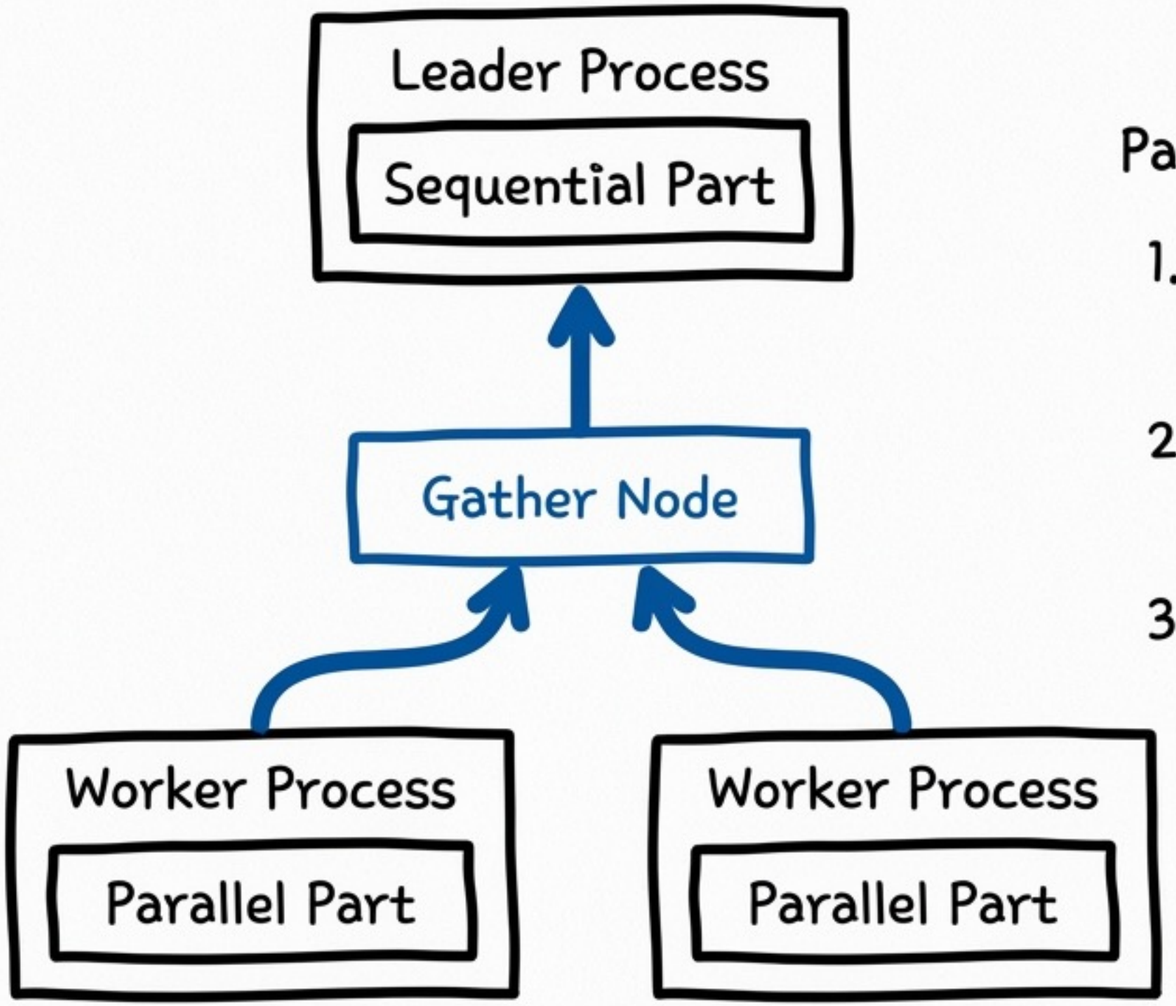
Total Cost = (Pages × I/O Cost)
+ (Tuples × CPU Cost)

I/O 비용: 순차 접근(1.0) vs 랜덤 접근(4.0)

CPU 비용: cpu_tuple_cost (기본값 0.01)

Insight: 테이블 페이지가 많을수록 스캔 비용은 선형적으로 증가

NotebookLM

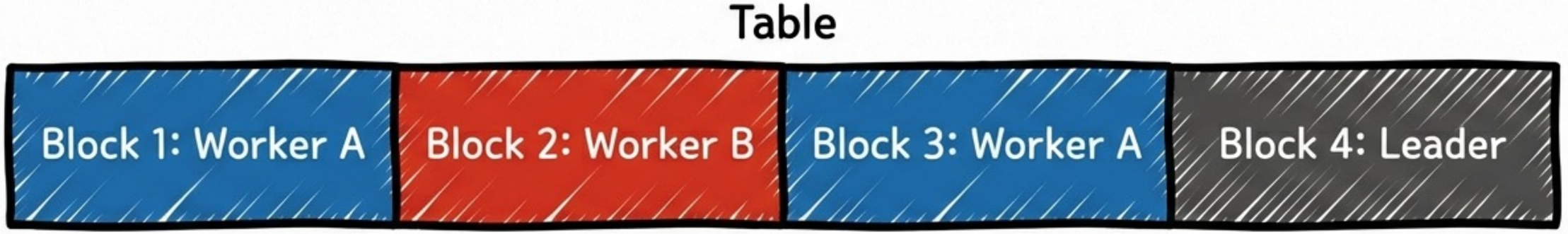


Parallel Execution Components

1. Leader
   병렬 계획 수립 및 실행 참여
2. Gather Node
   결과 수집 (Tuple Transfer)
3. Worker
   실제 쿼리 병렬 수행

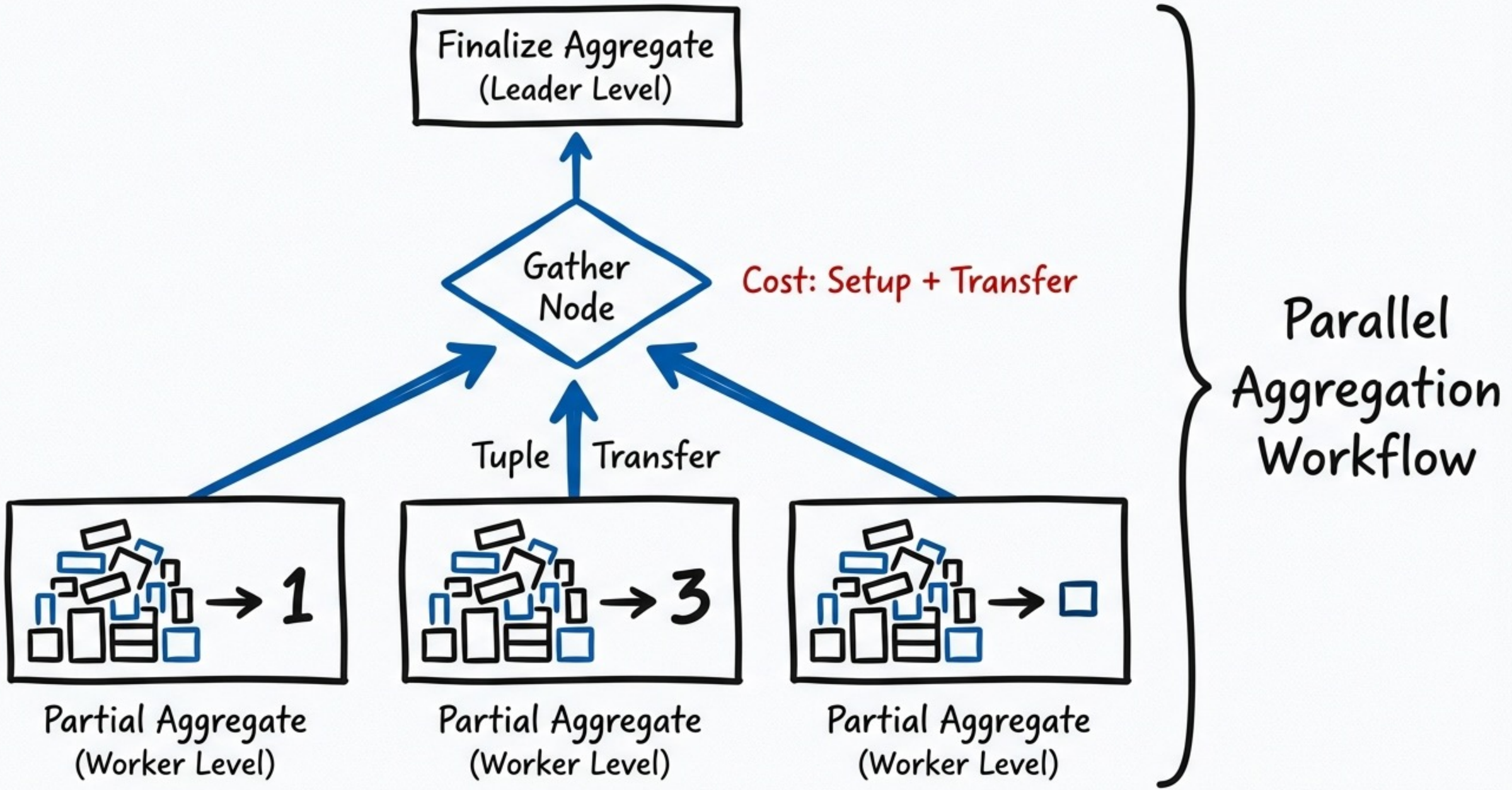
NotebookLM

병렬 순차 스캔 (Parallel Sequential Scan)



- 블록 할당 (Block Allocation): 중복 스캔 방지를 위해 연속된 페이지를 할당
- OS의 관점: 여러 프로세스가 동시에 읽으므로 Random I/O로 인식됨 (Prefetch 효과 감소)
- 효율성: 단순 스캔보다는 집계(Aggregation) 등 연산이 많을 때 유리

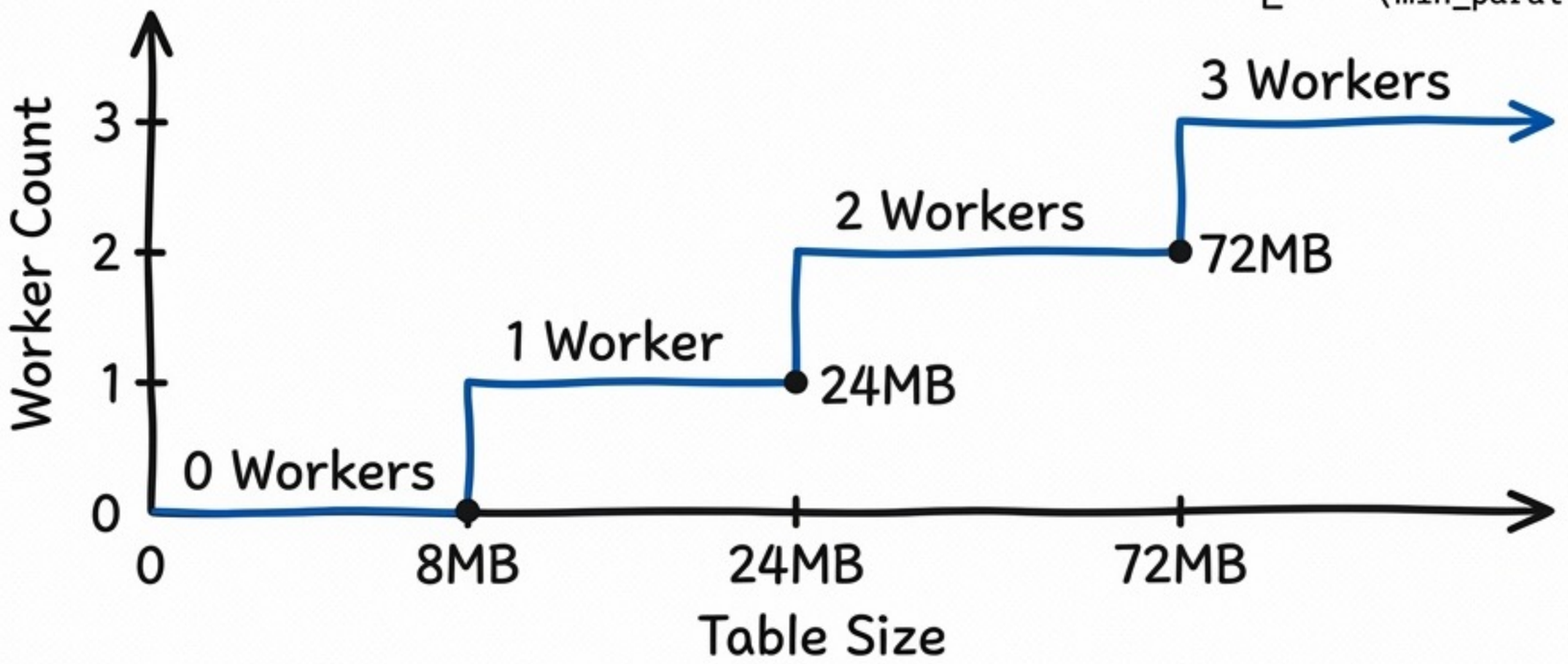
NotebookLM


Finalize Aggregate
(Leader Level)
Gather
Node
Cost: Setup + Transfer
Tuple
Transfer
→ 1
→ 3
→
Partial Aggregate
(Worker Level)
Partial Aggregate
(Worker Level)
Partial Aggregate
(Worker Level)
Parallel
Aggregation
Workflow

NotebookLM

병렬 실행 워커 수 결정 규칙 (Worker Allocation Rules)

$$1 + \left\lfloor \log_3 \left(\frac{\text{table size}}{\text{min_parallel_table_scan_size}} \right) \right\rfloor$$



- 최소 조건: 8MB 미만은 병렬 실행 안 함 (`min_parallel_table_scan_size`)
- 증가 규칙: 테이블 크기가 3배 증가할 때마다 워커 1명 추가 (Logarithmic)
- 최대 제한: `max_parallel_workers_per_gather` 설정값

NotebookLM

병렬 처리가 제한되는 경우 (Restrictions)



데이터 수정
(UPDATE/DELETE)
(단, 서브쿼리는 가능)

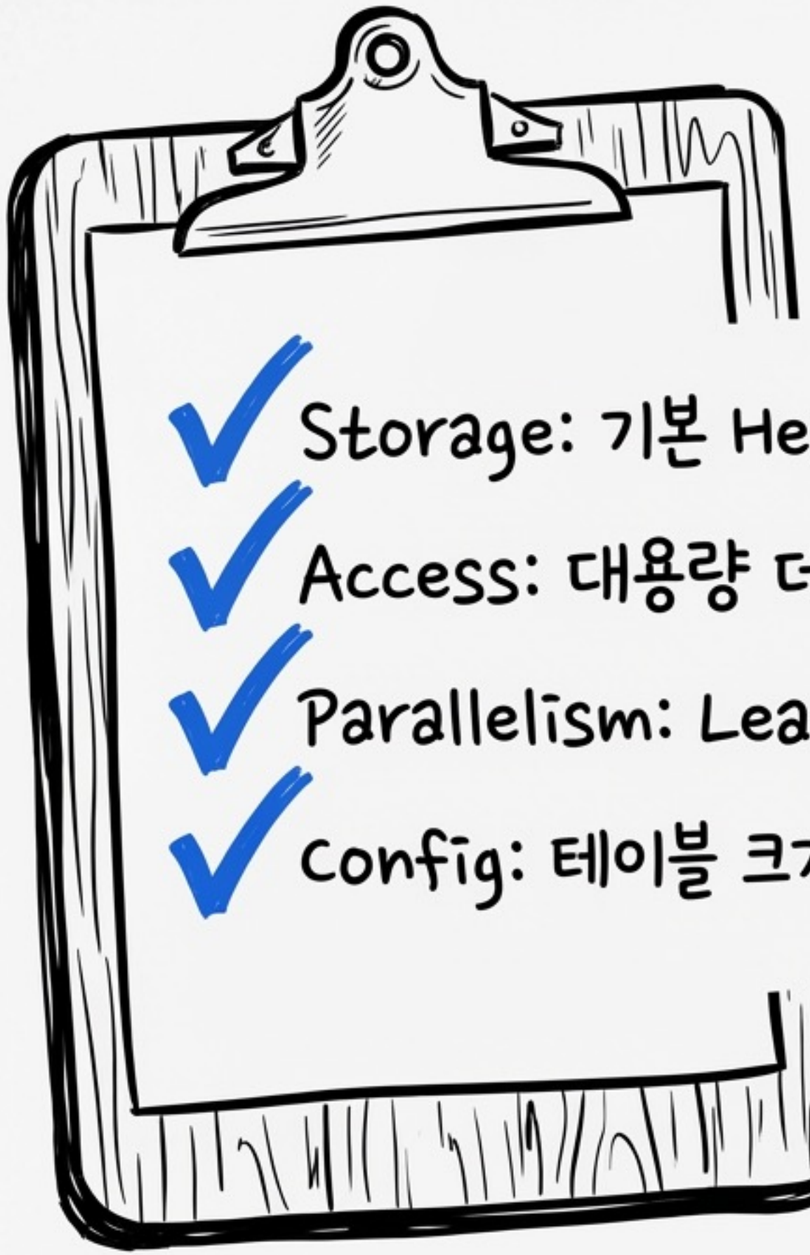


쿼리 제한
임시 테이블(Temp Table)
커서(Cursor) / 루프



함수 제한
PARALLEL UNSAFE Functions

NotebookLM



- ✓ Storage: 기본 Heap 엔진과 플러그형 아키텍처
- ✓ Access: 대용량 데이터는 순차 스캔(Seq Scan)이 유리
- ✓ Parallelism: Leader-Worker 구조로 스캔 속도 가속화
- ✓ Config: 테이블 크기(8MB 규칙)에 따라 워커 수 자동 결정



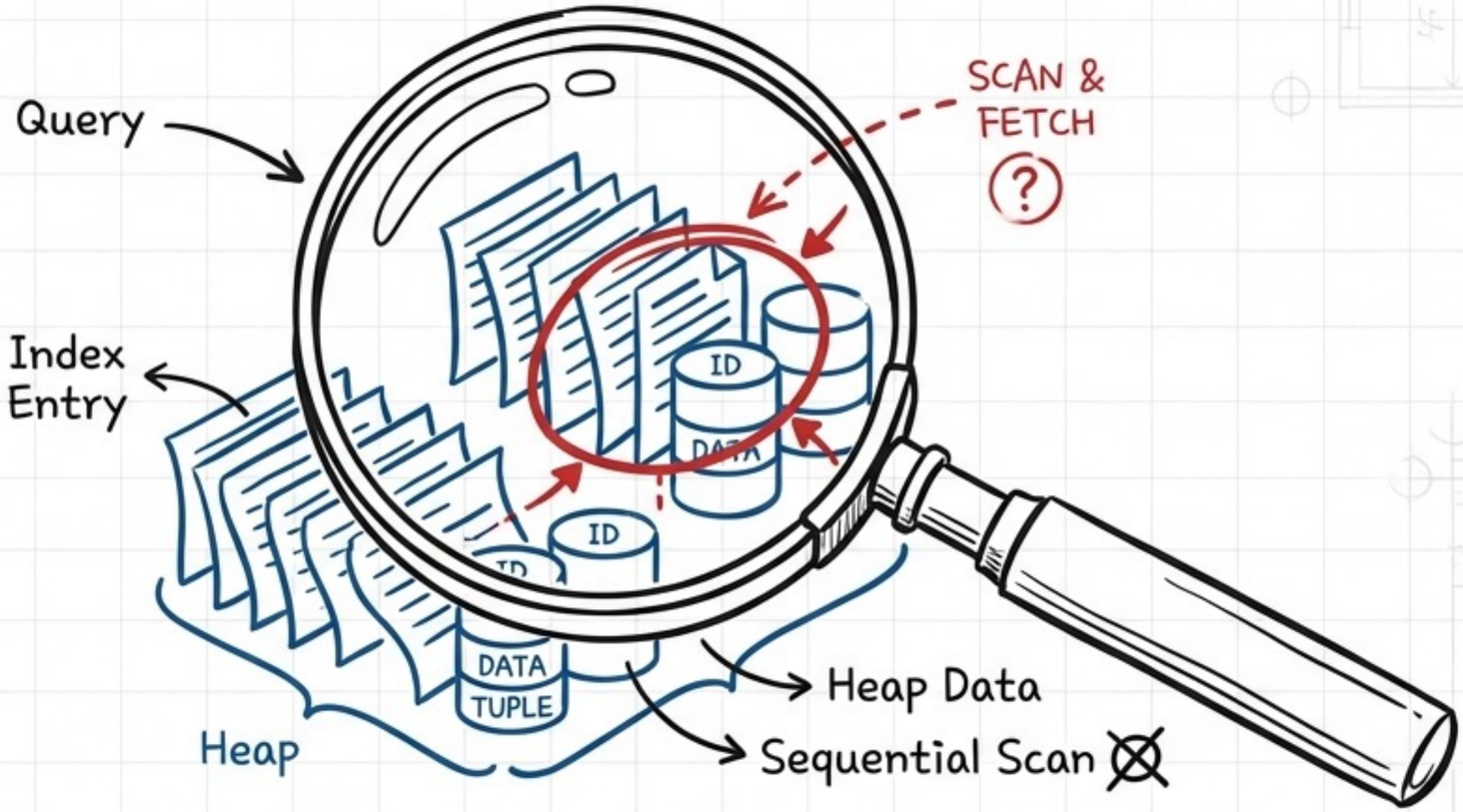
End of Chapter 18

NotebookLM

- **19. Index Access Methods**

PostgreSQL 인덱스 액세스 방법 (Index Access Methods)

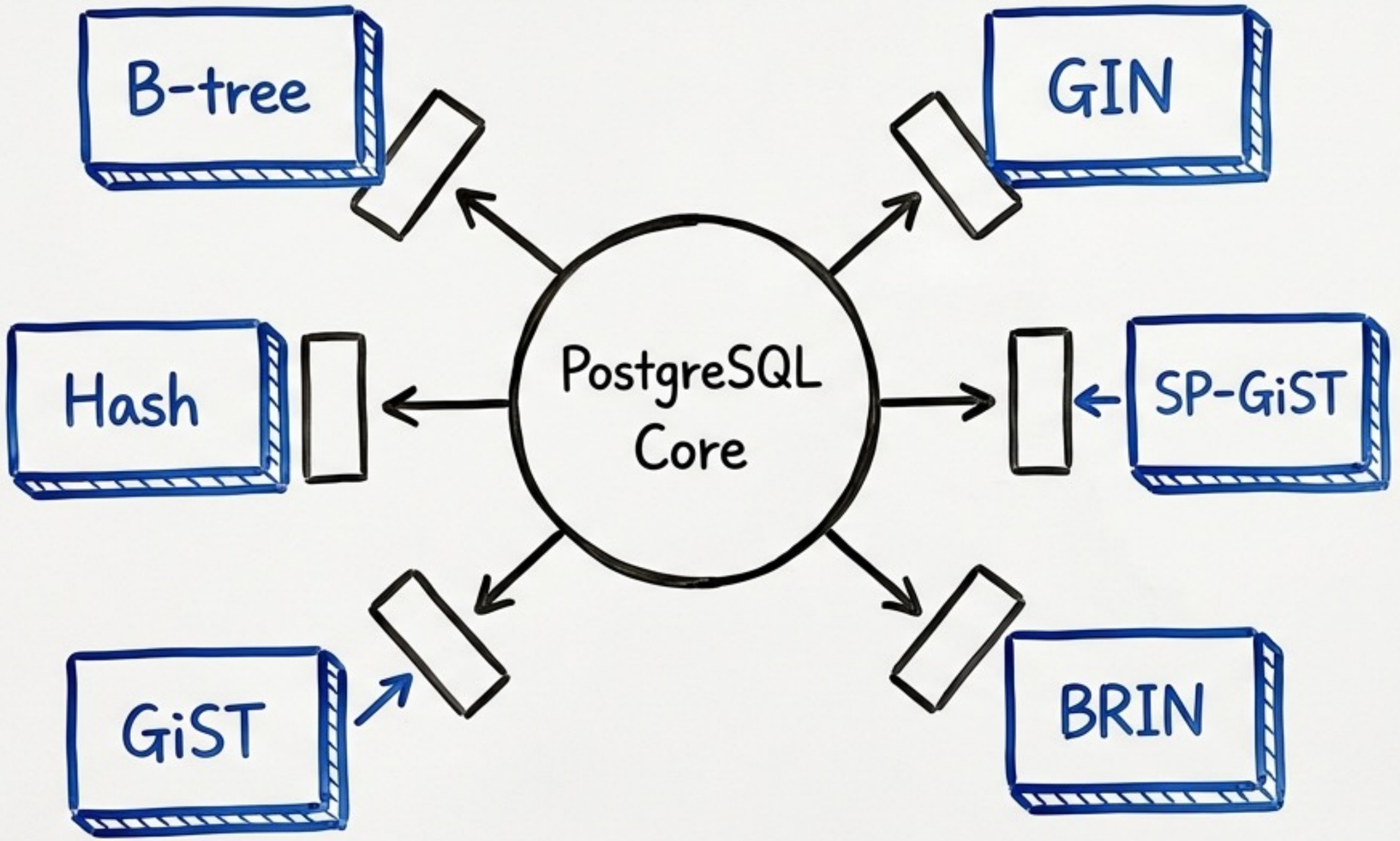
19장: 인덱싱 엔진과 확장성의 이해



PostgreSQL Internal Architecture Series

NotebookLM

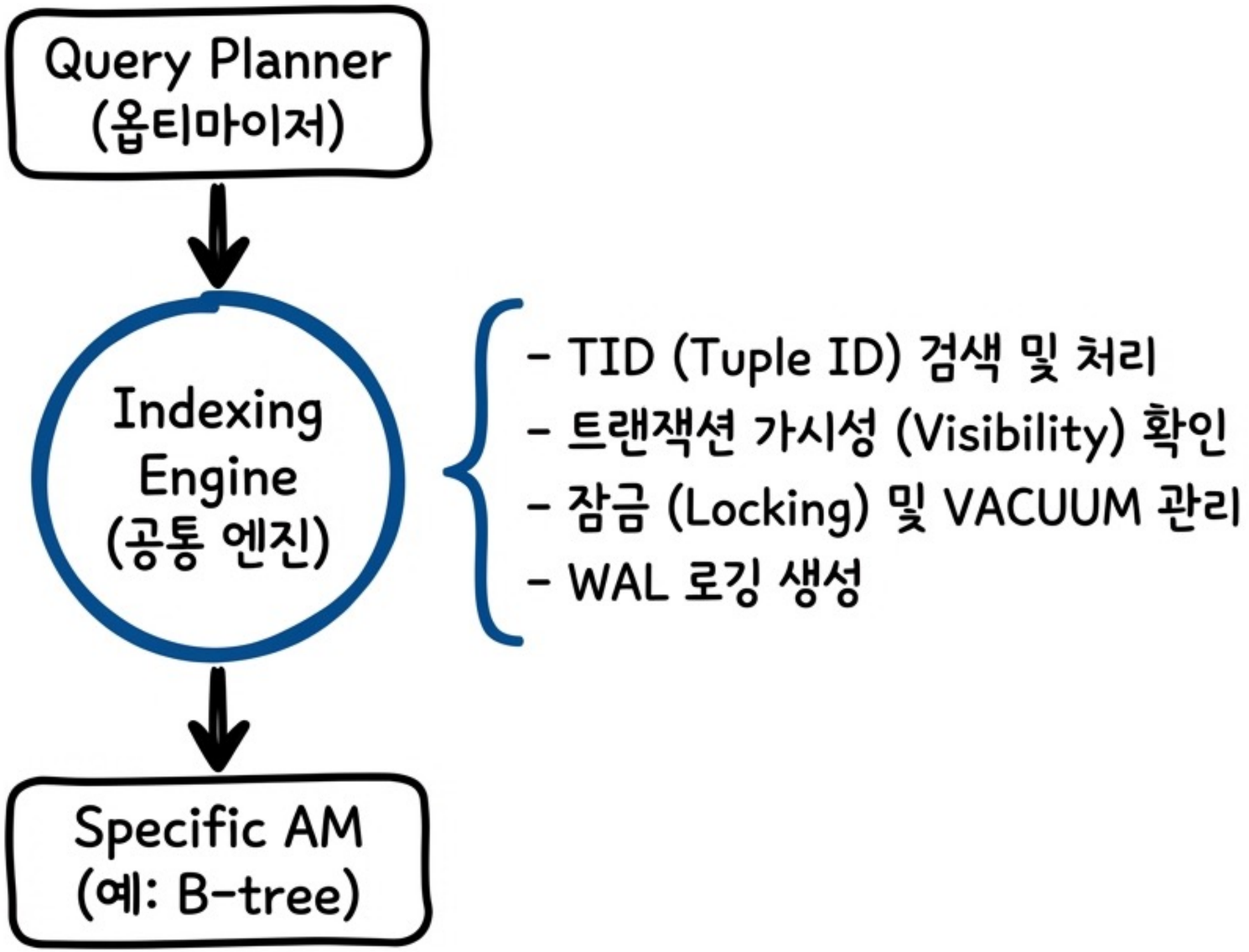
인덱스는 '플러그인' 구조입니다



- 확장성 (Extensibility): 코어를 수정하지 않고 새로운 방식을 추가 가능
- 6가지 내장 메소드: pg_am 시스템 카탈로그에 정의됨
- 역할: 힙(Heap) 데이터를 기반으로 한 보조 구조물 (언제든 재생성 가능)

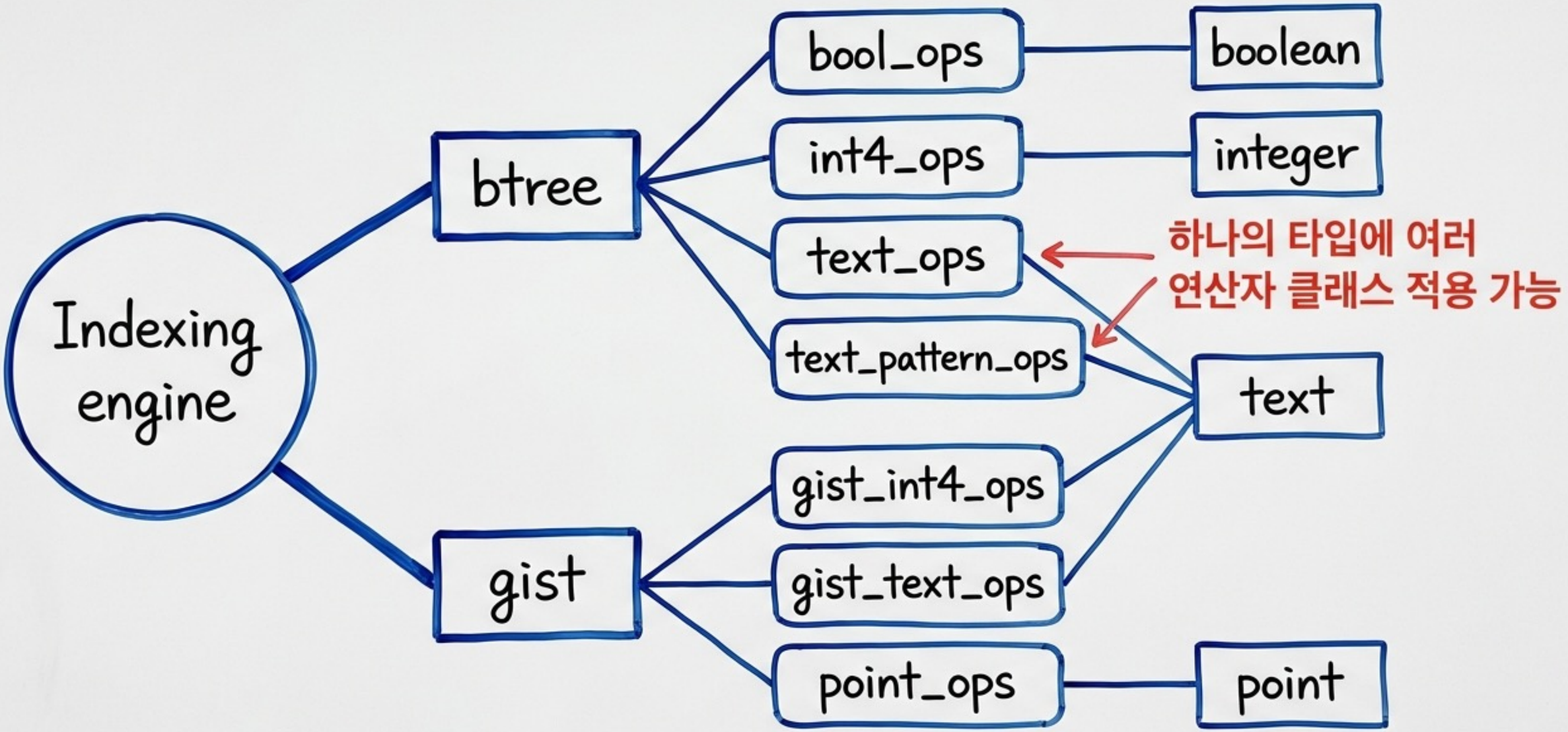
NotebookLM

공통 인덱싱 엔진의 역할



NotebookLM

데이터와 인덱스의 연결고리: 연산자 클래스 (Operator Class)



NotebookLM

전략 번호 (Strategy Number)와 의미론

B-tree Strategy Mapping

Strategy	Meaning
Strategy 1	작다 (<)
Strategy 2	작거나 같다 (<=)
Strategy 3	같다 (=)
Strategy 4	크거나 같다 (>=)
Strategy 5	크다 (>)



Index Method

인덱스는 연산자 이름(예: <, >)을 직접 알지 못합니다.

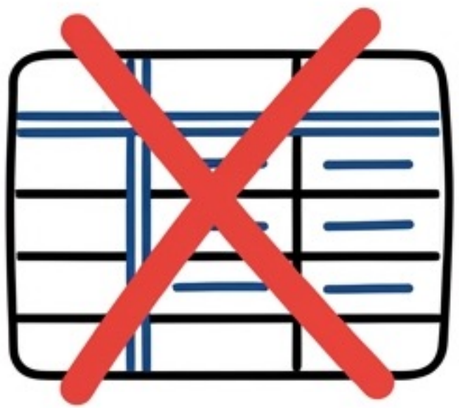
대신, 추상화된 '전략 번호'를 통해 논리적 동작을 수행합니다.

NotebookLM

사례 연구: text_ops vs text_pattern_ops

text_ops (Standard)

LIKE 'ELENA%'



Locale 설정에 따라
인덱스 사용 불가

text_pattern_ops

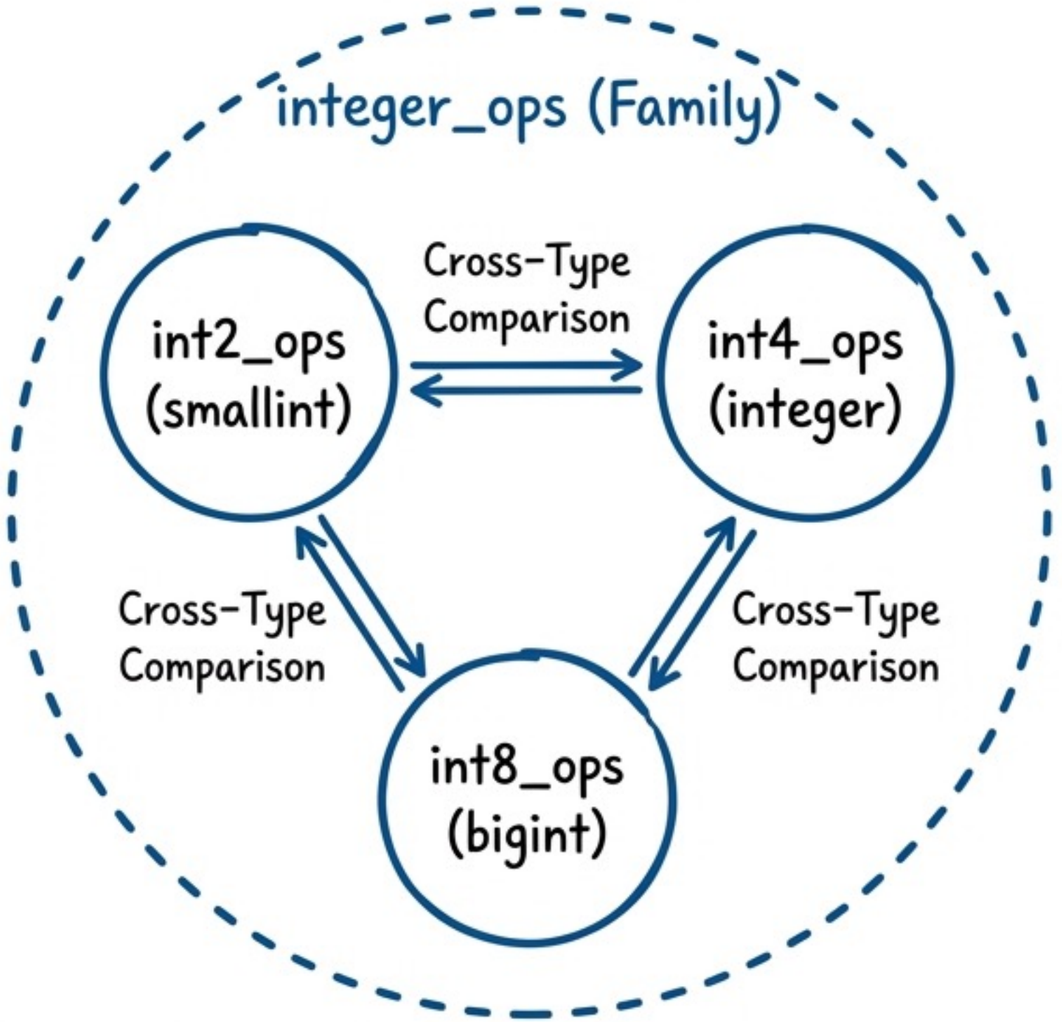
LIKE 'ELENA%'

(name ~>=~ 'ELENA')
AND
(name ~<~ 'ELENB')

Collation 무시,
문자 단위 비교 성공

NotebookLM

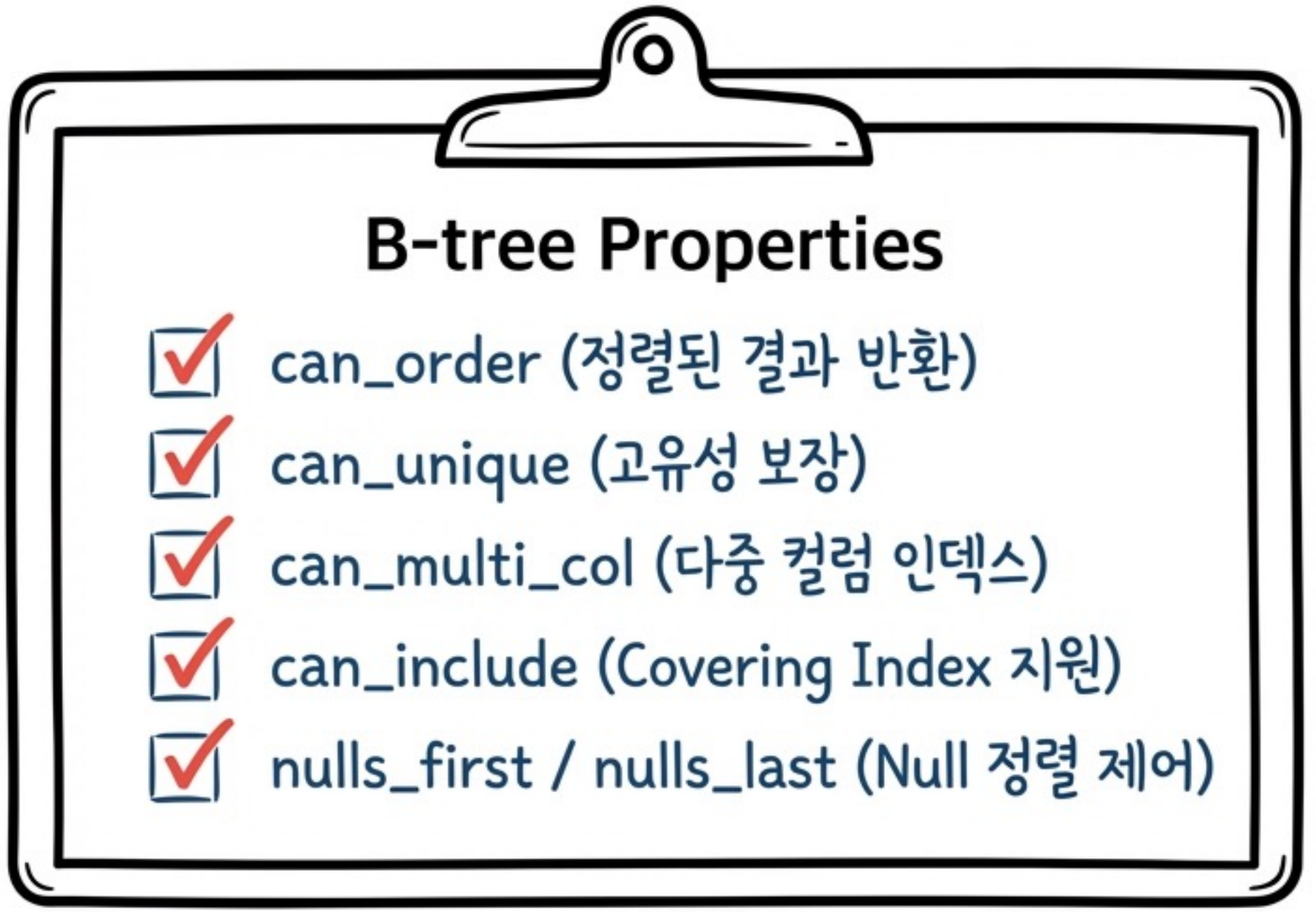
연산자 패밀리 (Operator Families)



- 유사한 데이터 타입들을 묶어주는 그룹
- 장점: 형변환(Casting) 없이 인덱스를 효율적으로 사용하여 비교 가능 (예: int4 = int8)

NotebookLM

액세스 메소드의 속성 (Access Method Properties)

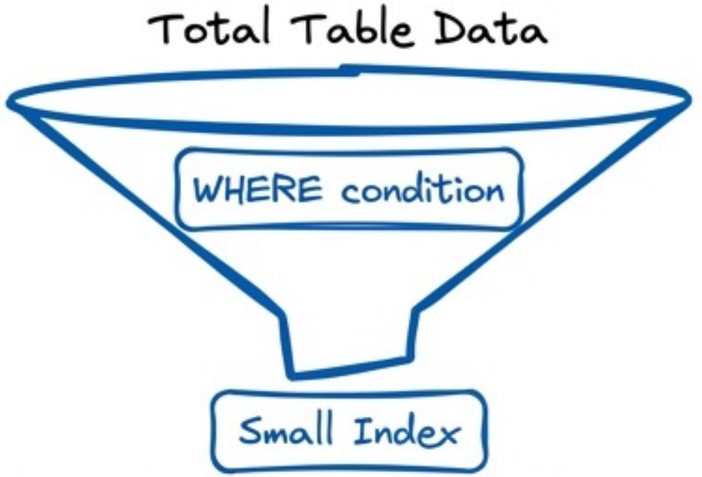


옵티마이저는 pg_indexam_has_property 함수를 통해
이 능력들을 확인하고 실행 계획을 짭니다.

NotebookLM

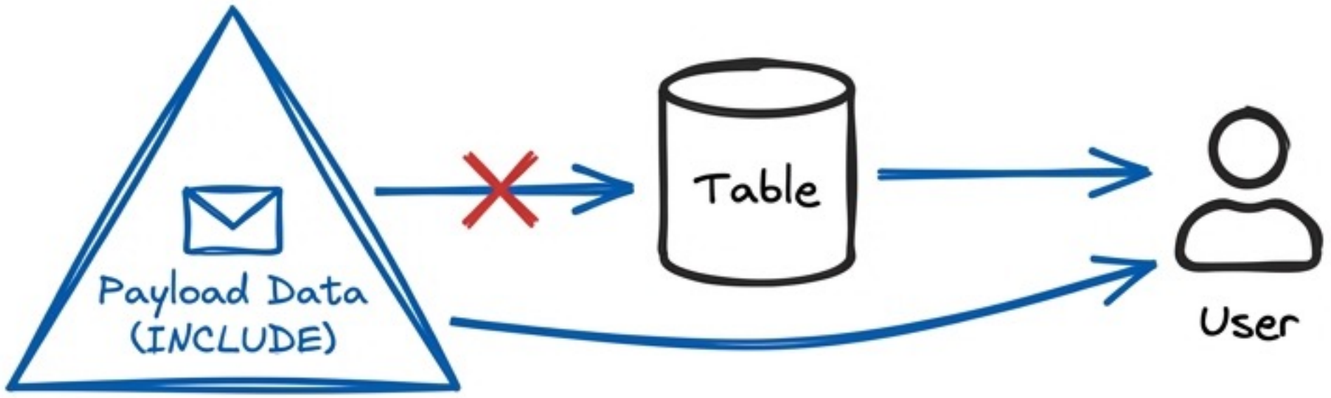
고급 최적화 기능

부분 인덱스 (Partial Index)



필요한 행만 인덱싱하여 공간 절약

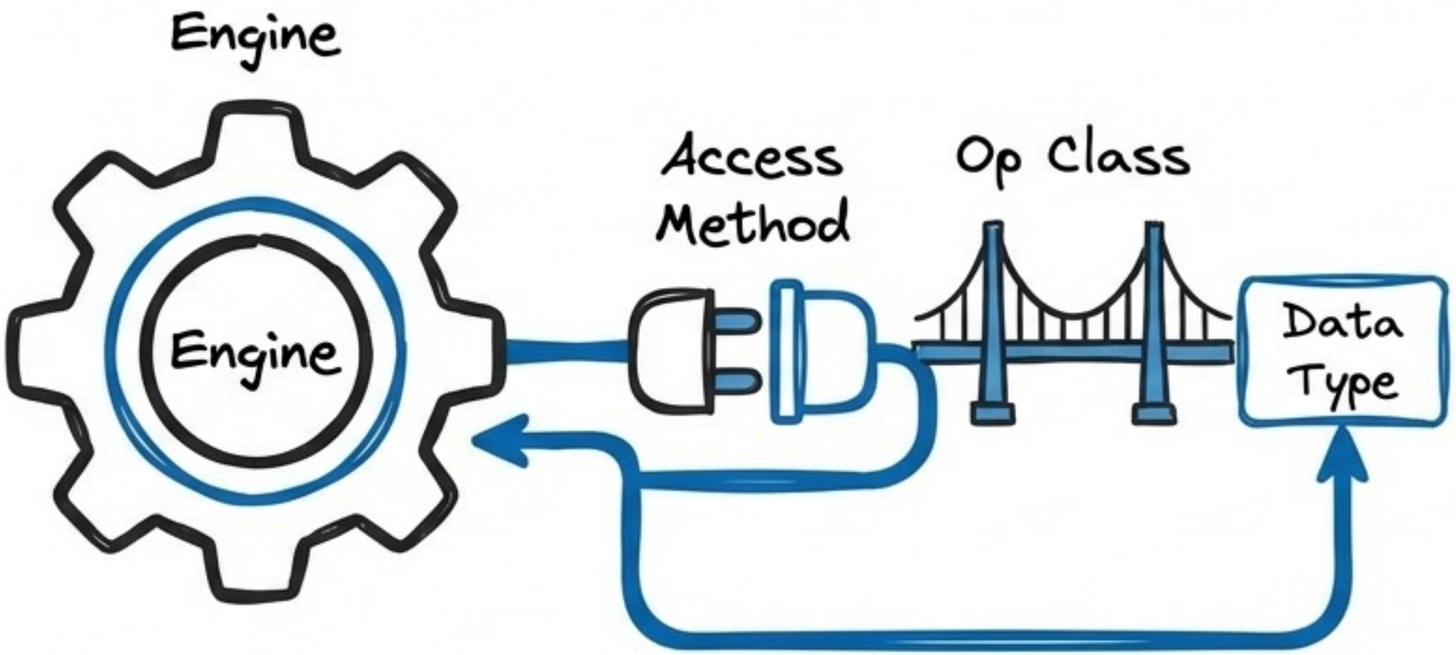
커버링 인덱스 (Index-Only Scan)



INCLUDE 절을 사용하여
힙 테이블 접근 없이 데이터 반환

NotebookLM

요약: 확장 가능한 아키텍처의 힘



1. **유연성** (Flexibility): 코어 수정 없이 **새로운 인덱싱 알고리즘** 수용.

2. **표준화** (Standardization): 모든 메소드는 WAL, Locking 등 **안전한 공통 인터페이스** 사용.

3. **다양성** (Diversity): 데이터 특성에 맞는 **최적의 도구(B-tree, BRIN, GiST)** 선택 가능.

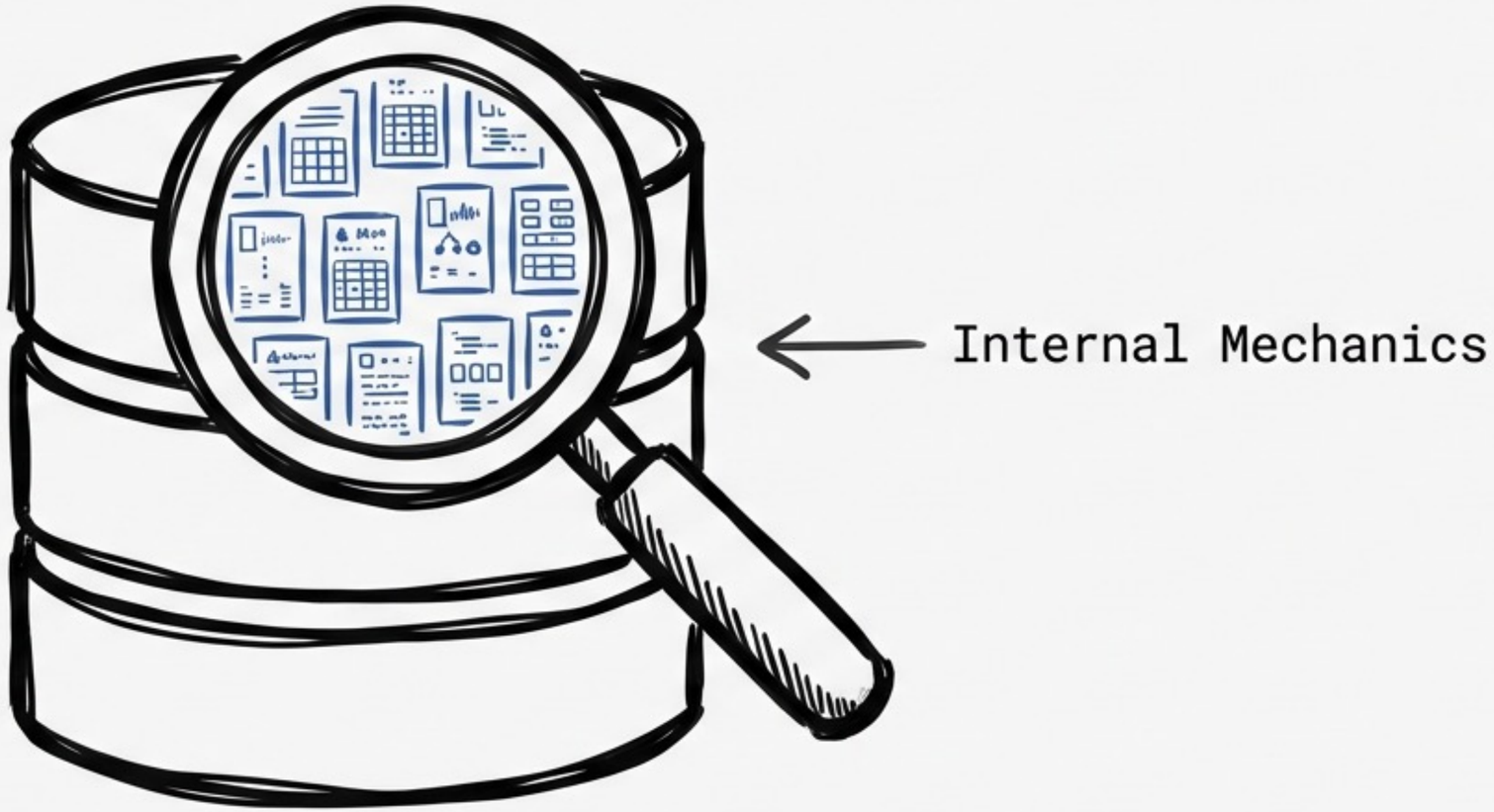
인덱스는 단순한 검색 도구가 아니라,
데이터 무결성과 성능을 지탱하는 프레임워크입니다.

NotebookLM

20. Index Scans

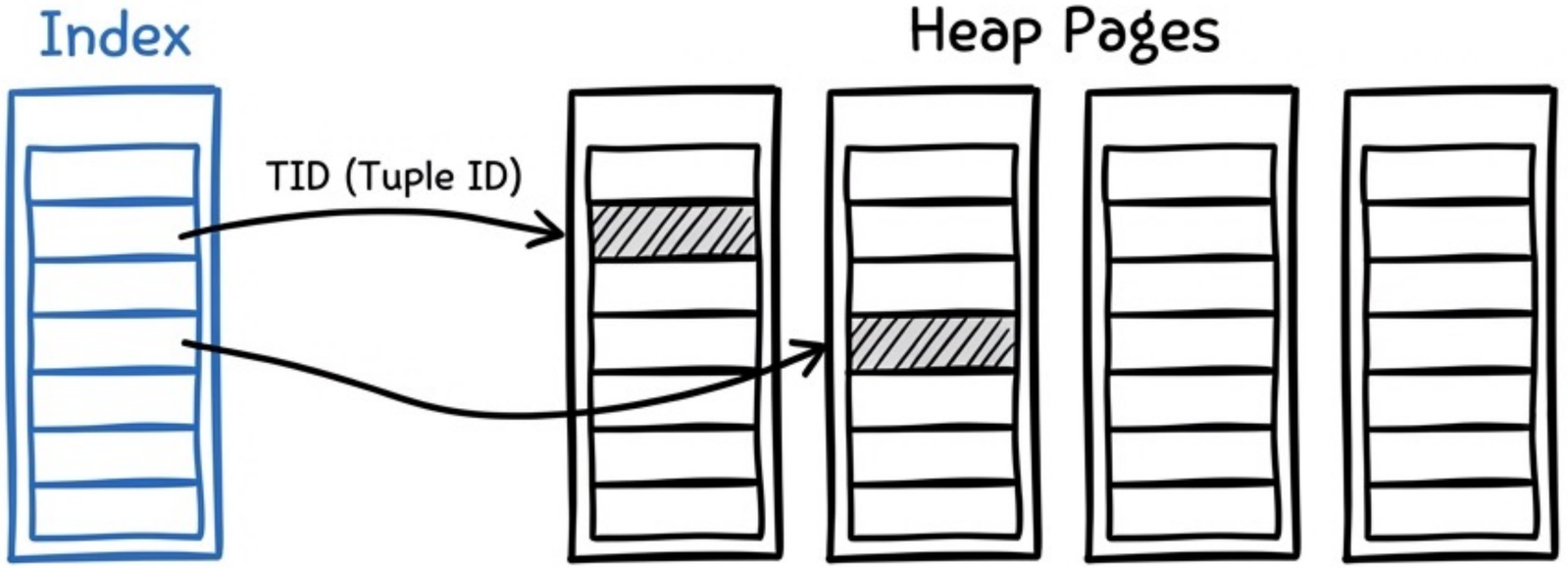
PostgreSQL 인덱스 스캔의 세계

쿼리 성능을 결정하는 내부 알고리즘의 이해



NotebookLM

기본 메커니즘: 인덱스 스캔 (Index Scan)

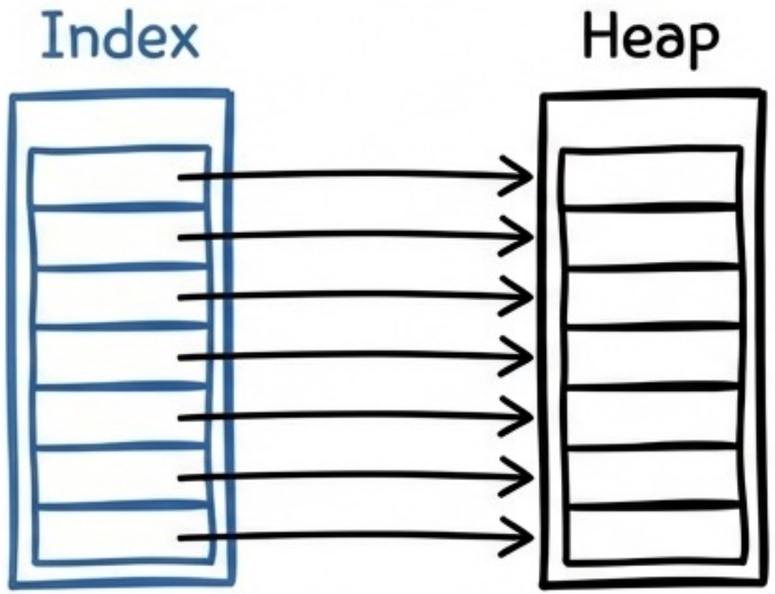


- 작동 원리: 인덱스에서 TID를 찾아 힙(Heap)의 튜플을 하나씩 조회.
- 실행 계획 노드: Index Scan
- 특징: 필터 조건(Index Cond)에 맞는 데이터를 즉시 반환.

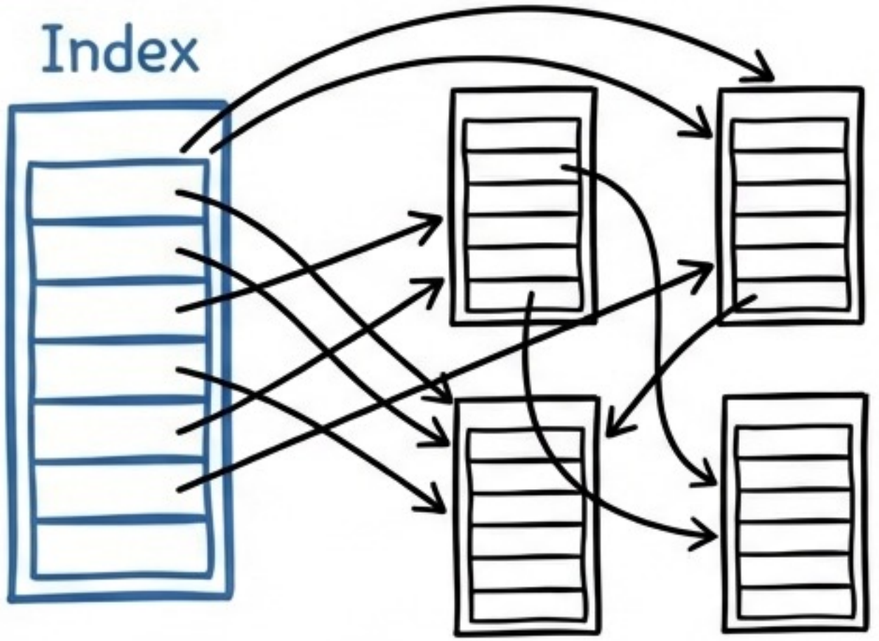
NotebookLM

성능의 핵심: 상관관계 (Correlation)

High Correlation (좋은 예)



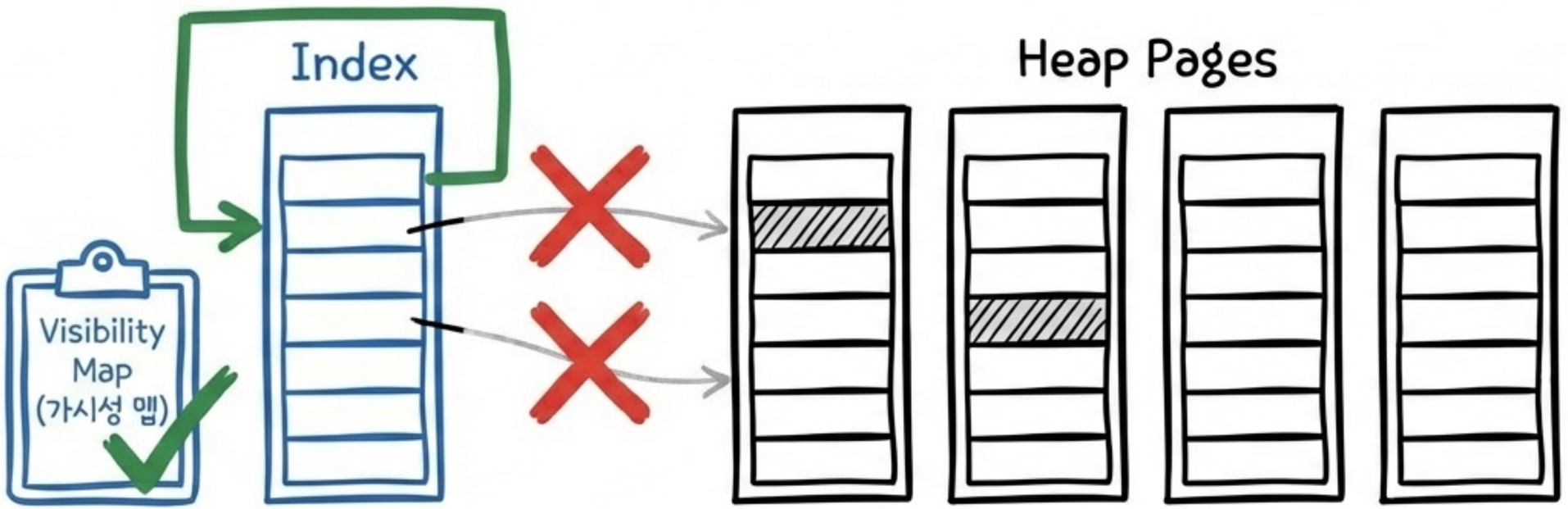
Low Correlation (나쁜 예)



상관관계가 낮으면 페이지를 반복해서 오가는 Random I/O 발생
→ 성능 저하 (`random_page_cost` 증가)

NotebookLM

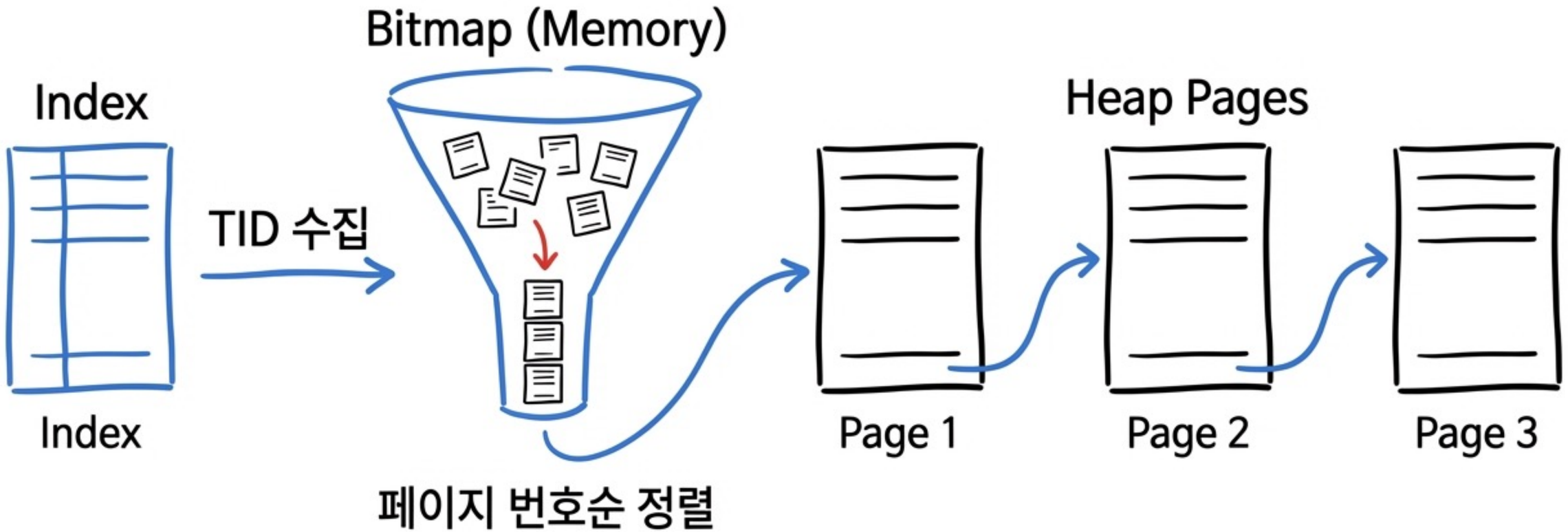
힙 접근 생략: 인덱스 전용 스캔 (Index Only Scan)



- 조건: 쿼리에 필요한 모든 컬럼이 인덱스에 포함 (Covering Index)
- 핵심: 가시성 맵(Visibility Map)이 'All Visible'일 때만 힙 접근 생략 가능
- 실행 계획 노드: `Index Only Scan`

NotebookLM

무작위 I/O 해결: 비트맵 스캔 (Bitmap Scan)

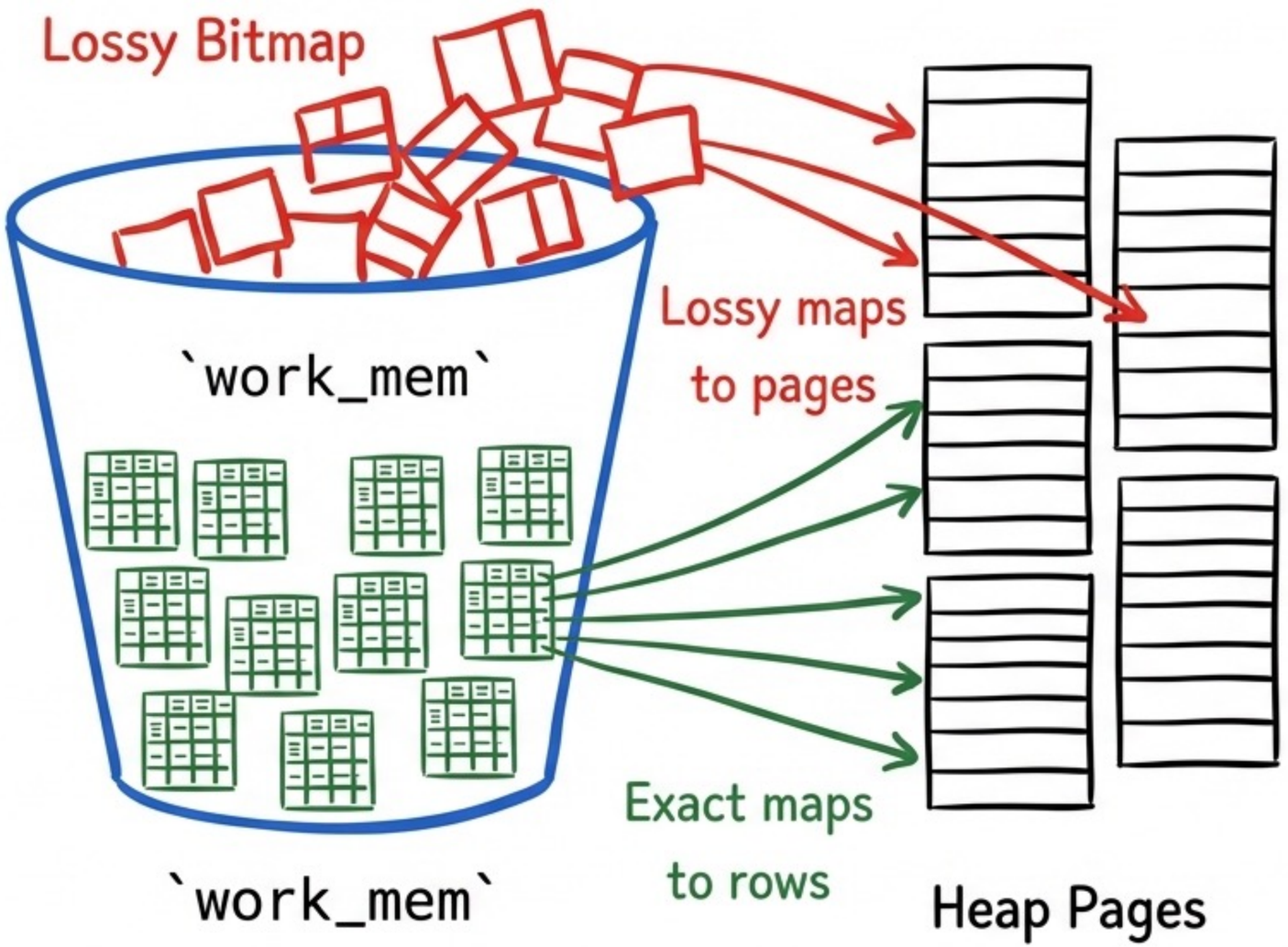


목적: 낮은 상관관계의 Random I/O를 순차적(Sequential) 접근으로 변환

1. `Bitmap Index Scan` (TID 수집) -> 2. `Bitmap Heap Scan` (정렬 후 접근)

NotebookLM

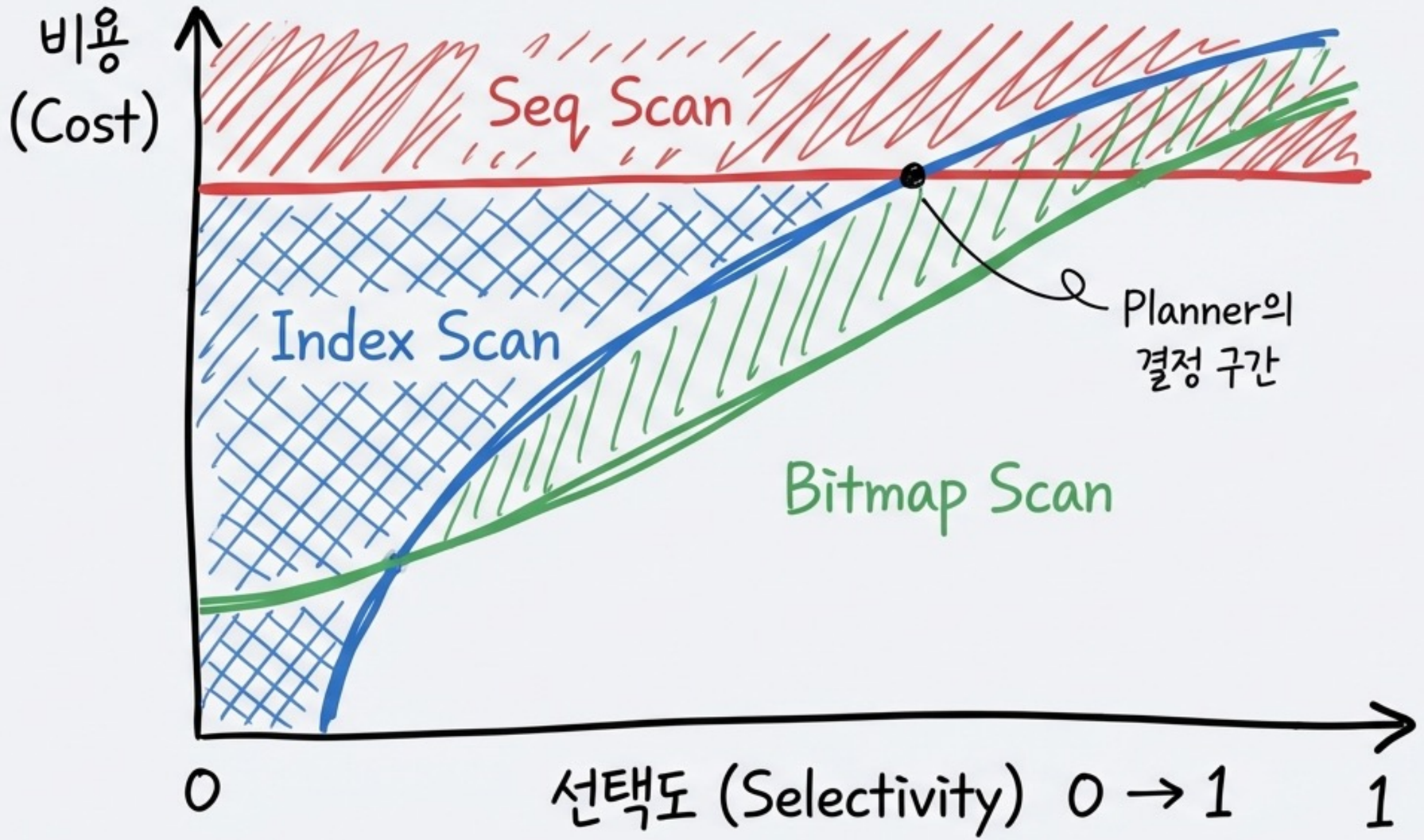
비트맵 정확도와 메모리 제한 (Lossy Bitmap)



- 메모리 부족 (`work_mem` 초과) 시 발생
- Lossy 모드: 튜플 위치 대신 '페이지' 위치만 기록
- 결과: `Recheck Cond` 발생 (페이지를 읽은 후 CPU가 다시 필터링)

NotebookLM

스캔 방식 비교: 언제 무엇을 쓰는가?



NotebookLM

핵심 요약 (Key Takeaways)



Index Scan

높은 상관관계(Correlation)
데이터의 소량 조회에 최적.



Index Only Scan

테이블 접근 최소화.
Visibility Map 관리가
성능의 열쇠 (`VACUUM` 중요).



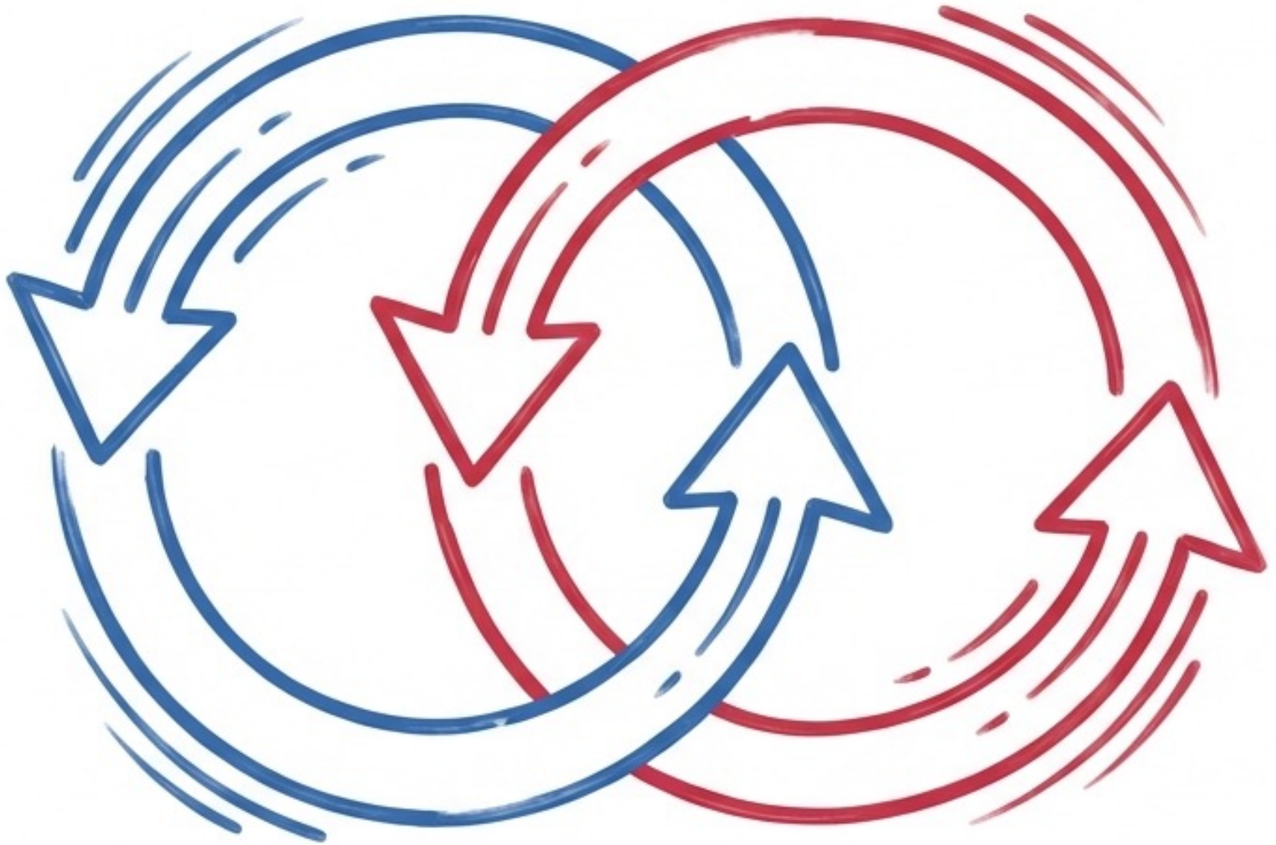
Bitmap Scan

다중 조건 결합(AND/OR)이나
상관관계가 낮은 데이터의
대량 조회에 강력.

NotebookLM

21. Nested Loop

PostgreSQL: 중첩 루프 조인 (Nested Loop Join)



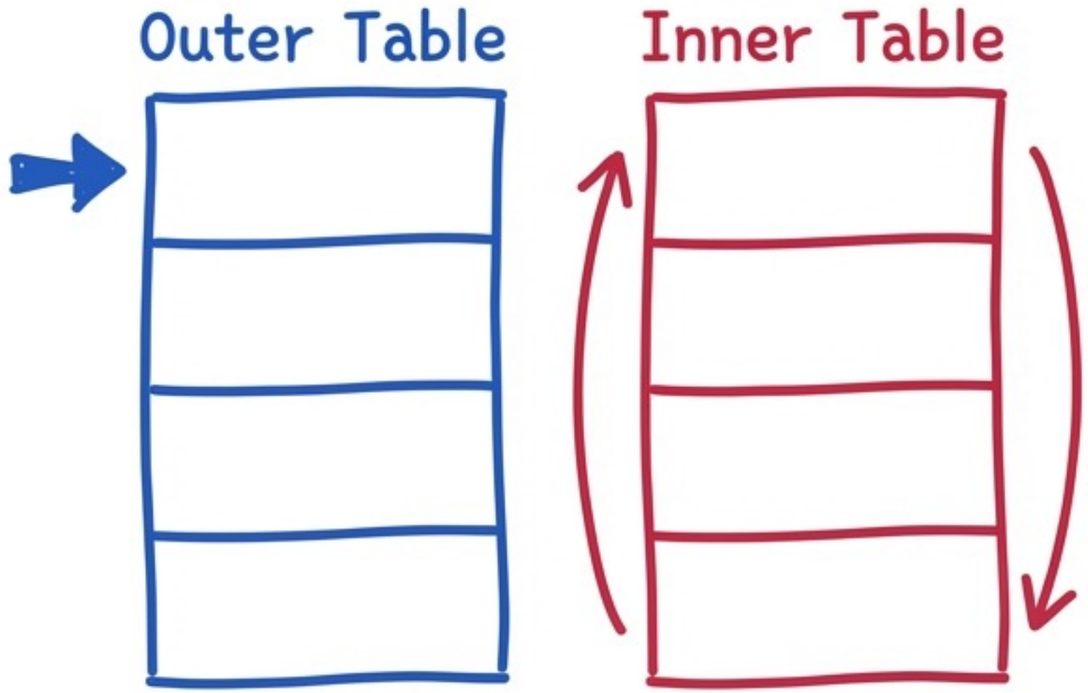
SQL 최적화와 내부 동작 원리 완전 정복

NotebookLM

알고리즘 (Algorithm)

```
FOR row_a IN Outer_Table LOOP
    FOR row_b IN Inner_Table LOOP
        IF row_a == row_b THEN
            RETURN (row_a, row_b)
        END IF
    END LOOP
END LOOP
```

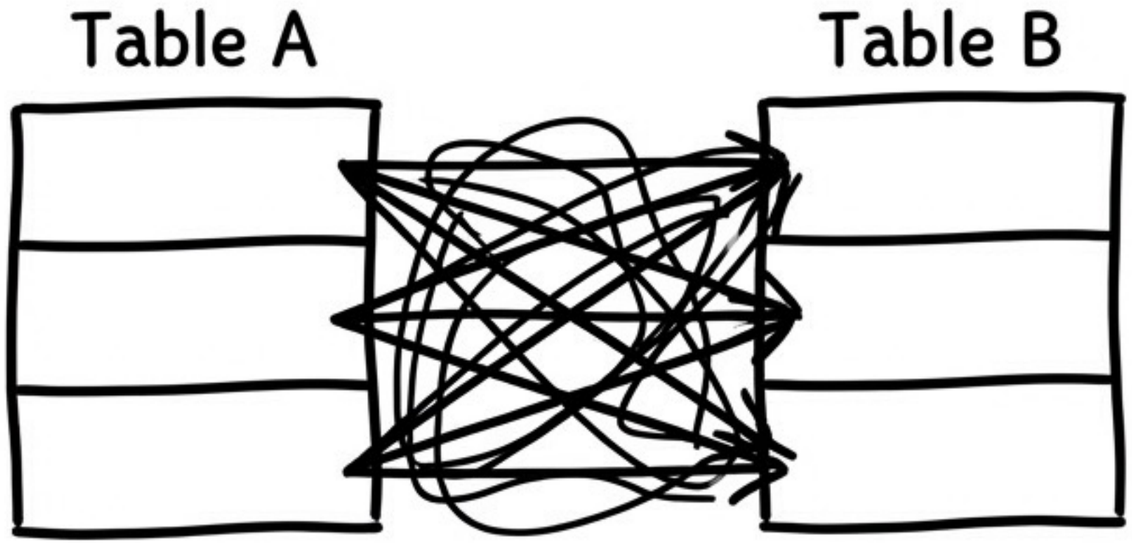
시각적 모델 (Visual Model)



- 기본 원리 (Basic Concept)
- Outer Table의 각 행마다 Inner Table 전체 순회
- 조건을 만족하는 즉시 결과 반환

NotebookLM

카테시안 곱 (Cartesian Product)

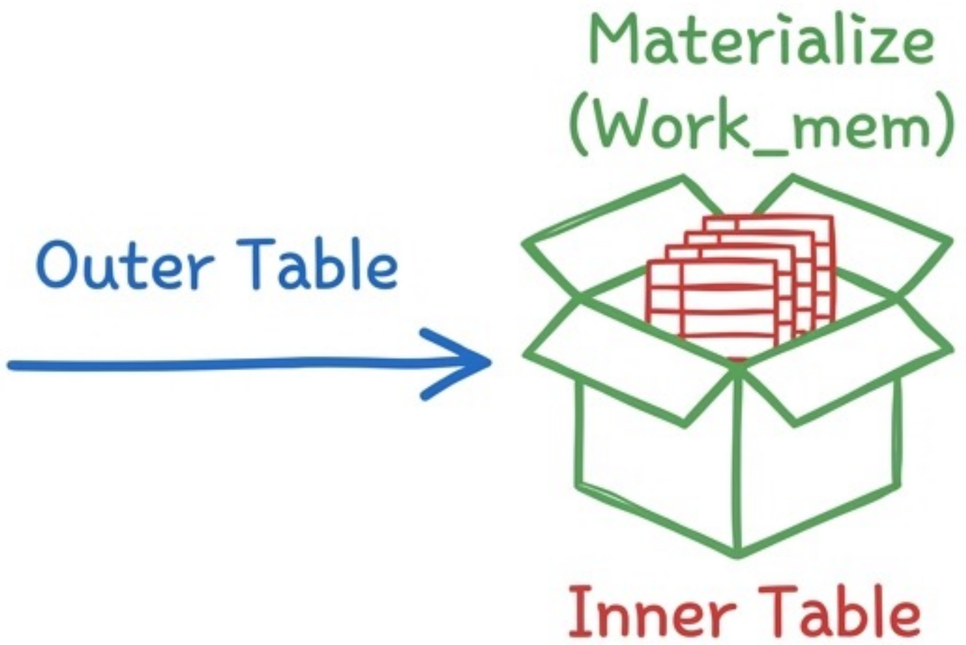


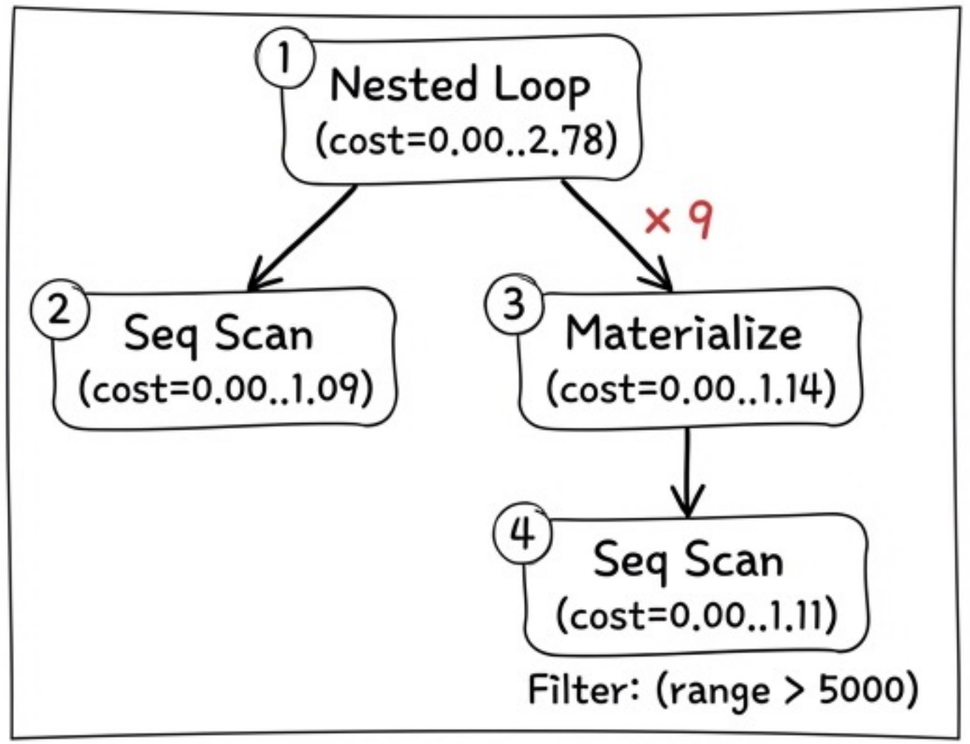
N × M

- 조인 조건이 없을 때 발생 (CROSS JOIN)
- 모든 가능한 행의 쌍(Pair)을 생성
- 비용(Cost): Outer 행 수 × Inner 행 접근 비용

NotebookLM

최적화 1: Materialization (물리화)

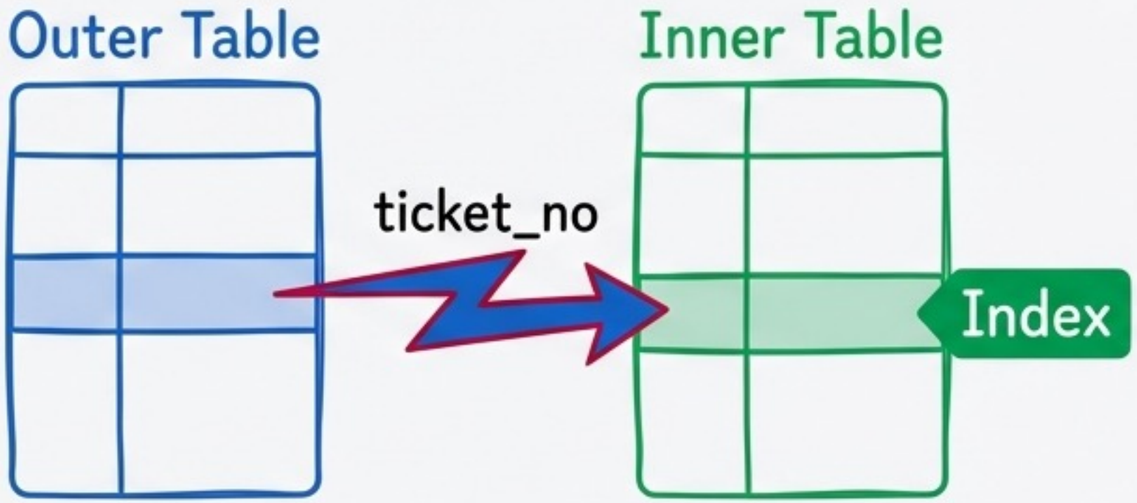




- Inner Table을 매번 디스크에서 읽는 비효율 제거
- 데이터를 메모리(Temp)에 저장하여 재사용
- 반복 접근 비용 최소화

NotebookLM

최적화 2: 매개 변수화된 조인 (Parameterized Join)



- Outer의 값을 Inner의 인덱스 조건으로 전달
- Full Scan 대신 Index Scan으로 필요한 행만 '콕 집어서' 조회
- 비용: Outer 행 수 × 인덱스 검색 비용

QUERY PLAN

Nested Loop
(cost=0.99..25.05 rows=3 width=136)

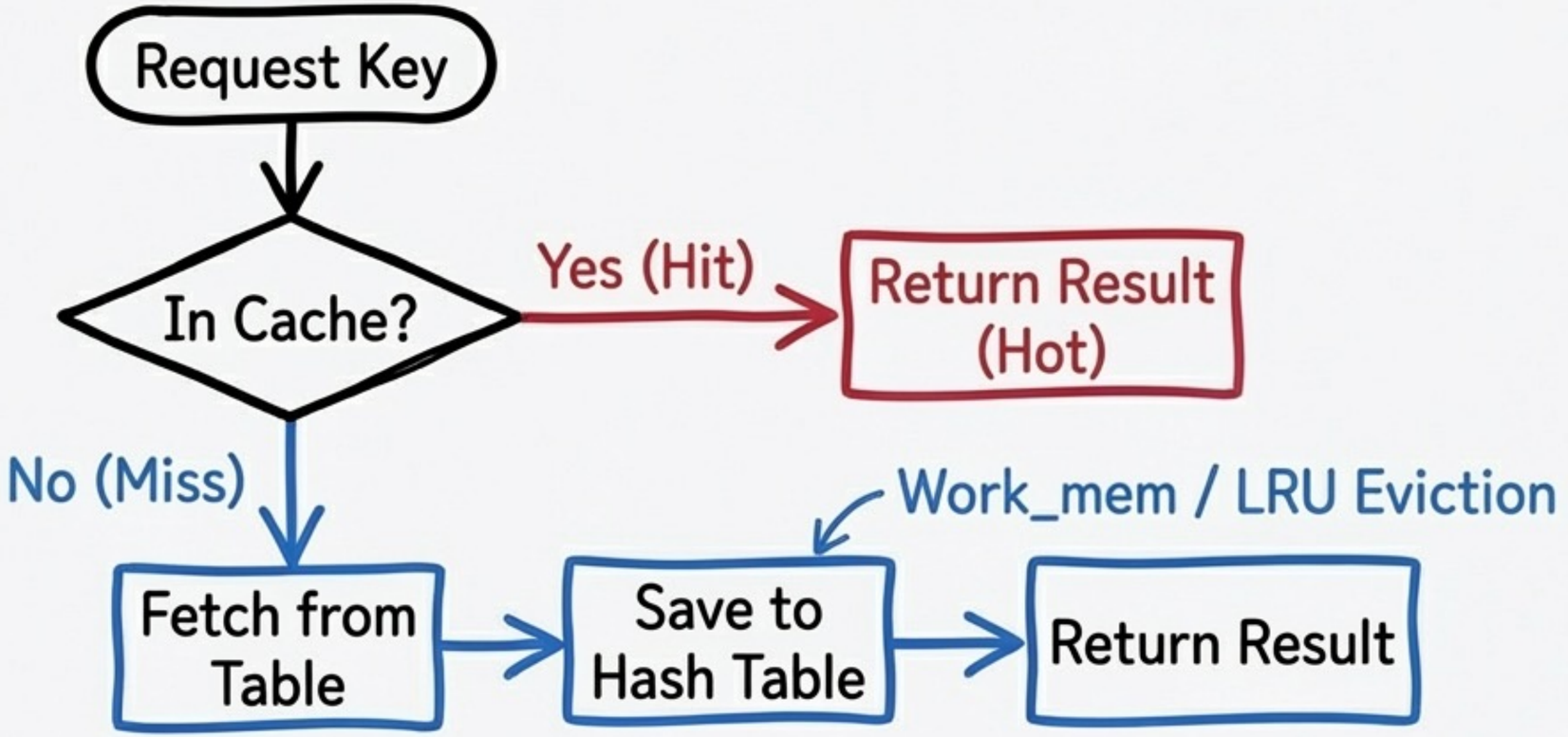
0.99..25.05 x 1

-> Index Scan using tickets_pkey
on tickets t
(cost=0.43..8.45 rows=1 width=104)

-> Index Scan using ticket_flights_pkey
on ticket_flights tf
(cost=0.56..16.58 rows=3 width=32)

NotebookLM

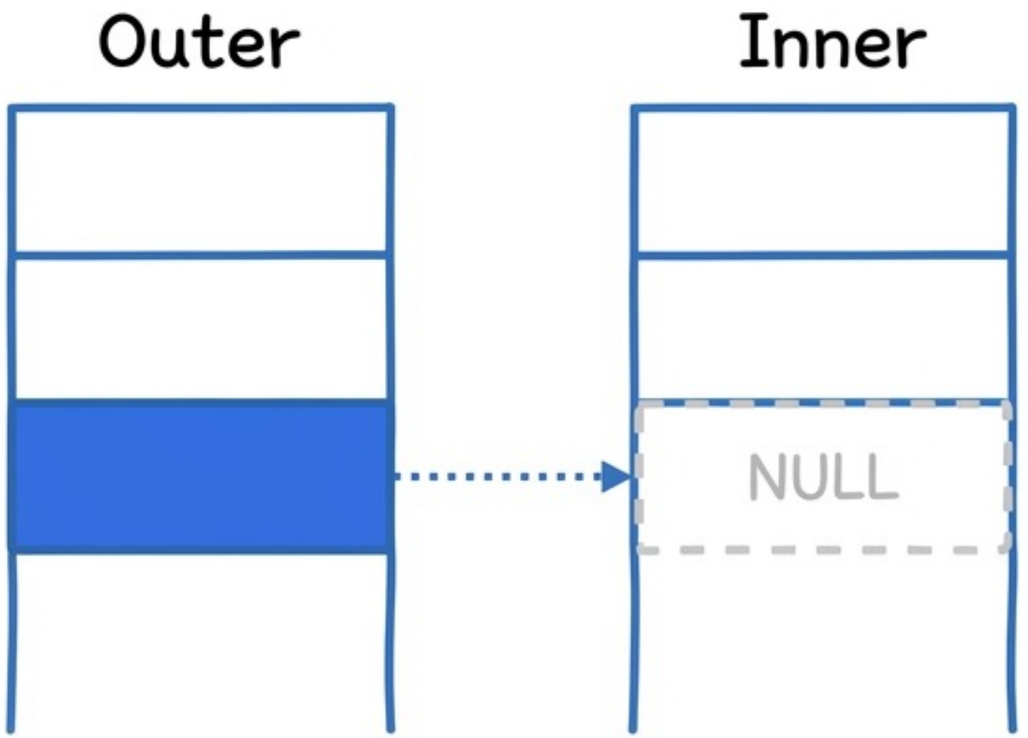
최적화 3: Memoize (결과 캐싱)



- 이미 조회한 키값과 결과를 메모리에 저장
- Hot Key: 자주 쓰는 값은 캐시에서 즉시 반환 (I/O 감소)
- Cold Key: 안 쓰는 값은 캐시에서 제거

NotebookLM

외부 조인 (Left Outer Join)

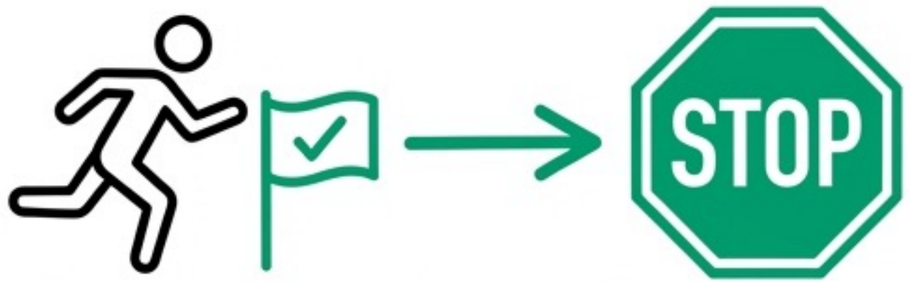


- Inner 루프에서 일치하는 행이 없으면?
- Outer 행 + NULL 값으로 결과 확장
- 행의 수를 줄이지 않고 보존

NotebookLM

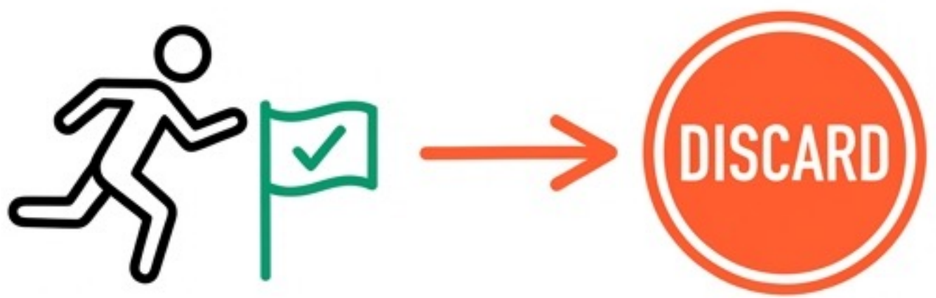
Semi & Anti Join (Short Circuit)

EXISTS (Semi Join)



첫 번째 일치 -> True 반환 -> 종료

NOT EXISTS (Anti Join)

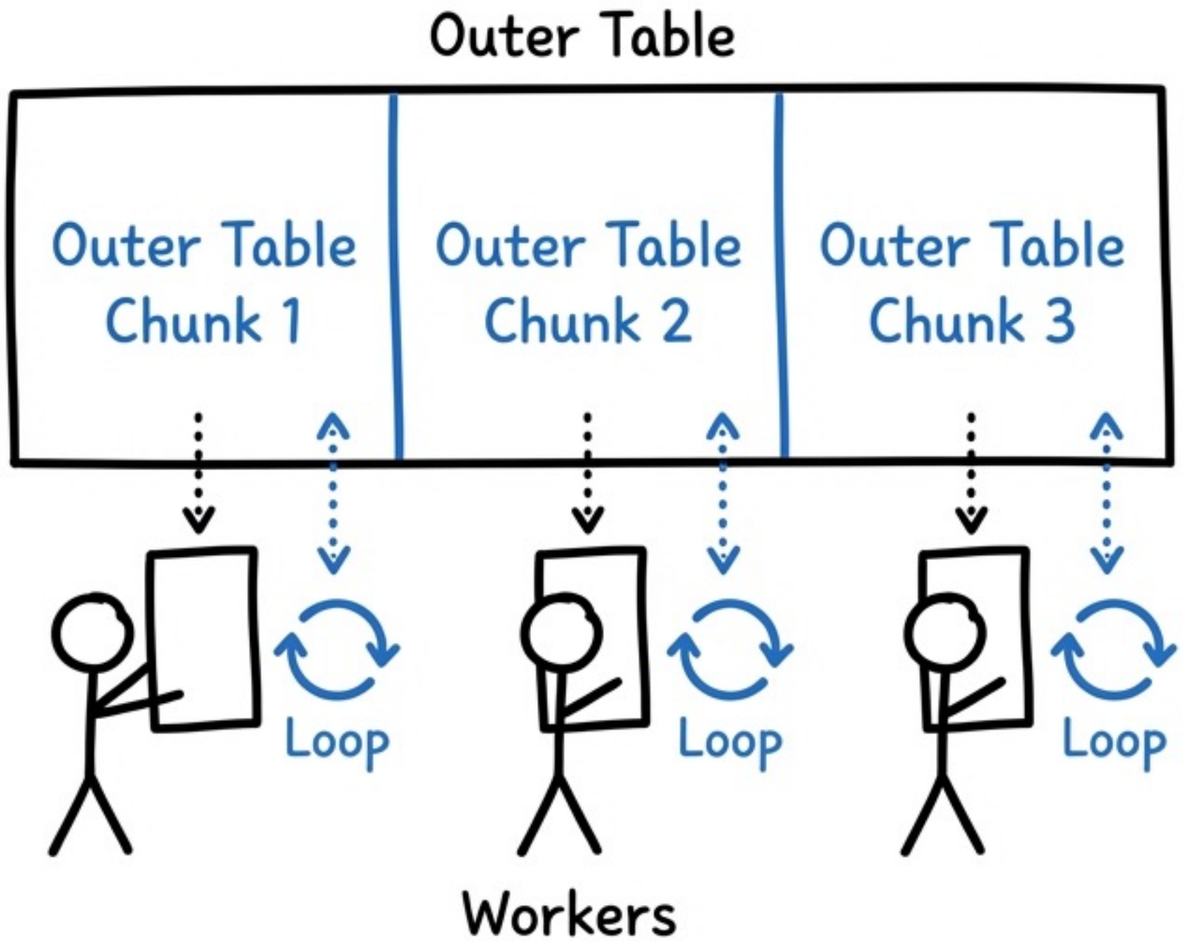


첫 번째 일치 -> False (제외) -> 종료

- Short Circuit: 모든 데이터를 확인할 필요 없이 즉시 루프 종료

NotebookLM

병렬 실행 (Parallel Mode)



- Outer Table: 여러 워커(Worker)가 범위를 분담하여 병렬 스캔
- Inner Table: 각 워커가 개별적으로 중첩 루프 수행
- 대용량 데이터 처리 시 속도 획기적 개선

NotebookLM

핵심 요약 & 전략

- [x] **1. 데이터셋 크기**
  - Outer 집합(Driving Table)이 작을수록 유리
- [x] **2. 인덱스 (Index)**
  - Inner 집합에 효율적인 인덱스 필수 (Index Scan)
- [x] **3. 캐싱 (Caching)**
  - 반복되는 패턴은 Memoize로 최적화

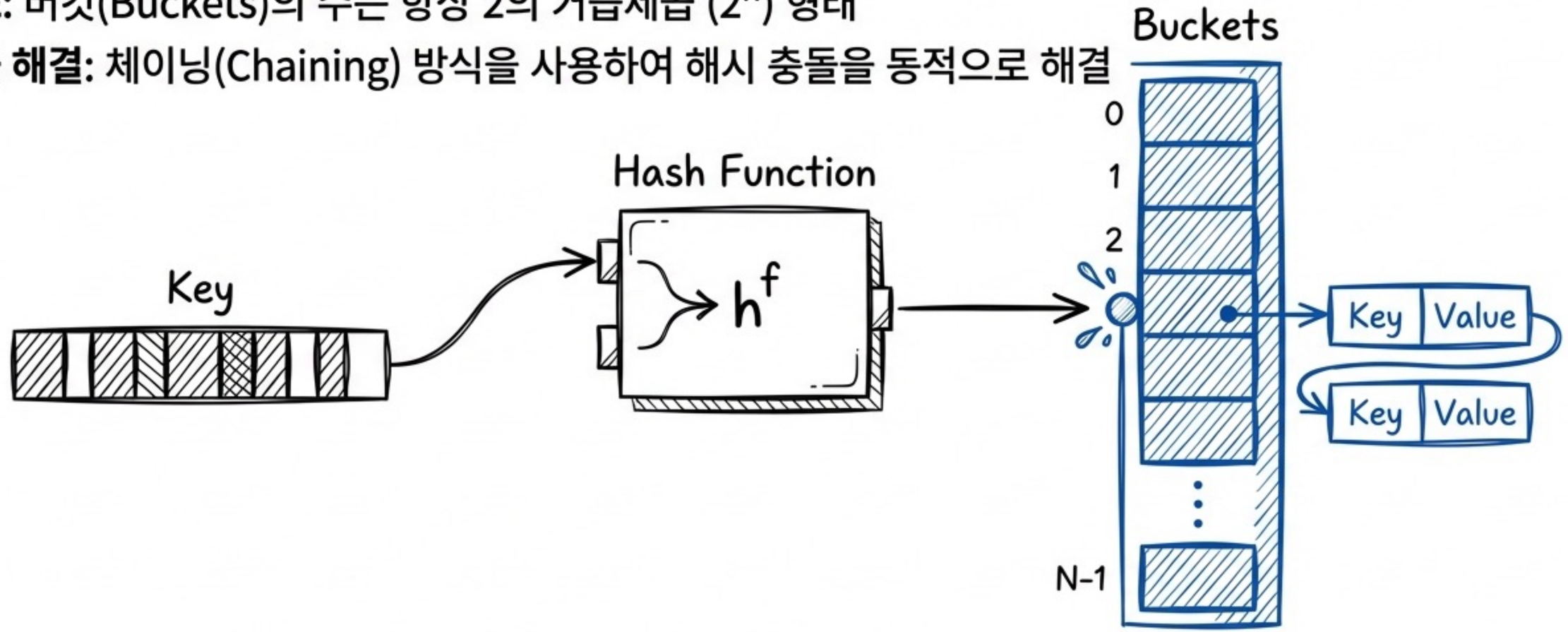
PostgreSQL은 비용(Cost) 기반으로 가장 효율적인 경로를 선택합니다.

NotebookLM

- **22. Hashing Join**

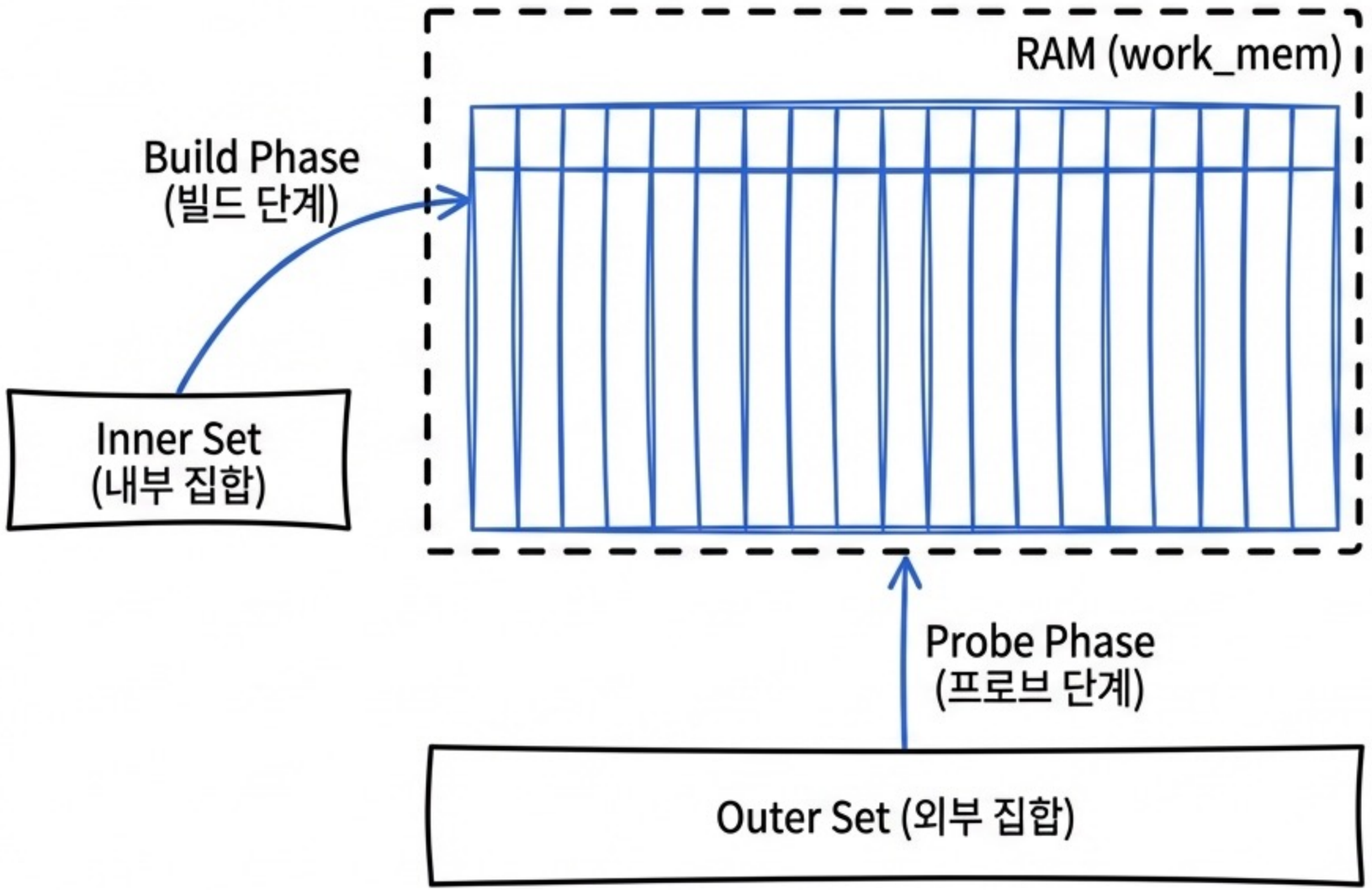
PostgreSQL 해시 조인: 해싱을 통한 고속 데이터 결합

- **기본 개념:** 사전에 구축된 해시 테이블을 사용하여 일치하는 행을 검색
- **핵심 원리:** 해시 키(Hash Key)와 값(Value) 쌍을 저장하여 O(1)에 가까운 빠른 접근 속도 보장
- **구조:** 버킷(Buckets)의 수는 항상 2의 거듭제곱 (2^n) 형태
- **충돌 해결:** 체이닝(Chaining) 방식을 사용하여 해시 충돌을 동적으로 해결



NotebookLM

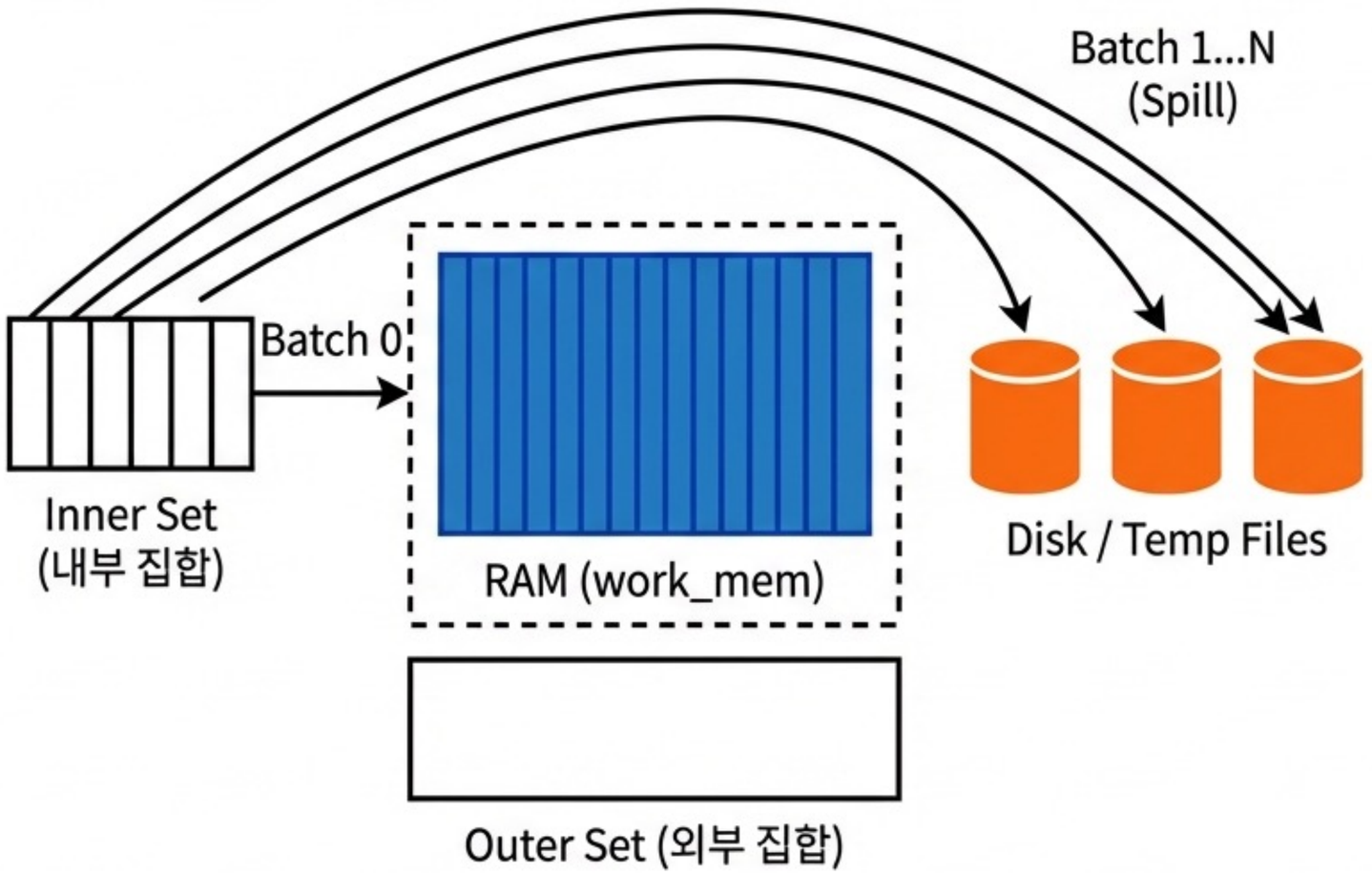
1-패스 해시 조인: 메모리가 충분할 때



- **빌드**(Build): 내부 집합(Inner Set) 전체를 스캔하여 메모리 내 해시 테이블 생성
- **프로브**(Probe): 외부 집합(Outer Set)을 스캔하며 해시 테이블과 대조하여 즉시 결과 반환
- **조건:** 전체 해시 테이블 크기 ≦ work_mem × hash_mem_multiplier
- **효율성:** 가장 빠르고 효율적인 방식 (디스크 I/O 없음)

NotebookLM

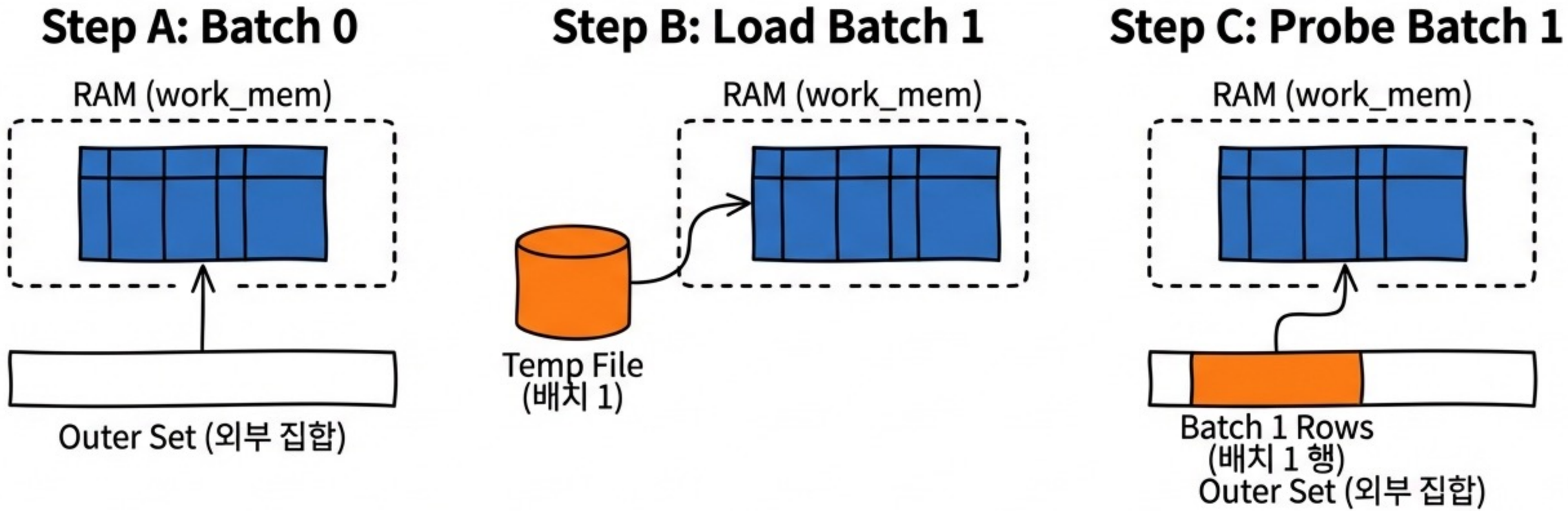
2-패스 해시 조인: 메모리 부족과 배치 분할



- **배치**(Batch) **분할**: 메모리가 부족하면 데이터를 여러 배치로 나누어 처리
- **작동 방식**:
  - 첫 번째 배치(Batch 0)는 RAM에 유지
  - 나머지 배치는 임시 파일(Temp Files)로 디스크에 기록
- **비용 발생**: 임시 파일 쓰기/읽기로 인한 I/O 오버헤드 증가
- **제한**: temp_file_limit을 초과하면 쿼리가 중단됨

NotebookLM

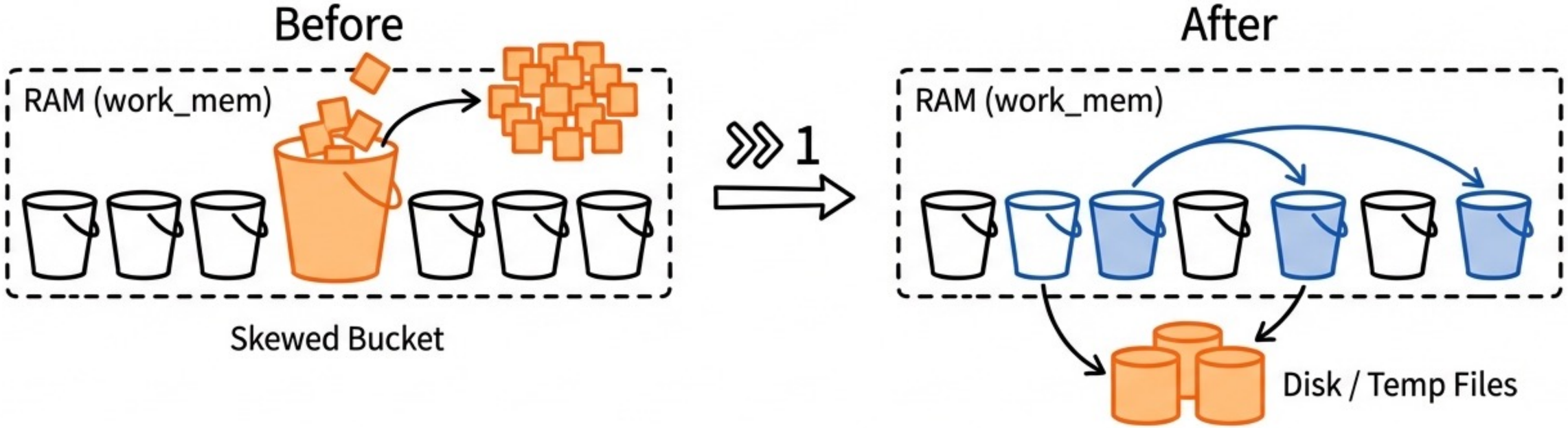
디스크에 저장된 배치의 처리 과정



- 순차적 처리: 메모리에 있는 해시 테이블이 비워지면, 다음 배치를 디스크에서 로드하여 재구축
- 독립적 매칭: 외부 집합(Outer Set)의 행도 동일한 배치 번호를 가진 내부 집합 배치와만 대조됨
- 임시 파일 삭제: 처리가 완료된 배치의 임시 파일은 즉시 삭제됨
- 성능 영향: 디스크 접근(Temp Read/Written)이 EXPLAIN (BUFFERS) 출력에 나타남

NotebookLM

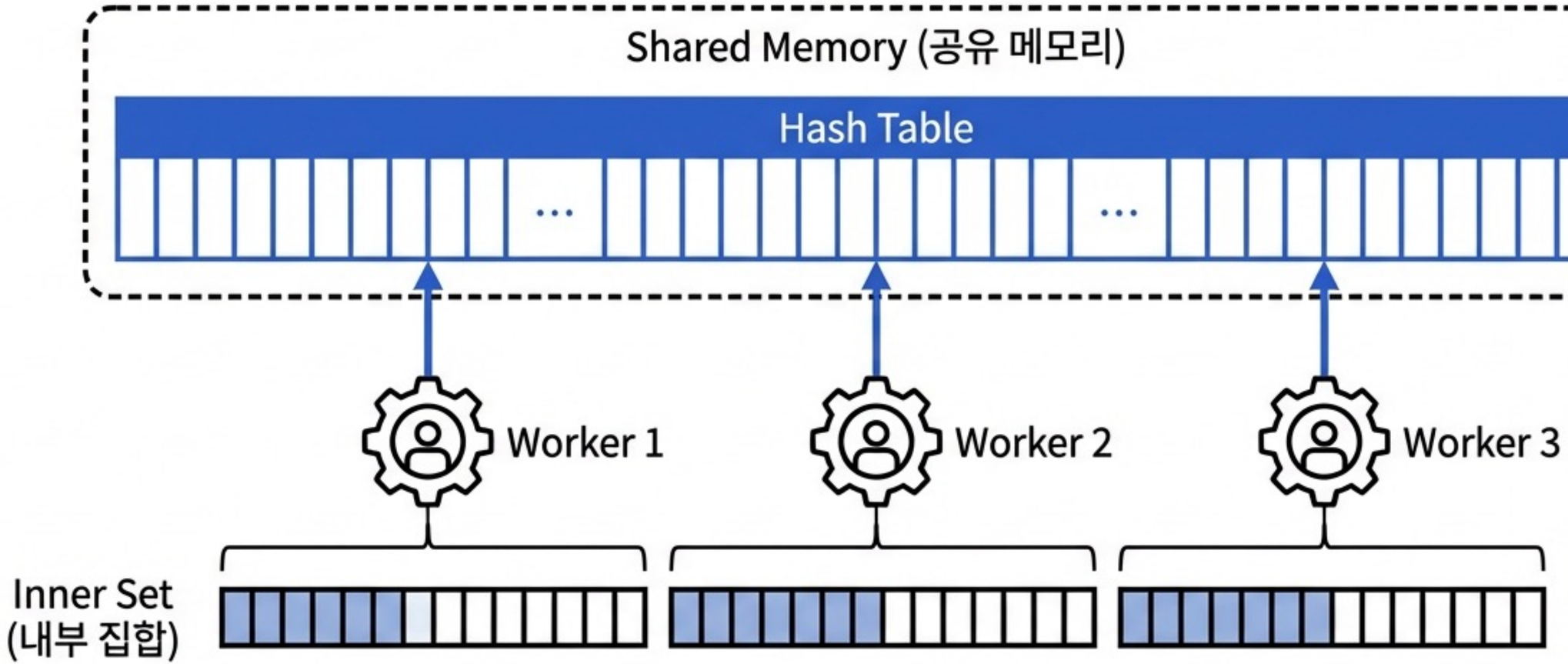
데이터 편향(Skew)과 동적 조정



- **문제 상황:** 데이터 분포가 비균일(Non-uniform)하여 특정 배치가 예상보다 커지는 경우 (MCV 등)
- **동적 조정:** 해시 테이블 구축 중 배치가 너무 커지면 배치 수를 즉시 두 배로 증가(Bit shifting)
- **결과:** 기존 배치의 약 절반은 메모리에 남고, 나머지 절반은 새로운 임시 파일로 이동
- **주의:** 키 값이 모든 행에서 동일한 경우(최악의 경우), 배치를 늘려도 해결되지 않음

NotebookLM

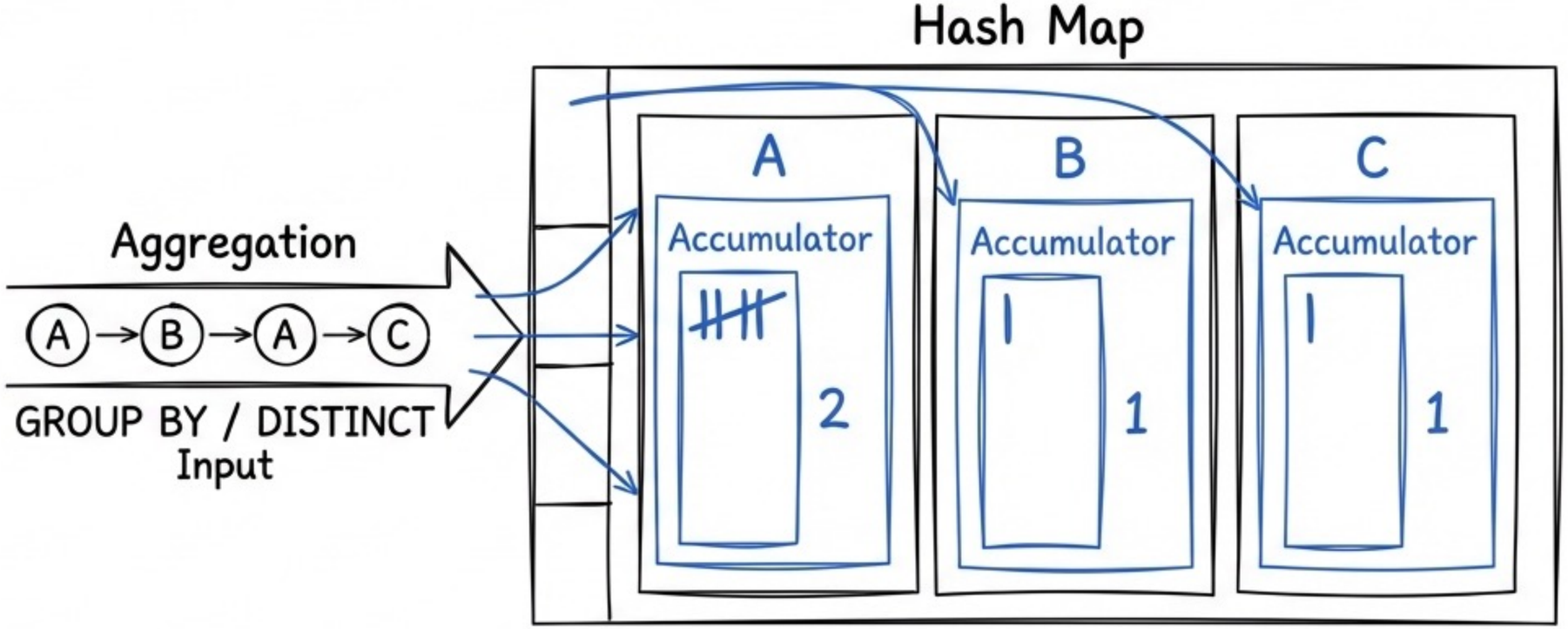
병렬 해시 조인: 공유 메모리 활용



- **핵심 차이:** 각 프로세스가 개별 테이블을 만드는 것이 아니라, 단일 공통 해시 테이블을 공유 메모리에 구축
- **협업:** 모든 병렬 프로세스(Workers)가 내부 집합을 스캔하여 함께 해시 테이블을 생성
- **장점:** 메모리 효율성 증대. 1-패스로 조인을 완료할 확률이 높아짐
- **동기화:** 배리어(Barrier)를 통해 모든 프로세스가 빌드 단계를 완료한 후 프로브 단계로 진입

NotebookLM

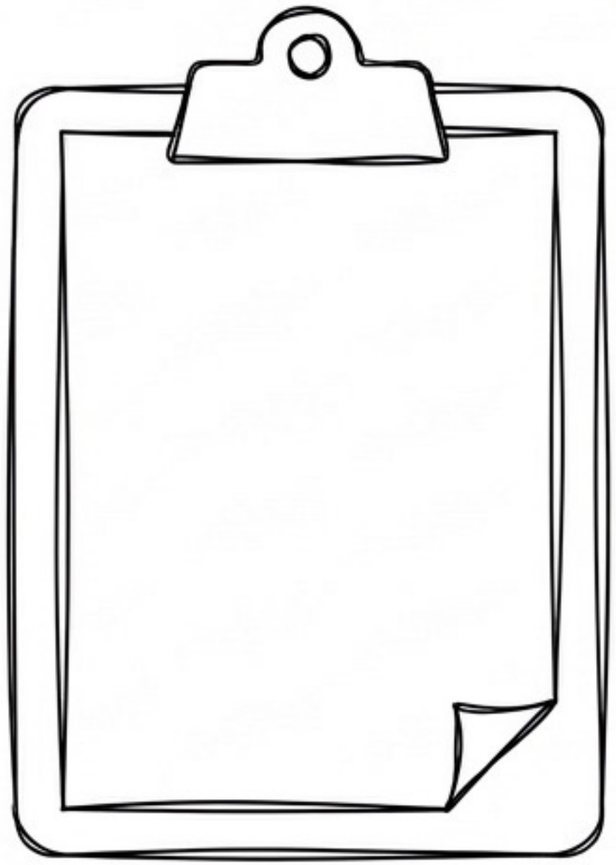
해시 집계: 조인을 넘어선 활용



- 적용: GROUP BY, DISTINCT, UNION 연산에서 중복 제거 및 값 집계에 사용
- HashAggregate 노드: 해시 테이블에 키가 없으면 추가, 있으면 집계 값 업데이트
- 메모리 초과 시: 해시 조인과 유사하게 파티션(Partition)으로 나누어 디스크에 기록(Spill)
- 최적화: 필요한 열만 해시 테이블에 포함하여 메모리 사용량 최소화

NotebookLM

요약 및 성능 최적화 체크리스트



✅ **메모리 튜닝:** work_mem과 hash_mem_multiplier를 적절히 설정하여 1-패스 조인 유도

✅ **쿼리 작성:** SELECT * 지양. 불필요한 열은 해시 테이블 크기를 키워 디스크 스필(Spill)을 유발함

✅ **통계 정보:** 최신 통계(ANALYZE)가 있어야 플래너가 정확한 배치 수를 추정 가능

✅ **비용 이해:** 1-패스(메모리) ≪ 2-패스(디스크). 디스크 I/O 발생 시 비용이 급격히 증가함

NotebookLM

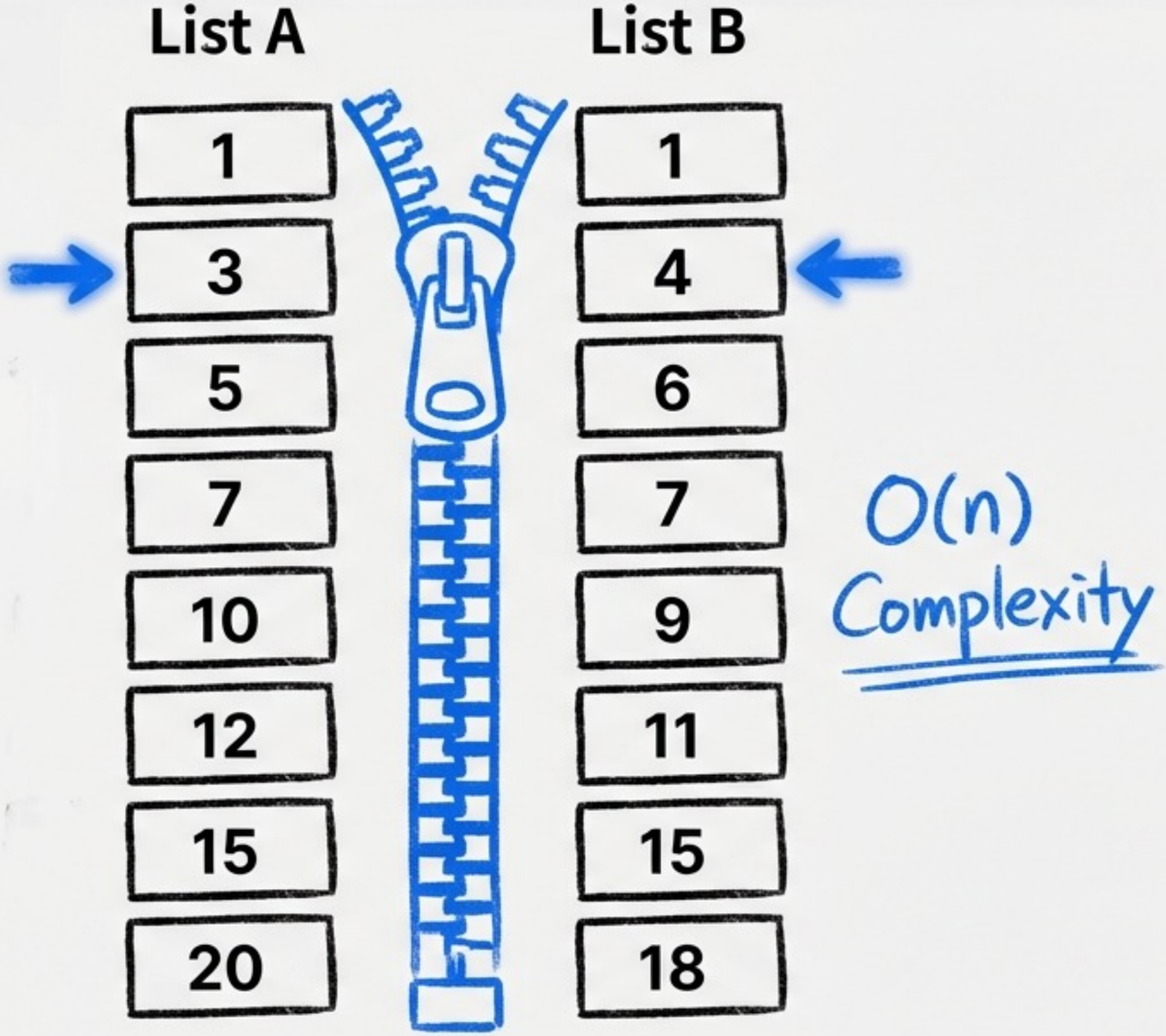
- **23. Sort Merge Join**

PostgreSQL 내부 구조: 정렬과 병합

쿼리 최적화를 위한 알고리즘 메커니즘



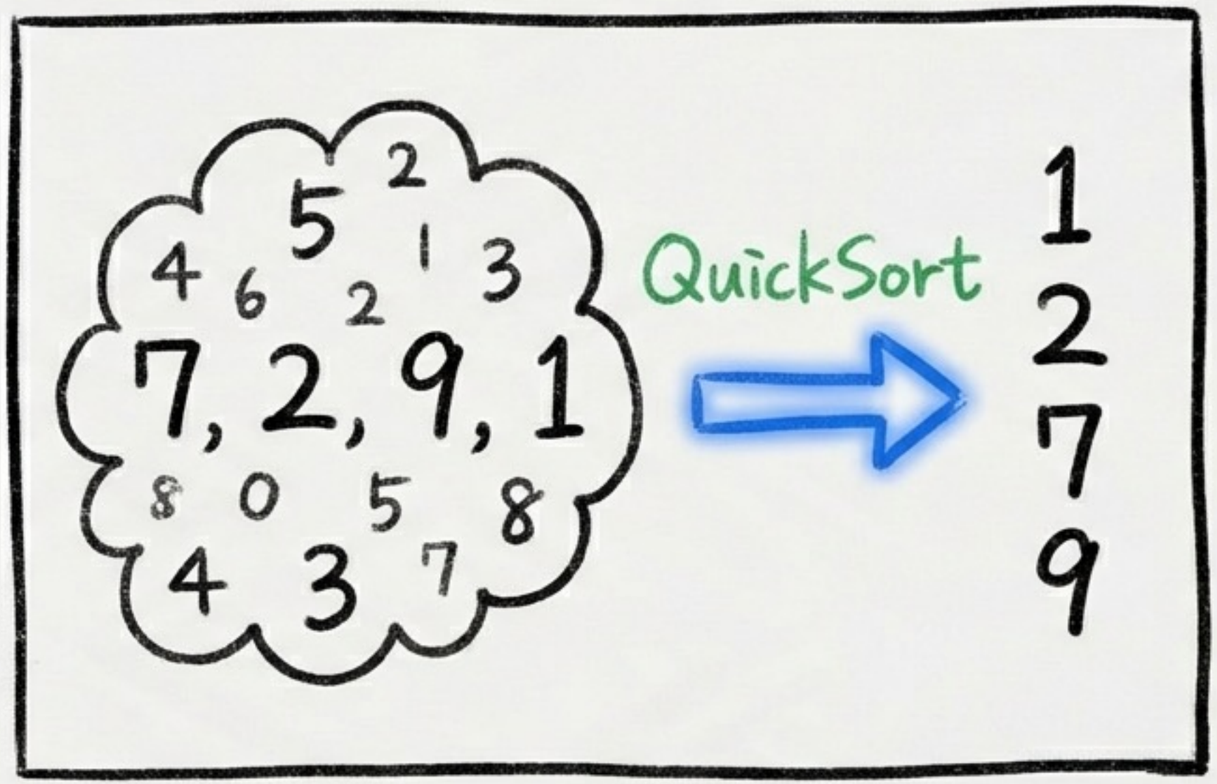
NotebookLM



병합 조인 (Merge Join): 정렬된 데이터의 힘

- 작동 원리: 두 개의 정렬된 데이터 세트를 '지퍼'처럼 맞물려 스캔.
- 효율성: 데이터가 정렬되어 있다면 가장 빠르고 메모리 효율적인 조인 방식.
- 포인터 이동:
  1. 키가 일치하면 조인 수행.
  2. 일치하지 않으면 작은 키 값을 가진 쪽의 포인터만 이동.
  3. 한쪽 세트가 끝나면 작업 종료 (Outer Join 제외).

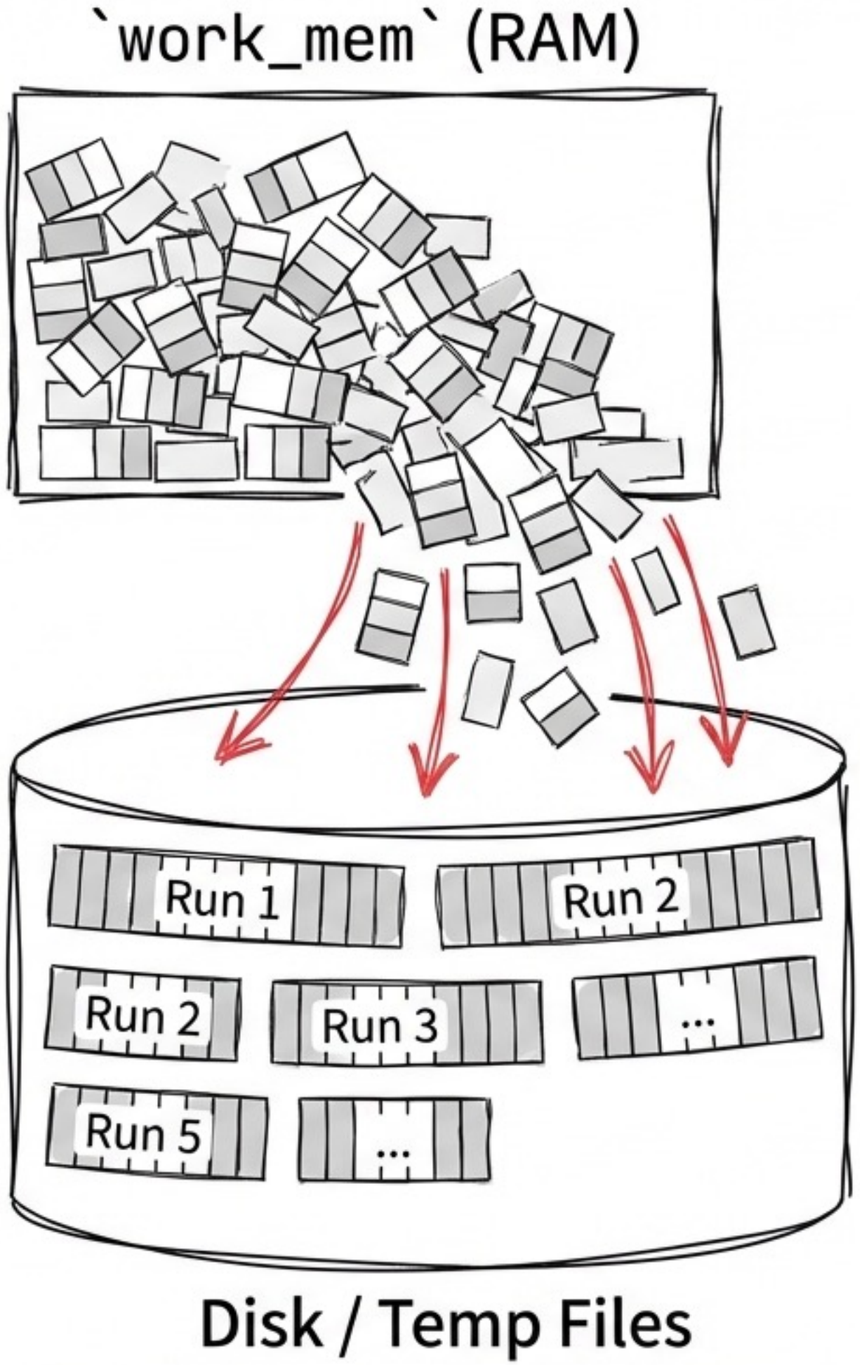
NotebookLM



메모리 정렬 (In-Memory Sort): QuickSort

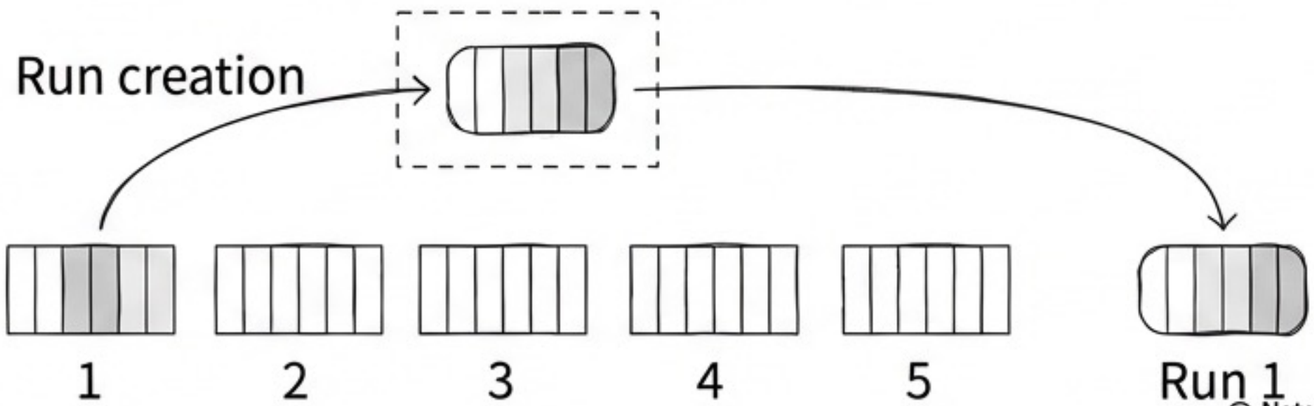
- 조건: 데이터 크기 ≤ work_mem (기본값: 4MB)
- 알고리즘: 퀵정렬 (QuickSort) 사용
- 특징: 디스크 I/O가 없어 매우 빠름
- 비용: O(n log n) 복잡도

NotebookLM

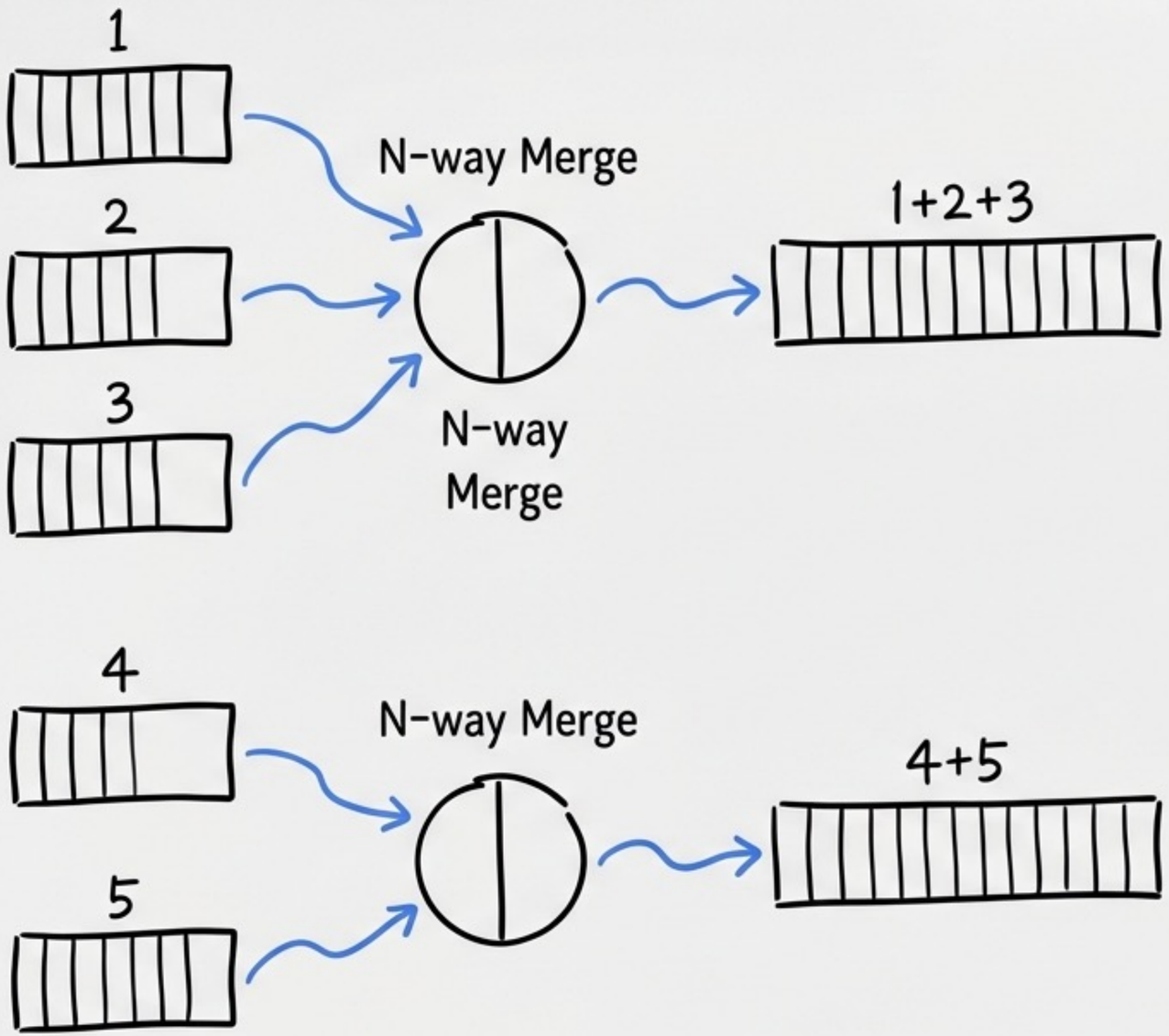


외부 병합 정렬 (External Merge Sort) – 1단계: 분할

- 문제: 데이터 크기 > `work_mem`
- 해결: 데이터를 메모리 크기만큼 잘라서(Run) 정렬 후 임시 파일로 저장
- 알고리즘: 내부적으로는 여전히 퀵정렬을 사용하여 Run을 생성
- I/O 발생: 이 시점부터 디스크 쓰기 비용 발생



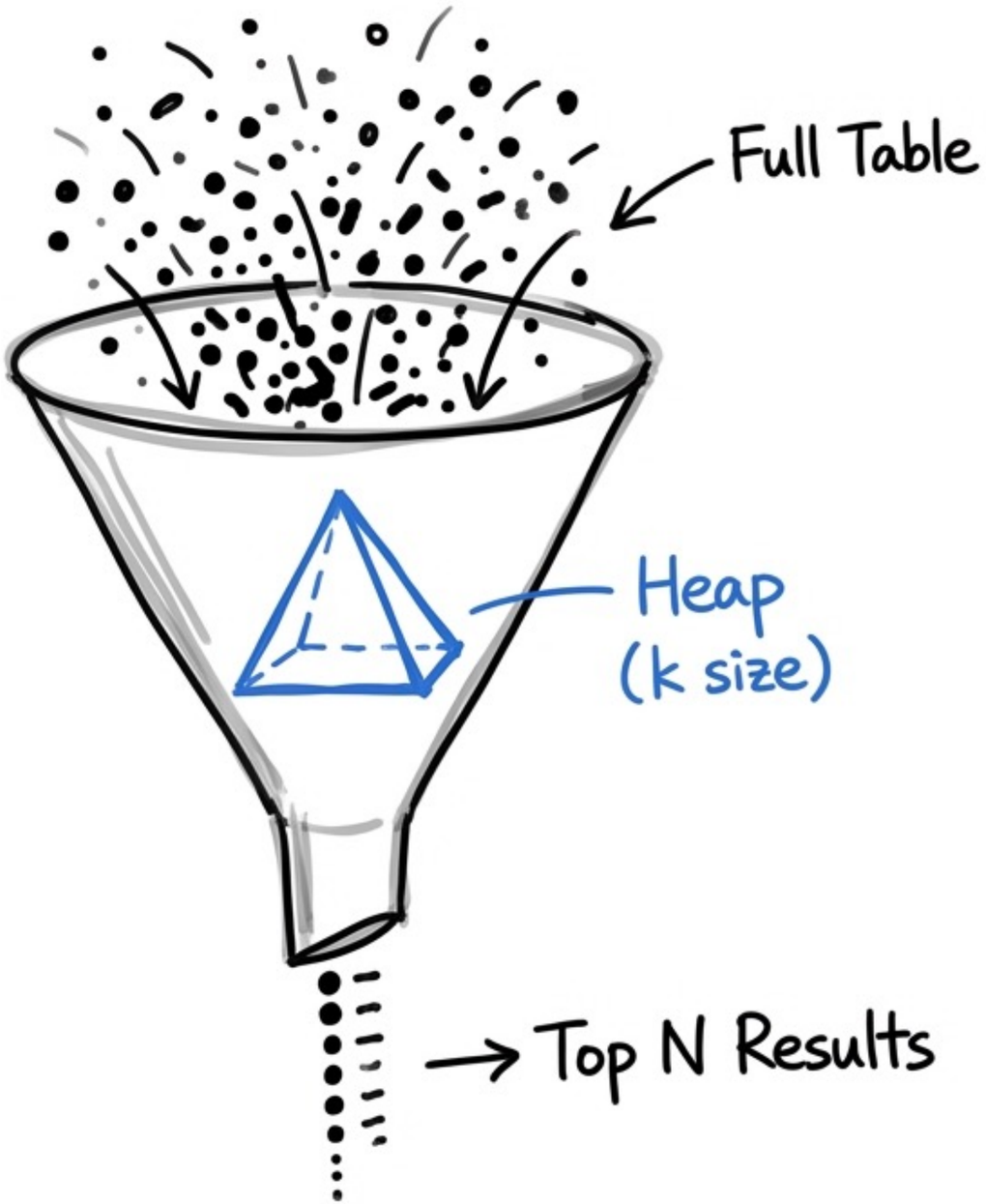
NotebookLM



외부 병합 정렬 – 2단계: 병합 (Merging Runs)

- 병합(Merge): 여러 개의 임시 파일(Run)을 하나로 합치는 과정
- 최적화 : 파일당 한 행씩만 메모리에 유지하며, 32페이지 단위로 배치 읽기(Batch Read)를 수행해 I/O 감소
- 한계 : 한 번에 병합 가능한 파일 수는 메모리에 의해 제한됨 (최대 500개)

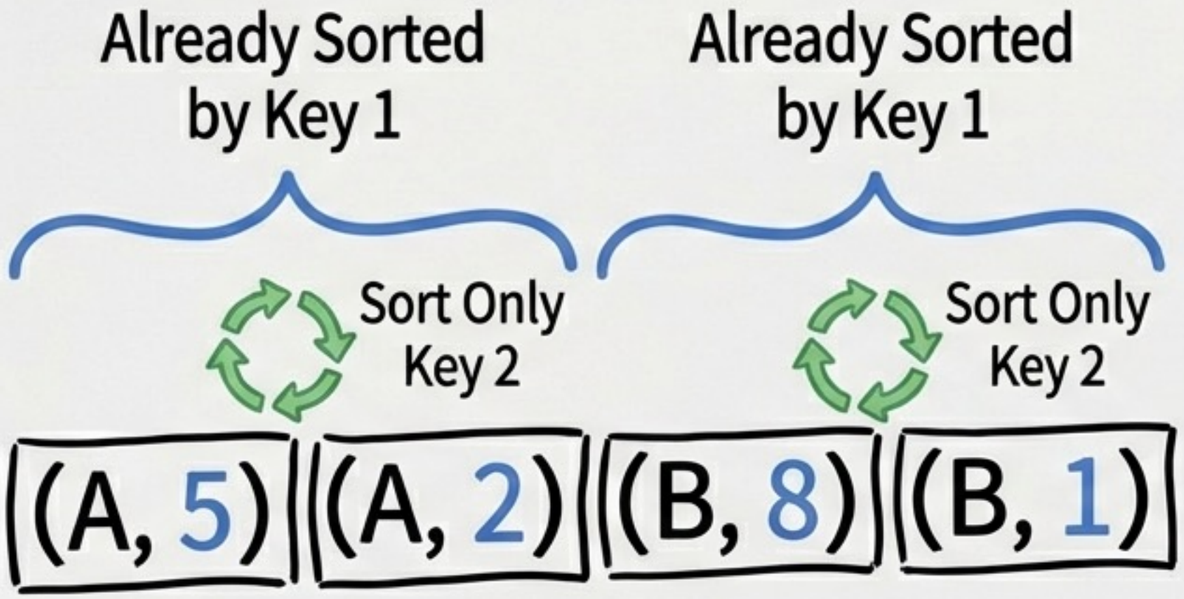
NotebookLM



최적화 1: Top-N 힙 정렬 (Heapsort)

- **사용 사례:** ORDER BY ... LIMIT k 쿼리
- **원리:** 전체 데이터를 정렬하지 않고, 상위 k개만 유지하는 '힙(Heap)' 자료구조 사용 '힙(Heap)' 자료구조 사용
- **장점:** 메모리 사용량 최소화. 전체 정렬보다 훨씬 빠름 (O(n log k))

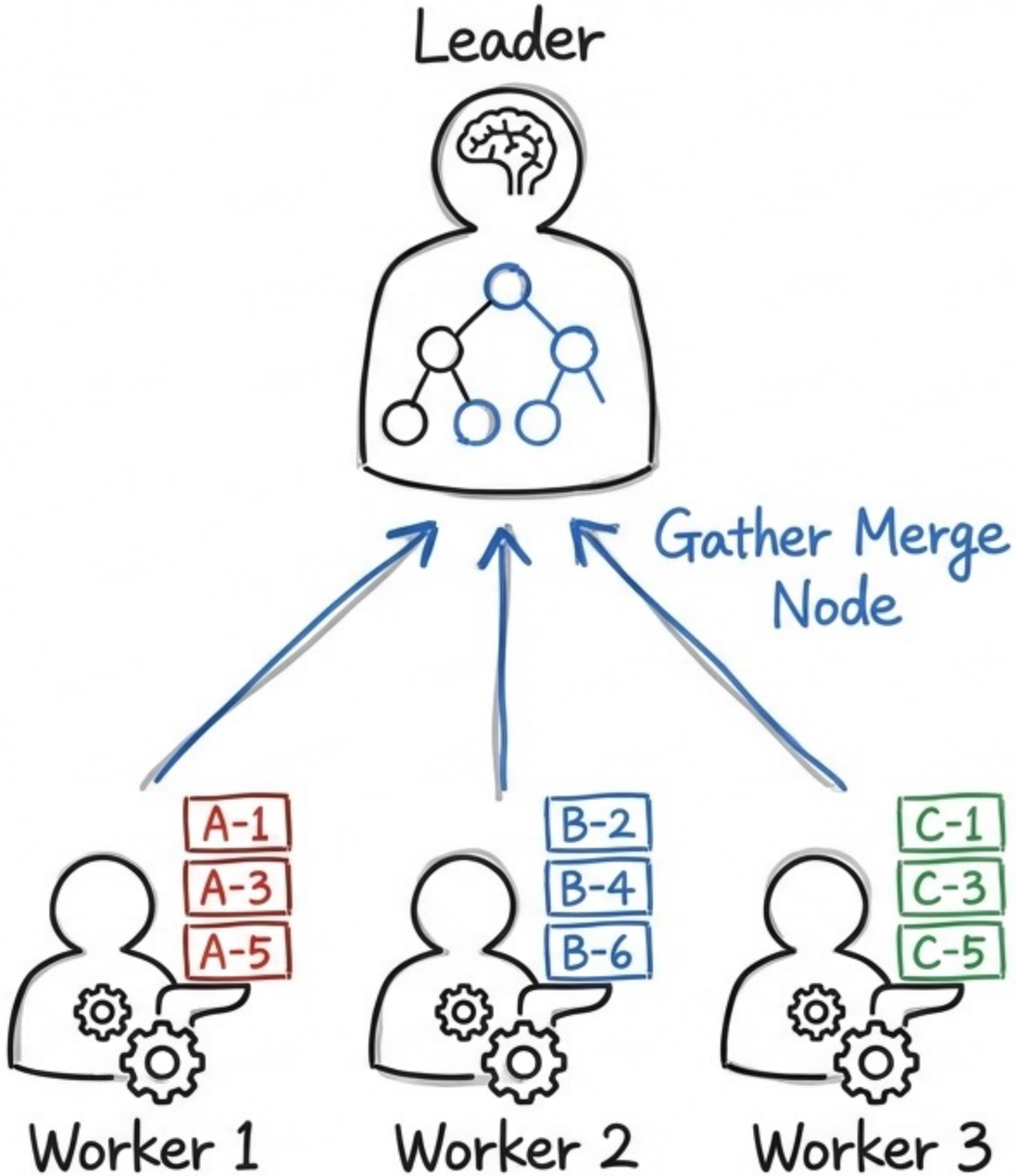
NotebookLM



최적화 2: 증분 정렬 (Incremental Sort)

- **상황:** 데이터가 첫 번째 키(Key 1)로 이미 정렬된 경우
- **방식:** 이미 정렬된 그룹(Key 1) 내에서만 나머지 키(Key 2...)를 정렬
- **효과:** 전체 데이터를 다시 정렬할 필요 없음. 메모리 절약 및 빠른 첫 행 반환

NotebookLM

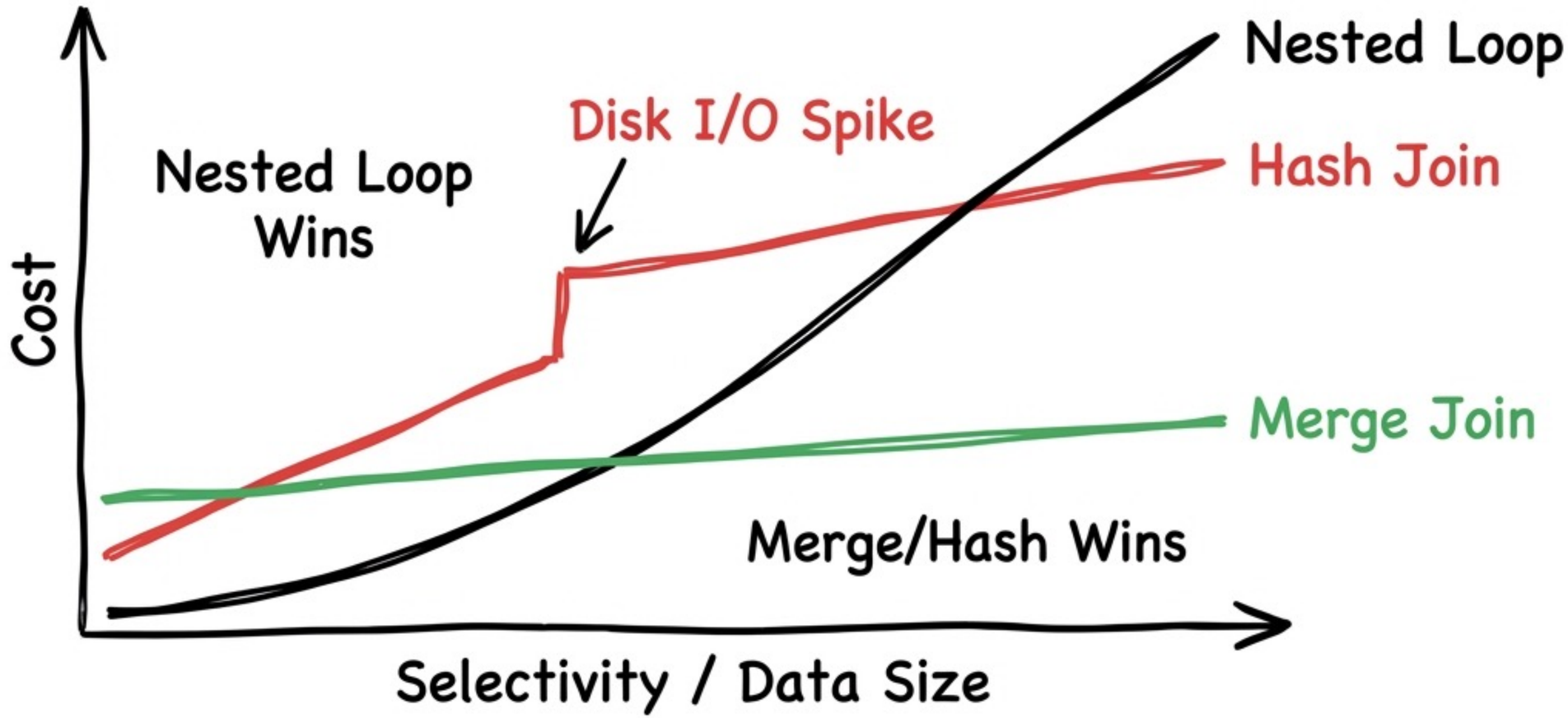


병렬 정렬 (Parallel Sorting & Gather Merge)

- Worker: 각자 할당된 데이터 분할을 정렬 (QuickSort/Top-N Heap)
- Leader (Gather Merge): 워커들의 정렬 순서를 유지하며 최종 취합
- 메커니즘: 리더는 '이진 힙(Binary Heap)'을 사용하여 워커들로부터 들어오는 데이터의 최소/최대값을 실시간으로 판별

NotebookLM

조인 전략 비교: 언제 병합 조인을 쓰는가?



- Small Data: Nested Loop 승리
- Large Data (Unsorted): Hash Join 승리
- Large Data (Sorted/Indexed): Merge Join 승리

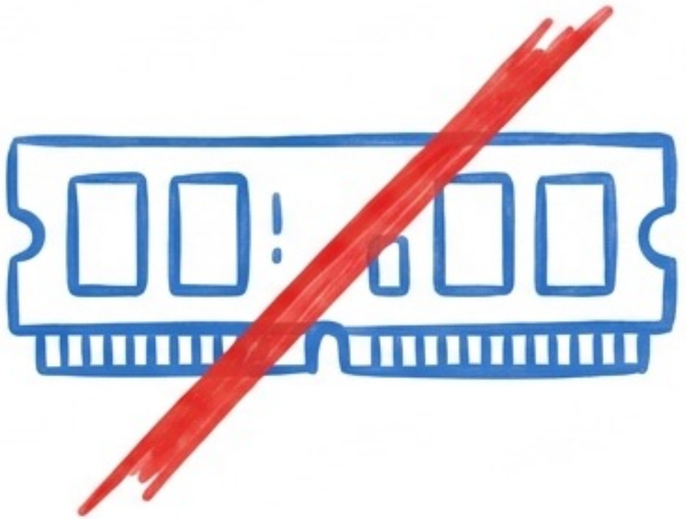
NotebookLM

핵심 요약 (Key Takeaways)



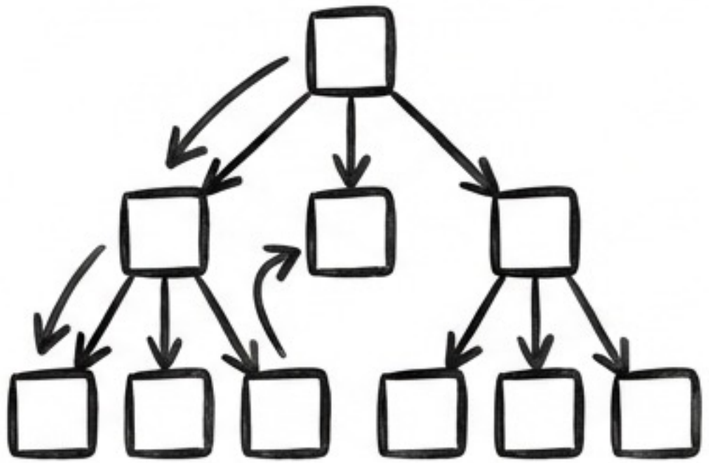
Merge Join

정렬된 데이터에 대해
가장 효율적인 조인 방식.



Work_mem

이 메모리 설정을 초과하면
디스크 기반 '외부 병합 정렬'이
발생하여 성능 저하 (I/O).



Index 활용

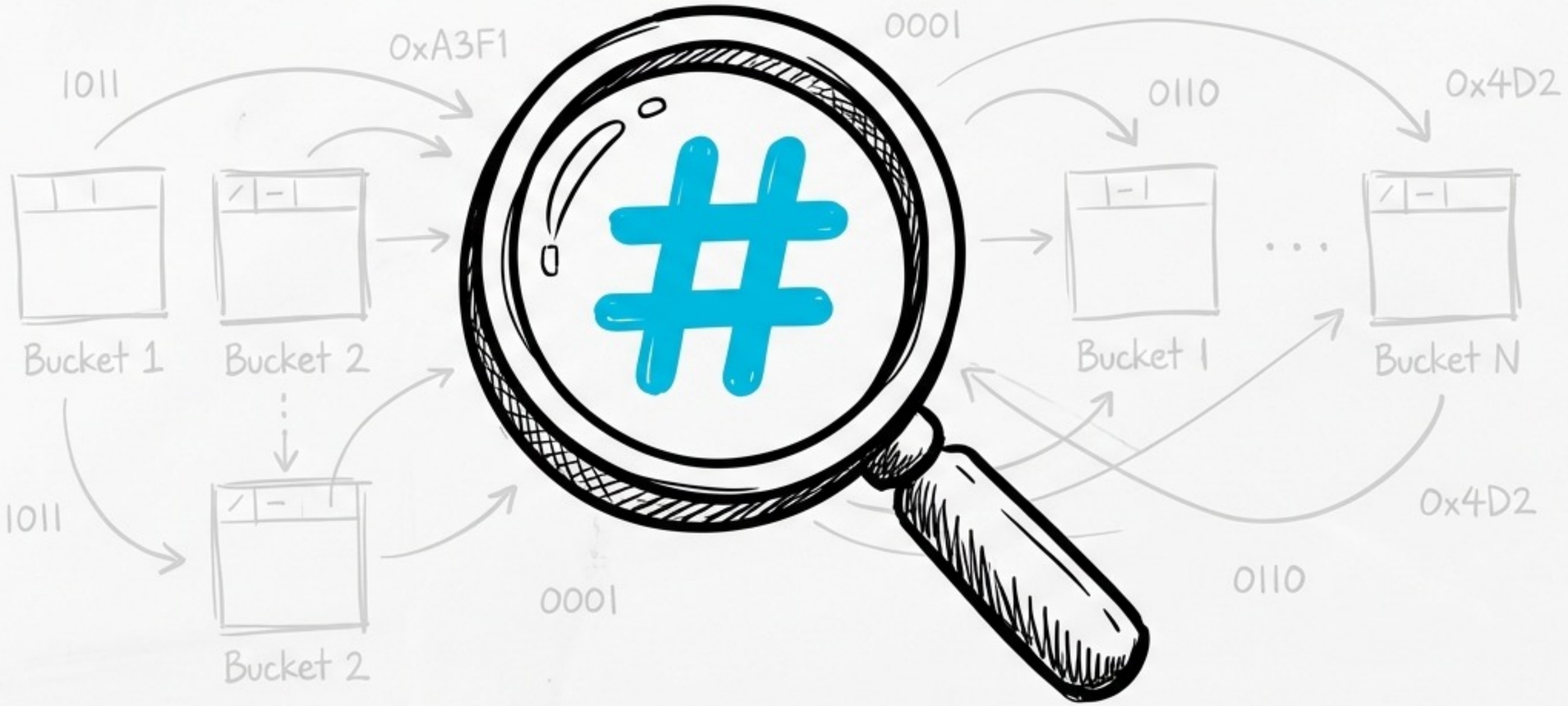
적절한 인덱스는 정렬
비용(Sort Node)을 제거하여
Merge Join의 성능을 극대화함.

NotebookLM

- **24. Hash**

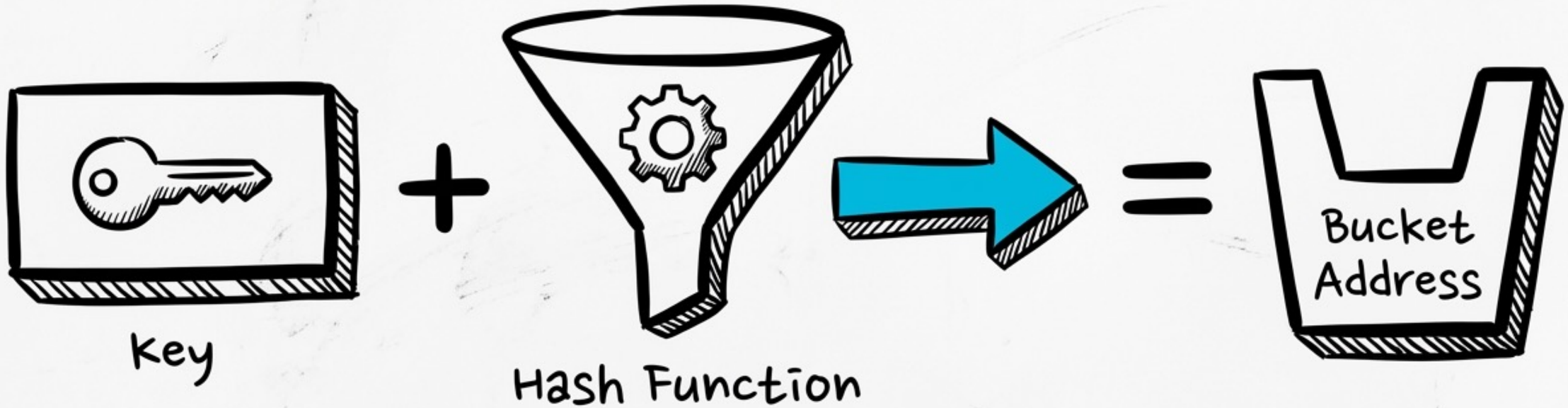
PostgreSQL 해시 인덱스 구조와 원리

기계적 해부학: 청사진에서 작동 원리까지



NotebookLM

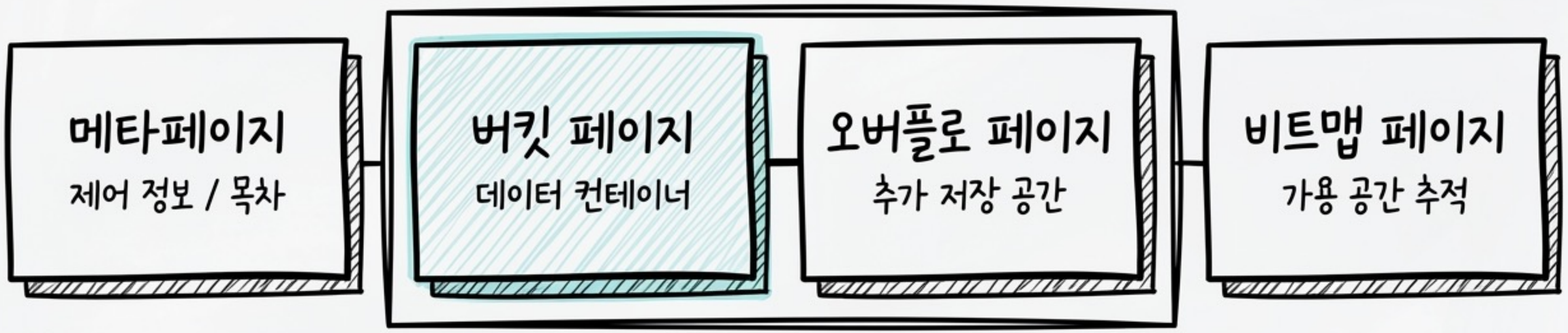
해시 인덱스란 무엇인가?



- 디스크 기반 해시 테이블
- 동등 조건(=`) 검색만 지원
- 인덱스 키 값은 저장하지 않음
- 해시 코드와 TID(Tuple ID)만 저장

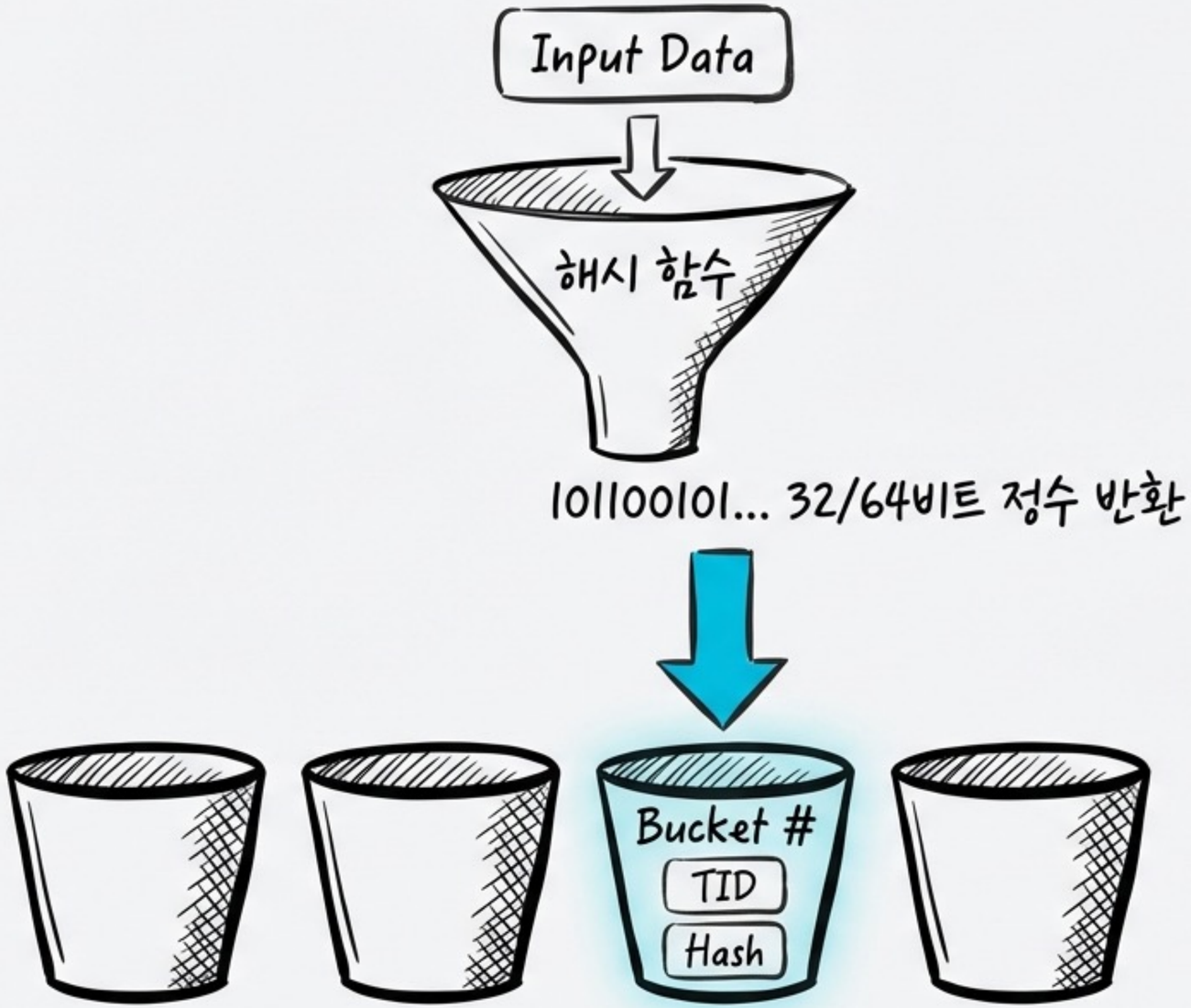
NotebookLM

4가지 페이지 레이아웃



NotebookLM

데이터 삽입의 흐름



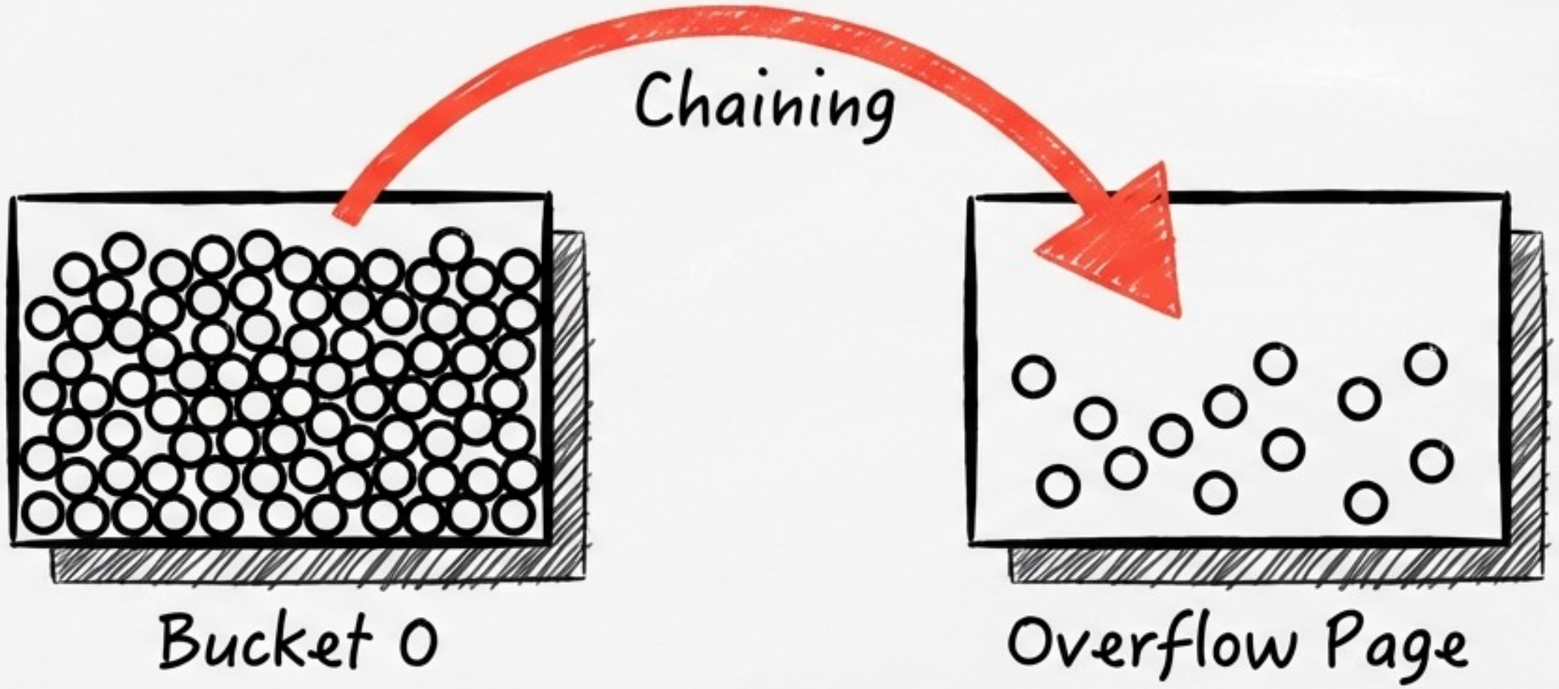
1. 해시 함수:
32비트 또는 64비트 정수 반환

2. 버킷 번호 결정:
해시 값의 하위 비트를 사용

3. 저장:
TID와 해시 코드를 버킷에 추가

NotebookLM

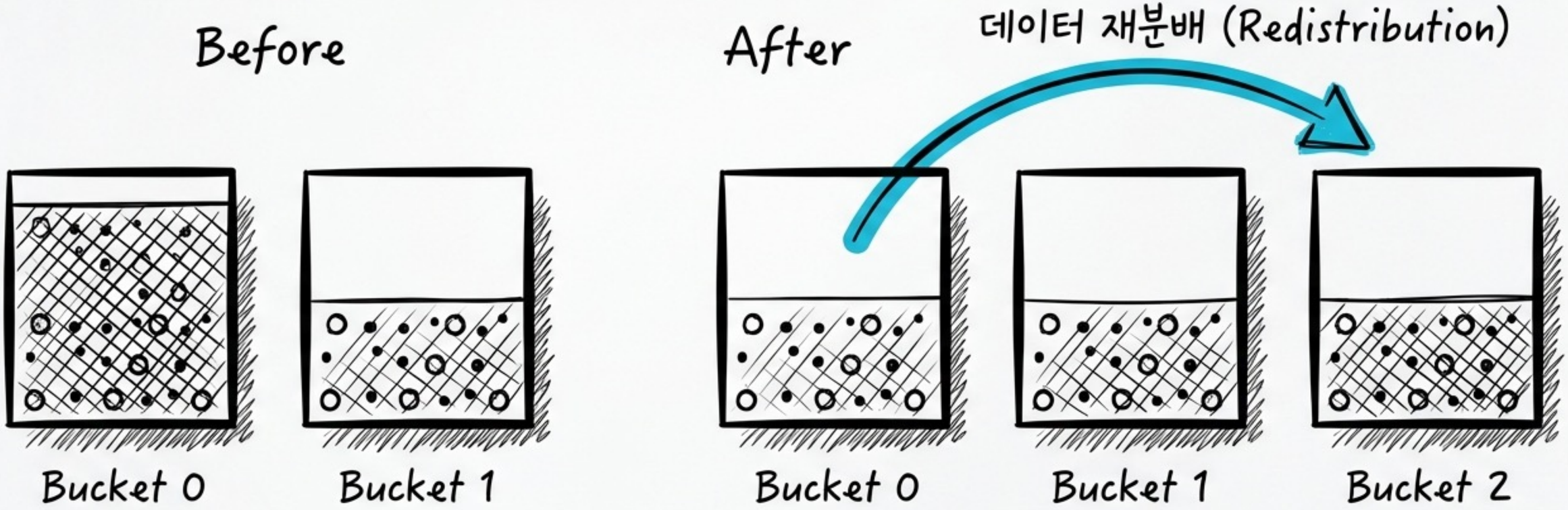
해시 충돌과 오버플로



- 발생 원인: 데이터가 ffactor(채우기 비율)를 초과할 때
- 처리 방식: 체이닝(Chaining) 방식으로 오버플로 페이지 연결
- 성능 이슈: 페이지가 분산되어 검색 속도 저하

NotebookLM

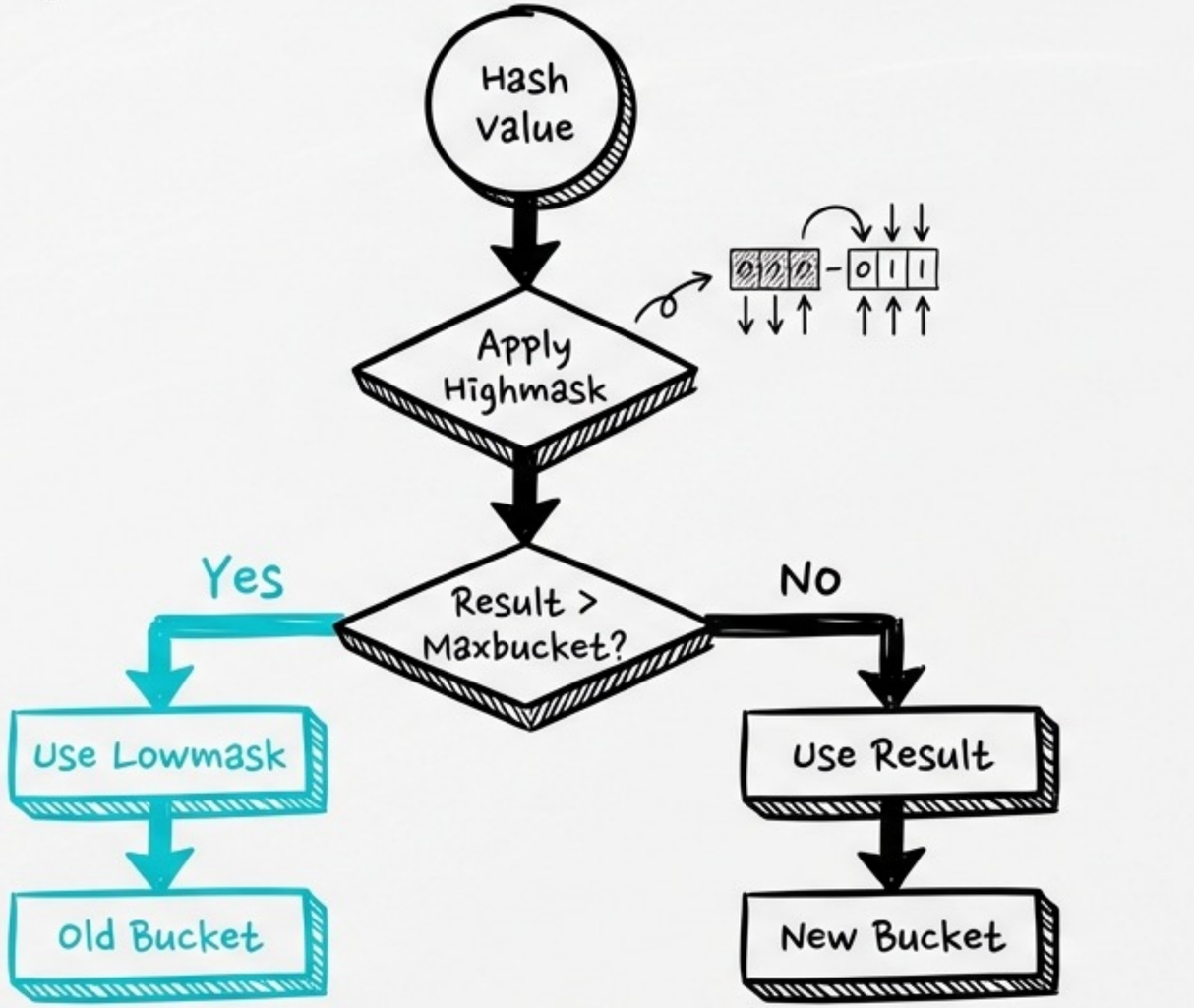
스플릿과 확장: 리니어 해싱



- **점진적 확장**: 전체 테이블 락(Lock) 없이 한 번에 하나의 버킷만 분할
- **재분배**: 해시 코드의 비트를 하나 더 사용하여 데이터 이동

NotebookLM

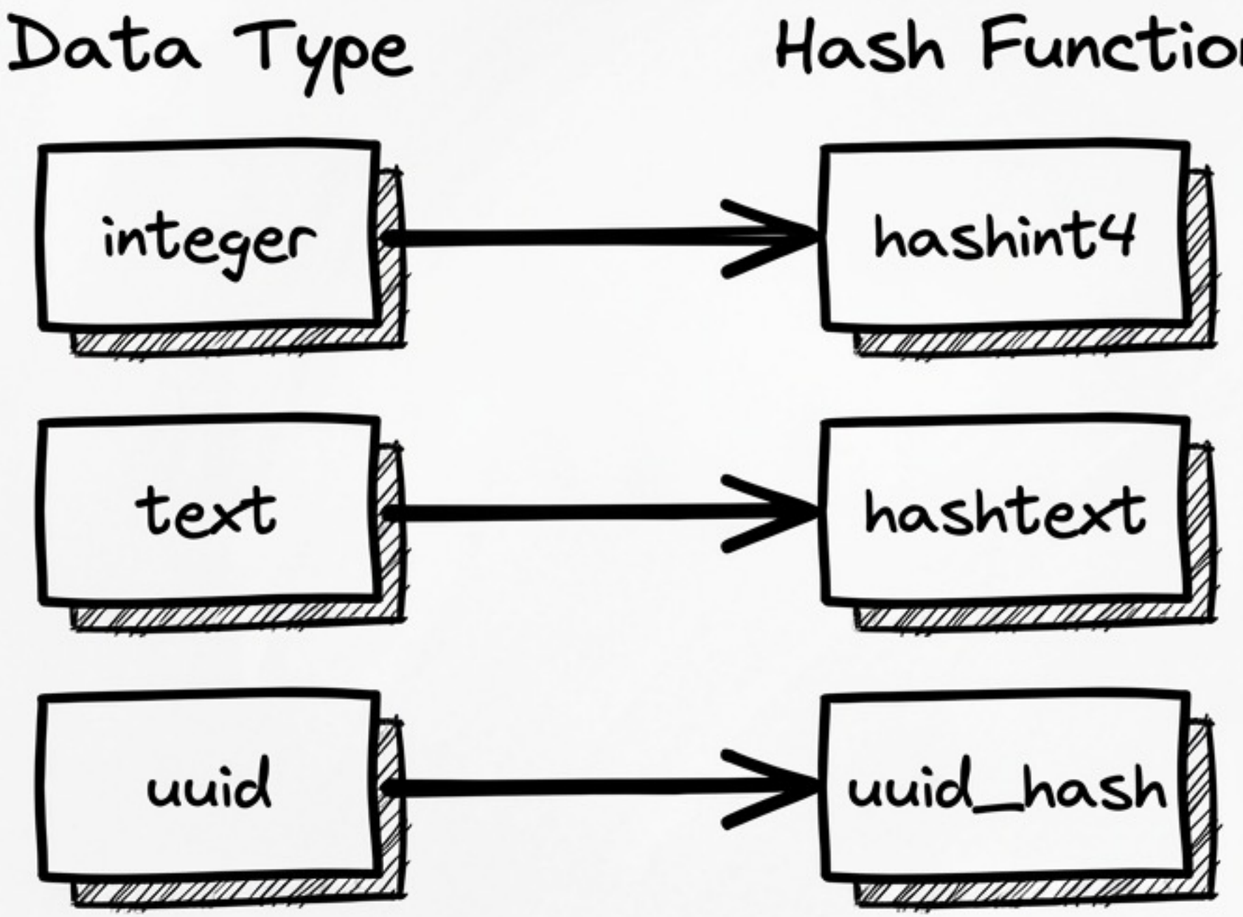
동적 주소 지정 로직



스플릿 진행 중에도 정확한 버킷을 찾아가는 알고리즘

NotebookLM

연산자 클래스와 함수



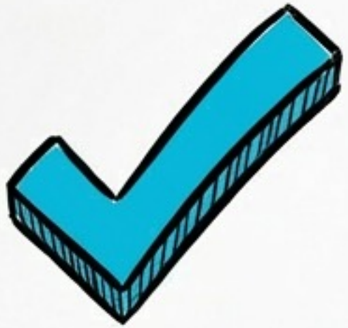
pg_opfamily: 데이터 타입별 해시 함수 매핑 관리

사용자 정의 데이터 타입도 지원 가능

NotebookLM

제약 사항 및 특징

Allowed



- 동등 비교 (`=`)
- WAL 로깅 지원 (PG 10+) - 크래시 안전
- 배타적 제약조건 (`EXCLUDE`)

Not Allowed



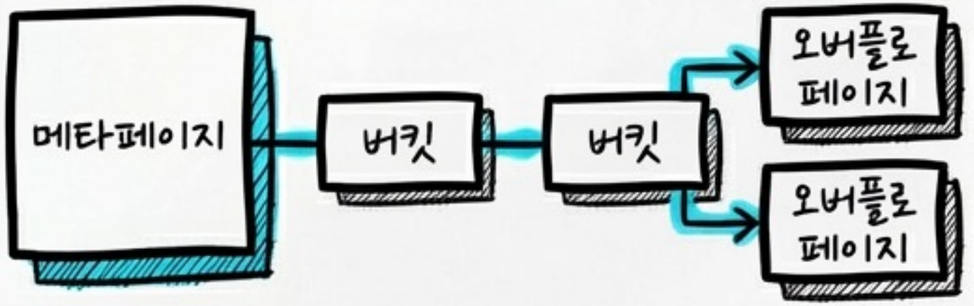
- 범위 검색 (`<, `>`)
- 정렬 (`ORDER BY`)
- 유일성 제약 (`UNIQUE`)
- 인덱스 전용 스캔

NotebookLM

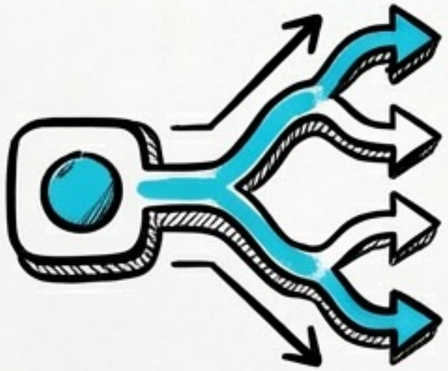
핵심 요약



목적: 대용량 데이터의 동등(=) 조건 고속 검색



구조: 메타페이지 + 버킷 + 오버플로 페이지



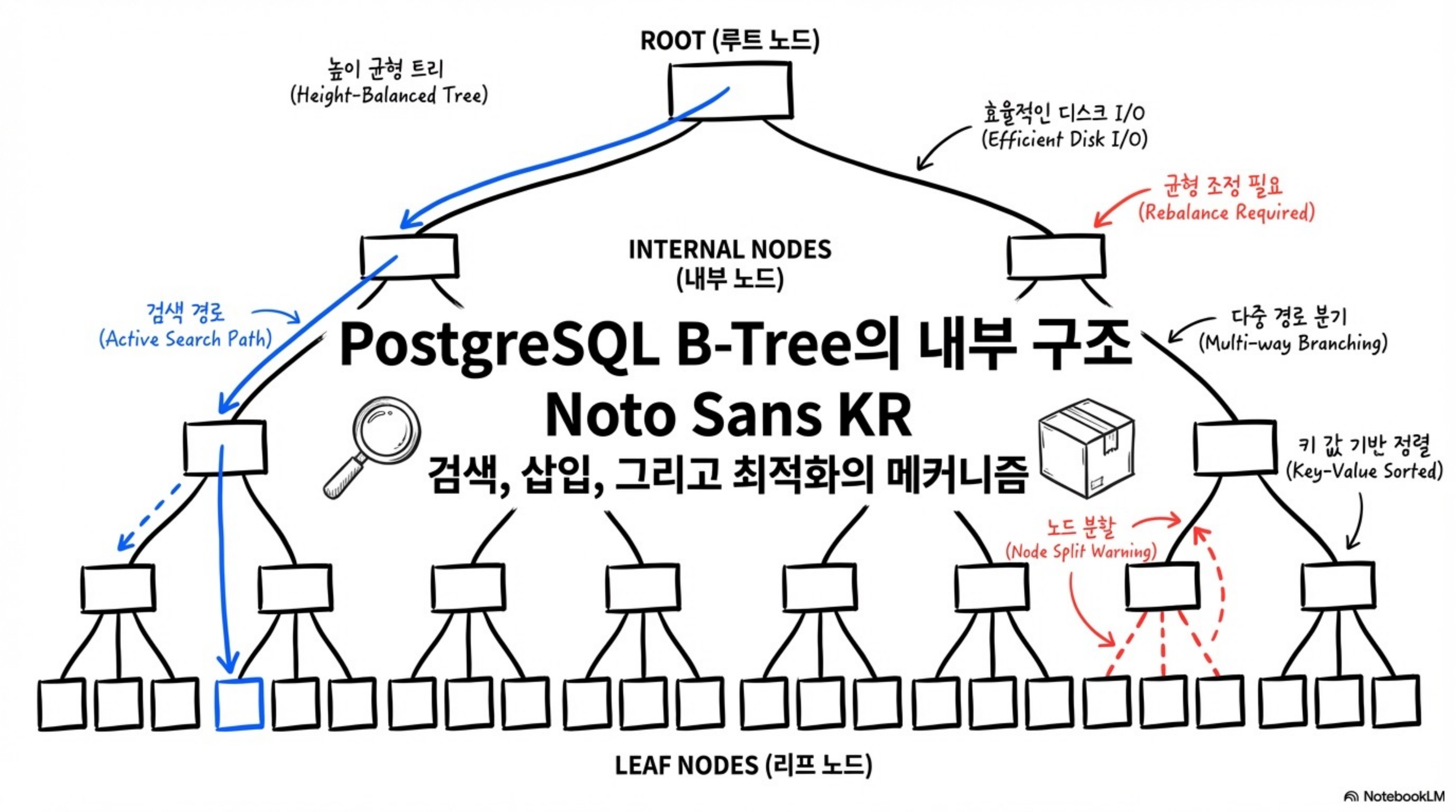
확장성: 리니어 해싱을 통한 점진적 스플릿



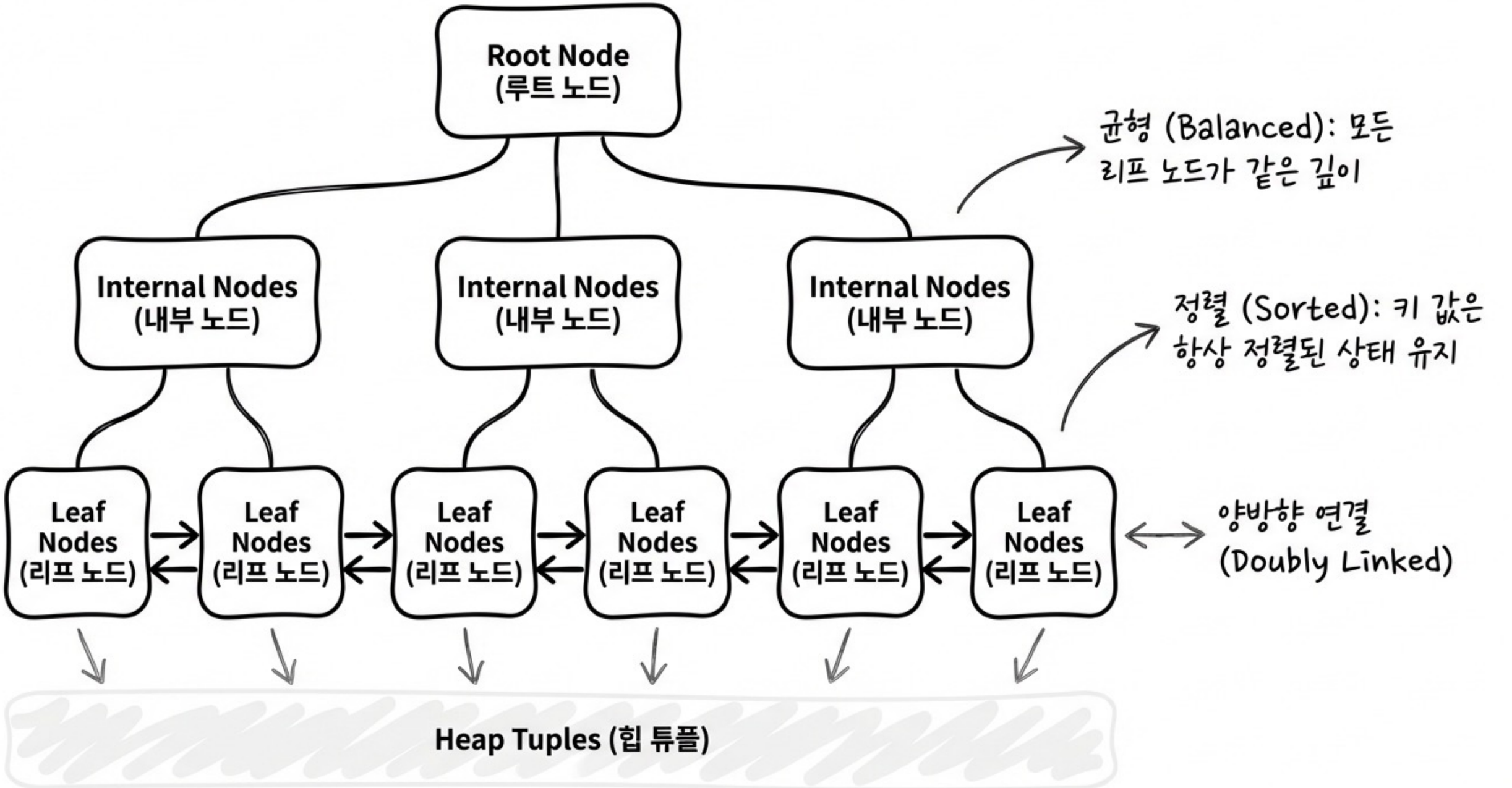
주의: 정렬 및 범위 검색 불가

NotebookLM

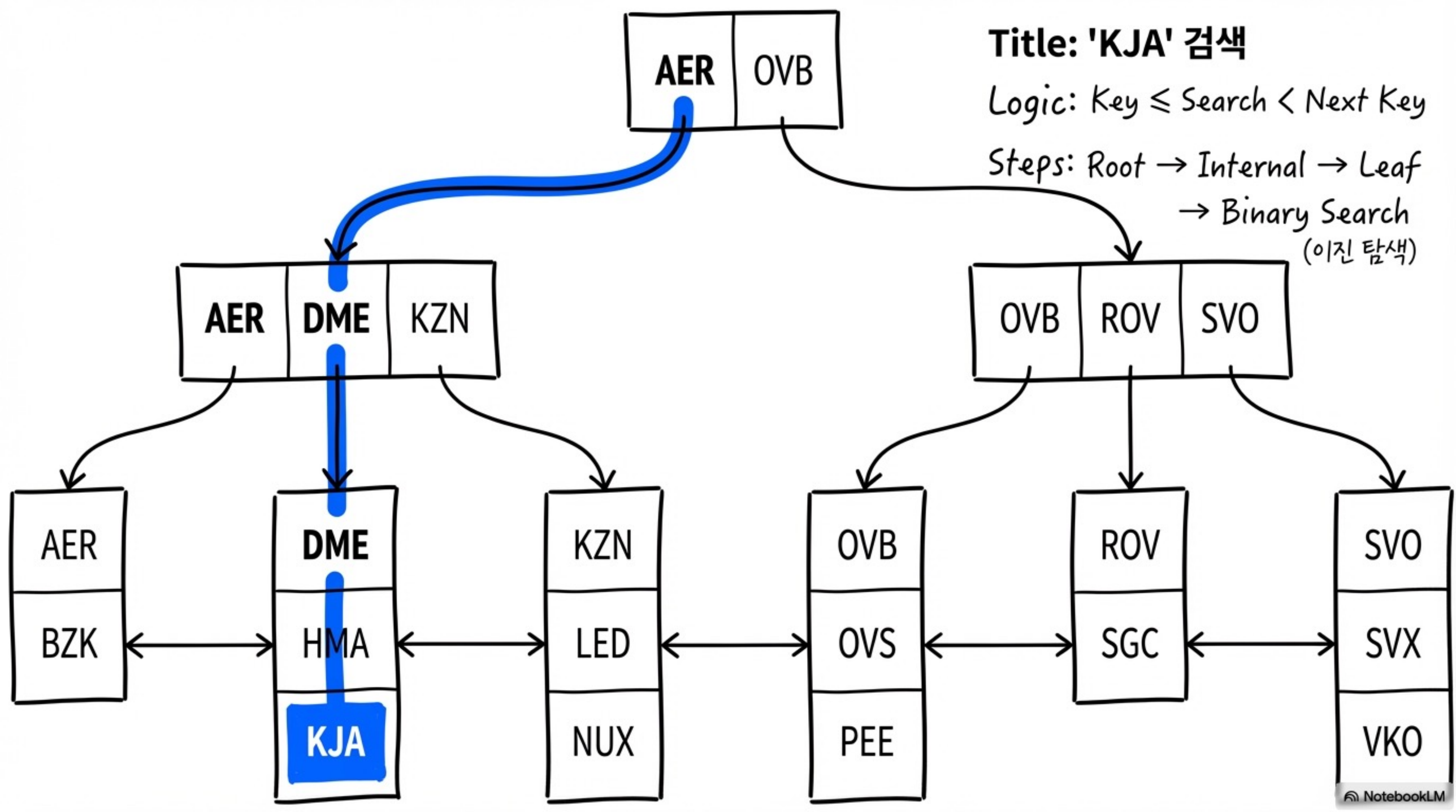
•25. B-tree


ROOT (루트 노드)
높이 균형 트리
(Height-Balanced Tree)
효율적인 디스크 I/O
(Efficient Disk I/O)
균형 조정 필요
(Rebalance Required)
INTERNAL NODES
(내부 노드)
검색 경로
(Active Search Path)
다중 경로 분기
(Multi-way Branching)
PostgreSQL B-Tree의 내부 구조
Noto Sans KR
검색, 삽입, 그리고 최적화의 메커니즘
키 값 기반 정렬
(Key-Value Sorted)
노드 분할
(Node Split Warning)
LEAF NODES (리프 노드)

NotebookLM


Root Node
(루트 노드)
Internal Nodes
(내부 노드)
Internal Nodes
(내부 노드)
Internal Nodes
(내부 노드)
Leaf
Nodes
(리프 노드)
Leaf
Nodes
(리프 노드)
Leaf
Nodes
(리프 노드)
Leaf
Nodes
(리프 노드)
Leaf
Nodes
(리프 노드)
Leaf
Nodes
(리프 노드)
Heap Tuples (힙 튜플)
균형 (Balanced): 모든
리프 노드가 같은 깊이
정렬 (Sorted): 키 값은
항상 정렬된 상태 유지
양방향 연결
(Doubly Linked)

NotebookLM


Title: 'KJA' 검색
Logic: Key ≤ Search < Next Key
Steps: Root → Internal → Leaf
→ Binary Search
(이진 탐색)
AER
OVB
AER
DME
KZN
OVB
ROV
SVO
AER
BZK
DME
HMA
KJA
KZN
LED
NUX
OVB
OVS
PEE
ROV
SGC
SVO
SVX
VKO
NotebookLM

Before

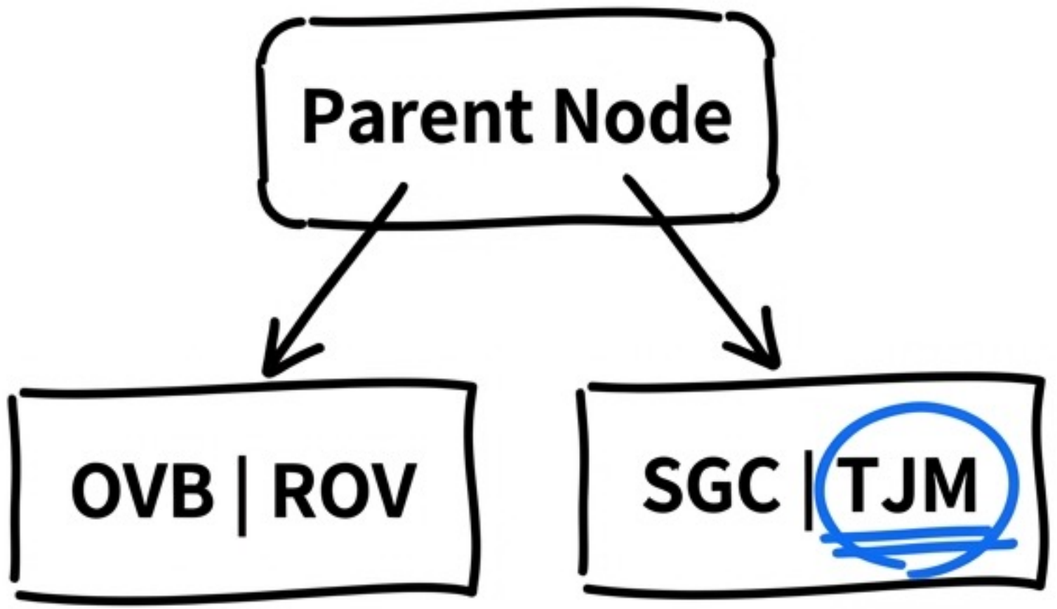
공간 부족 (Page Full)

Insert 'TJM'

OVB | ROV | SGC

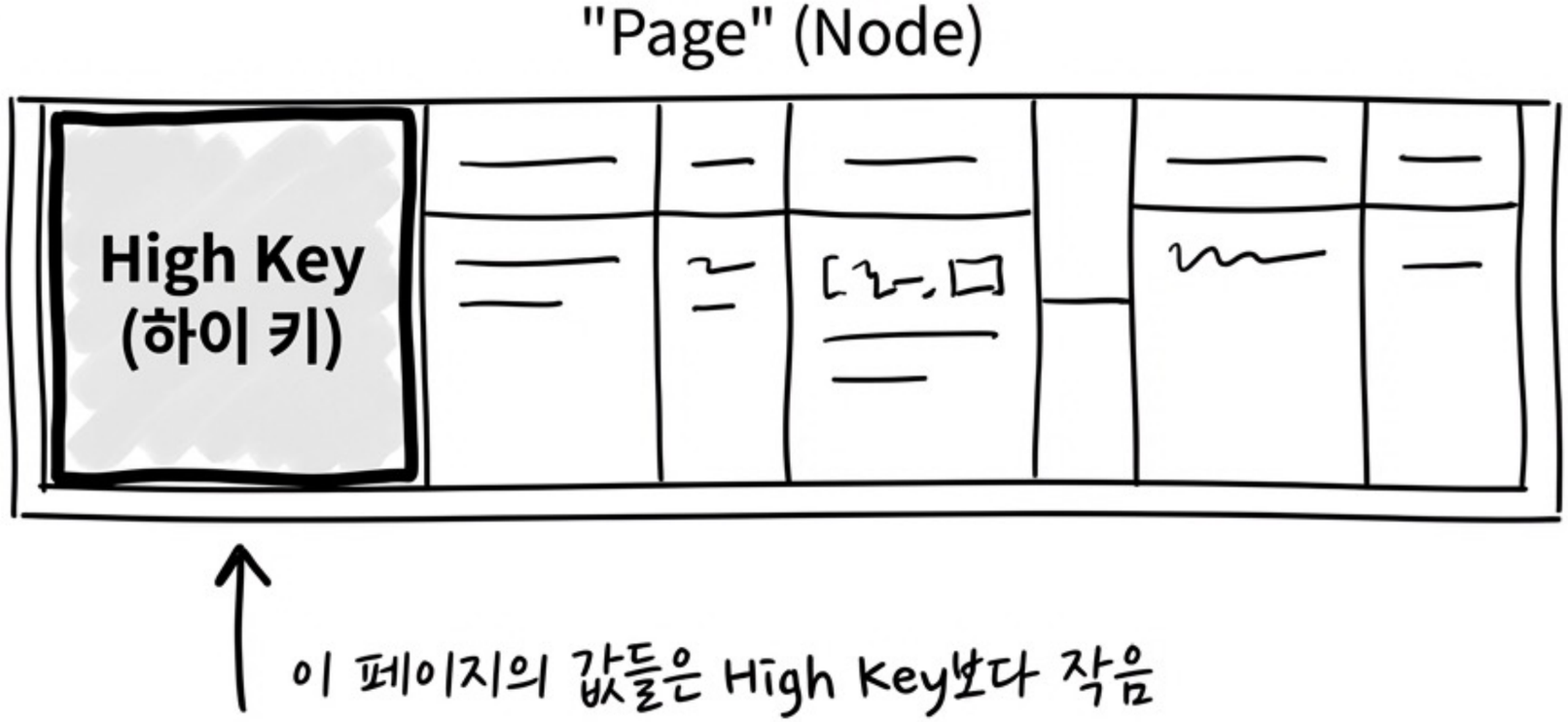
After

페이지 분할 (Page Split)



Result: 트리 균형 유지, 부모 노드에 포인터 추가

NotebookLM

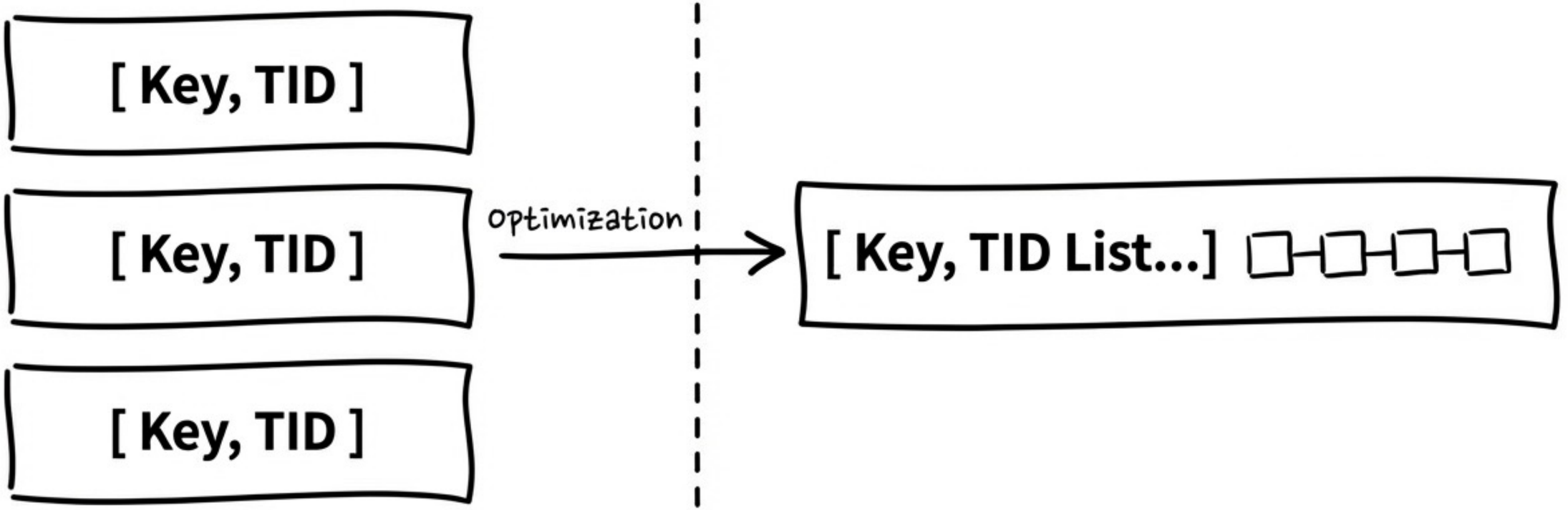


- Role: 페이지 스캔 범위 결정
- Note: 우측 끝 페이지에는 없음

NotebookLM

Standard (일반 저장)

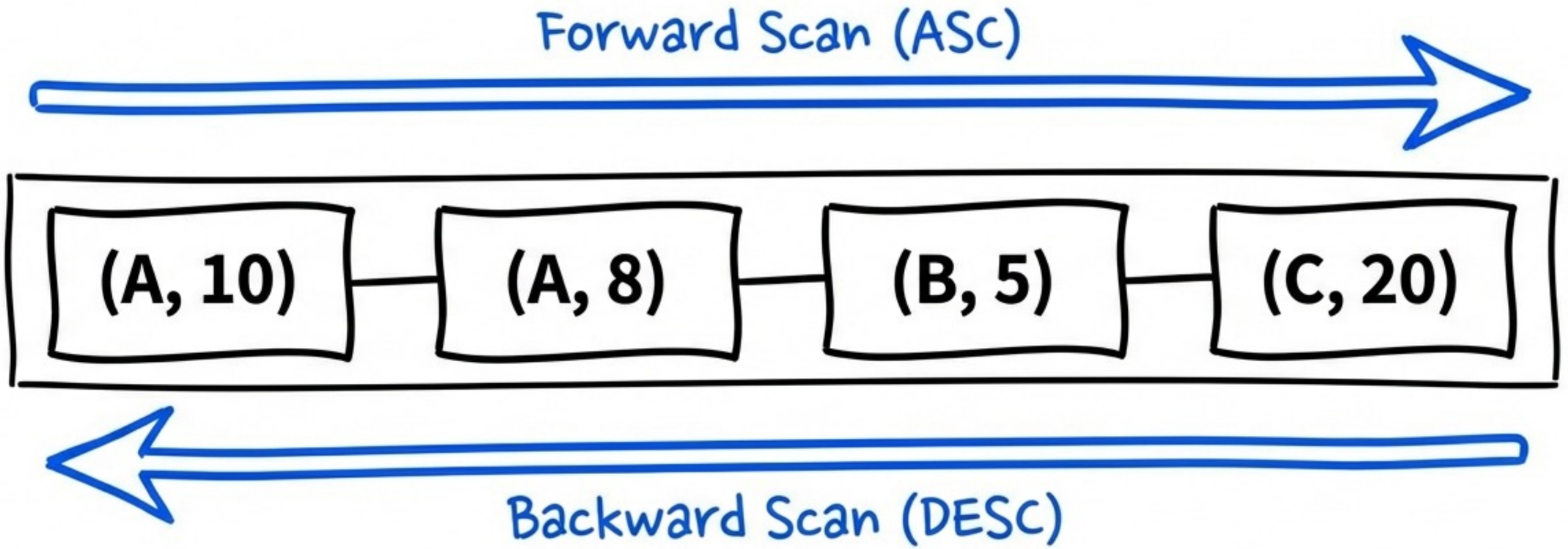
Deduplicated (중복 제거)



☞ - Deduplication (중복 제거): 포스팅 리스트(Posting List) 사용
- Benefit: 인덱스 크기 감소, 페이지 분할 방지
- Constraint: 단순 이진 비교가 가능한 데이터만 지원

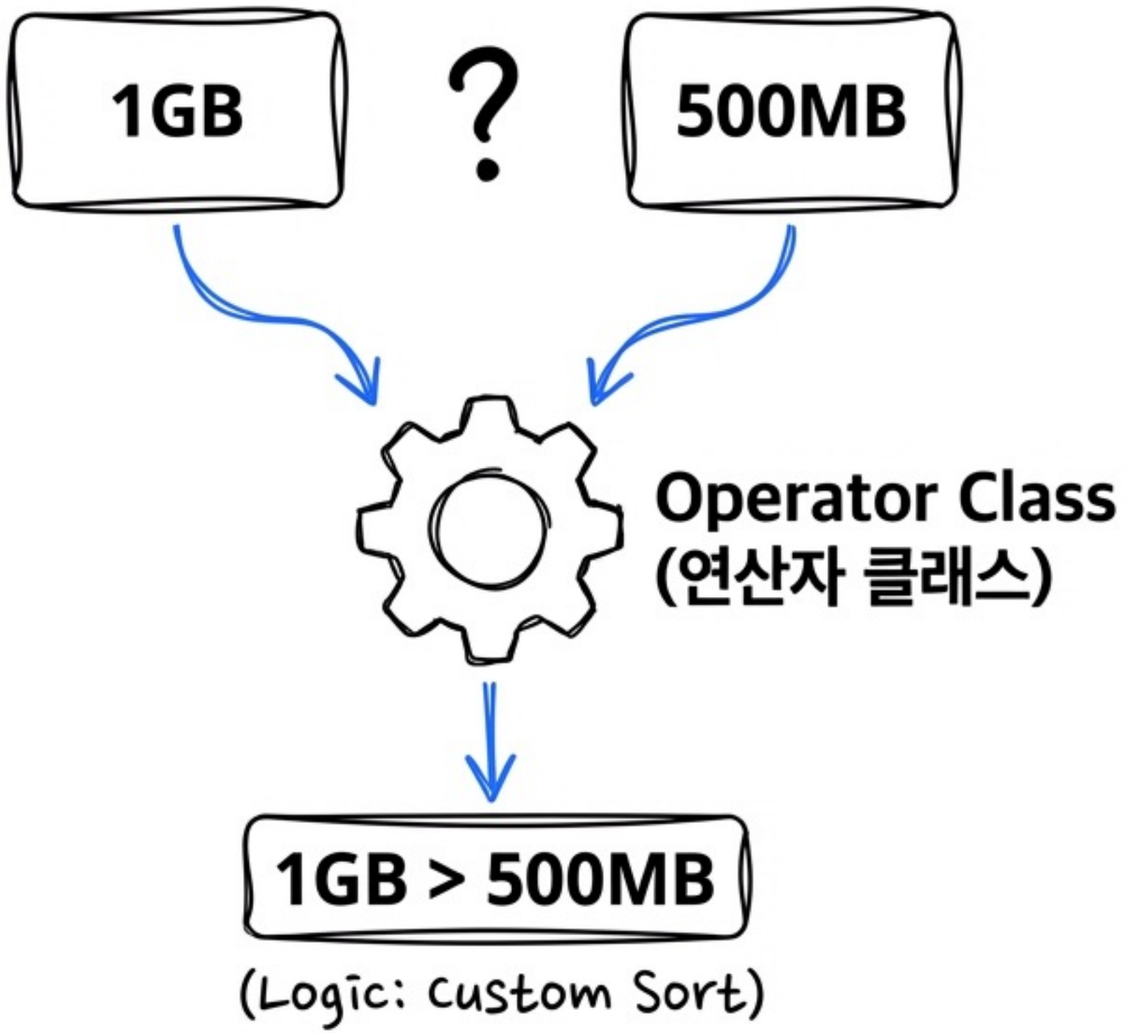
NotebookLM

Index: (Name ASC, Ref DESC)



☞ - 다중 컬럼 (Multi-Column): 첫 번째 컬럼부터 순서대로 정렬
- Scan Direction: ORDER BY DESC 쿼리는 역방향 스캔으로 처리
- Efficiency: 인덱스 정의 순서가 검색 효율 결정

NotebookLM



Strategies (전략)

1. < → (Less than)
2. <= → (Less or Equal)
3. = → (Equal)
4. >= → (Greater or Equal)
5. > → (Greater)

☞ Role: 데이터 타입의 비교 및 정렬 규칙 정의

NotebookLM

- **정렬 및 유일성 (Order & Unique):**
  데이터를 정렬하고 유일성을 보장하는 유일한 인덱스
- **NULL 처리:** NULLS FIRST/LAST 지원
- **다중 컬럼 (Multi-Column):** 여러 컬럼에 대한 복합 인덱스 지원
- **양방향 탐색 (Bidirectional):** ASC, DESC 모두 효율적 처리
- **커버링 인덱스:** INCLUDE 절을 통한 부가 정보 저장 가능

PostgreSQL의 가장 범용적이고 강력한 인덱스

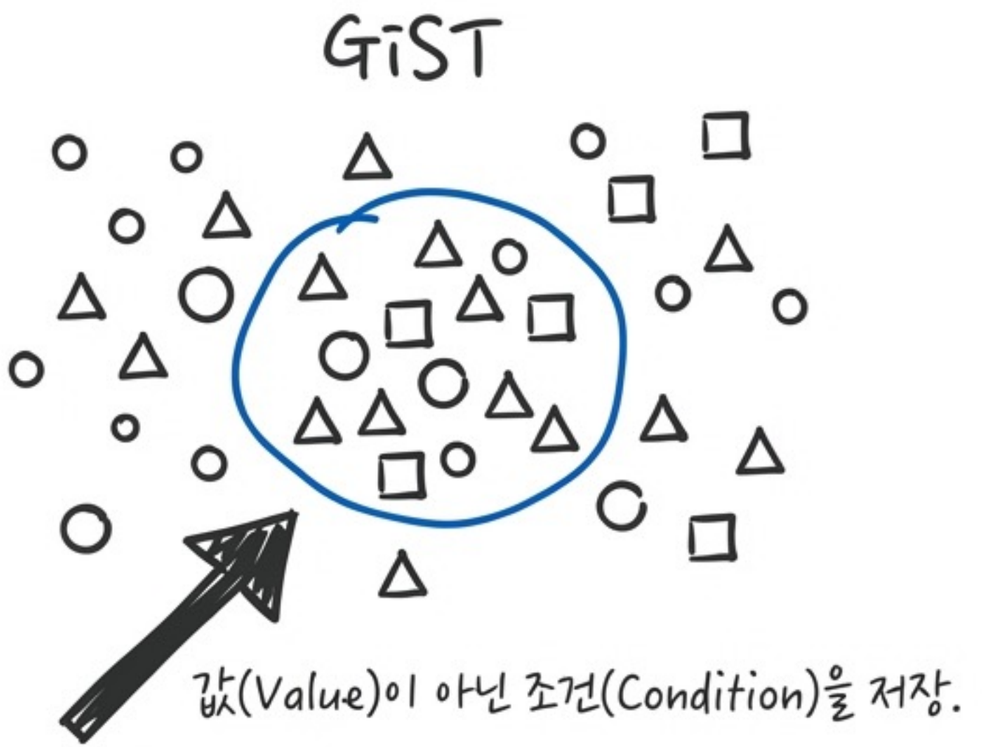
NotebookLM

- **26. GiST**

PostgreSQL GiST

복잡한 데이터를 위한 인덱싱 프레임워크





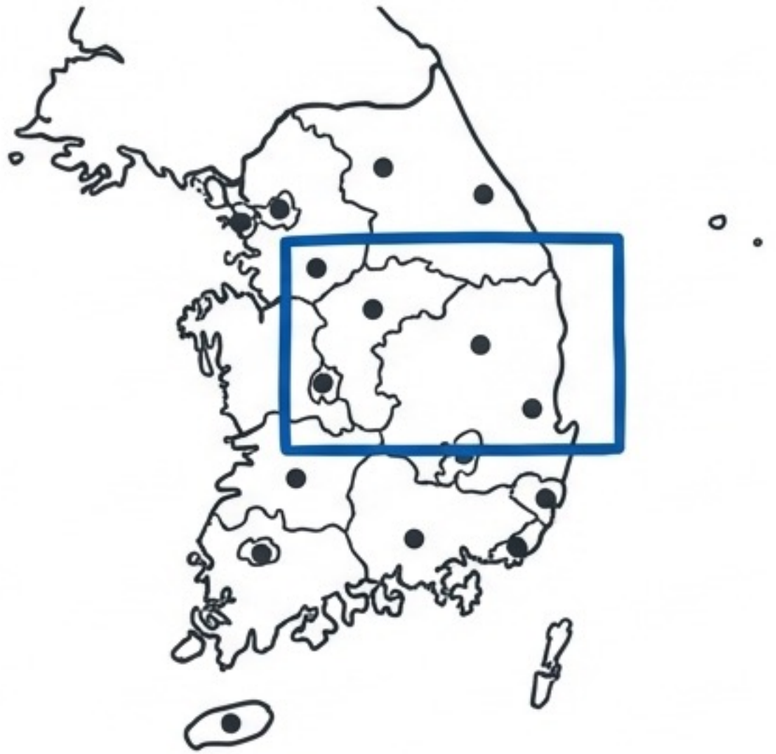
순서를 넘어, 관계를 인덱싱하다

NotebookLM

B-Tree의 한계와 GiST의 접근



?

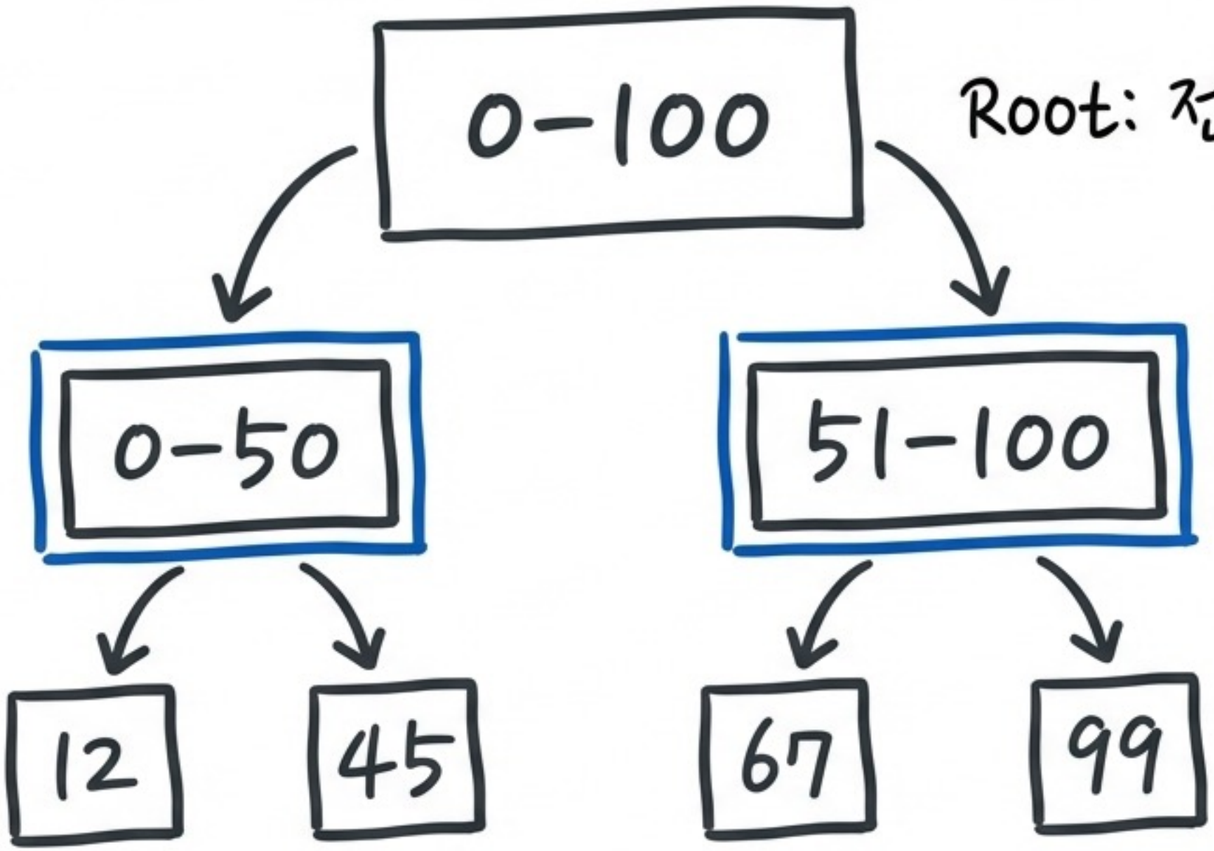


- 정렬(Sort)과 일치(=)에 강력함
- 공간/다차원 데이터에는 한계

- 데이터의 **상대적 위치**와 **포함 관계**
- 임의의 기준으로 그룹화 (Grouping)

NotebookLM

균형 검색 트리 구조



Root: 전체 조건

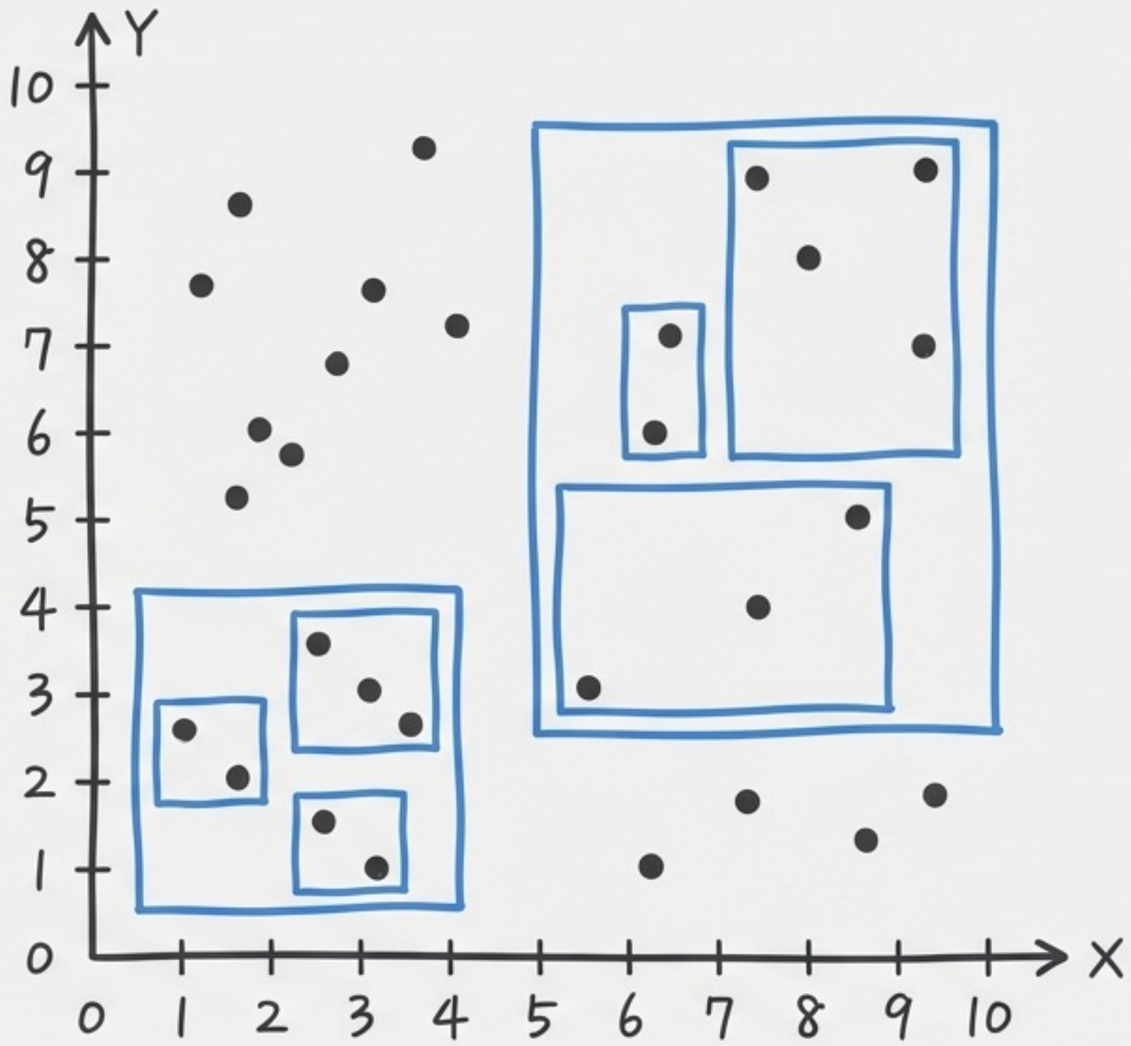
Internal Node: 하위 트리를 포함하는 조건(Predicate)

Leaf Node: 실제 데이터 (TID)

내부 항목의 조건은 자식 항목들의 합집합(Union)입니다.

NotebookLM

R-Tree: 기하학적 데이터 인덱싱



경계 상자 (Bounding Box)

점들을 감싸는 사각형을 계층적으로 구성

검색 원리

찾는 영역과 **겹치는**(Overlap) 상자만 탐색

일관성 함수 (Consistency Function)

"이 상자 안에 데이터가 있을 가능성이 있는가?"

NotebookLM

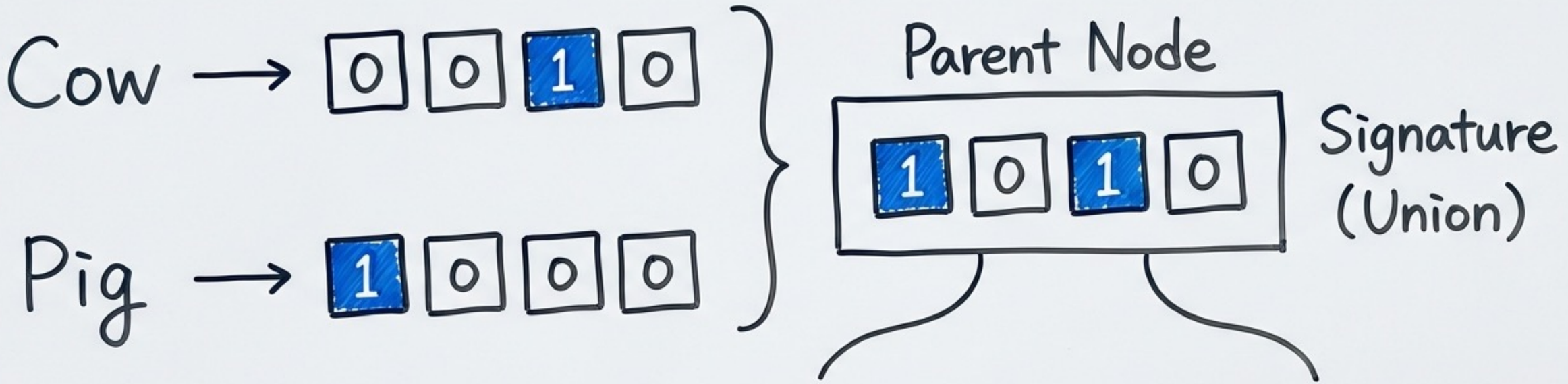
k-NN (최근접 이웃) 검색 효율성



- **거리 연산자 : <->**
- **가지치기 (Pruning) :**
  가까운 상자부터 우선 탐색
- **효율성 : 모든 점을 계산하지 않음**

NotebookLM

전문 검색 (Full Text Search)과 RD-Tree



- 시그니처 (Signature) : 단어 → 해시 → 비트열
- 블룸 필터 (Bloom Filter) : 비트맵의 합집합(Union) 저장
- 검색 : 쿼리 비트가 켜져 있는 노드만 방문

NotebookLM

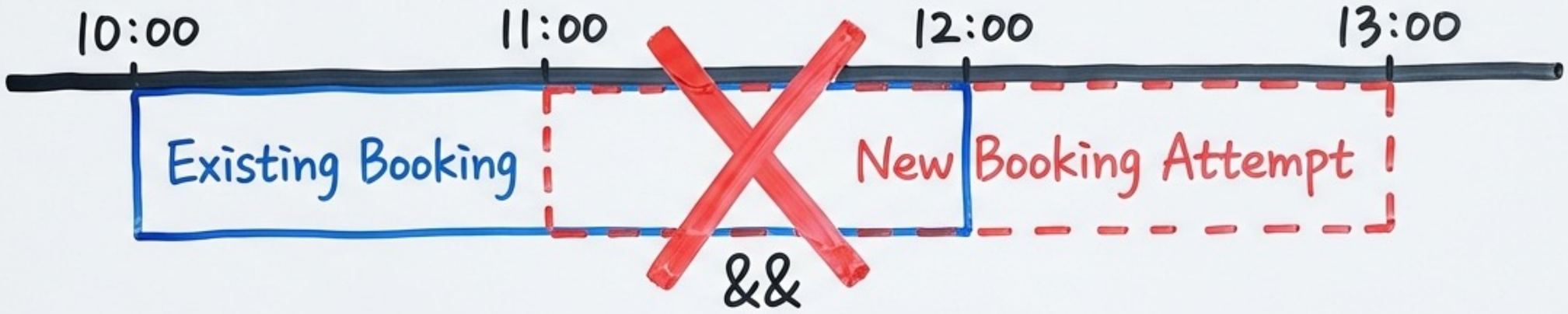
거짓 양성 (False Positive)과 재확인



- 압축의 대가: 해시 충돌 가능성
- Recheck Cond: 테이블(Heap)에서 실제 데이터 확인
- 손실 압축: 시그니처가 짧을수록 인덱스는 작지만 재확인 비용 증가

NotebookLM

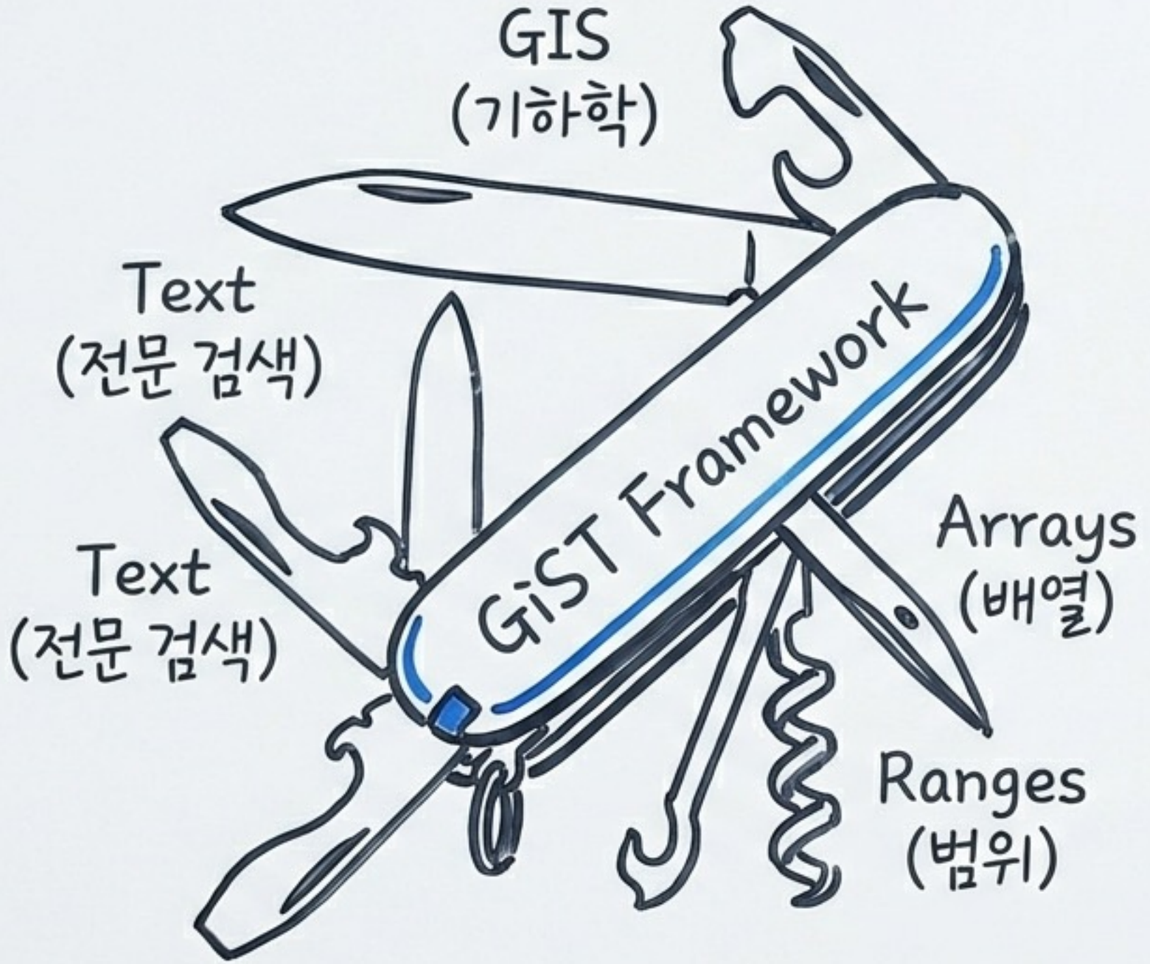
배타적 제약 조건 (Exclusion Constraints)



- 개념 : "겹치지 않음"을 보장 (예: 예약 시스템)
- 연산자 : && (Overlap) 등으로 충돌 감지
- 활용 : 시간, 범위, 기하학 데이터의 무결성

NotebookLM

요약: GiST는 인덱스 제작 키트다



- **프레임워크 :** 연산자 클래스에 따라 동작 결정
- **핵심 철학 :** “데이터의 모양(Shape)과 관계(Relation)를 이해”
- **B-Tree와의 차이 :** 단순 정렬이 아닌 복잡한 조건 처리

NotebookLM

- **27. SP-GiST**

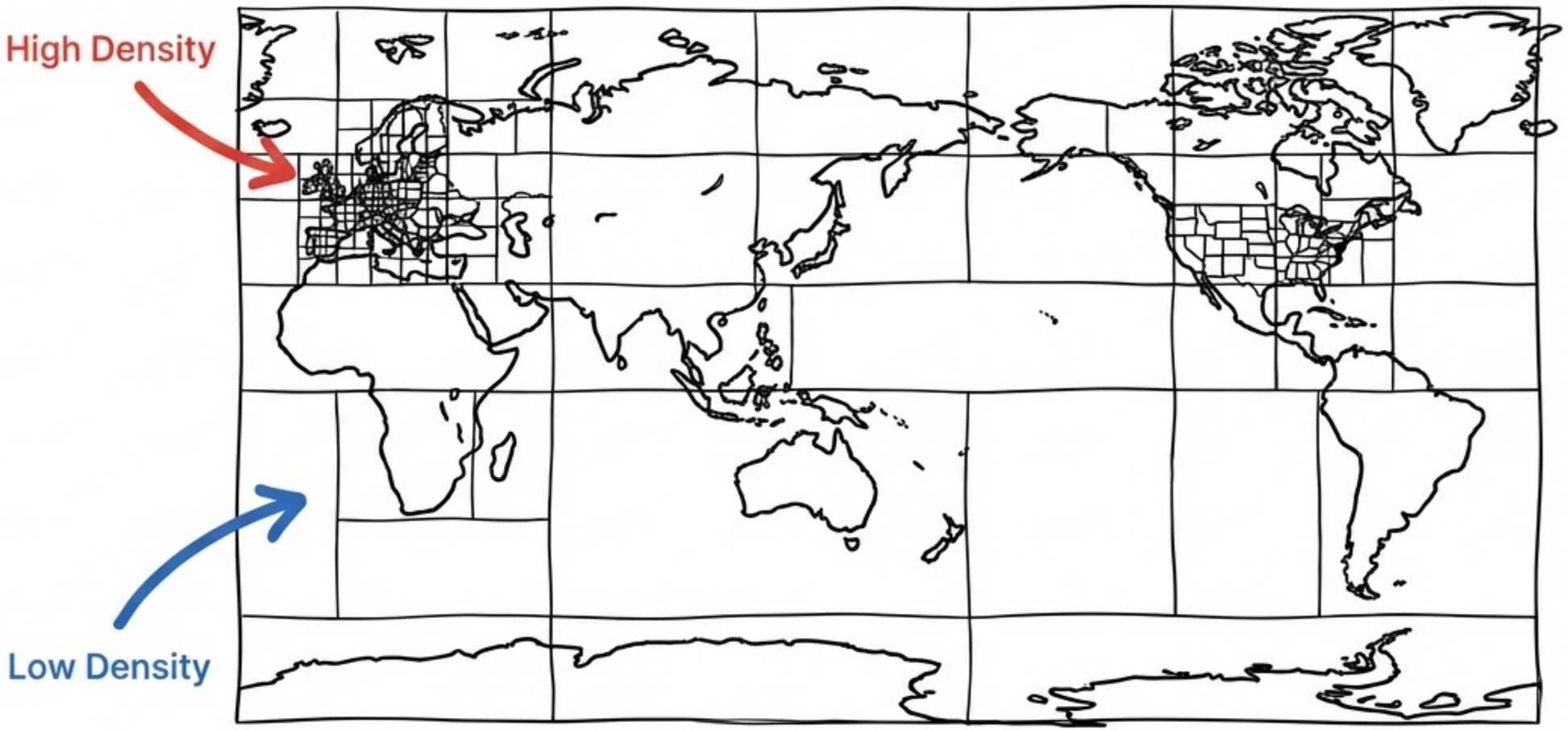


SP-GiST: 공간 분할의 미학

PostgreSQL의 Divide and Conquer 인덱싱 전략

NotebookLM

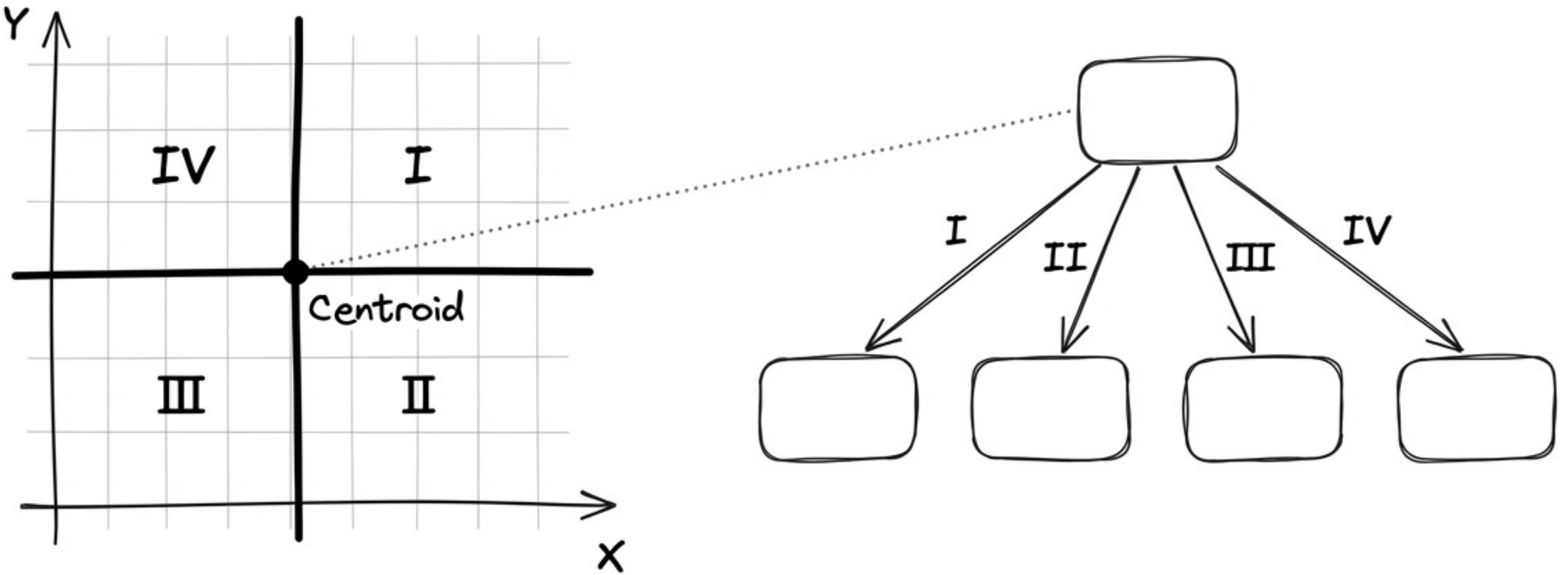
SP-GiST란 무엇인가?



- **Space Partitioning (공간 분할):** 검색 공간을 겹치지 않는(Non-overlapping) 영역으로 재귀적 분할.
- **Unbalanced Tree:** 데이터 밀도가 높은 곳은 더 잘게 쪼갠다 (깊이가 달라짐).

NotebookLM

쿼드트리 (Quadtree) - 2차원 공간의 4분할

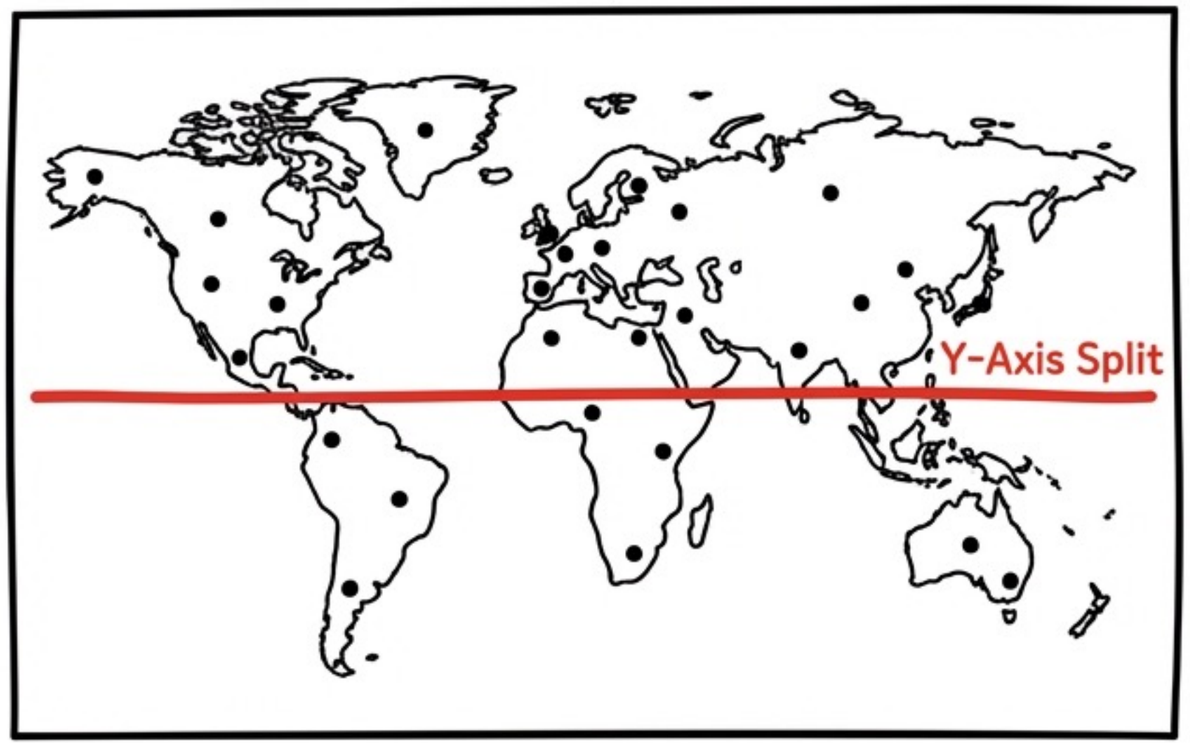


- **Mechanism:** 중심점(Centroid)을 기준으로 4개의 사분면(I, II, III, IV)으로 분할.
- **Role:** 중심점은 하위 노드의 범위를 결정하는 조건(Prefix)이 됨.
- **Use Case:** 위도/경도(Point) 데이터 인덱싱에 최적.

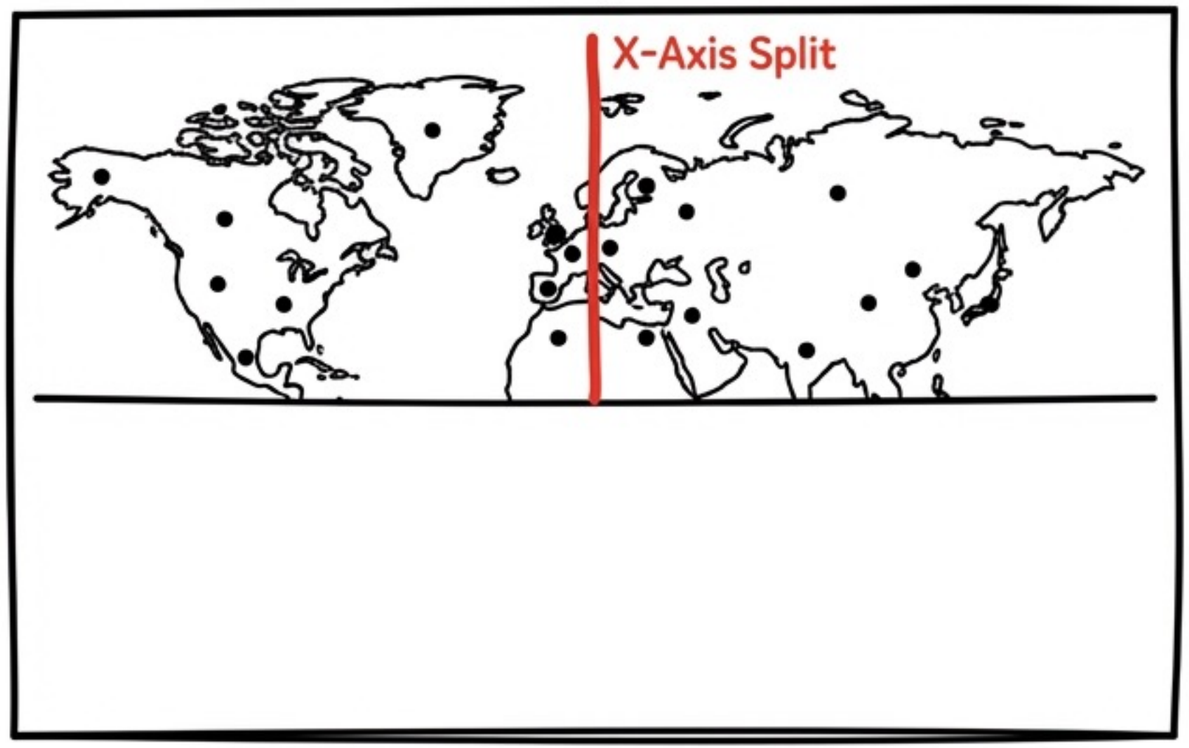
NotebookLM

k-D 트리 - 축을 교차하는 분할

Step 1



Step 2



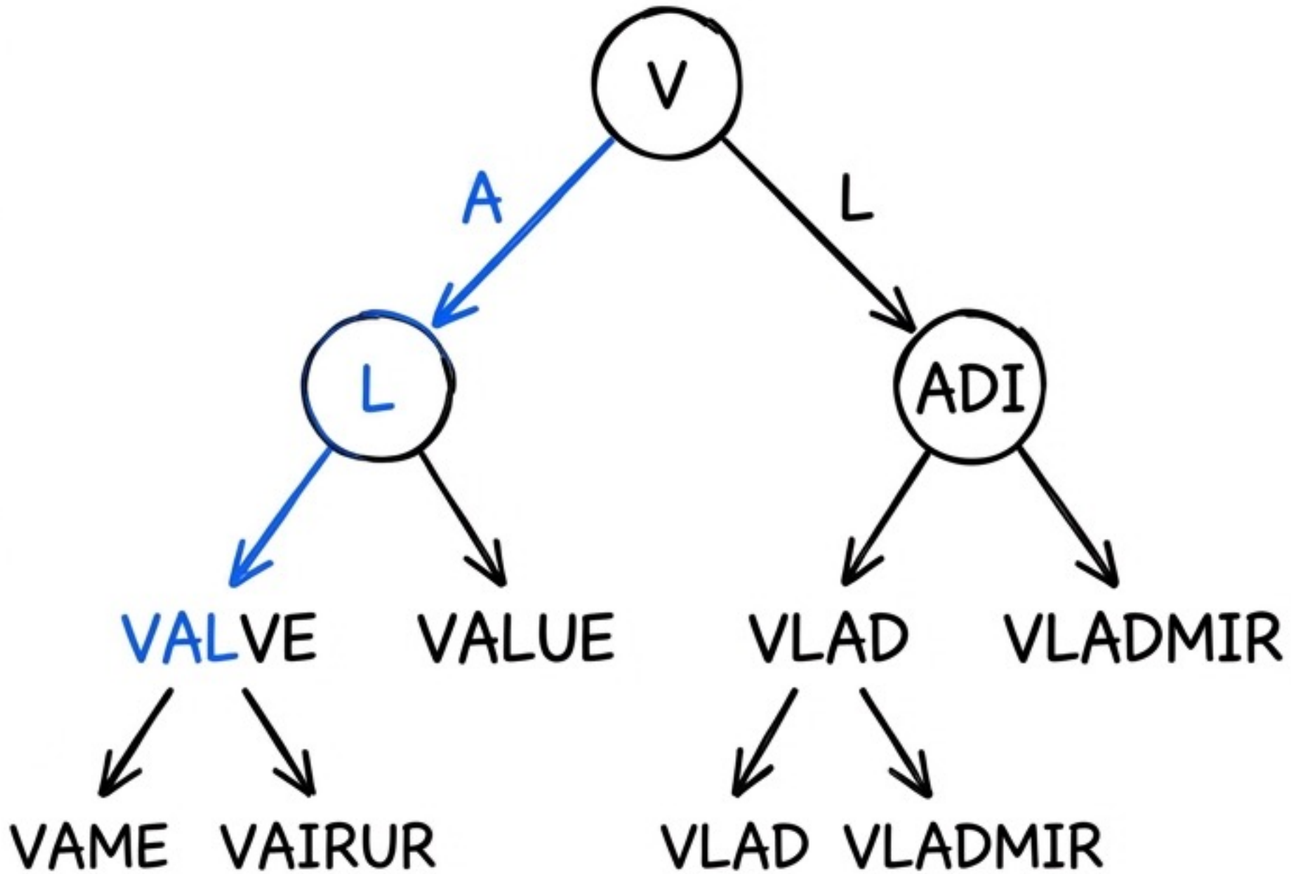
** Mechanism: 한 번에 하나의 차원(축)만 분할.

** Process: Y축 기준 분할 (위도) → X축 기준 분할 (경도) 반복.

** Advantage: 차원이 높아져도 유연하게 대응 가능.

NotebookLM

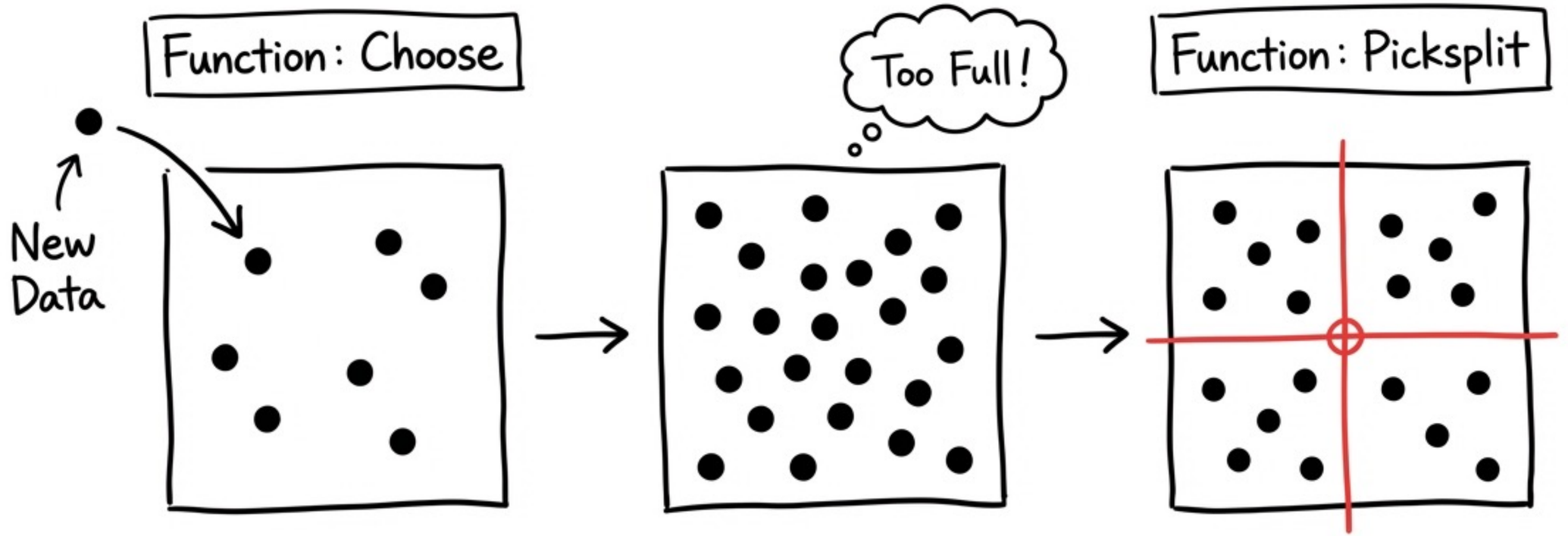
Radix 트리 - 텍스트도 공간이다



- **Concept:** '공간'은 알파벳의 나열. 공통 접두사(Prefix)를 내부 노드에 저장.
- **Efficiency:** 중복된 텍스트를 저장하지 않아 B-Tree보다 훨씬 컴팩트함.
- **Example:** 'V' → 'A' → 'L' 분기 구조.

NotebookLM

작동 원리 - 삽입과 분할 (Insertion & Split)



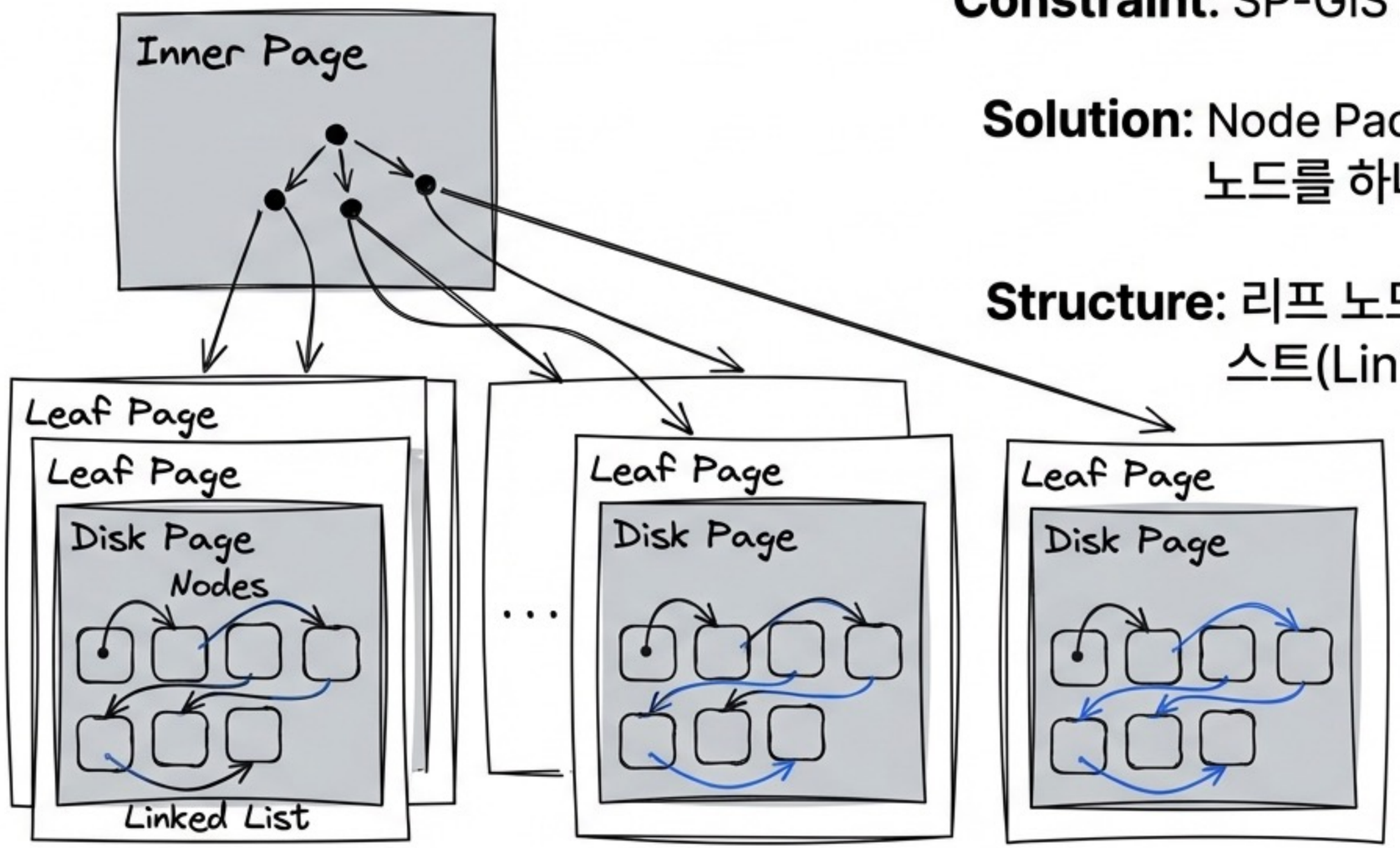
- **Function 1 (Choose):** 데이터가 속할 하위 노드를 선택하여 이동.

  Scenario: 노드에 공간이 부족하다면?

- **Function 2 (Picksplit):** 새로운 중심점(Centroid)을 계산하여 노드를 분할하고 데이터를 재분배.

NotebookLM

물리적 구조 – 페이지 패킹 (Physical Storage)



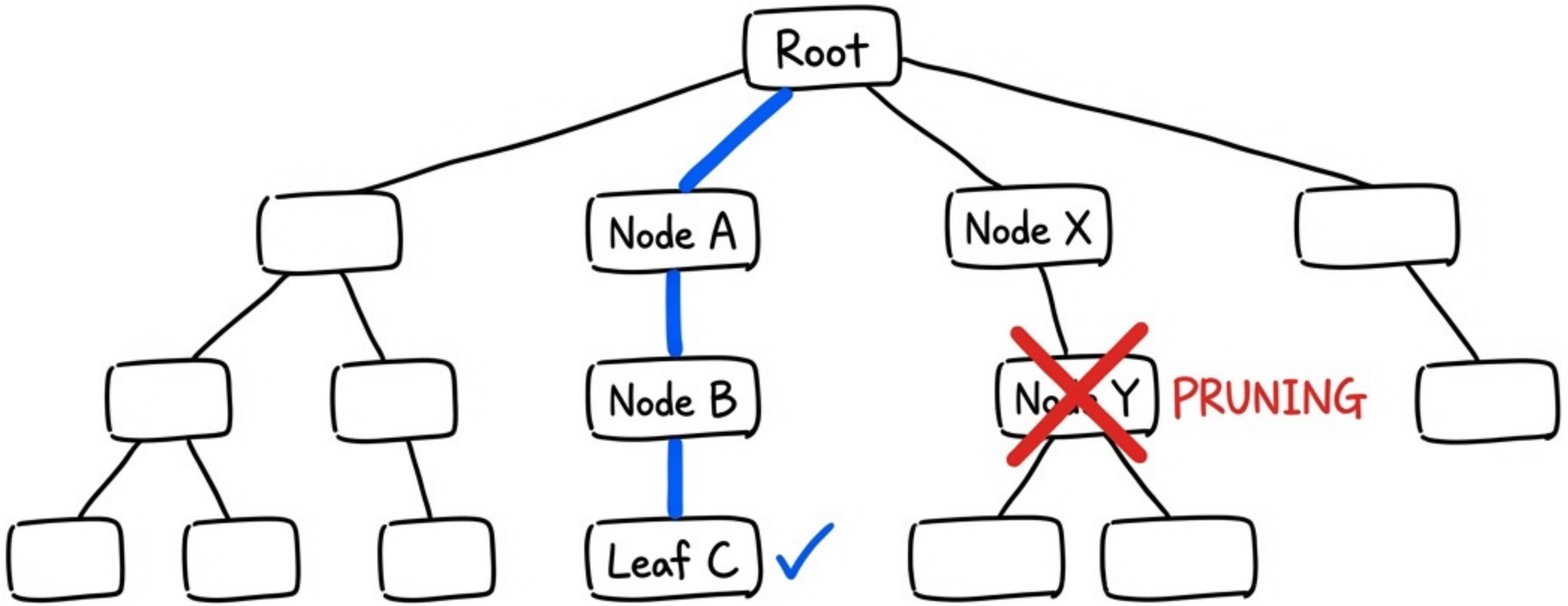
Constraint: SP-GiST 노드는 작고 트리는 불균형함.

Solution: Node Packing. I/O 최소화를 위해 여러 노드를 하나의 디스크 페이지에 밀집.

Structure: 리프 노드들은 페이지 내에서 연결 리스트(Linked Lists)로 관리.

NotebookLM

검색 알고리즘 - 가지치기의 기술



- **Method:** Depth-First Search (깊이 우선 탐색).
- **Inner Node Check:** '이 쪽으로 내려갈 가치가 있는가?' (일관성 함수).
- **Leaf Node Check:** '이 값이 조건과 일치하는가?'

NotebookLM

속성과 제약 (Properties & Constraints)



지원함 (YES)

- Index-Only Scan (일부 경우)
- k-NN (최근접 이웃)
- Exclude 제약 조건



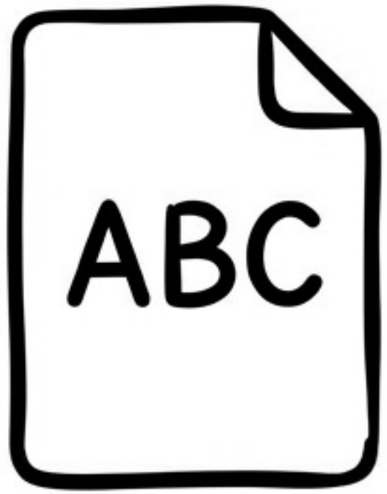
지원 안 함 (NO)

- ORDER BY (정렬 불가)
- Multi-column (다중 컬럼 불가)
- Uniqueness (유니크 제약 불가)

NotebookLM

요약 - 언제 SP-GiST를 사용하는가?







Condition: 데이터 분포가 불균형하거나(Skewed), 겹치지 않는 파티션이 명확할 때.

- **GIS (PostGIS):** 중첩이 많은 스파게티 데이터.
- **Text Search:** URL, 이름 등 접두사 검색.
- **Network:** IP 주소 대역 관리.

NotebookLM

•28. GIN

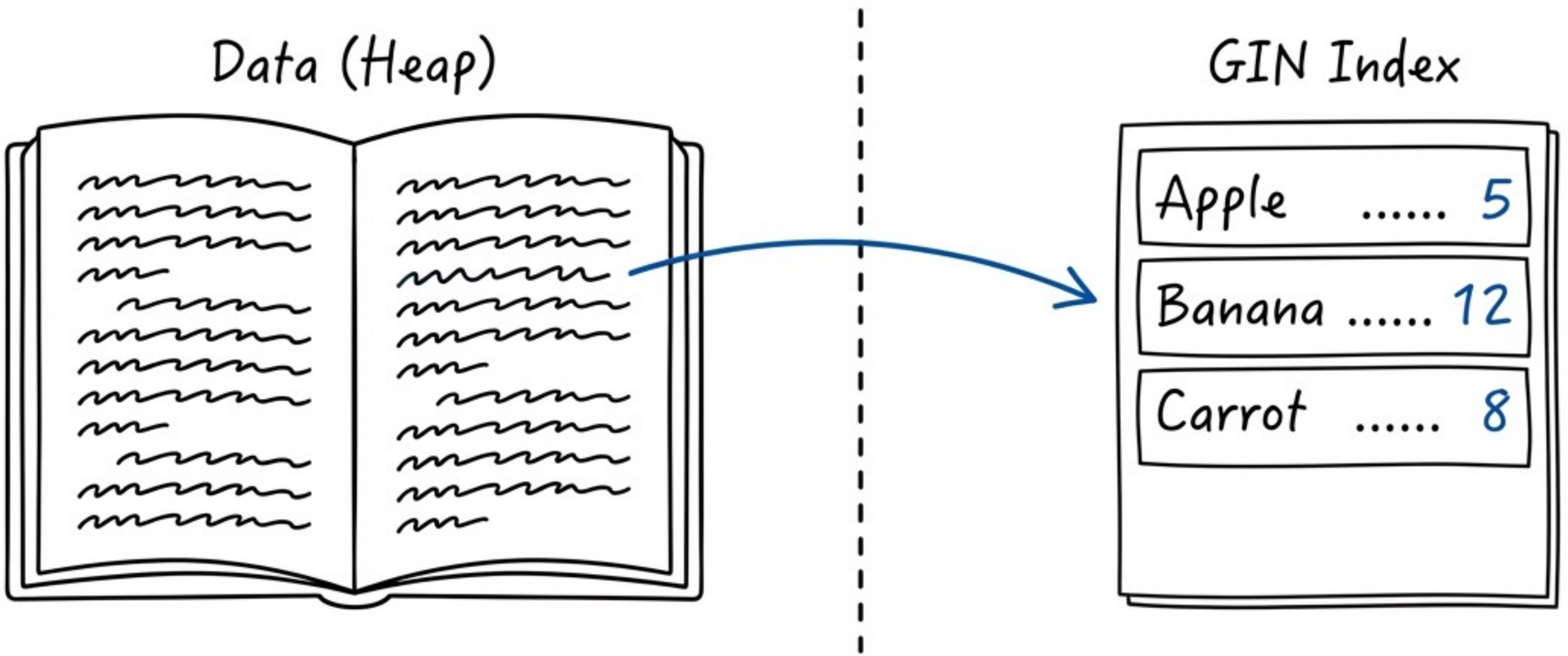


PostgreSQL GIN 인덱스: 복합 데이터 검색의 핵심

Searching the Unsearchable — 텍스트, 배열, JSON 고속 검색을 위한 아키텍처

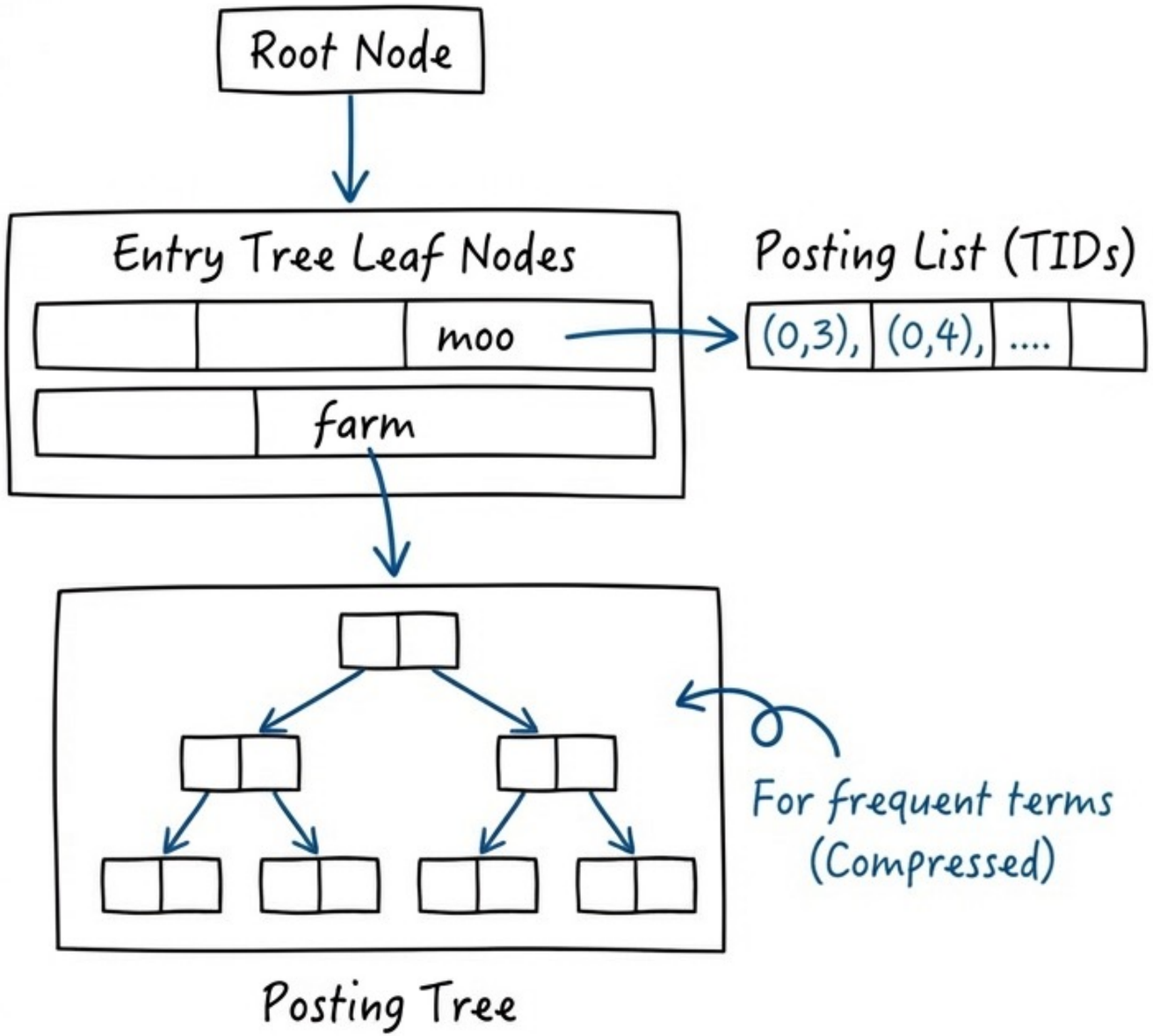
NotebookLM

역색인(Inverted Index)이란?



- 책 뒤의 '색인'과 같은 원리: 키워드(Key) → 페이지 번호(TID)
- 데이터의 요소(Element)를 추출하여 테이블 행(Row)에 매핑

NotebookLM

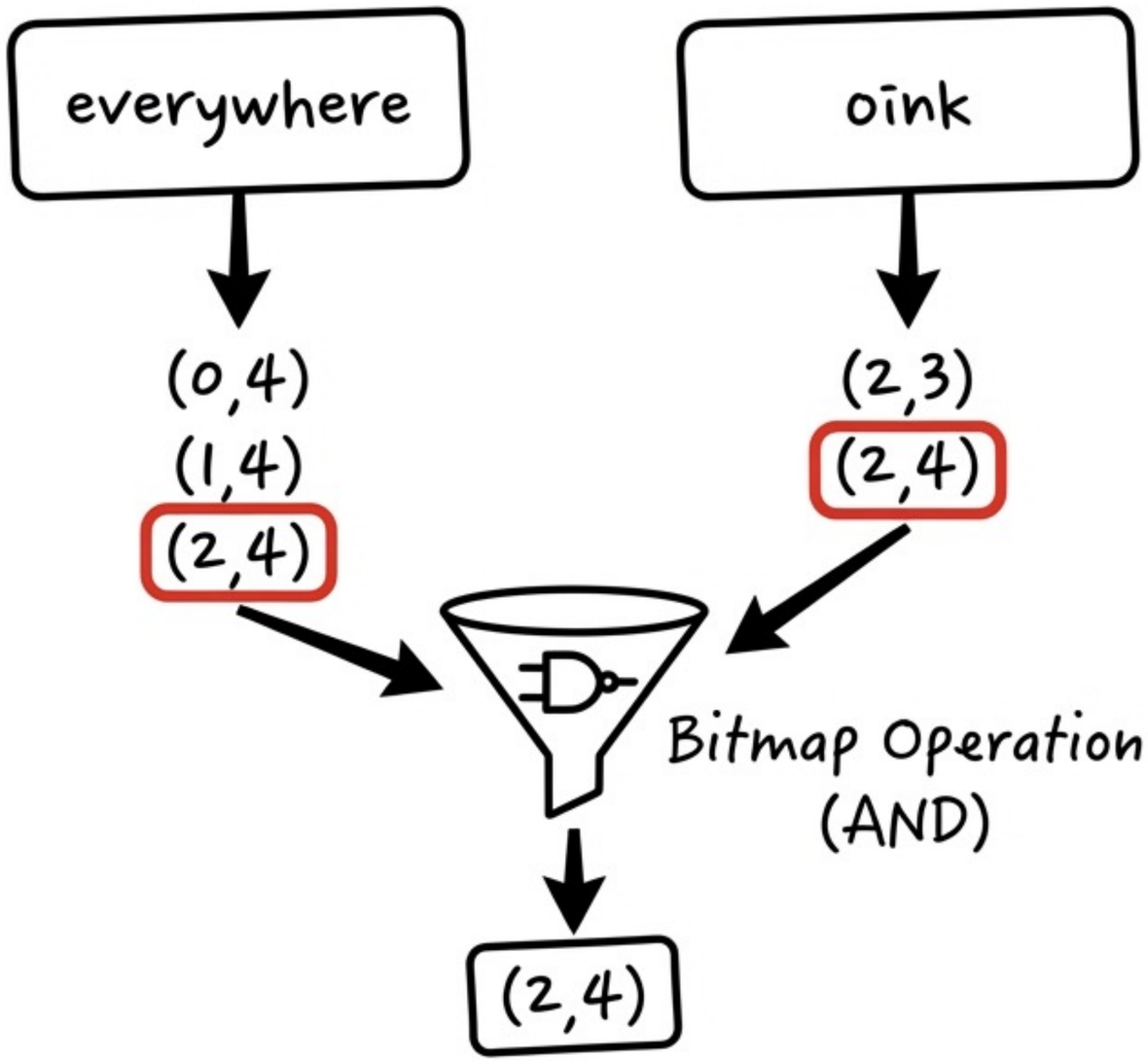


GIN의 내부 구조: Entry Tree와 Posting List

- Entry Tree: 모든 요소(Key)는 인덱스에 단 한 번만 저장됨
- Posting List: 해당 요소가 포함된 문서 ID(TID)들의 리스트
- Posting Tree: 리스트가 너무 길어지면 별도의 B-Tree로 변환 (압축 및 검색 효율)

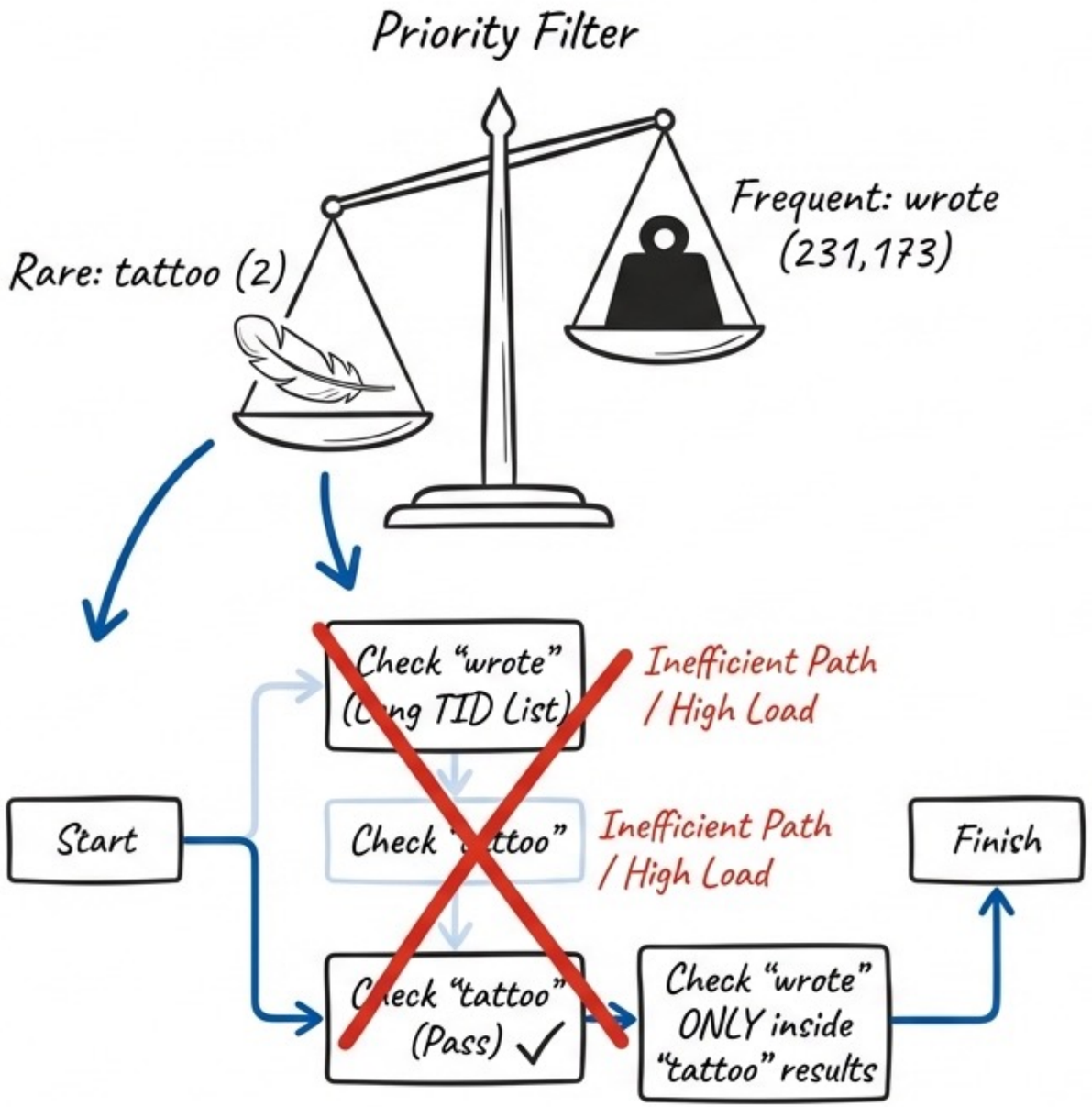
NotebookLM

검색 알고리즘: 리스트 병합과 비트맵(Bitmap) 연산



- 각 키(Key)를 개별적으로 스캔
- TID 리스트를 가져와 메모리 상에서 비트맵(Bitmap) 연산 수행 (AND, OR)
- 결과: 테이블을 직접 확인하지 않고도 고속으로 교집합/합집합 도출

NotebookLM

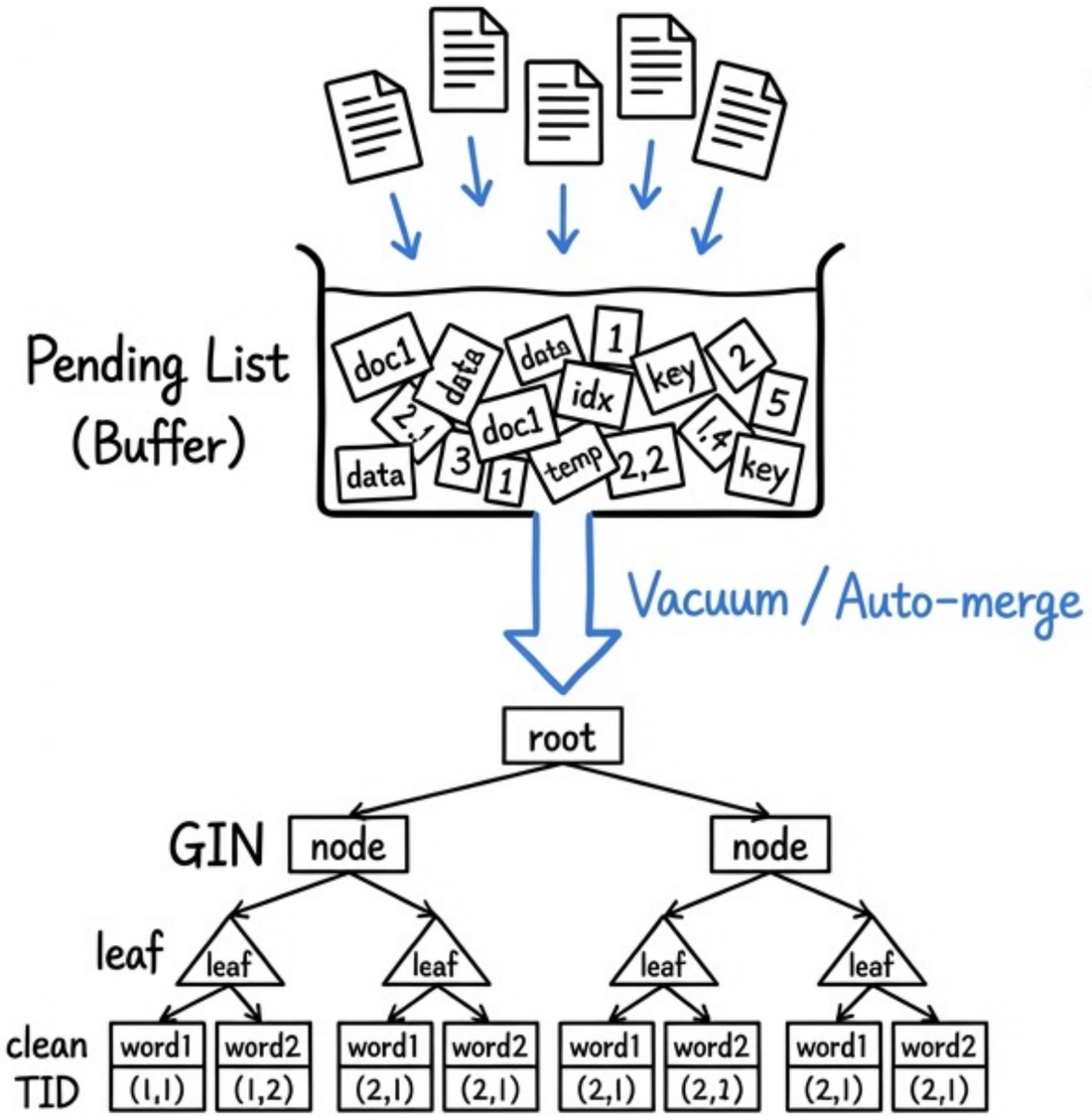


검색 최적화: 희귀한 단어(Rare) 우선 검색 전략

- 빈번한 단어(Frequent)는 TID 리스트가 매우 길어 비효율적
- 전략: 희귀한 단어(tattoo)의 결과를 먼저 확보한 후, 그 결과 내에서만 빈번한 단어(wrote)를 확인
- 불필요한 페이지 스캔을 건너뛰어 성능 극대화

NotebookLM

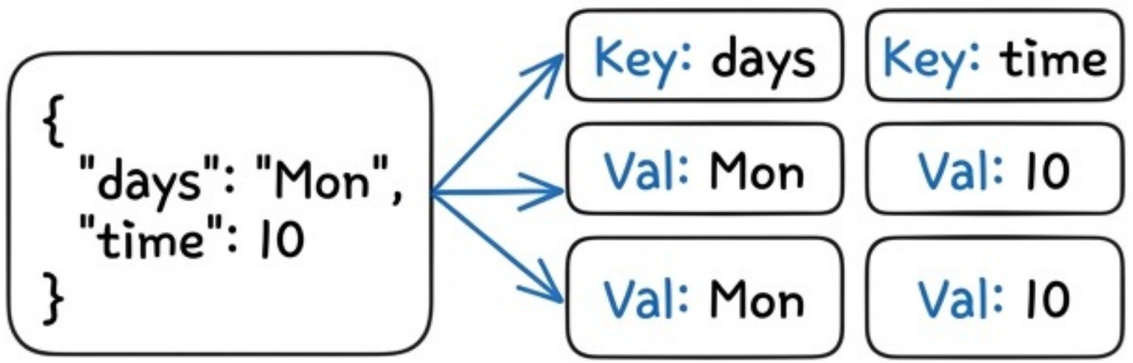
쓰기 성능과 Pending List: Fast Update 메커니즘



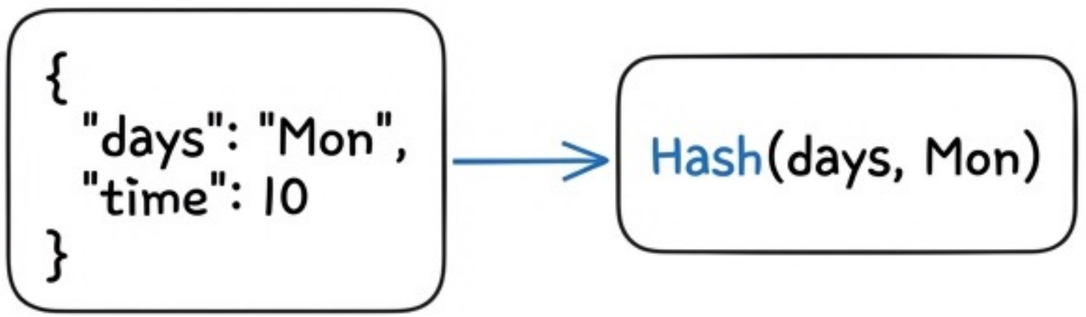
- 문제: 1개의 문서 저장 = 수십/수백 개의 인덱스 항목 업데이트 (쓰기 비용 높음)
- 해결: fastupdate (Pending List)
  - 변경 사항을 정렬되지 않은 리스트에 임시 저장 후, 나중에 일괄 병합(Bulk Merge)

NotebookLM

jsonb_ops



jsonb_path_ops



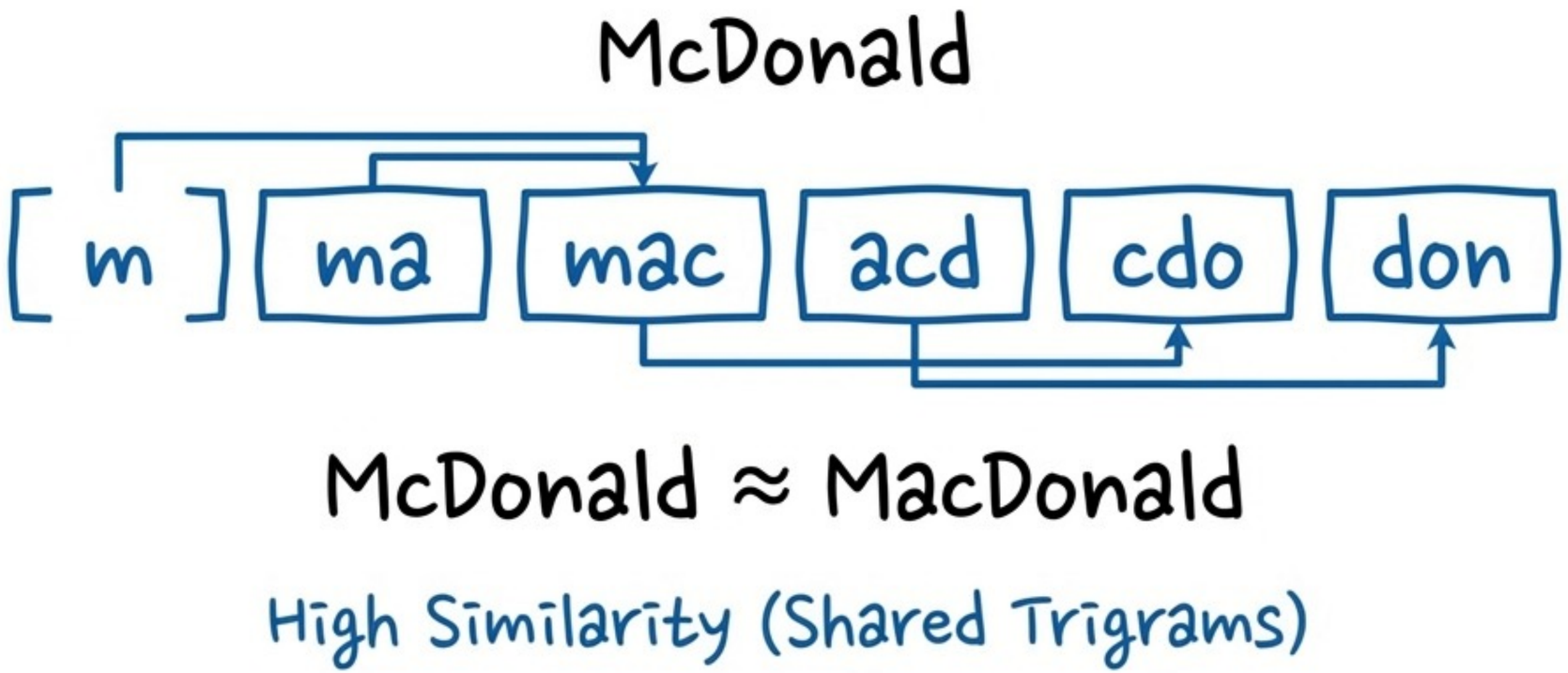
JSONB 인덱싱: jsonb_ops vs jsonb_path_ops

- **jsonb_ops** (Default): 모든 키와 값을 개별 인덱싱. 유연함 **(?, ?|** 연산자 지원)
- **jsonb_path_ops**: 경로(Path)의 해시값만 인덱싱. 인덱스 크기가 작고 검색이 빠름 **(@>** 연산자 최적화)

NotebookLM

pg_trgm과 유사도 검색: 오타가 있어도 찾는다

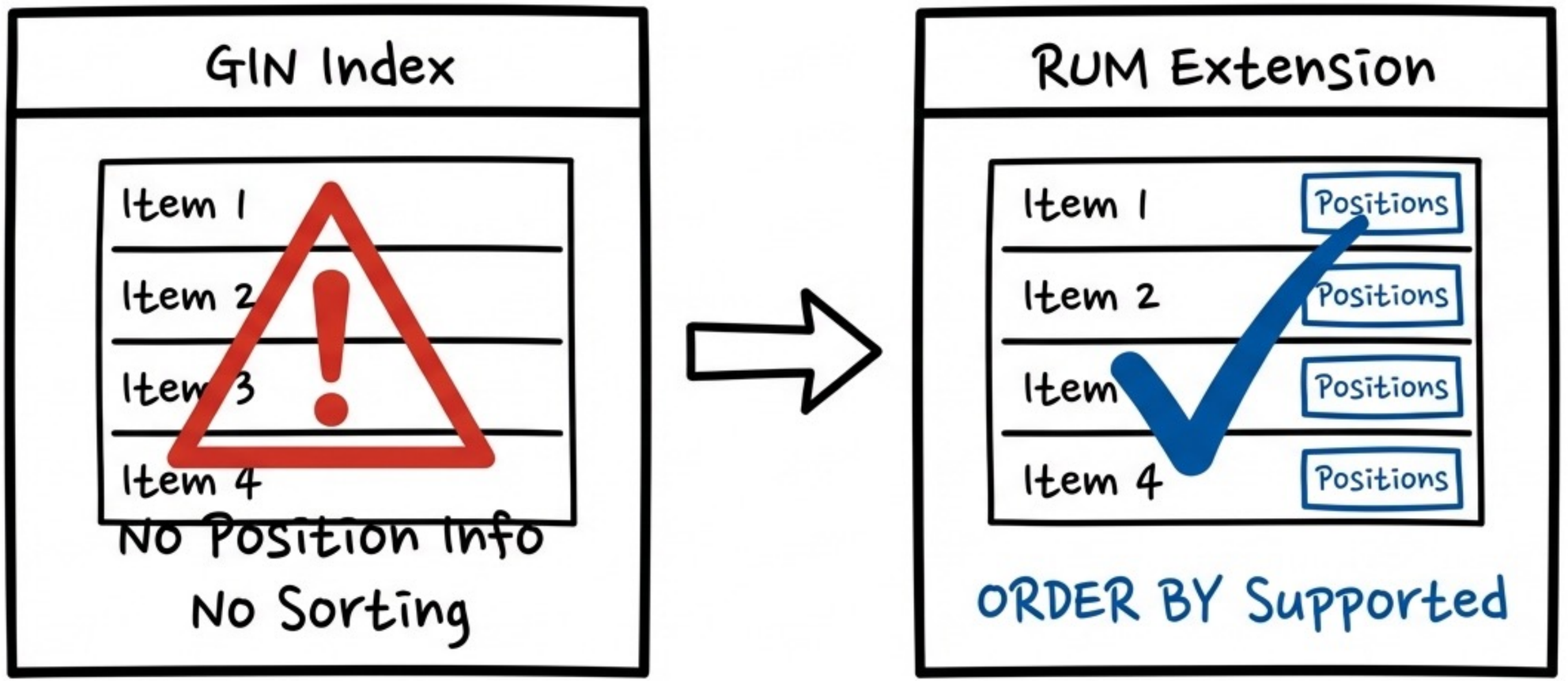
- Trigram: 문자열을 3글자 단위로 분해하여 인덱싱
- LIKE, ILIKE, 정규표현식 검색 가속화
- 유사도(Similarity) 측정을 통해 오타 교정 및 퍼지(Fuzzy) 검색 지원



NotebookLM

한계점과 대안: 정렬 불가, 그리고 RUM 인덱스

- GIN의 한계: 위치 정보 부재로 구문 검색(Phrase Search) 및 랭킹(Ranking) 정렬 시 비용 발생
- RUM 확장: GIN + 위치 정보/추가 필드
- 검색 결과 정렬(ORDER BY)이 필수적인 경우 RUM 사용 고려



NotebookLM

요약: GIN 인덱스, 언제 써야 할까?

결론: '요소(Element)' 기반의 복합 데이터 검색에는 GIN이 정답.

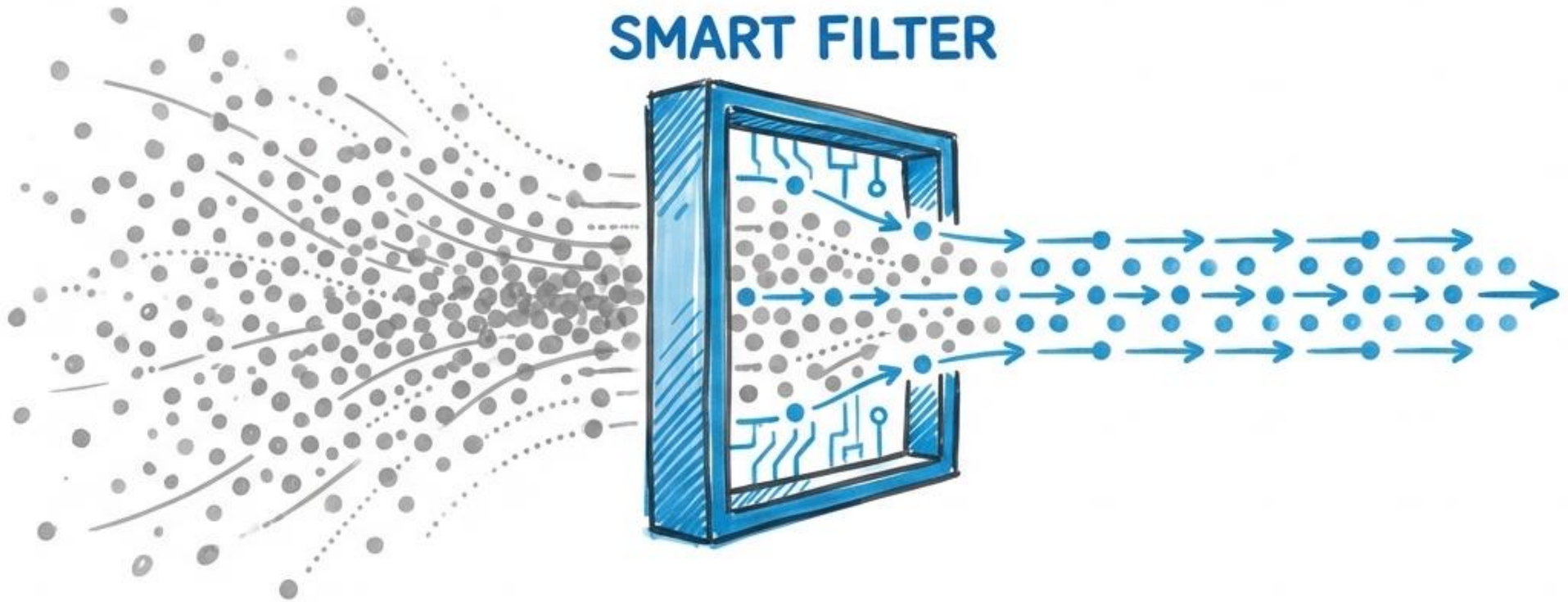


NotebookLM

•29. BRIN

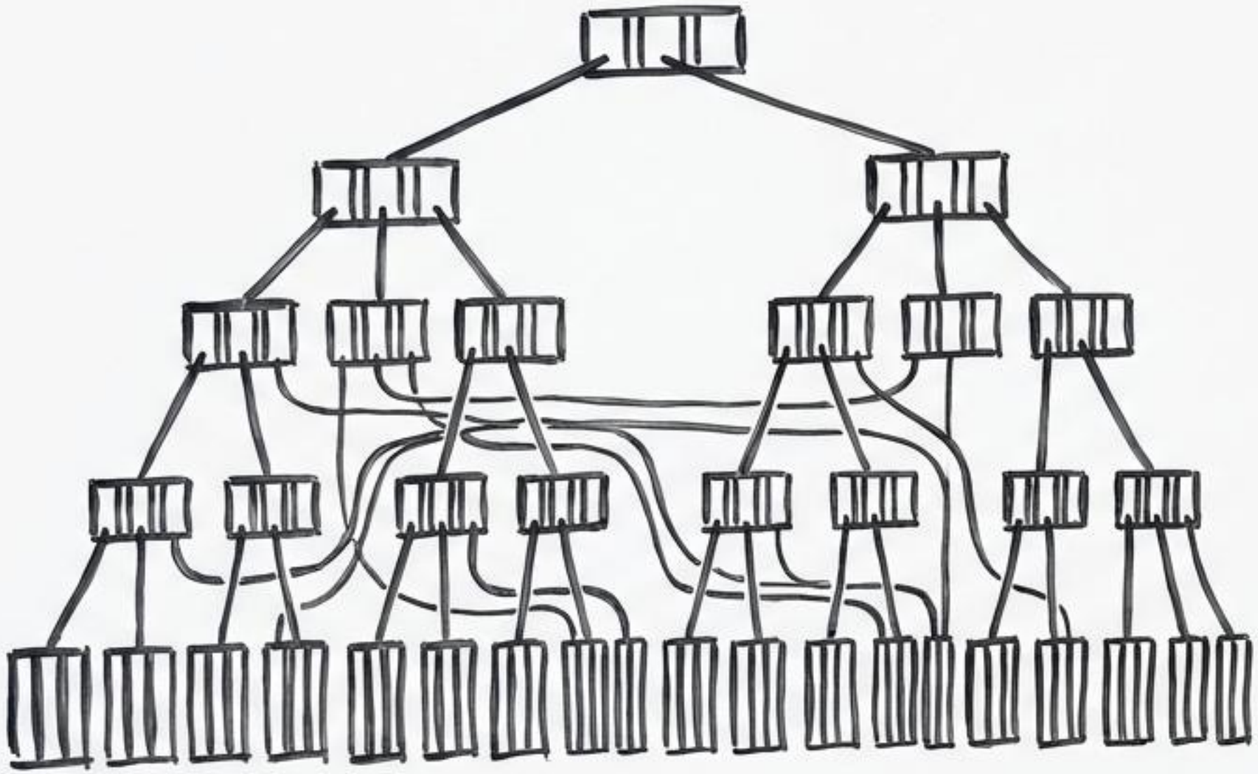
테라바이트를 정복하는 기술: PostgreSQL BRIN 인덱스

B-Tree의 한계를 넘어서는 '스마트 필터' 전략



NotebookLM

B-Tree의 역설: 데이터가 커지면 인덱스도 짐이 된다



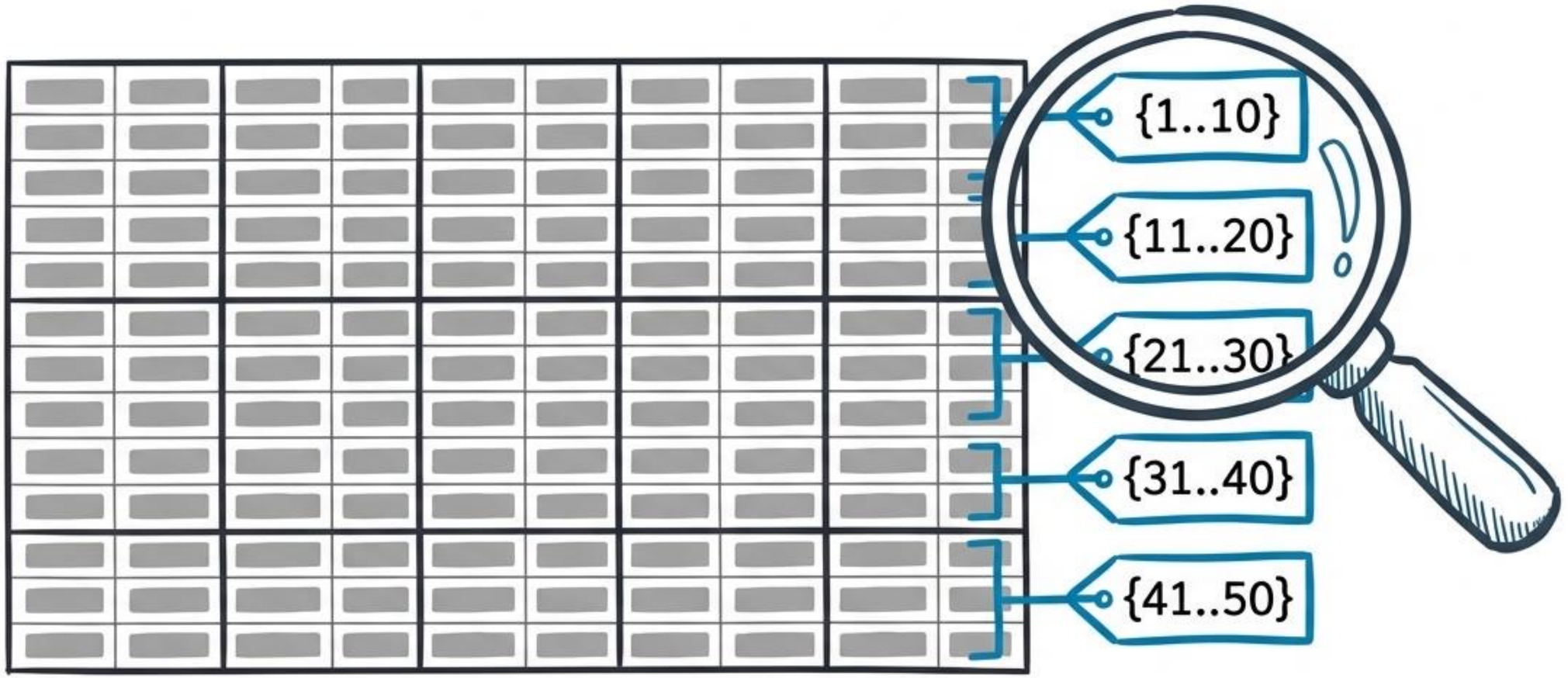
B-Tree Index
210 MB



BRIN Index
184 kB

NotebookLM

새로운 철학: 위치를 찍지 말고, 범위를 요약하라

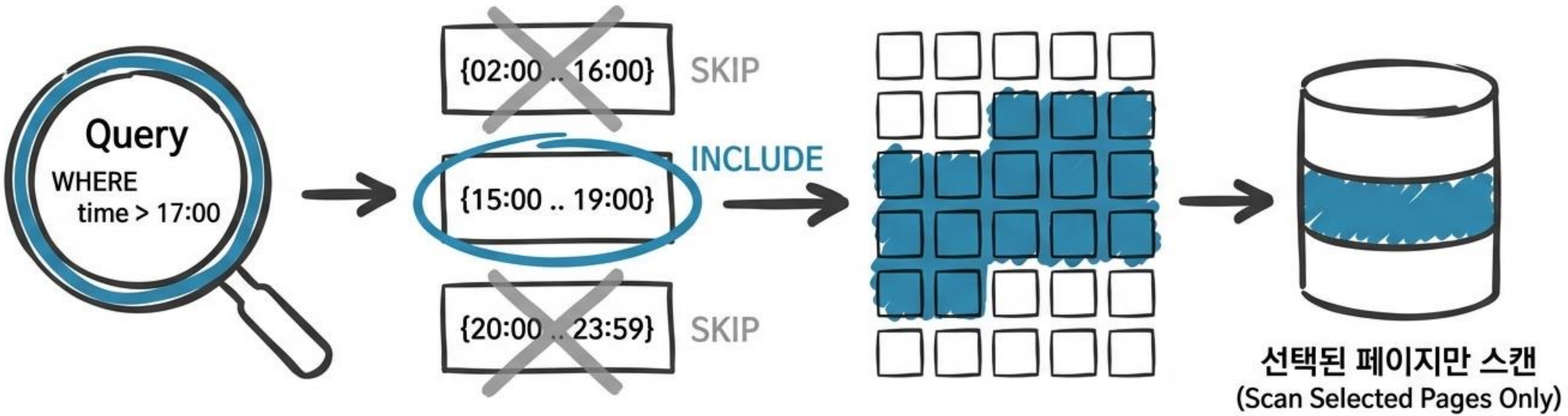


1. Divide (나누기): 테이블을 논리적인 '블록 범위'로 나눕니다.
2. Summarize (요약하기): 각 범위의 최소값(Min)과 최대값(Max)만 저장합니다.
3. Filter (거르기): 정확한 위치 대신 불필요한 데이터를 건너뛰는(Skip) 것에 집중합니다.

NotebookLM

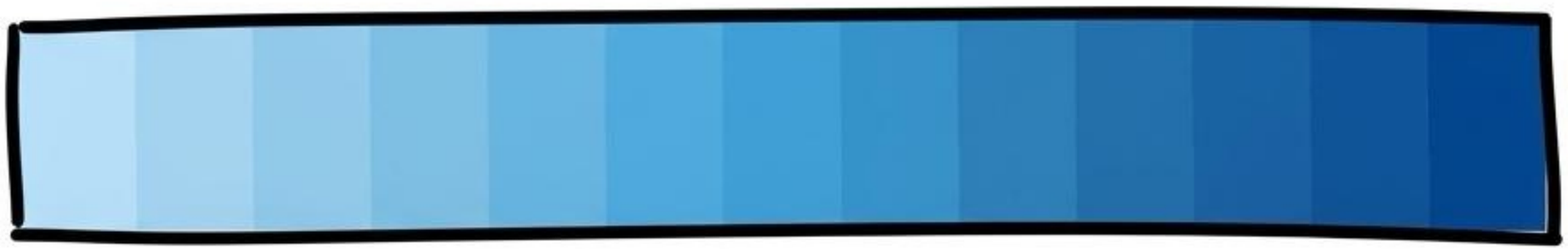
작동 원리: 손실 비트맵 스캔 (Lossy Bitmap Scan)

인덱스가 페이지 전체를 선택하면, 이후 재확인(Re-check) 과정을 거칩니다.

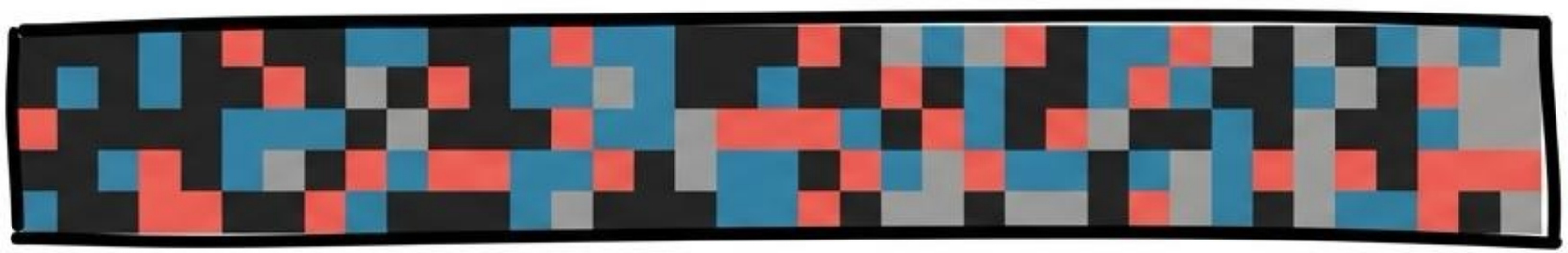


NotebookLM

성공의 열쇠: 데이터 정렬 (Correlation)



High Correlation



Low Correlation

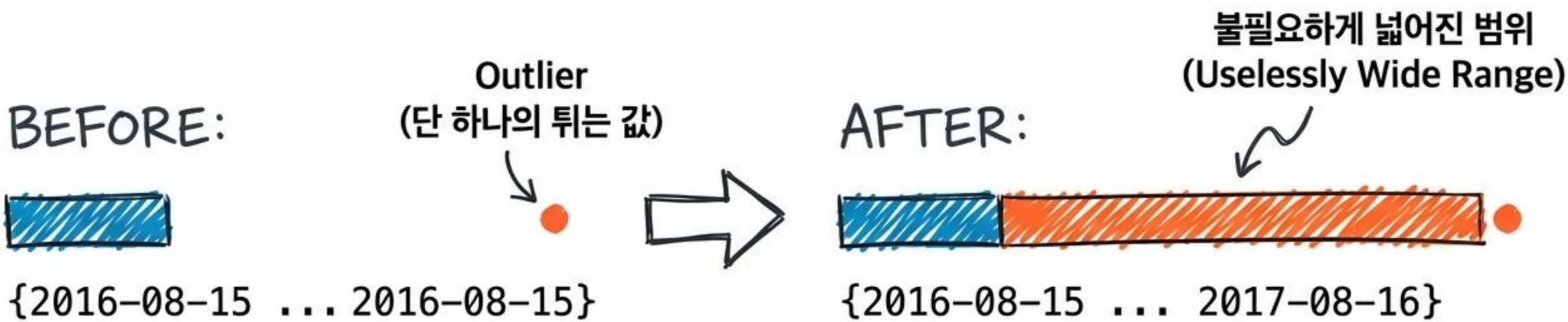


물리적 저장 순서와 논리적 값의 순서가 일치할 때(예: 시계열),
단 0.3%의 페이지만 읽고 검색을 완료합니다.

NotebookLM

위기: 아웃라이어(Outlier)가 효율을 망칠 때

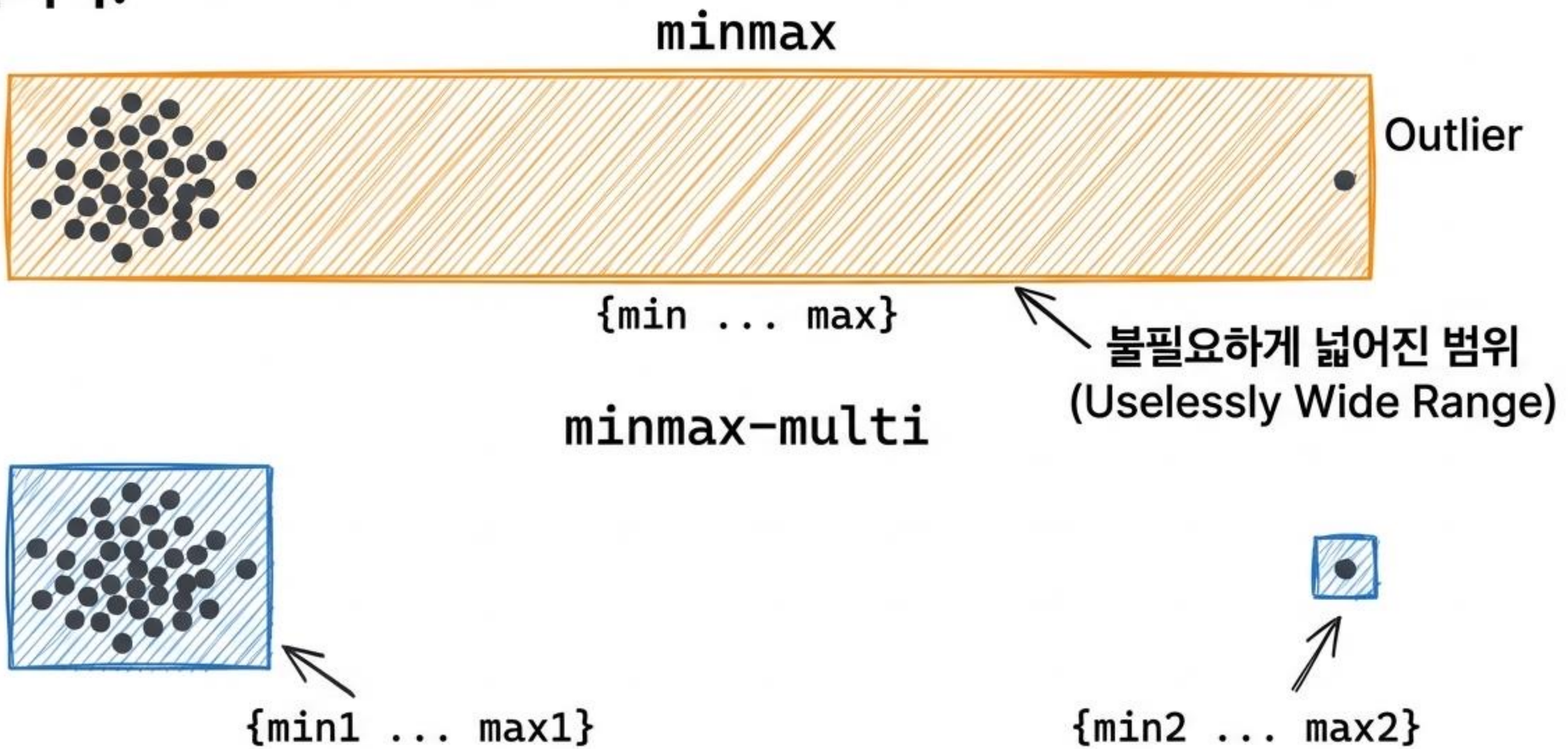
단 하나의 튀는 값이 범위 전체를 불필요하게 넓혀, 인덱스의 변별력을 0으로 만듭니다.



NotebookLM

해결책: `minmax-multi`로 정밀도 회복하기

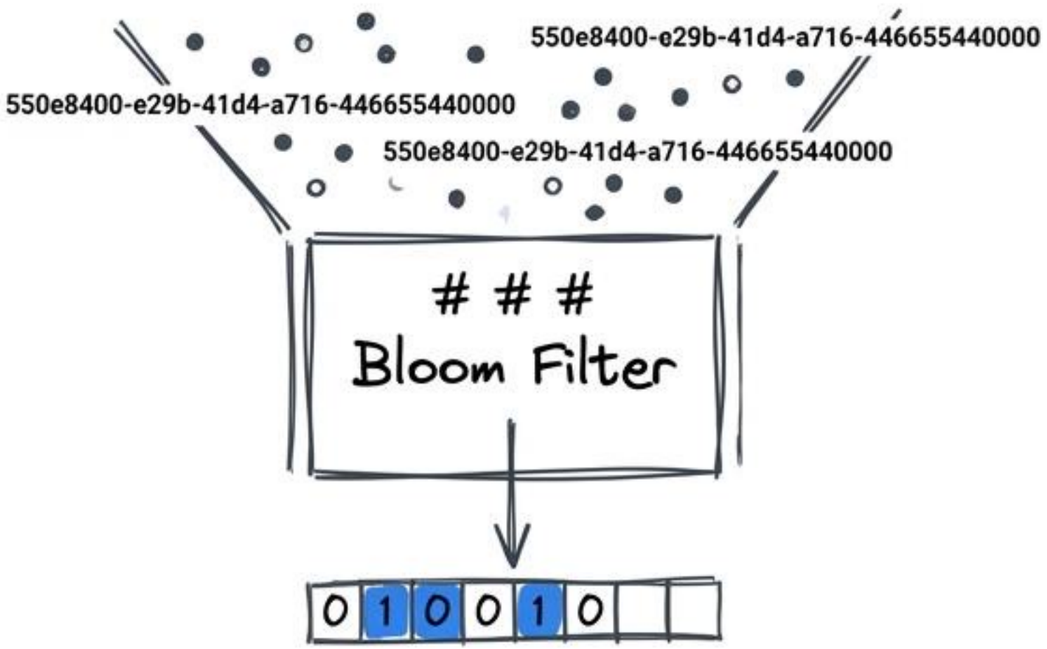
단일 범위 대신, 블록당 여러 개의 작은 범위를 저장하여 아웃라이어 문제를 해결합니다.



NotebookLM

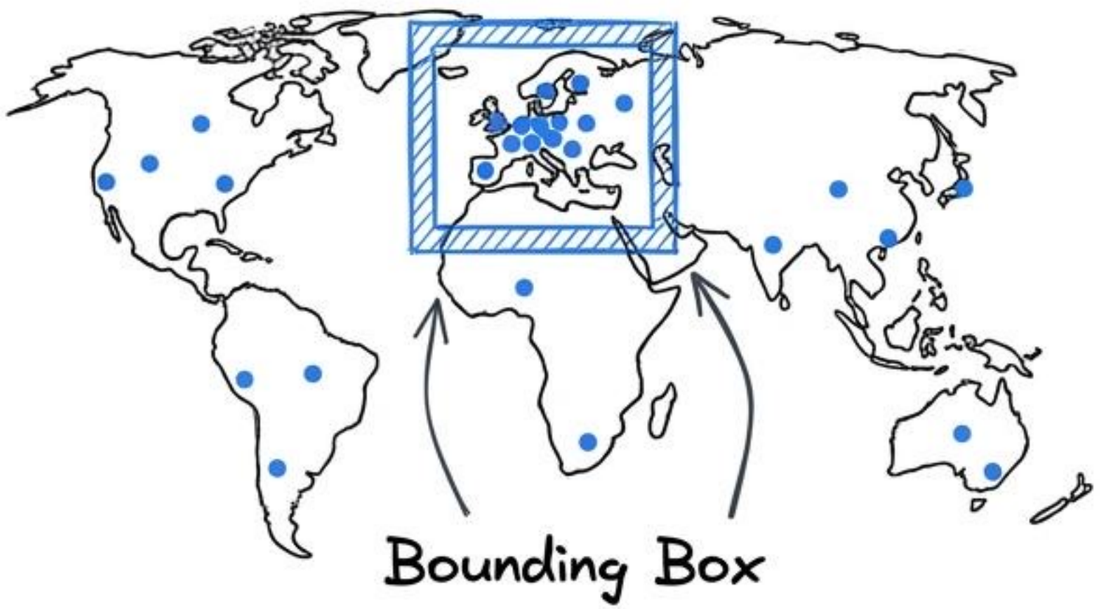
특수 무기: 정렬되지 않은 데이터와 공간 데이터

bloom 인덱스



낮은 상관관계 데이터용
(For Low-Correlation Data)

inclusion 인덱스



공간 및 범위 데이터용
(For Spatial and Range Data)

NotebookLM

BRIN 사용 체크리스트 (The Playbook)



☑ 1. 초대용량 (Massive)



☑ 2. Append-mostly



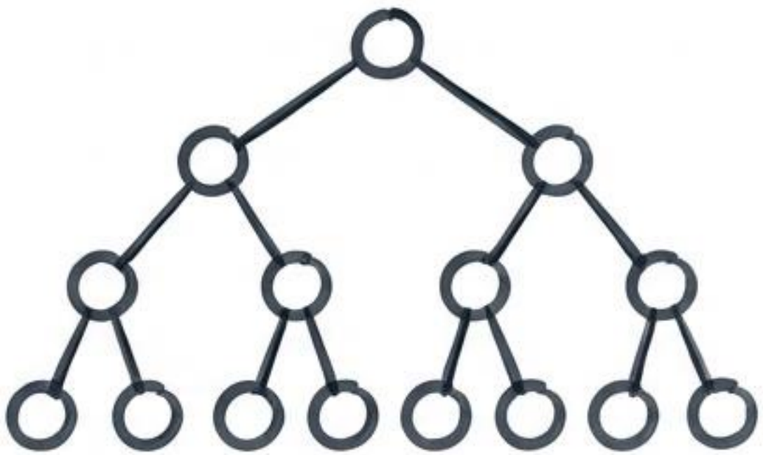
☑ 3. 높은 정렬도



☑ 4. 범위 필터링

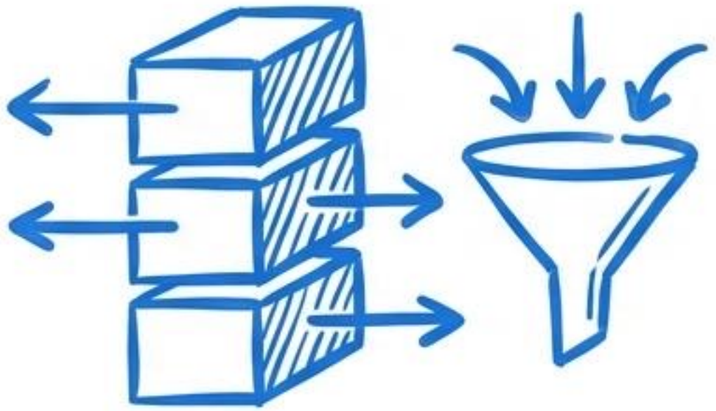
NotebookLM

결론: 대체제가 아닌, 강력한 특수 목적 도구



B-Tree
(General Purpose)





BRIN
(Specialized Tool)

Think Ranges. Think BRIN.

NotebookLM

Thank You

Questions?

NotebookLM