aws FOR GAMES

GAMES ON AWS 2026

GS Neotek | MEGAZONE CLOUD

GAMES ON AWS 2026

aws FOR GAMES

찰나의 행동마저 기억하는 AI NPC 만들기:

플레이 맥락을 실시간으로 기억하는 엔진, Live Memory 개발 여정

정윤호
엔지니어, 넥슨

배승종
엔지니어, 넥슨

강정희
솔루션스 아키텍트, AWS

GS Neotek | MEGAZONE CLOUD

AGENDA

목차

01	컨텍스트의 중요성	컨텍스트 최적화의 중요성과 AWS AgentCore Memory
02	배경	게임 도메인의 AI 에이전트, 그리고 행동 맥락이라는 자산
03	기억 모델링	구조화된 로그를 그래프 기억으로 바꾸는 방법
04	기억 저장소	FastGraph — 수천만 개의 기억 그래프를 담는 DB를 직접 만들다
05	마무리	만들며 배운 것과 앞으로의 과제

GAMES ON AWS 2026

aws FOR GAMES

01

컨텍스트의 중요성과 AWS AgentCore Memory

컨텍스트를 인식하는 AI 에이전트를 만들기 위해서는 무엇이 필요할까요?

강정희

솔루션스 아키텍트, AWS

GS Neotek | MEGAZONE CLOUD

컨텍스트 윈도우의 구성요소

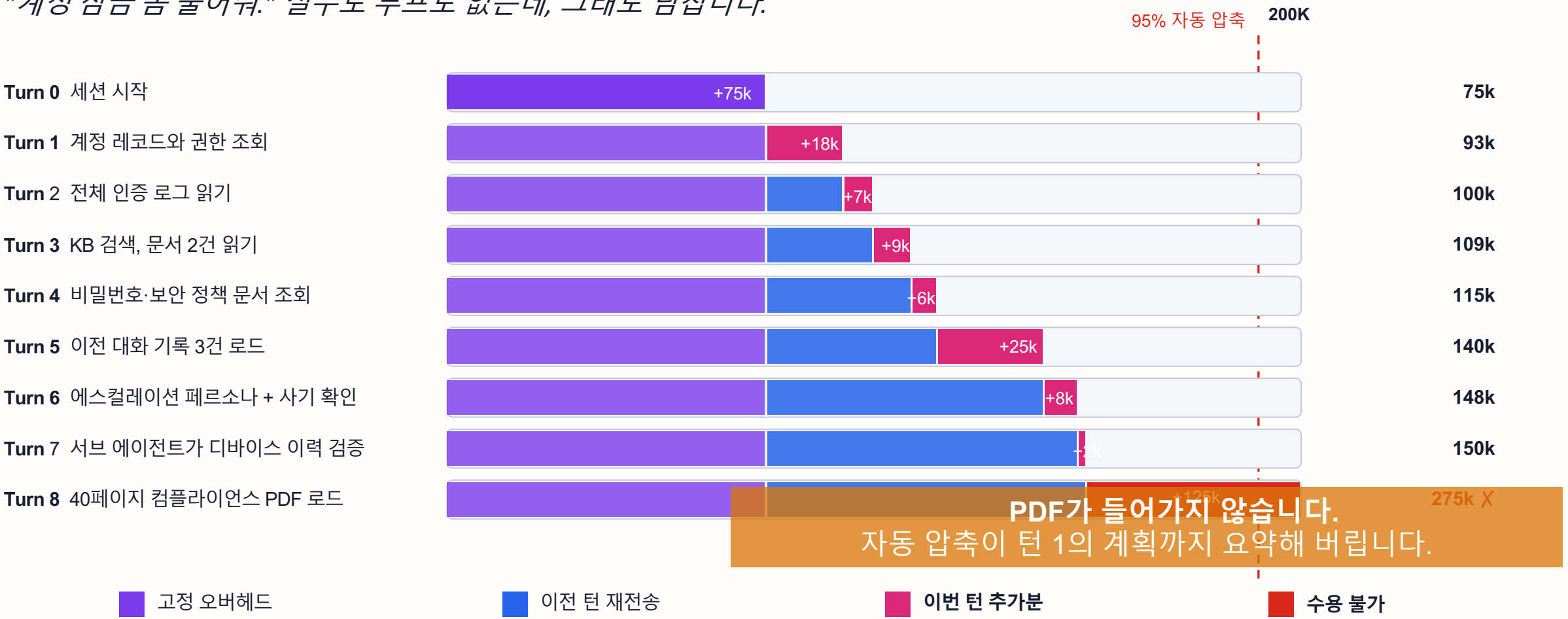
컨텍스트 윈도우는 시스템 프롬프트, 대화 히스토리, 검색된 지식, 작업 상태를 포함한 텍스트 전체입니다. 모델이 한 번의 호출에서 해석하고 추론하는 데 쓰는 모든 것입니다.

시스템 프롬프트	히스토리	작업 상태	검색된 컨텍스트
역할, 규칙, 도구 정의.	턴과 도구 호출의 누적 로그.	스크래치패드, 현재 계획, 활성 도구 출력.	온디맨드로 가져오는 RAG 청크와 메모리 레코드.

평범한 아홉 턴이면 윈도우가 바닥납니다

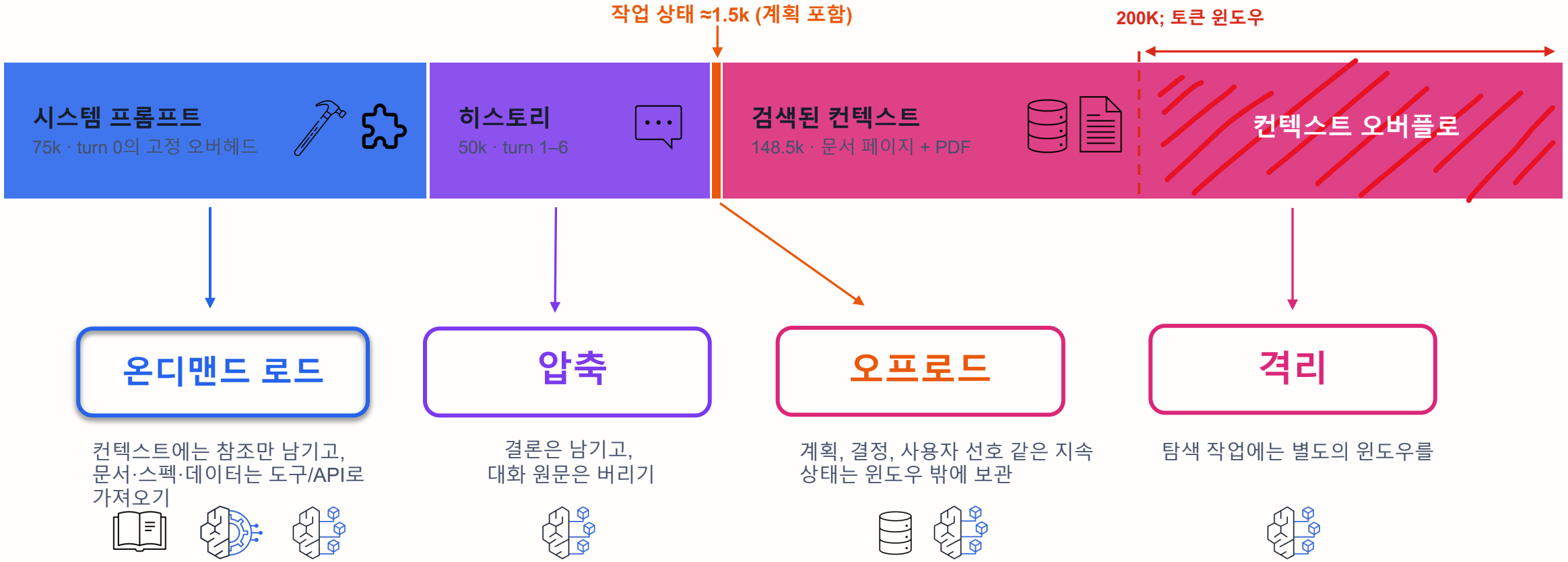
비대해지는 과정

"계정 잠금 좀 풀어줘." 실수도 루프도 없는데, 그래도 넘칩니다.



플레이북: 네 가지 기법

최적화 방법



네 가지 기법 중 세 가지는 하나의 서비스로

온디맨드 로드

Semantic + Episodic 메모리

질의에 따라 관련 사실과 과거 경험을 벡터 스토어에서 검색합니다.

압축

Summary 전략 + STM

압축된 세션 요약이 최근 메시지(마지막 k개)와 함께 프롬프트에 실립니다.

오프로드

STM + 로컬 파일 시스템

계획, 상태, 아티팩트는 윈도우 밖에 보관되어 재시작과 새 세션에도 유지됩니다.

격리

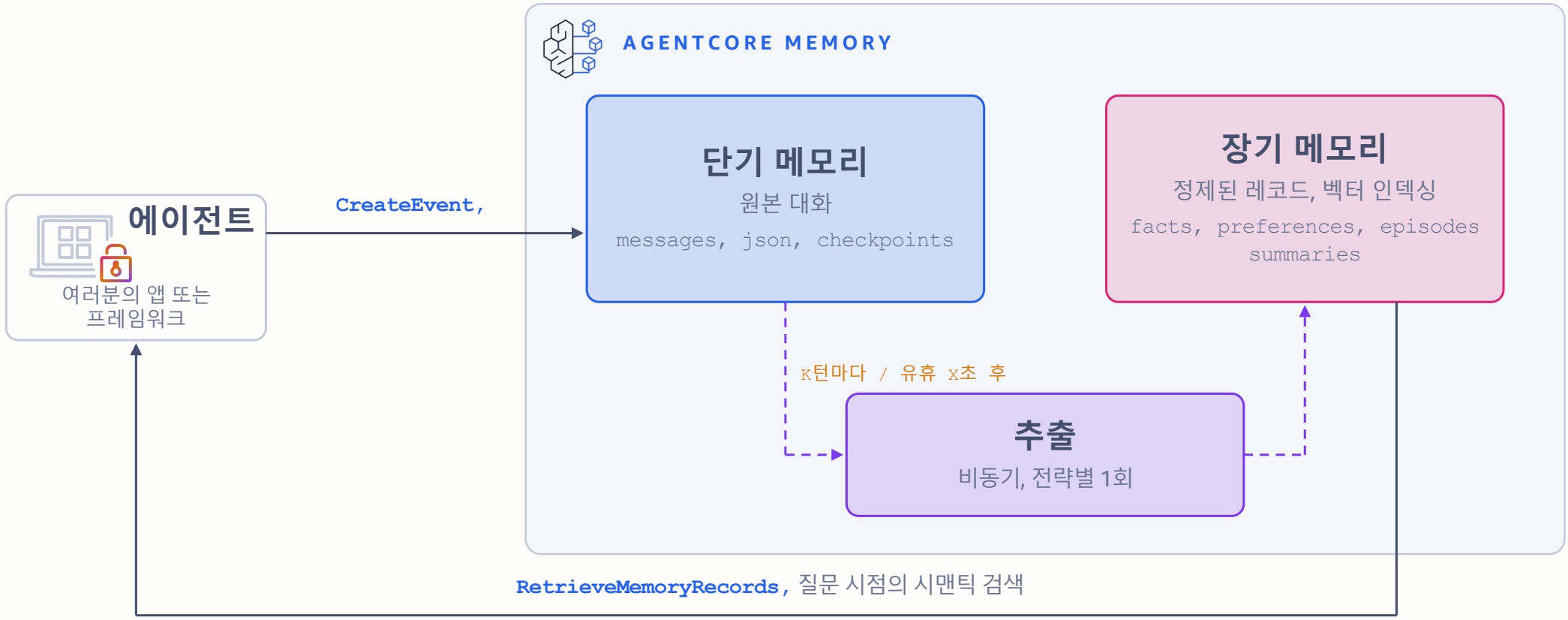
컨텍스트 격리

서브 에이전트는 자신만의 윈도우를 유지하고, 필요한 상태만 의도적으로 공유해야 합니다.

AgentCore Memory : CreateEvent로 턴을 기록하고, 추출이 레코드를 만들고, 검색은 관련 있는 것만 주입합니다.

두 개의 저장소, 하나의 서비스

아키텍처



서버리스, 저장 시 암호화

장기 메모리 (LTM)

핵심 개념: 기본 제공 전략



Session Summaries: 세션의 압축된 표현.

"반품 접수, 라벨 이메일 발송" "어젯밤 레이드: 2회 전멸 후 클리어, 전리품 분배 완료"

User Preferences: 사용자의 호불호, 상호작용 스타일, 선택.

"채팅보다 이메일 후속 연락 선호" "근접 무기 선호 · 파밍보다 보스전 · 음성 안내 끔"

Semantic Facts: 사실과 도메인 특화 정보.

"5월 3일 QC-45 헤드폰 구매" "길드 '새벽서리' 소속 · 주 캐릭터 Lv.72 버서커"

Episodic Memory: 지능적인 추출과 리플렉션 과정을 통해 에이전트가 자신의 경험으로부터 스스로 학습하는 능력.

상황, 턴, 평가 "화염 저항 없이 2페이즈 진입 → 전멸 · 원인: 저항 부족"

어제 함께 잡은 보스와, 그때 실패한 이유까지

기억하는 NPC를 만들 수 있을까?

Episodic Memory (일화 기억)



에피소드

상황(Situation): 파티가 용암 거인 2페이즈에서 반복 전멸

의도(Intent): 공략 실패의 근본 원인 파악

턴(Turns):

Step 1: 사고: "진형이 문제인가" -> 행동: 힐러 위치 변경
-> 관찰: "생존 12초 증가"

Step 2: 사고: "피해 원인 분석" -> 행동: 전투 로그 리뷰
-> 관찰: "원인은 화염 피해"

평가(Assessment): 화염 저항 부족이 근본 원인 — 물약 선제 사용 필요

리플렉션

제목: 용암 거인 2페이즈 공략 패턴

사용 사례: 동일 보스 재도전 · 유사 화염 몬스터 전투

힌트: 저항 물약을 선제 사용할 것. 1페이즈에서 쿨다운을 아낄 것. 흔한 함정: 딜 부족으로 오판하는 것.

에이전트가 자신의 경험을 보존하고 그 위에서 추론합니다. 인과적 일관성 유지와 목표 분할로 가능해집니다.

에이전트는 자신의 경험에서 교훈을 뽑아냅니다. 손으로 작성한 규칙이 아니라 학습된 패턴입니다.

게임의 기억 원천은 채팅이 아니라 행동 로그

서사는 대화가 아니라 전투·파밍·거래 이벤트에 쌓입니다

대화 메시지 기본 추출 대상

USER “어제 그 보스 다시 가자”

NPC “지난번엔 화염 저항이 발목을 잡았죠”

USER “이번엔 물약 챙겼어”

구조화된 행동 이벤트 (JSON) 진짜 서사의 원천

```
{ "event": "boss_raid_result",
  "boss": "lava_golem", "phase": 2,
  "result": "fail", "cause": "fire_resist_low",
  "party": ["u-17", "u-42", "u-77"] }
```

지금까지의 제약 추출을 거치려면 모든 콘텐츠를 먼저 단기 메모리 이벤트로 저장해야 했습니다 (CreateEvent 경유).

이벤트에는 대화 이상이 담깁니다

단기 메모리 : 메타데이터와 페이로드

```
# 정확 일치 검색을 위한 이벤트 태깅
client.create_event(
  ...,
  metadata = {
    "priority": "high",
              "channel": "chat",
  },
)

# 또는 임의의 JSON 상태를 체크포인트
client.create_event(
  ...,
  payload = [{"blob": {
    "plan": ["diagnose", "refund"],
    "step": 2,
  }}],
)
```

대화형

역할 태그가 붙은 턴. 장기 메모리 추출이 읽는 대상입니다.

blob / JSON

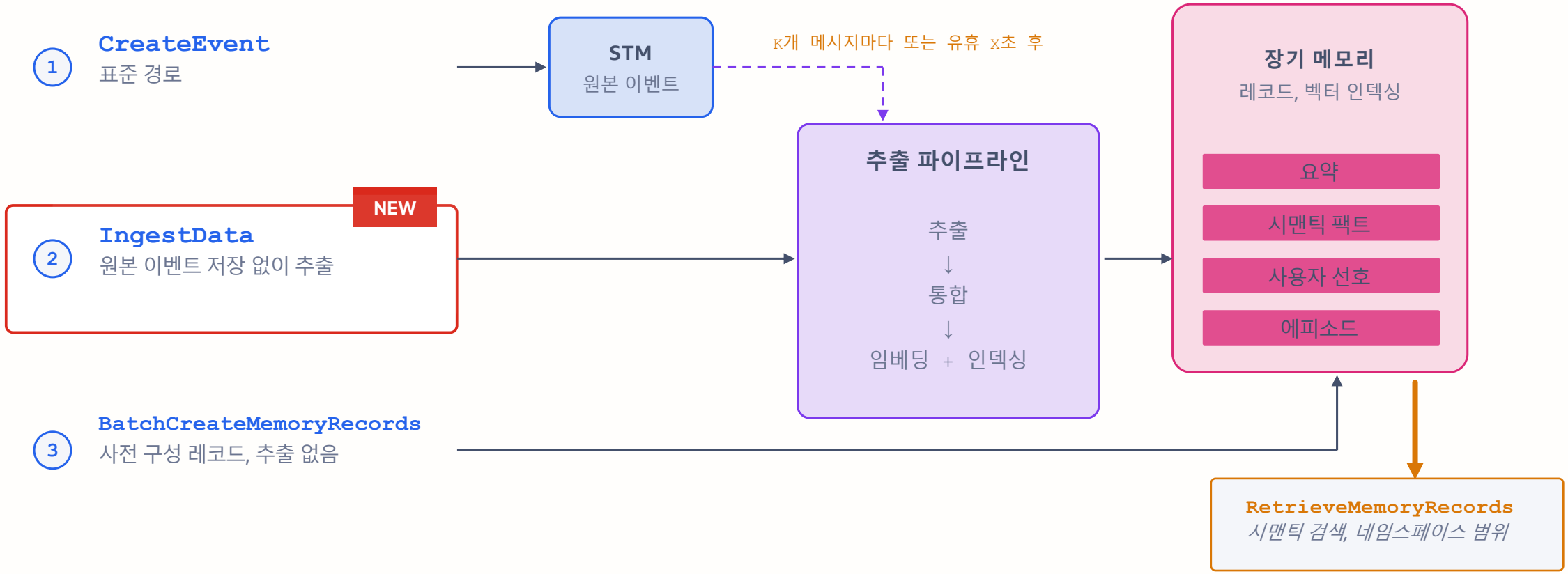
계획, 스크래치패드, 도구 결과 등 무엇이든 체크포인트할 수 있습니다. 추출은 blob을 건너뜁니다.

metadata

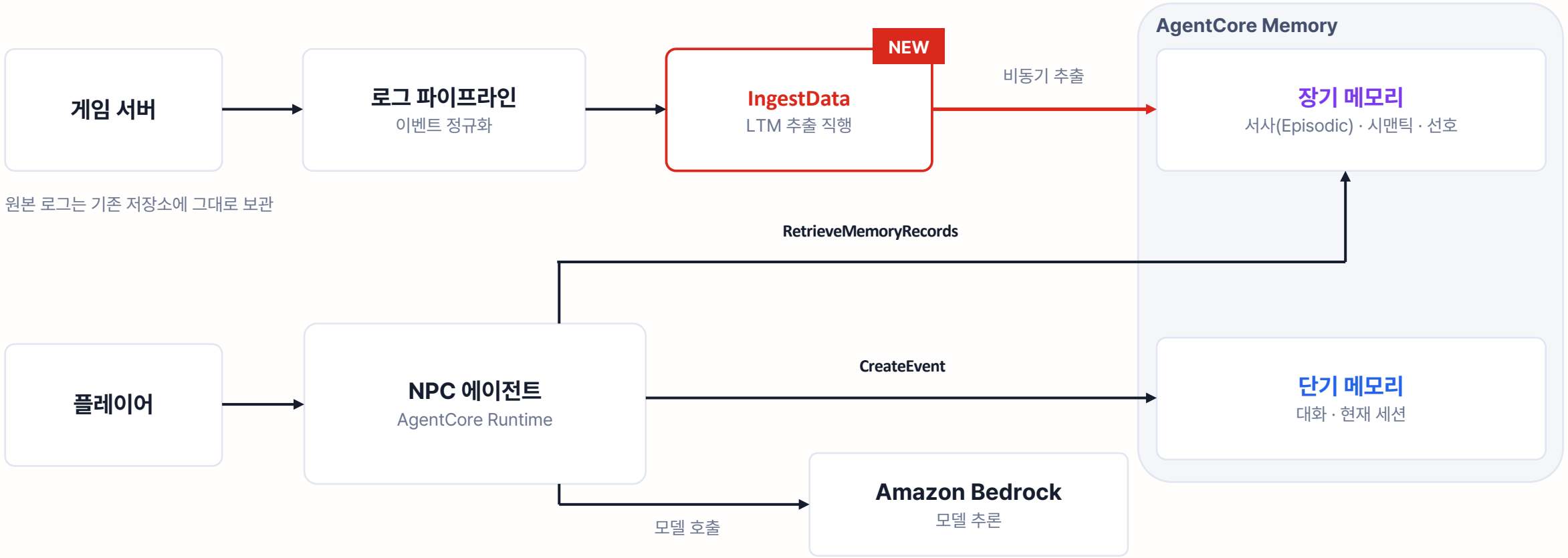
키-값 태그. ListEvents와 검색에서 정확 일치 필터로 돌아옵니다. 임베딩은 쓰이지 않습니다.

장기 메모리로 들어가는 세 가지 경로

게임 로그의 진입 경로



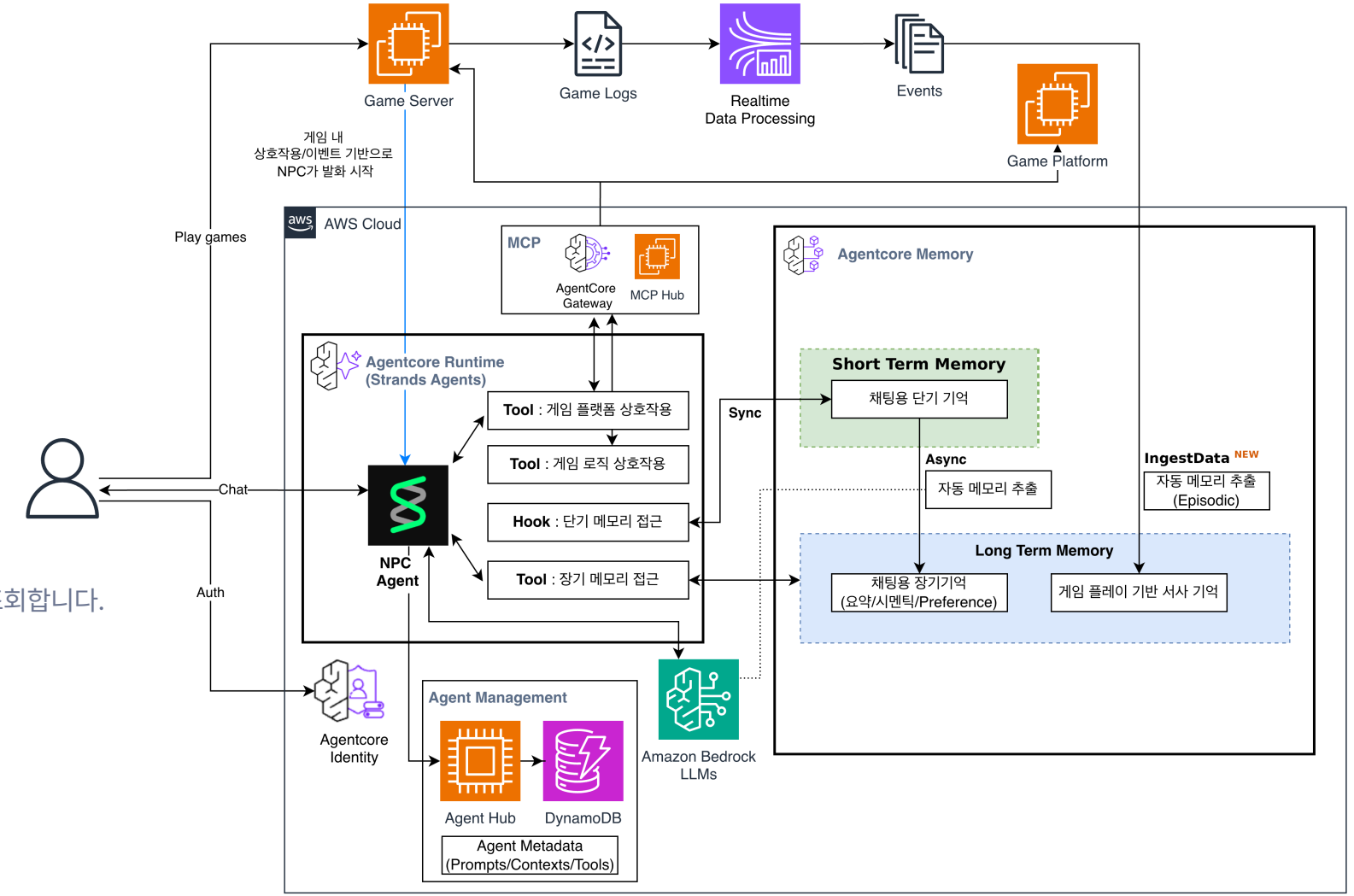
참고 아키텍처 — 게임 데이터와 NPC를 연결하기



플레이어 ID와 네임스페이스가 기억의 조회 범위를 가릅니다 — 각 플레이어의 NPC는 자기 기억만 봅니다

서사 기억을 활용한 Game NPC 예시 아키텍처 - AgentCore Memory

- **게임 이벤트 → LTM 직행**
  IngestData로 STM 저장 없이 서사 기억을 추출합니다
- **채팅은 표준 경로**
  CreateEvent로 단기 기억에 쌓고 비동기 추출합니다.
- **검색은 하나로**
  NPC는 RetrieveMemoryRecords 하나로 두 기억을 모두 조회합니다.



02

배경

게임 도메인에서 AI 에이전트가 마주치는 문제, 그리고 행동 맥락이라는 새로운 자산

게임 도메인의 AI 에이전트

USE CASE 01

AI NPC

유저와의 서사를 기억하고, 세계관 안에서 맥락에 맞게 반응하는 NPC

"포션 하나 안 쓰고 갱단을?
...내가 판 포션은 장식이었나 봐요."

USE CASE 02

게임 AI 어시스턴트

인게임 맥락을 아는 게임 도우미 — 어제의 플레이를 이어주는 에이전트

"저번 퀘스트, 마지막 단계만 남았어요 — 이어서 하시겠어요?"

USE CASE 03

AI 대화형 게임

AI 캐릭터 채팅처럼 대화가 플레이가 되는 게임 — 기억이 곧 세이브 데이터

대화 기록이 유일한 게임 상태
— 기억 시스템이 곧 게임 엔진

유저 맥락을 잘 기억하는 것의 중요성

구분	스크립트 NPC	기억 없는 AI NPC	기억하는 AI NPC
대화	정해진 대사만	자유로운 대화	자유로운 대화
서사	정해진 서사를 따른다	매번 처음부터 다시	유저와 함께 쌓인다
유저를 대하는 방식	모두에게 같은 이야기	매번 처음 만난 사람	나를 기억하는 상대
몰입감	**무난하다**	**금방 깨진다**	**깊어진다**

행동 맥락과 대화 맥락

구분	행동 맥락	대화 맥락
소스 데이터	유저가 직접 겪은 사건	에이전트와 나눈 대화 기록
데이터 형태	정형 — 구조화된 이벤트 로그	비정형 — 자연어
추출 방식	규칙 기반 · 결정적	LLM 기반 · 비결정적
데이터 양	유저당 하루 수천 건 이상	세션당 수십~수백 턴
성격	객관적 사실	주관적 생각 · 감정
예시	방금 강화에 실패함	“강화에 실패해서 속상해”

유저가 100시간을 플레이하는 동안, 대화가 3시간이었다면

97%

행동 맥락이 담을 수 있는 경험의 비중 — 대화 맥락은 전체의 3%에 불과하다

행동 맥락의 중요성

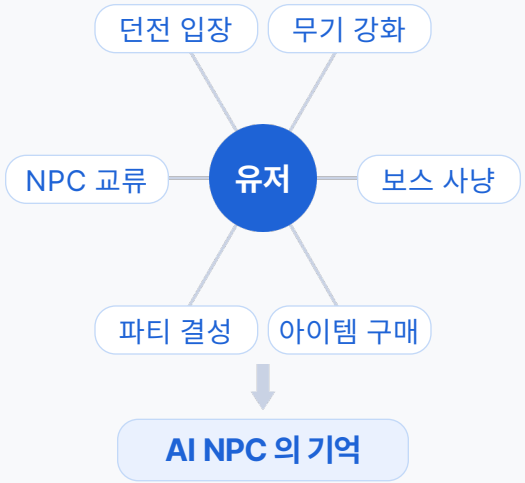
기억의 원천

일반 대화형 AI



말해 준 것만 안다

게임 AI



말하지 않아도 플레이하는 동안 계속 쌓인다

방금 한 행동을 아는 NPC

대장간	보스 처치	다시 대장간
무기 강화	강화한 무기로	NPC 의 첫마디는?

맥락 없음 "어서 오게. 오늘은 무엇을 강화하러 왔나?"

맥락 있음

"내가 어제 강화시켜 준 무기로 보스를 물리치고 오는 길이구먼. 자랑스럽네."

"이미 13번이나 강화한 무기일세. 이제 이쯤 해도 되지 않겠나...?"

행동 맥락의 진입 장벽

01	**수집** 채팅 로그만 모으면 되는 대화 맥락과 달리, 유저 주변의 모든 이벤트를 식별해야 한다	**모든 이벤트가 대상** 전투 · 성장 · 거래 · 제작 · 소셜 …
02	**규모와 실시간성** 유저도 많고 유저당 데이터도 많은데, 행동 맥락은 바로 반영되어야 한다	**초당 수백만 건 유입** 실시간 변환 파이프라인 필요
03	**게임 도메인의 특수성** 게임 안의 개념 · 상호작용 체계 · 해석 규칙은 그 게임 안에서만 유의미하다	**게임마다 다른 체계** 타이틀별 개념 · 해석 규칙 재사용 불가
04	**선행 연구 부족** 에이전트 메모리 연구는 대부분 대화 맥락을 다루고, 행동 맥락의 레퍼런스는 드물다	**대화 맥락 중심의 연구** 행동 로그의 도메인 종속 · 일반화 어려움

적합한 시스템이 없다면,
Do It Yourself!

Live Memory

- ✓ 유저별 행동 맥락
- ✓ 대규모 · 실시간
- ✓ 감당 가능한 비용

03

기억 모델링

구조화된 로그를 그래프 기억으로 — 결정적 파이프라인이 만드는 기억의 구조

행동 로그 — 쌓기만 해서는 기억이 되지 않는다

로그로 쌓으면 — 파편

- 18:02 ItemUpgrade **M_08 W_01 destroyed**
- 18:04 ItemEquip **W_02**
- 18:10 PartyJoin **CH_13**
- 19:12 DungeonEnter **D_42**
- 19:41 BossKill **B_07 D_42**
- 19:42 ItemGet **I_55 D_42**
- 20:05 ItemUpgrade **M_08 W_02 succeeded**

관계형 DB — 매번 JOIN



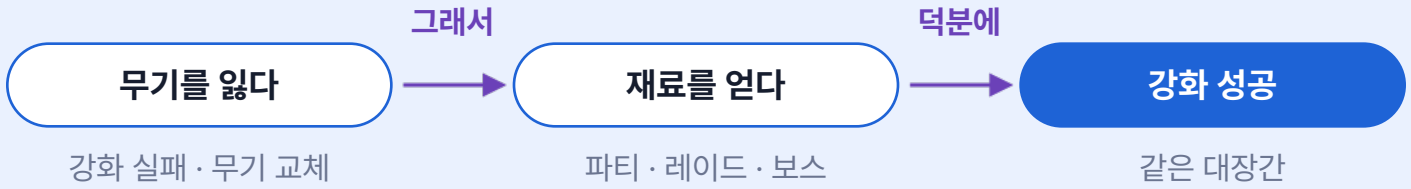
벡터 DB — 순서·인과가 묻힘



기억은

사건 사이의 연결

18:02 ~ 20:05 · 대장간 → 레이드 → 다시 대장간



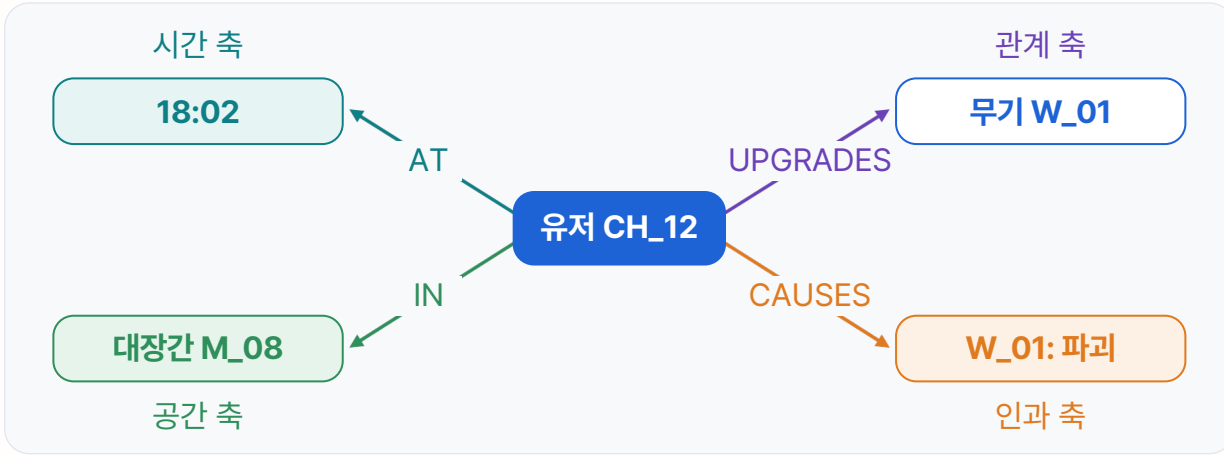
행동 로그 — 이미 연결의 단서를 품고 있다

이미 필드로 구조화된 행동 로그

```
{ "logtime": "18:02",
  "charid": "CH_12",
  "logtype": "ItemUpgrade",
  "mapid": "M_08",
  "itemid": "W_01",
  "result": "destroyed" }
```

규칙 기반 파서

실시간 변환
모델 호출 없음



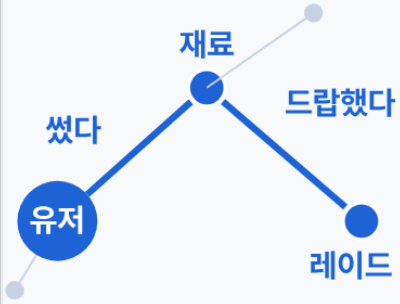
기억 회상 — 다섯 종류의 질문

1 단일 홉
Single-hop



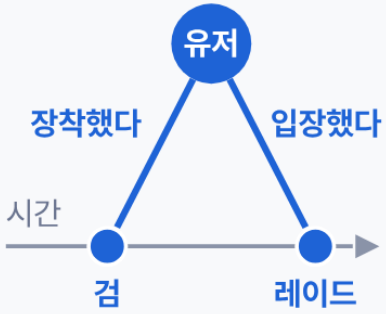
"아까 같이 파티 맺은
그 친구, 누구였지?"

2 다중 홉
Multi-hop



"강화에 썼던 그 재료,
어느 레이드에서
나온 거였죠?"

3 시간적 추론
Temporal



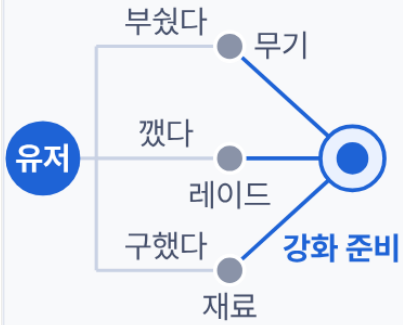
"레이드에 들어가시기
전에 새로 장착하신
무기가 이 검이었나요?"

4 기억 정정
Conflict resolution



"아, 맞다. 그 무기는
부서졌으니 지금 쓰시는 건
새로 장착하신 검이죠!"

5 유도된 상태
Derived-state



"지난번 실패에도
레이드에서 재료까지
구해 오셨으니, 이번
강화는 꼭 성공시켜
드릴게요!"

기억 회상 — 최신 연구들의 공통 기반은 그래프 구조

회상 단계	평가하는 인지 능력	대표 벤치마크 (공식 태스크)	최고 권위 SOTA · 표준 모델	SOTA 의 핵심 메커니즘
단일 홉 Single-hop	즉각적 단서 포착 및 사실 회상	**LoCoMo** · Single-hop **LongMemEval** · Information Extraction	**GAM** (ACL 2026) **LightRAG** (EMNLP 2025)	원시 로그와 엔티티를 **지식 그래프 구조로 구조화**하여 단서 탐색 시 노이즈 간섭 차단
다중 홉 Multi-hop	기억의 연쇄적 연결 및 인과 추론	**LoCoMo** · Multi-hop **LongMemEval** · Multi-Session Reasoning	**HippoRAG** (NeurIPS 2024) **GAM** (ACL 2026)	무거운 조인 연산 없이 **지식 그래프 순회**로 분산된 맥락을 직접 연결
시간적 추론 Temporal	사건의 선후 관계 / 타임라인 정렬	**LoCoMo** · Temporal Reasoning **LongMemEval** · Temporal Reasoning	**GAM** (ACL 2026) **Zep Agent Memory**	엣지에 타임스탬프와 사건 순서 정보를 직접 바인딩하는 **시계열 그래프 구조**
기억 정정 Conflict resolution	최신 맥락 반영 및 모순 해결	**LongMemEval** · Knowledge Updates	**A-Mem** (NeurIPS 2025) **HippoRAG 2** (ICML 2025)	새로운 사건 유입 시 낡은 연결을 끊고 **동적 그래프 진화**를 통해 모순 교정
유도된 상태 Derived-state	누적 경험 기반 종합 상태 도출	[실전 시스템 워크로드] 대화형 벤치마크 외 게임 시스템 영역	**MemGPT** (ICLR 2024)	수치 연산은 고속 캐시로 분리하되, **상태 전이 히스토리를 그래프로 역추적**하여 그 이유를 복원

기억 회상 — 그래프 구조가 필요한가?

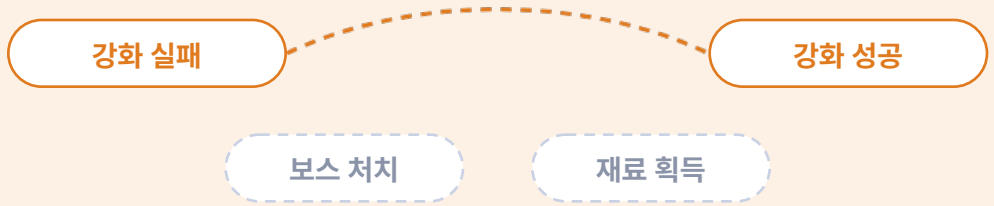
ACL 2026 실증 분석 연구가 던진 반론, 그리고 그 답

연구의 결론

잘 설계된 그래프라면, 그렇다

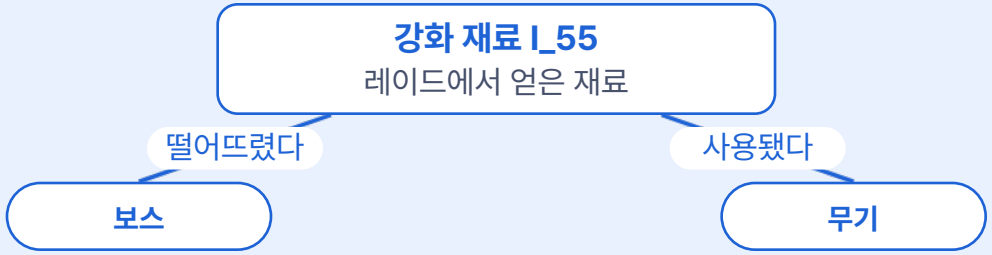
반론이 맞는 경우 · 그래프 없는 방식보다 낮다

유사도로 엮은 그래프



여전히 유효한 경우 · 비교한 방식 중 가장 높다

엔티티 연결 + 맥락 서술



출처 · Hu et al., “Does Memory Need Graphs? A Unified Framework and Empirical Analysis for Long-Term Dialog Memory”, ACL 2026

슈퍼 허브 노드 문제

필드를 그대로 노드로 옮긴 결과, 유저 중심의 성게 그래프

1. 유저가 몬스터를 잡았다

유저 — 잡았다 → 몬스터 B_07

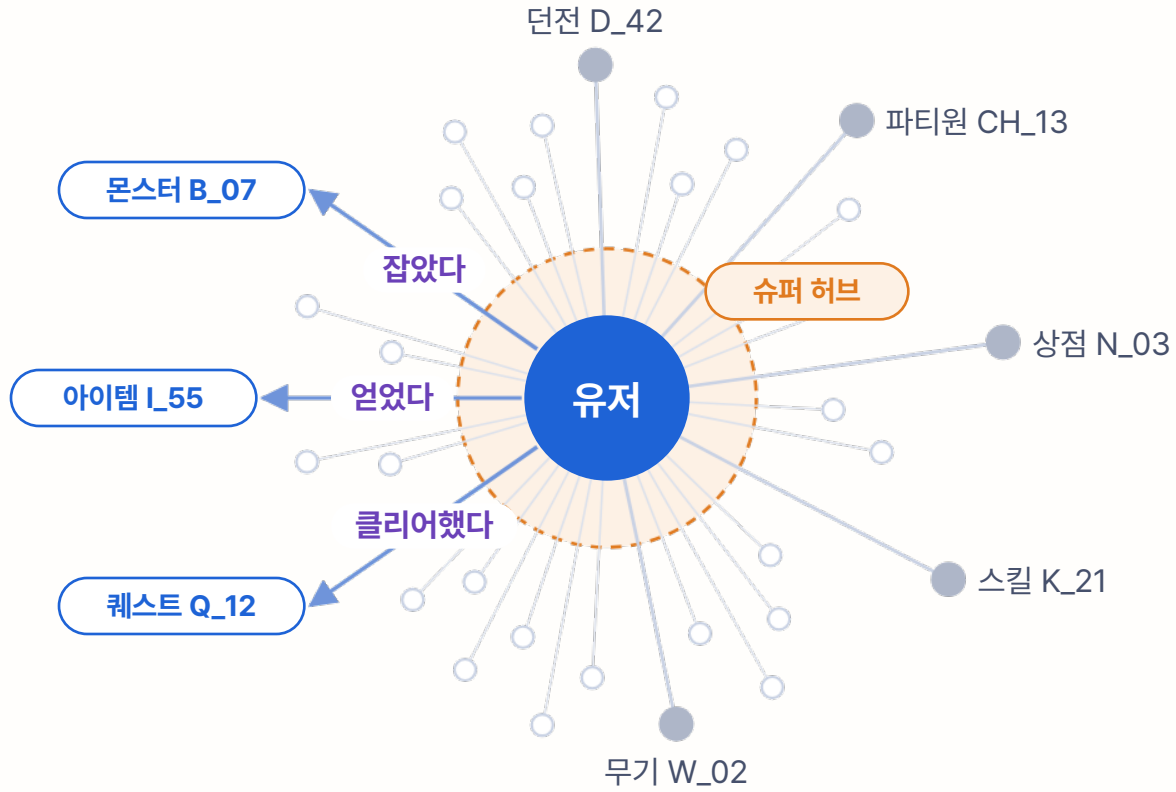
2. 유저가 아이템을 얻었다

유저 — 얻었다 → 아이템 I_55

3. 유저가 퀘스트를 클리어했다

유저 — 클리어했다 → 퀘스트 Q_12

그래프로 옮기면

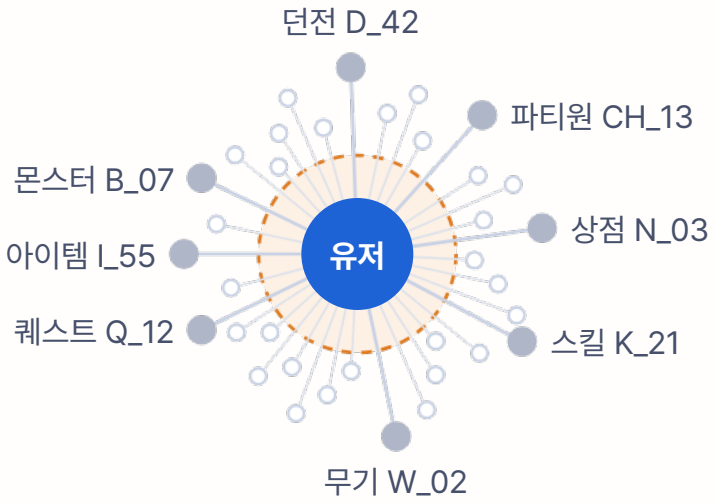


선행 연구 · Paul et al., 2026 (arXiv:2603.27910) · Lau et al., 2026 (arXiv:2602.01965)

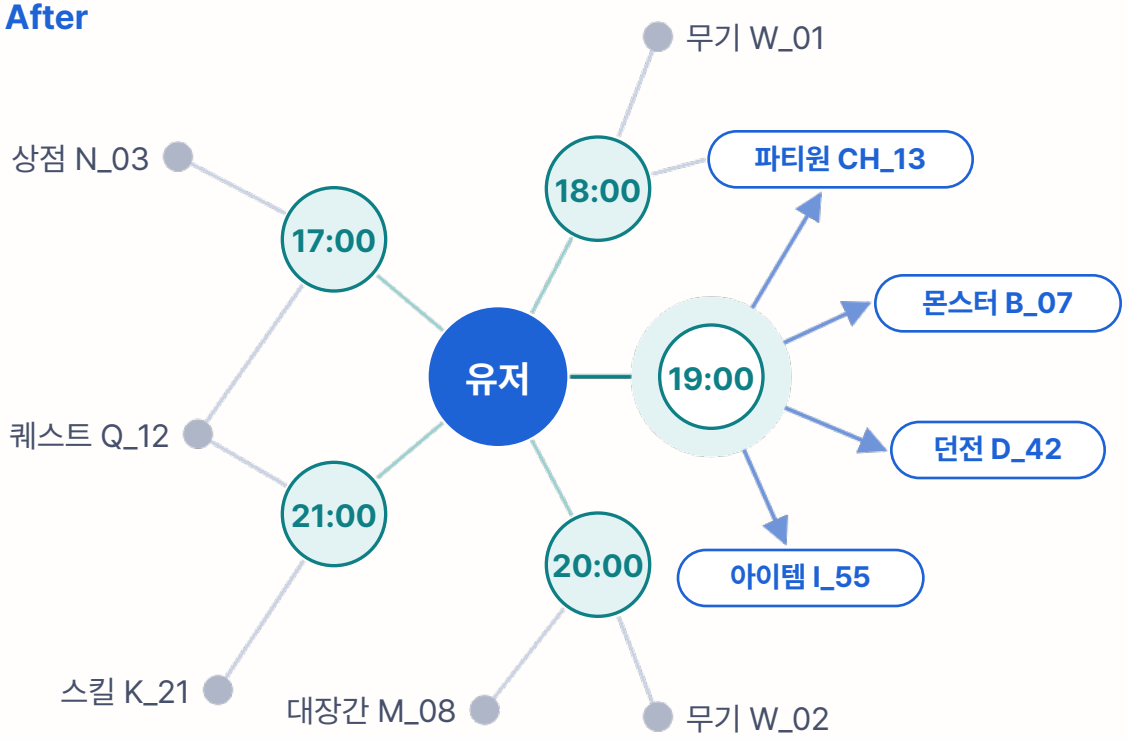
시간 노드 도입

시간 노드가 같은 시간대에 일어난 사건들을 하나로 묶는다

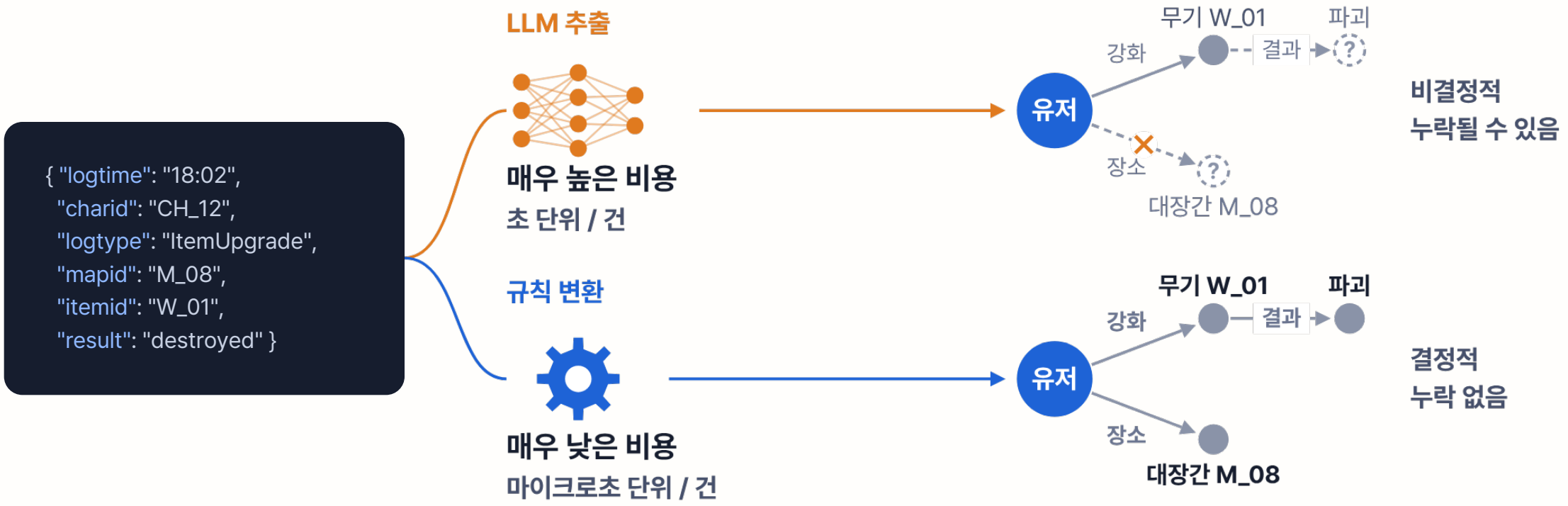
Before



After



기억 추출 방식 비교 — LLM 추출과 규칙 변환



기억 추출 규칙의 자동 생성

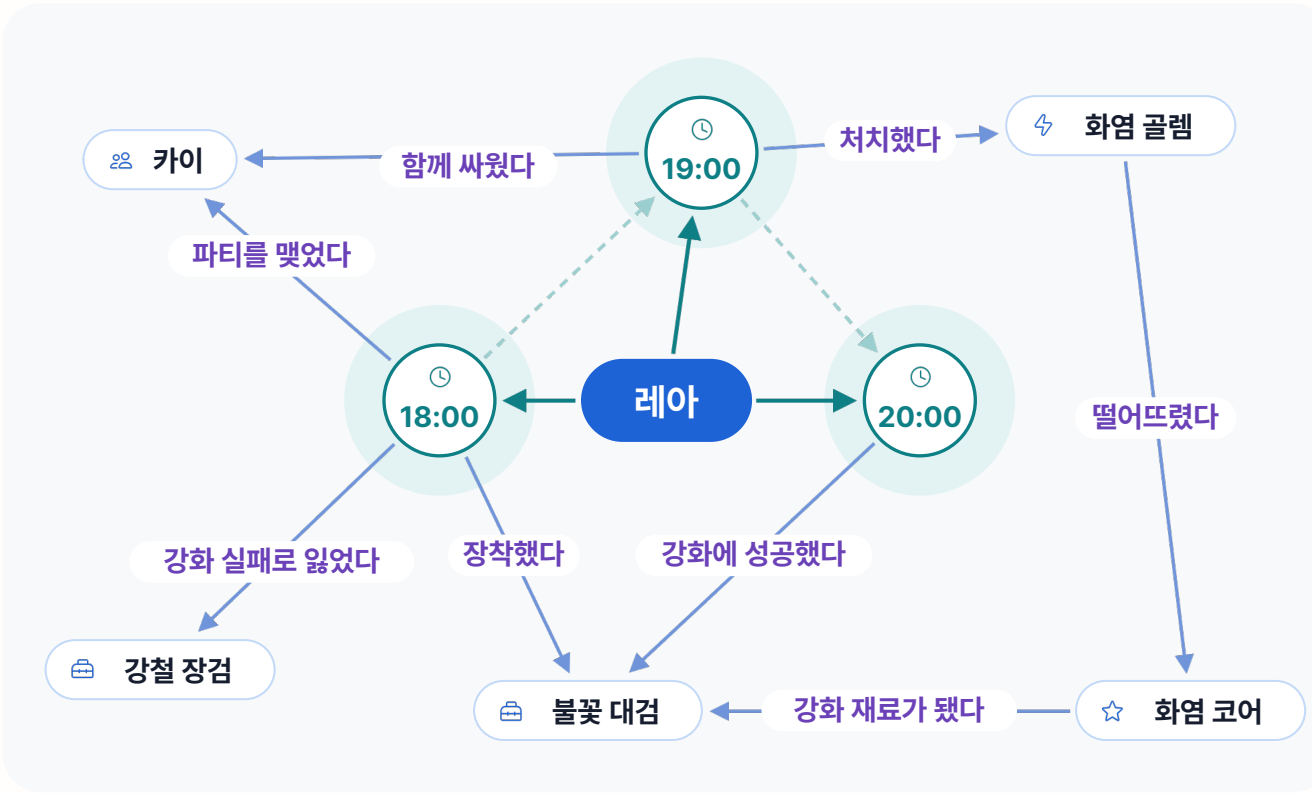


로그 구조를 해석해 만들기에, 규칙에 기억 그래프의 의미 서술이 담긴다

생성 · 검증 · 평가의 반복은 AI 에이전트가 맡고, 사람은 마지막 승격만 판단한다

차가운 로그가 아니라,
해석된 서사 기억으로 쌓인다

18:02 ItemUpgrade **M_08 W_01 destroyed**
18:04 ItemEquip **W_02**
18:10 PartyJoin **CH_13**
19:12 DungeonEnter **D_42**
19:41 BossKill **B_07 D_42**
19:42 ItemGet **I_55 D_42**
20:05 ItemUpgrade **M_08 W_02 succeeded**



04

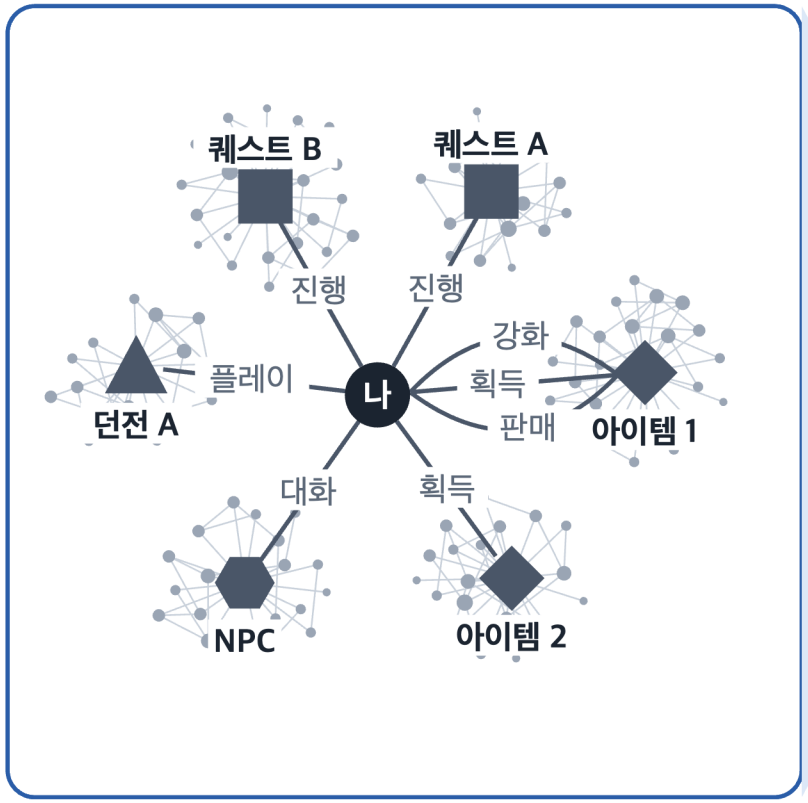
기억 저장소

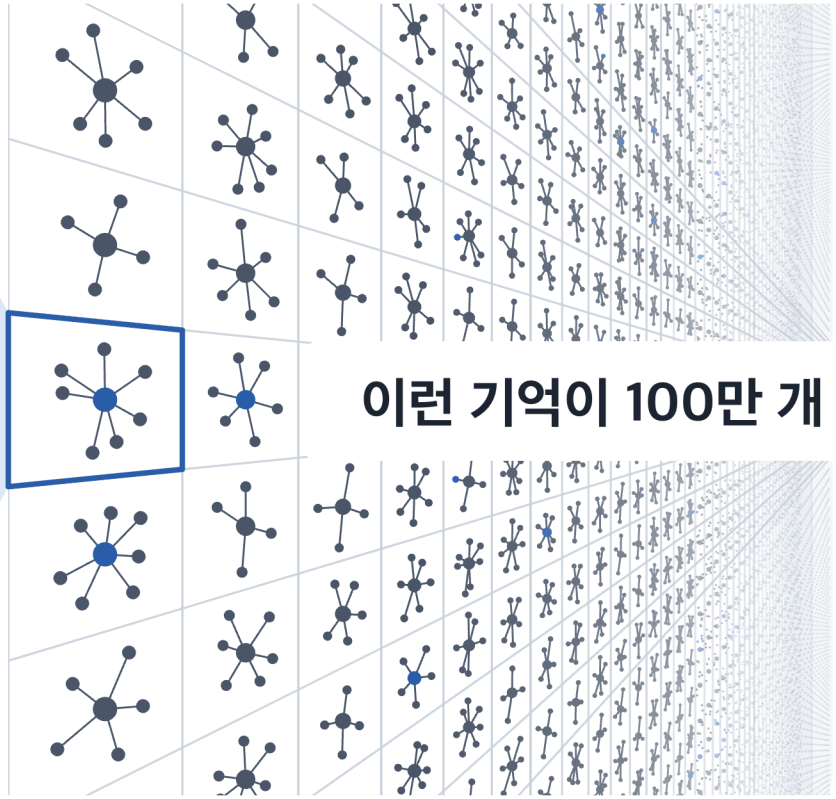
멀티테넌시 대규모 실시간 쓰기를 위한 In-house Graph DB

라이브메모리의 워크로드

노드
10만

엣지
40만

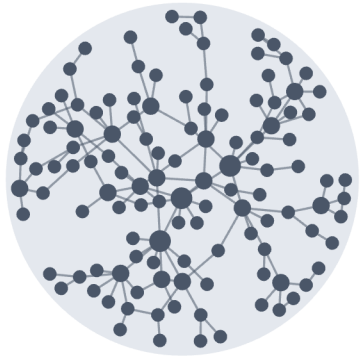




전통적인 워크로드와의 차이점

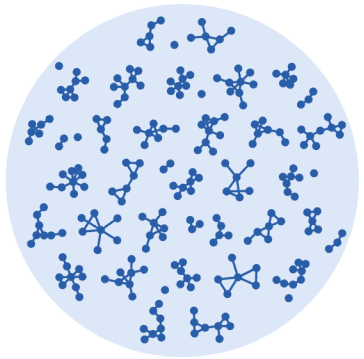
Facebook 700억 × 1개

소셜 그래프 · Facebook, 2011 · 노드 7.2억 + 엣지 687억



Nexon 50만 × 100만 개

Live Memory: 100만 유저의 기억 그래프



기존 그래프 DB 제품으로론 안될까?

	제품 A	제품 B	제품 C	제품 D
멀티테넌시 수백만 그래프	부분 충족	불가	부분 충족	충족
쓰기 수평 확장 100만 RPS	부분 충족	불가	불가	충족
종합 두 요구사항 모두 충족	미충족	미충족	미충족	충족

제품 D 를 사용한다면?

그래프 하나

215 MB

노드 33만
엣지 148만

× 100만
× 2 (HA)

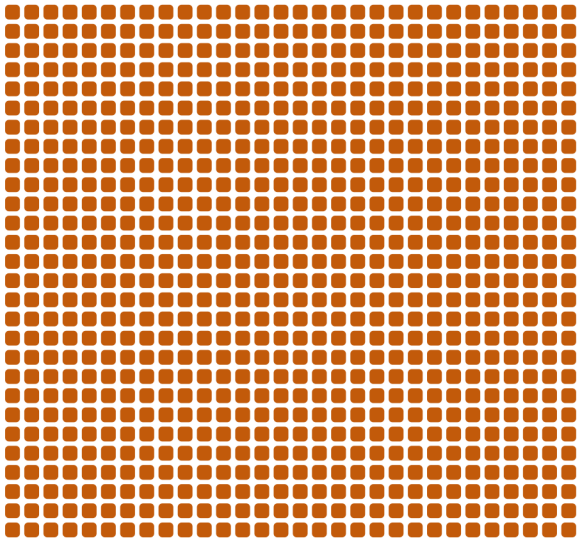
합계 메모리

430 TB

환산

인스턴스 840대

한 칸 = r7g.16xlarge 1대 (512 GB)



연 비용

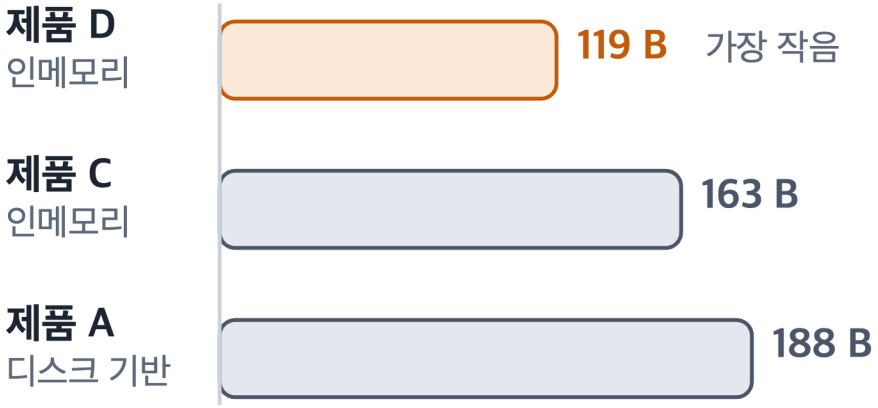
412억 원

LDBC SNB Interactive SF0.1 기준 측정
예상 워크로드의 약 3.6배

왜 비쌌을까?

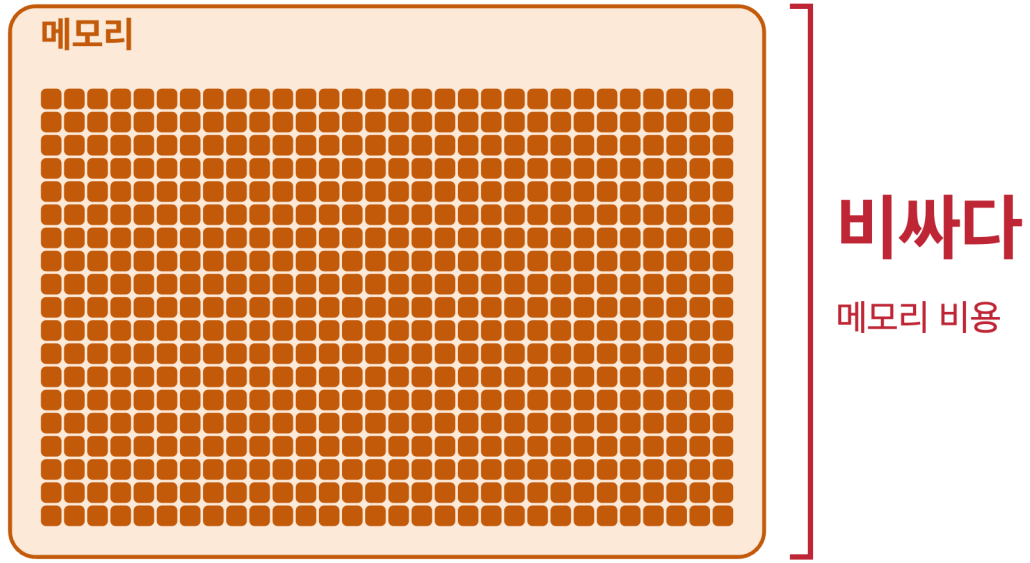
경쟁 제품 대비 가장 최적화된 레이아웃

LDBC SNB 그래프 1개의 엔트리(노드·엣지)당 메모리

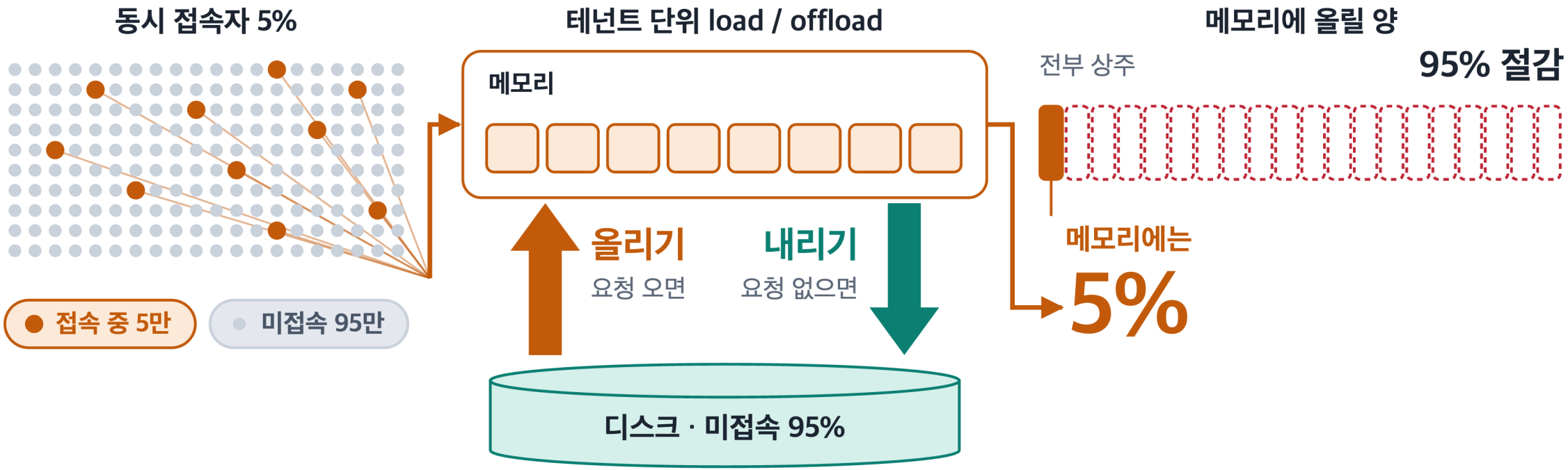


그런데

그래프 100만 개 전부 상주

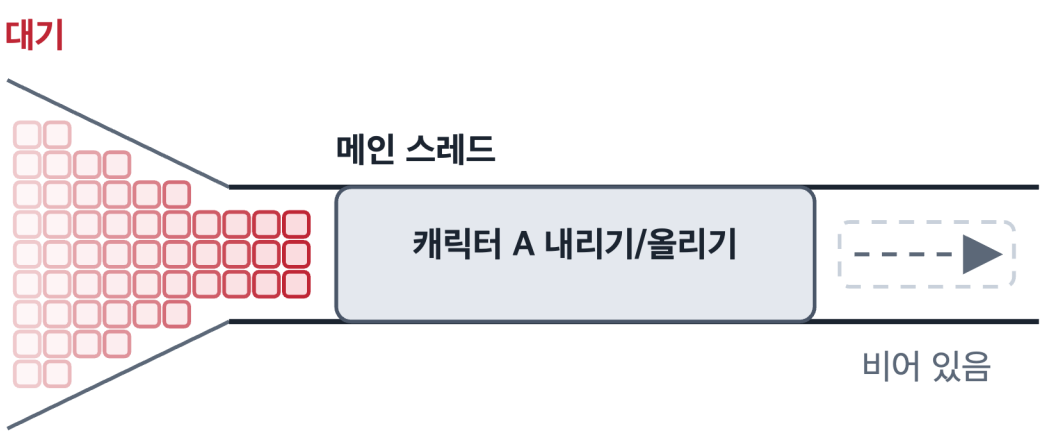


Offloading: 메모리엔 접속 중인 유저만

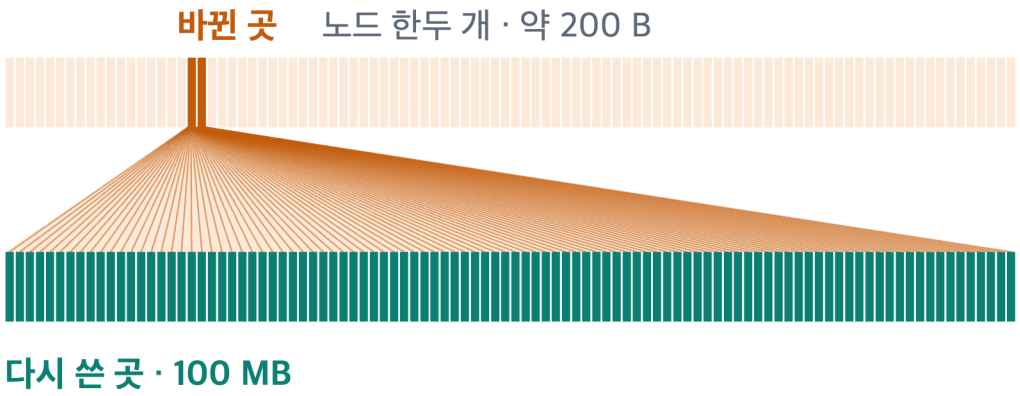


제품 D 위에 offloading 을 직접 구현한다면?

메인 스레드 블로킹

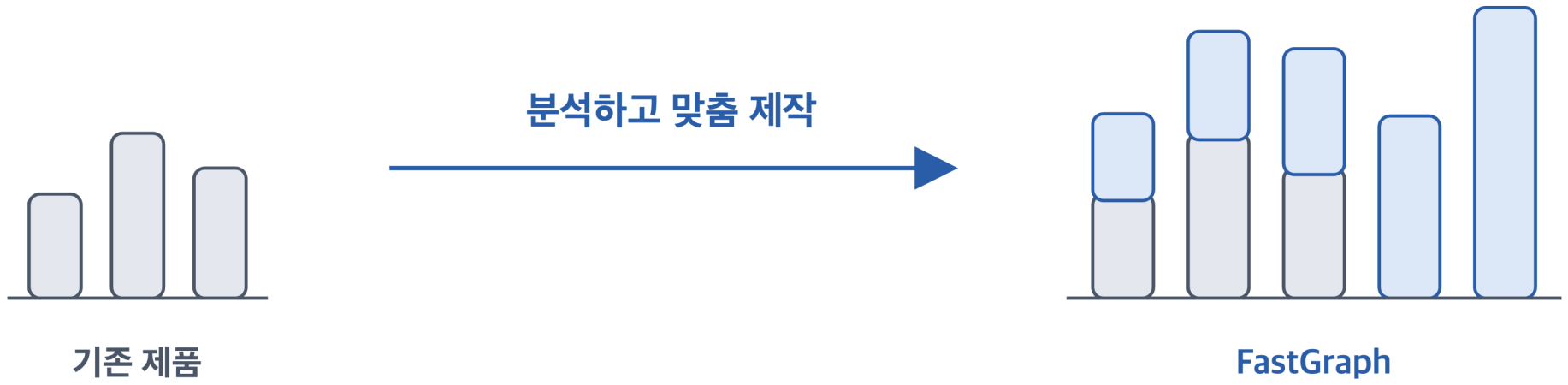


쓰기 증폭



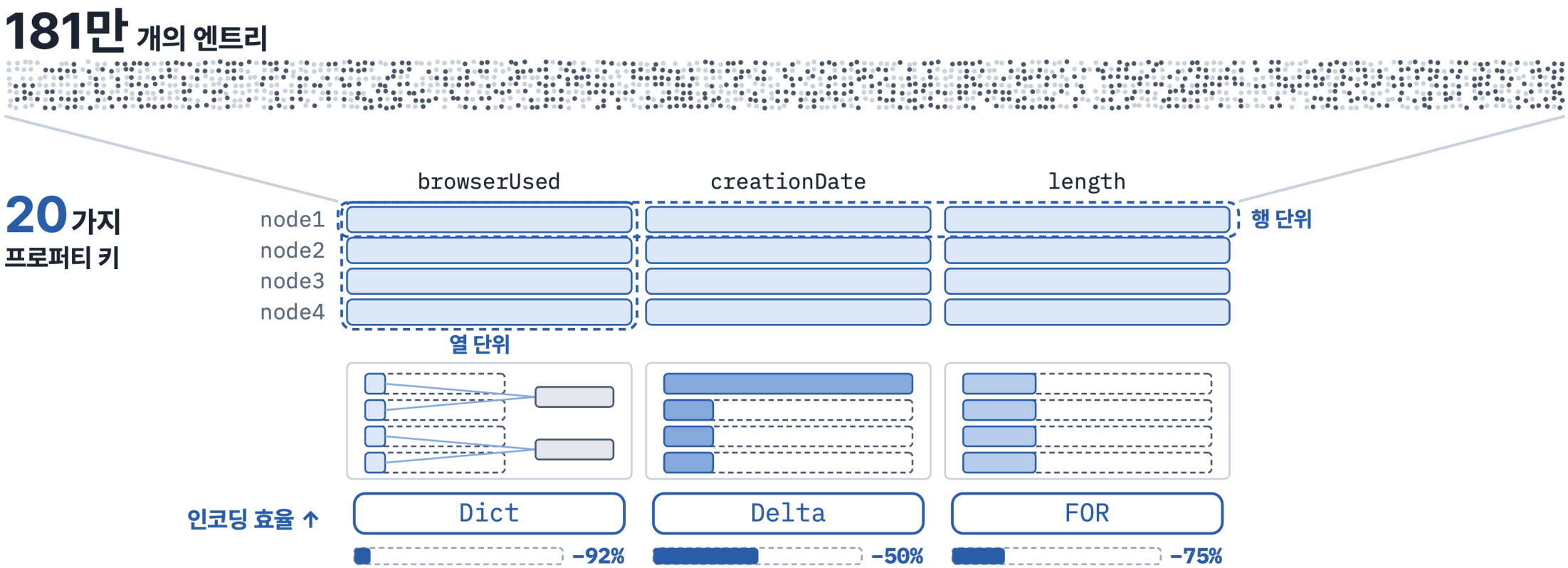
메인 스레드에서 직렬화 · 직렬화 비용(그래프 크기에 비례): 제품 D 공식 문서

FastGraph: in-house 멀티 테넌트 그래프DB



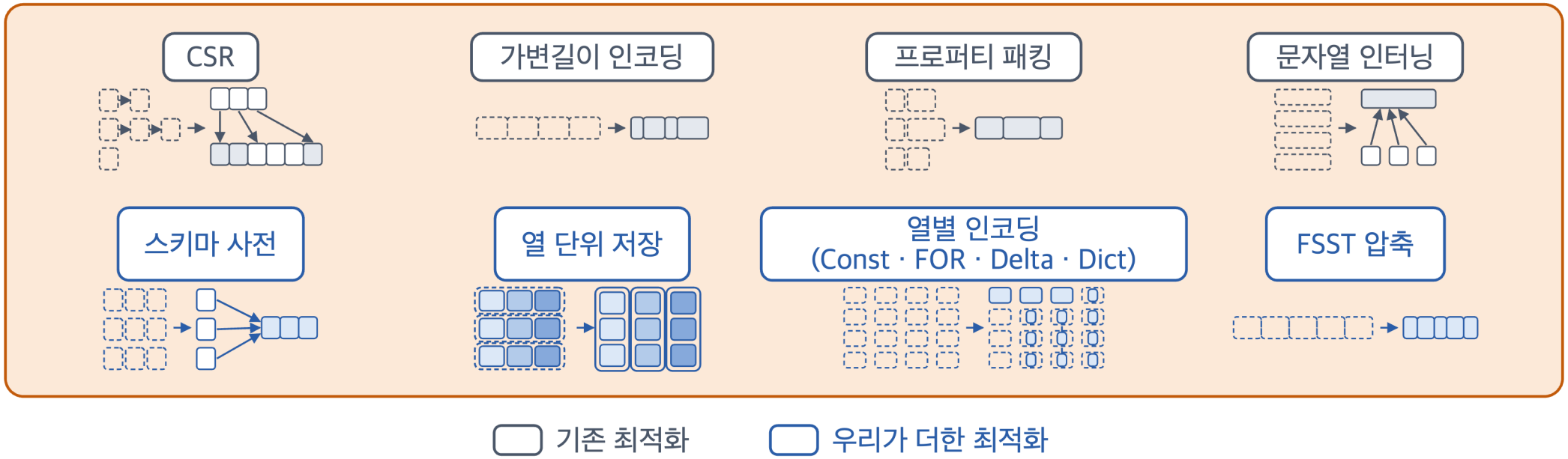
핵심 가설: 그래프 DB에 쌓이는 프로퍼티 키의 카디널리티는 작다

컬럼형 레이아웃에 특화된 워크로드

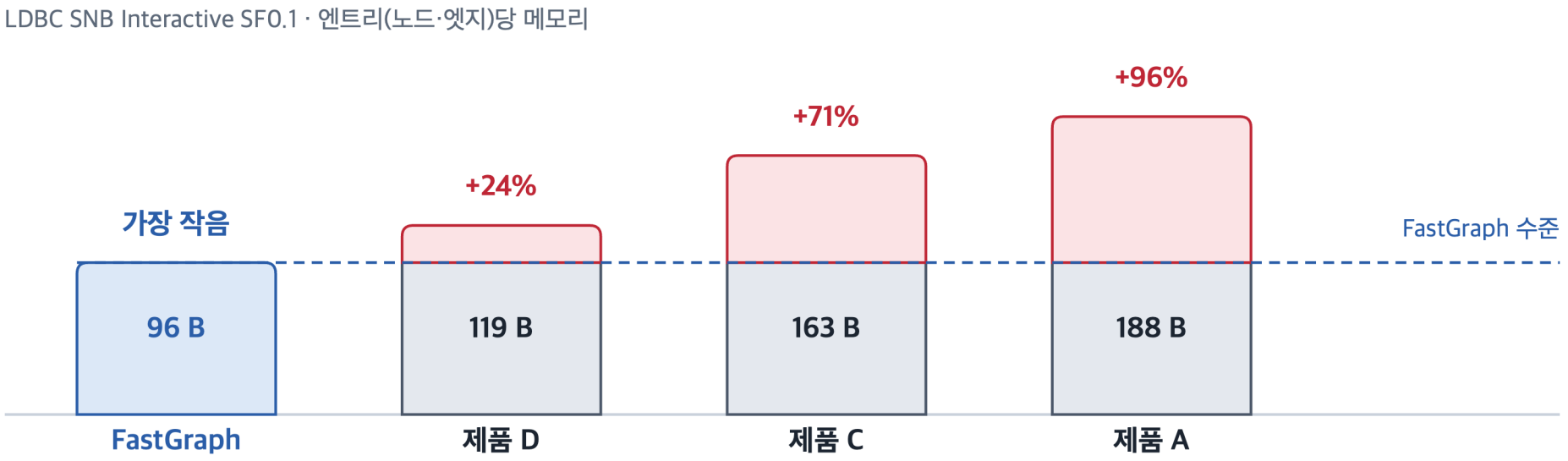


LDBC SNB Interactive SF0.1 · Comment 노드 기준 · Delta 는 계산값

기존 최적화 기법을 가져오고 컬럼형 레이아웃을 적용

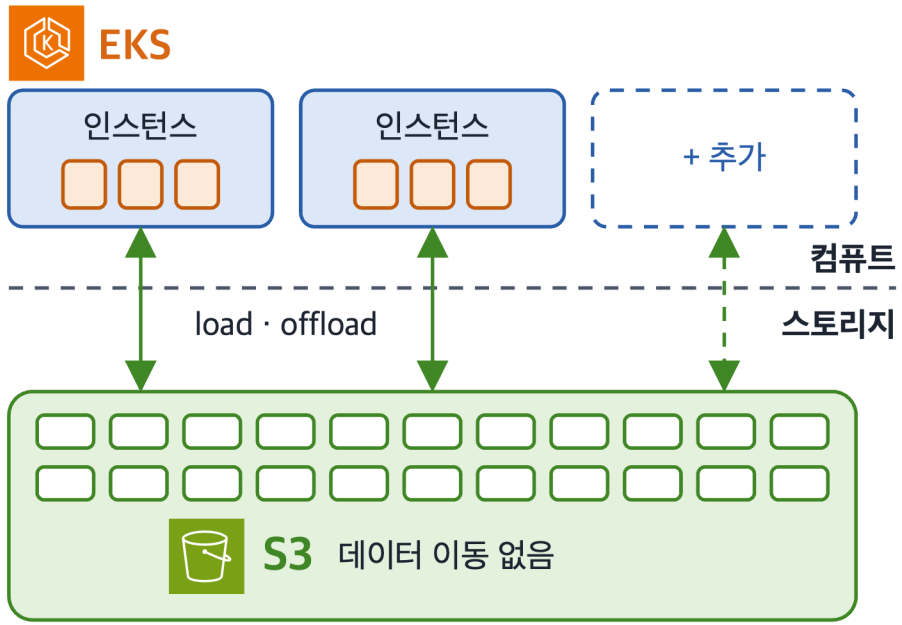


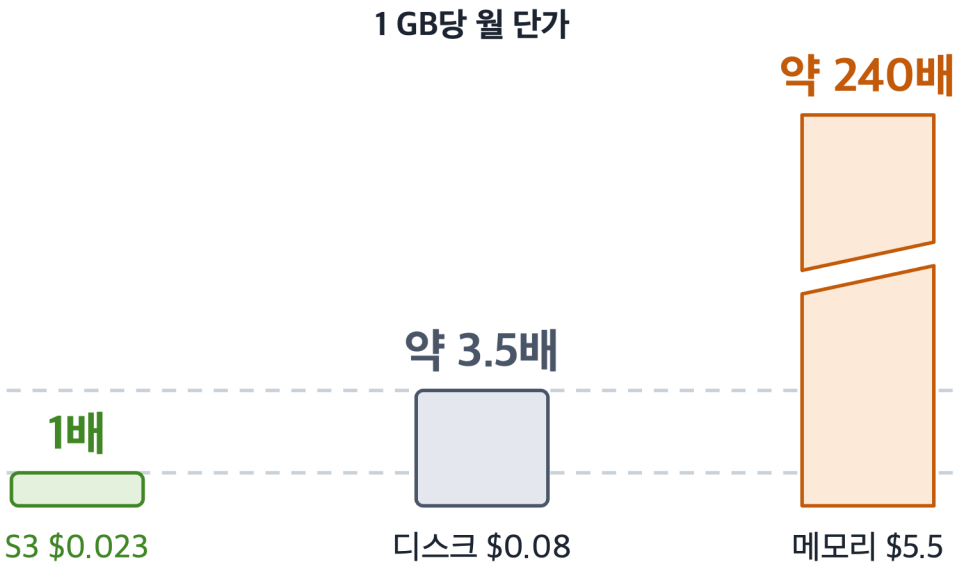
그 결과


LDBC SNB Interactive SF0.1 · 엔트리(노드·엣지)당 메모리
+96%
+71%
+24%
가장 작음
FastGraph 수준
96 B
119 B
163 B
188 B
FastGraph
제품 D
제품 C
제품 A

Offloading

Disaggregated Storage 아키텍처

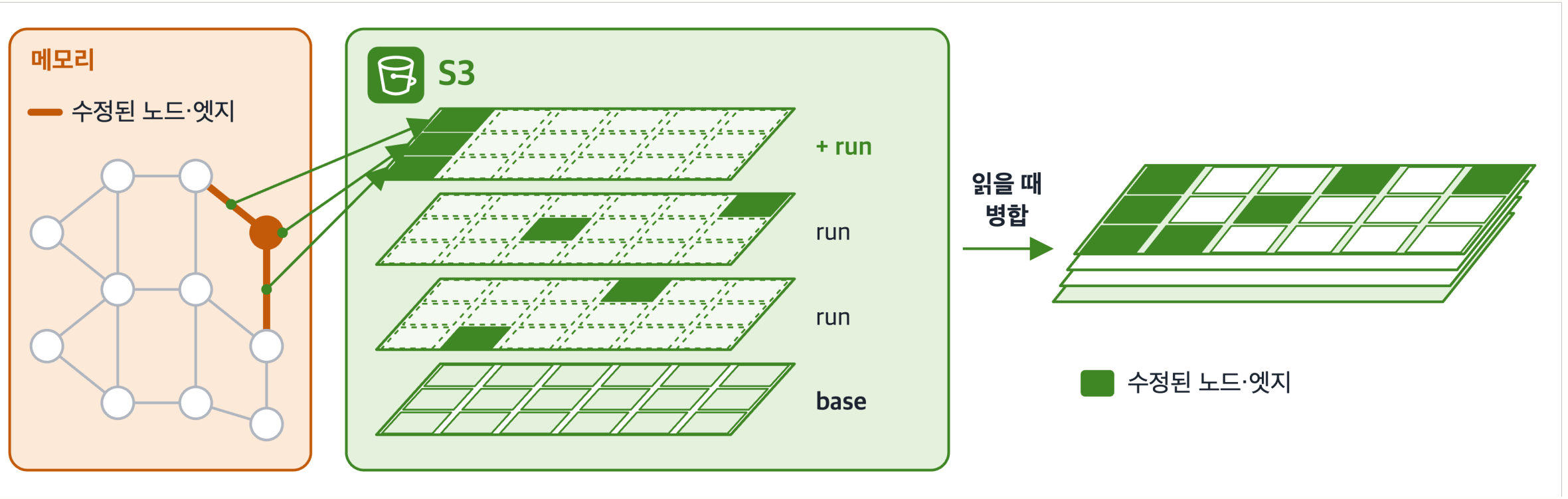




가격 기준: S3 Standard $0.023/GB·월 · EBS gp3 $0.08/GB·월 · 메모리 약 $5.5/GB·월(r7g.12xlarge)

Offloading

쓰기 증폭을 최소화하기 위한 차분 저장 방식



최종 비용 비교

LDBC SNB Interactive SF0.1 측정 기준 · 그래프 100만 개 · 제품 D는 HA를 위해 2배 · 막대 길이는 용량에 비례

제품 D
전량 메모리



430 TB
연 412억 원

메모리 8.6 TB

FastGraph
메모리 + S3



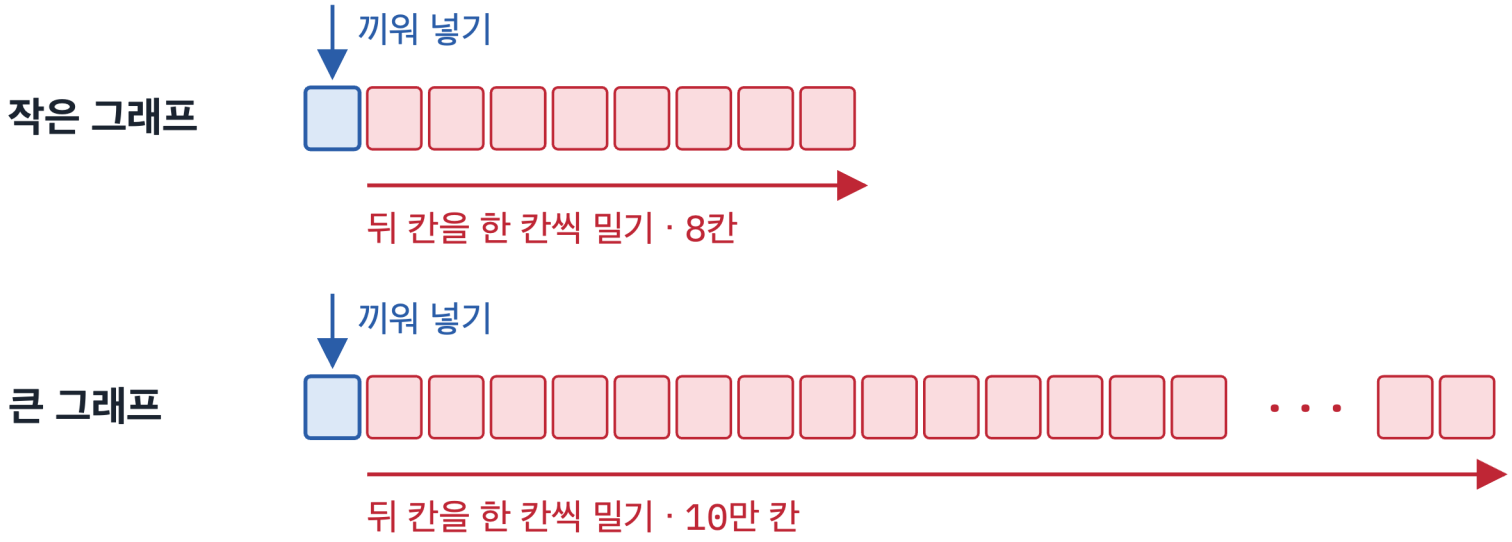
S3 15.3 TB

8.6 + 15.3 TB
연 8.3억 원

약 98%
감소

트레이드오프: 컴팩트 vs 쓰기 성능

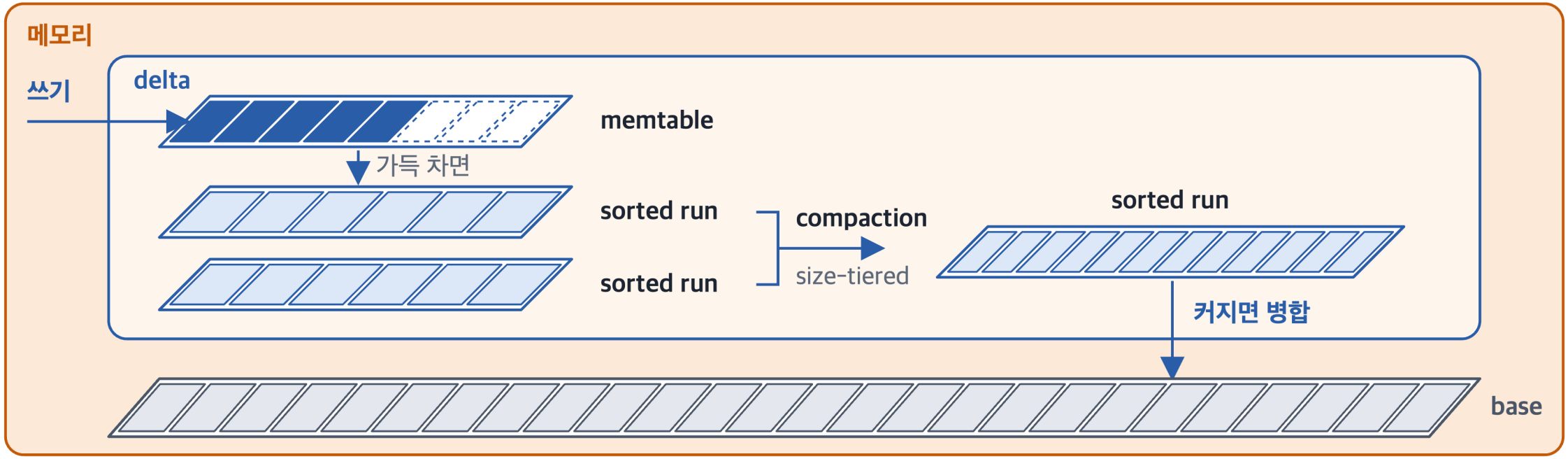
CSR의 컬럼 한 줄에 엣지 1건 삽입



O(N)

쓰기 1건의 비용

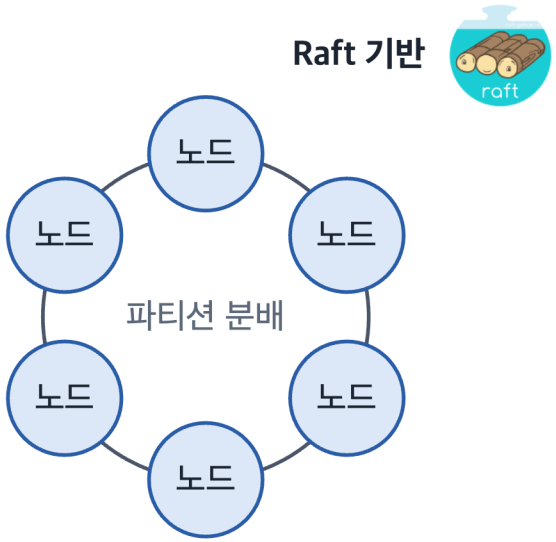
쓰기에 최적화된 LSM 트리 구조 도입



멀티 테넌시 지원

100만 개의 그래프를 효율적으로 담기

분산 처리

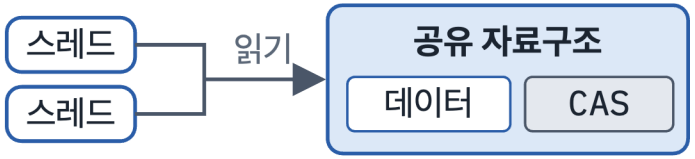


경합 최적화

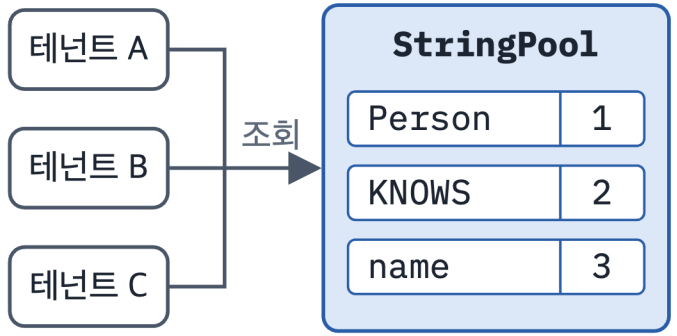
False Sharing 방지 · 캐시라인 패딩



Lock-Free 읽기



문자열 공유 사전



라벨 · 관계 타입 · 키 · 외부 ID

그 결과

LinkBench · 단일 테넌트

동시 요청 8 · 읽기·쓰기 혼합 처리량 (req/s)



넥슨 게임 실데이터 · 멀티 테넌트

동시 요청 100 · 쓰기 처리량 (req/s)

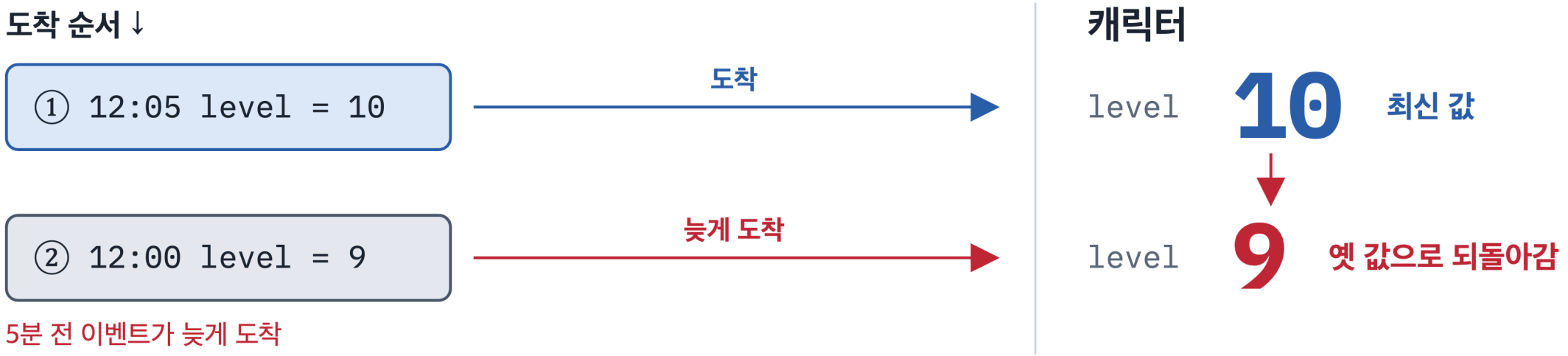


EKS 같은 리소스(pod 1, 3 vCPU/ 6GiB)에서 측정(2026-10)

순서 섞임: 이벤트 소싱의 고질병

도착 순서 ≠ 생성 순서

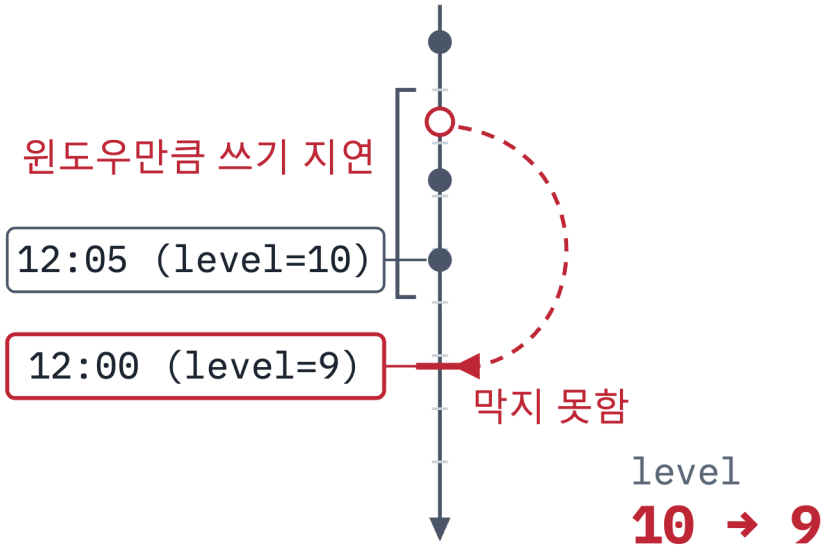
늦게 온 이벤트가 최신 값을 되돌림



순서 섞임: DB 밖에서 해결하는 방법

정렬 윈도우

이벤트 시각



복잡한 쿼리로 대응

CLIENT · Cypher

```
ON MATCH SET
 n.level = CASE
  WHEN n.level_at < row.when
  THEN row.level ELSE n.level END,
 n.level_at = CASE
  WHEN n.level_at < row.when
  THEN row.when ELSE n.level_at END
··· hp, zone 블록도 같은 모양
```

쿼리 2배

write →

키	값
level	10
level_at	12:05
hp	87
hp_at	12:05
zone	A1
zone_at	12:05

프로퍼티 2배

순서 섞임: DB 안에서 해결하는 방법

엔진 레벨 LWW(Last-Write-Win) 지원

엔진이 비교

참고: 카산드라 방식



프로퍼티마다 최신 값으로 갱신

쿼리 증폭 해소

CLIENT: Cypher

```
MERGE (n {id: row.id})
ON MATCH SET n += row.props
```

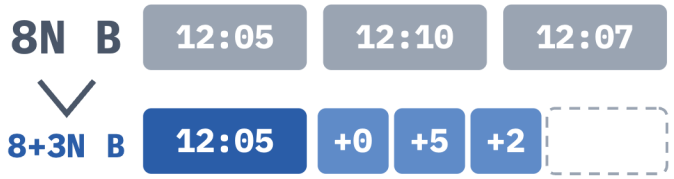
metadata: timestamp = 12:05

write →

쿼리 parsing/plan 효율 ↑

delta 인코딩으로 작게

키	값	시각
level	10	12:05
hp	87	12:10
zone	A1	12:07



05

마무리

Live Memory의 next step

나를 기억하는 AI NPC



aws FOR GAMES

감사합니다

GS Neotek | MEGAZONE CLOUD