Session I : Klaytn Architecture

우준희 / Ground X

Data Layer Architecture & Optimizations

우준희, Melvin

Software Engineer, Ground X

- Klaytn Test-net (Aspen, Baobab) 개발
- Klaytn Main-net (Cypress) 개발
- Performance, Storage 최적화 작업

Software Engineer, SQL Optimizer Team, SAP Labs Korea

- SAP's in-memory database HANA
- SQL Optimizer Engine
    - Join Optimization

Seoul National University

- 전기 컴퓨터 공학 전공, 학사

TABLE OF CONTENTS

01. 메인넷 이전 데이터 영역의 구조 및 문제점

1-1. 메인넷 이전 데이터 영역의 구조

1-2. 메인넷 이전 데이터 영역의 문제

1-3. 메인넷 이전 성능

1-1. 메인넷 이전 데이터 영역의 구조

- 어카운트 데이터를 저장하는 자료 구조

- Merkle Patricia Trie 구조로,
  어카운트 데이터는 Trie의 Leaf Node에 저장
  (= 어카운트 개수가 늘어나면 Trie가 커짐)

- 따라서 어카운트 개수가 늘어나면
  전체 State Trie 를 메모리 상에서 관리 불가
  (= 일부는 디스크 영역에서 관리)

- key-value store DB

- 하나의 LevelDB에 모든 데이터 저장
  e.g., Block, Receipts, State ...

1-2.
메인넷 이전 데이터 영역의 구조의 문제

- 어카운트 데이터를 저장하는 자료 구조

- Merkle Patricia Trie 구조로,
  어카운트 데이터는 Trie의 Leaf Node에 저장
  (= 어카운트 개수가 늘어나면 Trie가 커짐)

- 따라서 어카운트 개수가 늘어나면
  전체 State Trie 를 메모리 상에서 관리 불가
  (= 일부는 디스크 영역에서 관리)

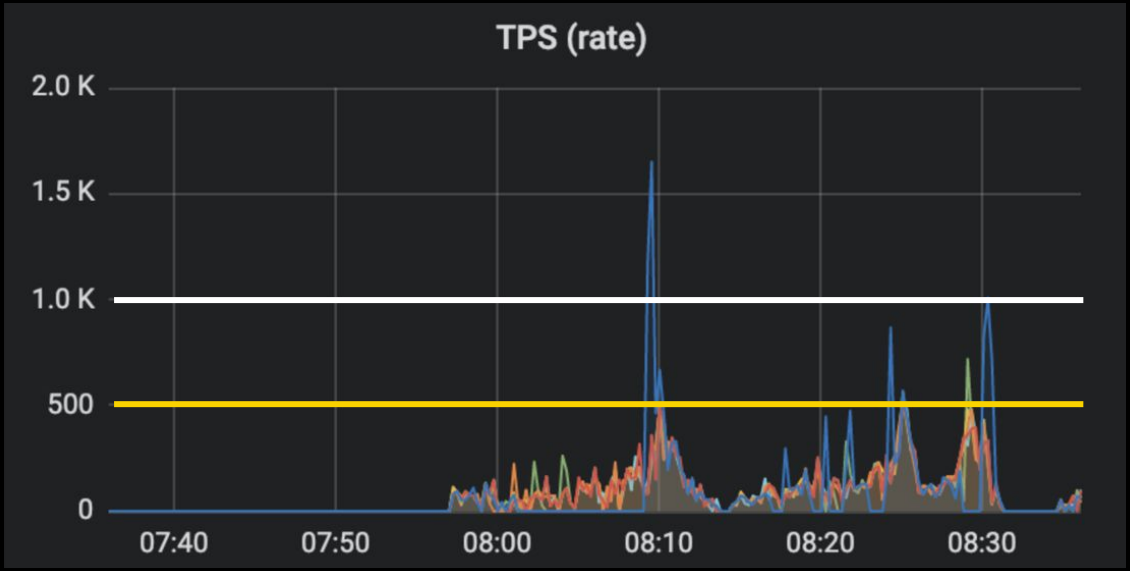
= 필요한 노드가 메모리에 없는 경우,
  디스크 접근 필요.
= 디스크 접근으로 인한 성능 저하.
= State Trie가 커질수록 이러한 문제가 심각해짐.

- key-value store DB

- 하나의 LevelDB에 모든 데이터 저장
  e.g., Block, Receipts, State ...

= 모든 데이터가 하나의 DB에 저장되어 있기 때문에 데이터 종류에 따른 병렬 처리 불가능.

1-3.
메인넷 이전 성능



어카운트 개수가 많아지면(500만개 이상), 평균 500 TPS 미만의 성능을 보임.

02.메인넷 이후 데이터 영역의 구조 및 적용된 최적화 기법

2-1. 메인넷 이후 성능

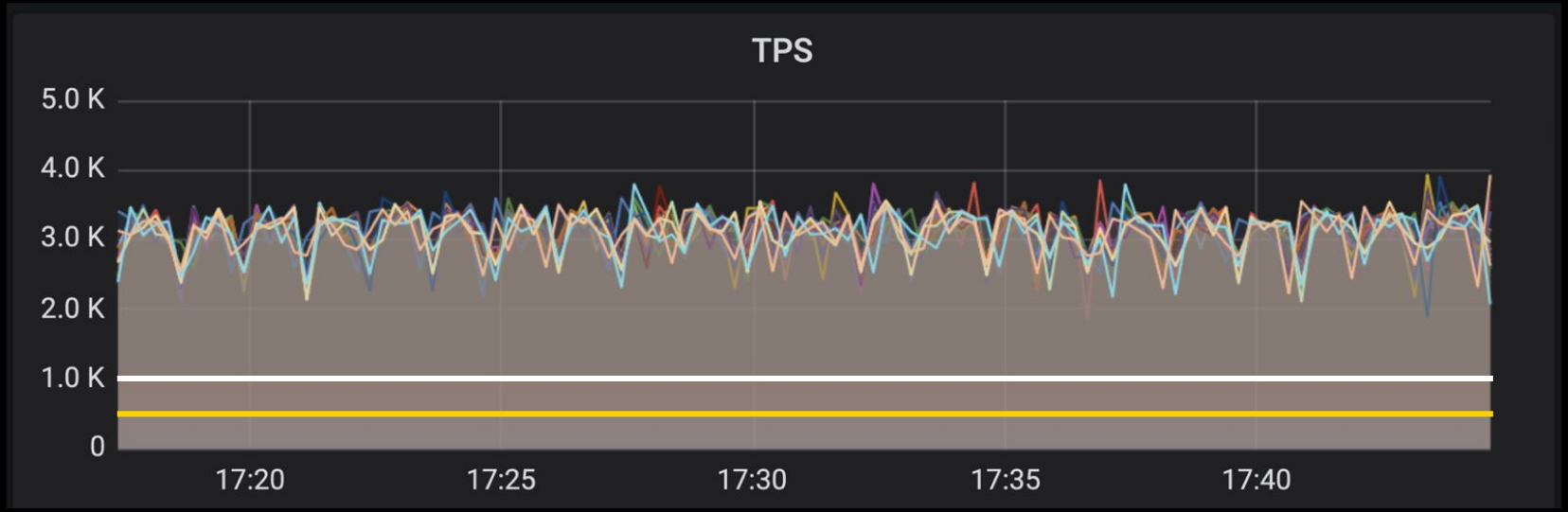
2-2. 메인넷 이후 데이터 영역의 구조

2-3. 메인넷 개발중 적용된 최적화 기법

2-3-1. 캐시

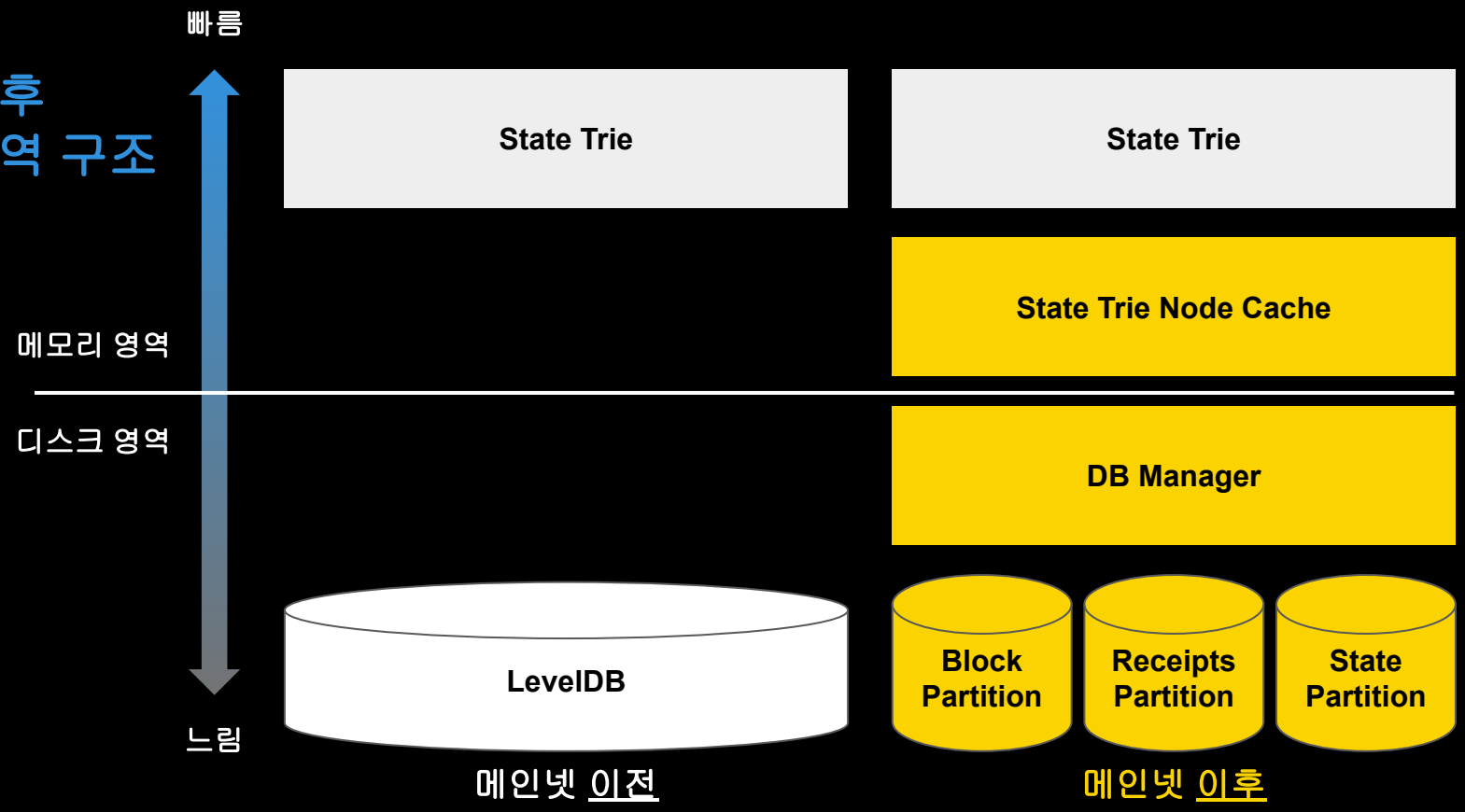
2-3-2. 파티셔닝

2-1.
메인넷 이후 성능

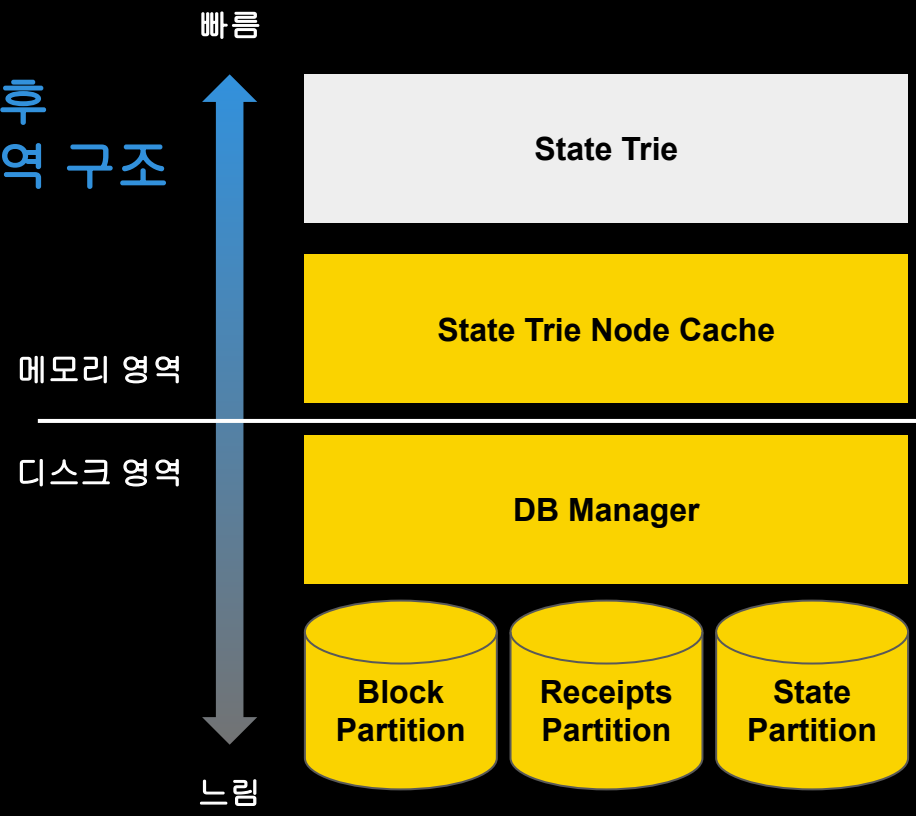


어카운트 개수가 많아도(500만개 이상), 평균 3000 TPS 정도(+2500↑)의 성능을 보임.

2-2.
메인넷 이후
데이터 영역 구조



2-2.
메인넷 이후
데이터 영역 구조



메인넷 이후

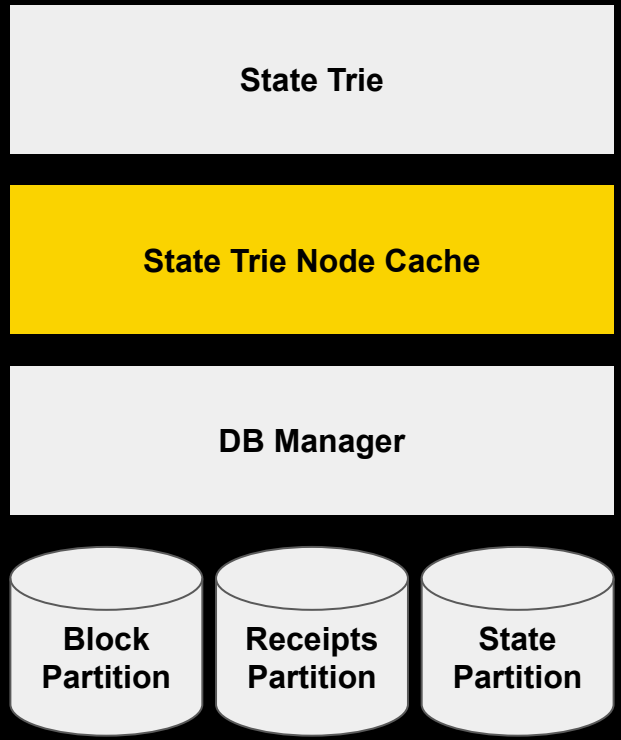
- State Trie 와 DB Manager 사이에서 State Trie 의 Node를 캐시해주는 역할.
- 디스크 영역에 대한 접근을 줄여준다.
- 아래 레벨의 구현을 자유롭게 가져가기 위해 key-value store DB를 바탕으로 구성된 DB 인터페이스.
- key-value store DB
- 데이터 종류에 따라 별개의 DB를 사용해서 병렬 처리를 가능하게 함.
  = 파티셔닝

2-3. 메인넷 개발중 적용된 최적화 기법

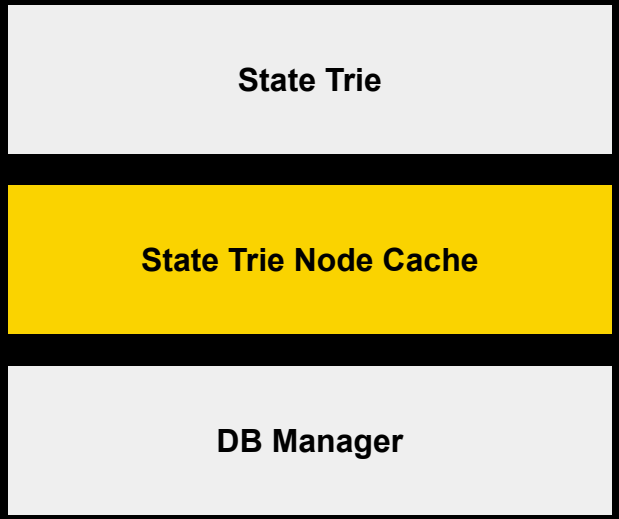
2-3-1. 캐시

2-3-2. 파티셔닝

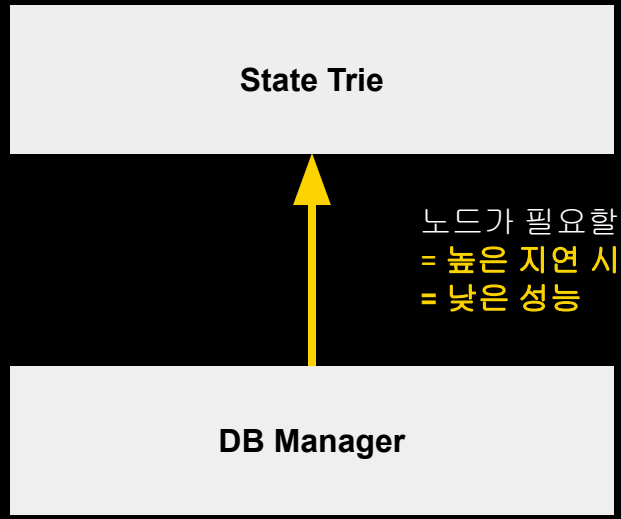
2-3-1.
캐시



2-3-1.
캐시



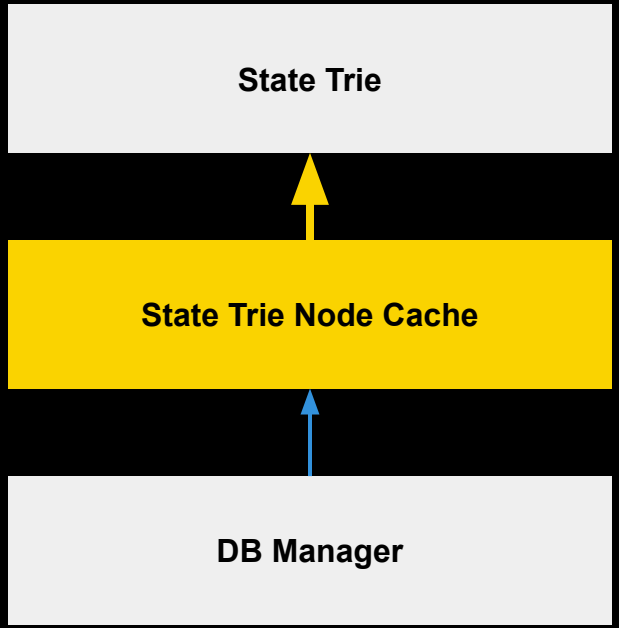
2-3-1.
캐시



노드가 필요할 때마다 디스크 영역에 접근
= 높은 지연 시간
= 낮은 성능

메인넷 이전

2-3-1.
캐시

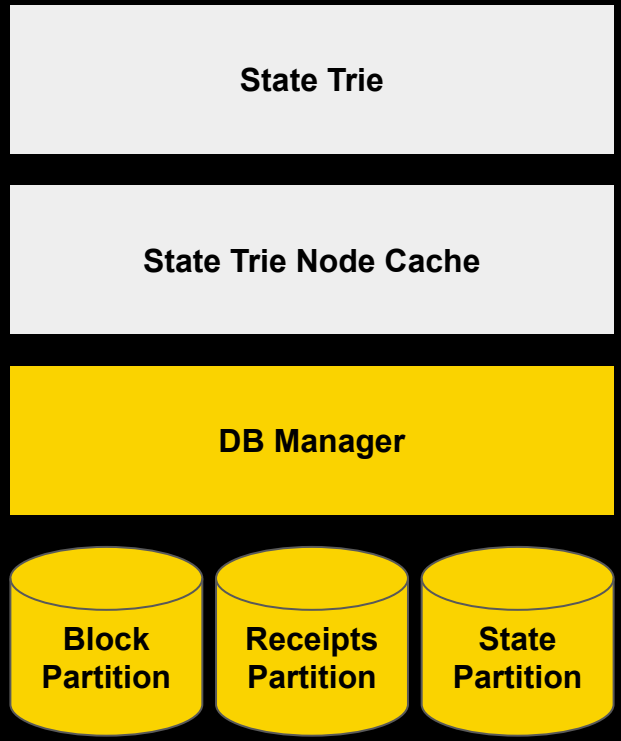


노드가 필요할 때, 대부분 캐시에 접근
= 낮은 지연 시간
= 높은 성능

메인넷 이전에 비해 디스크 접근이 줄어듬.
= 전체어카운트 500만/활성어카운트 100만 상황에서 캐시 미스 발생하지 않음

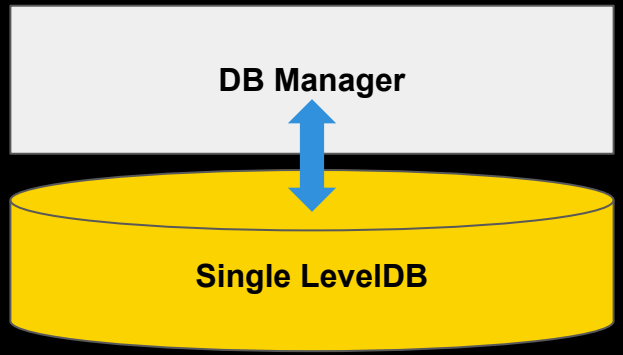
메인넷 이후

2-2.
파티셔닝



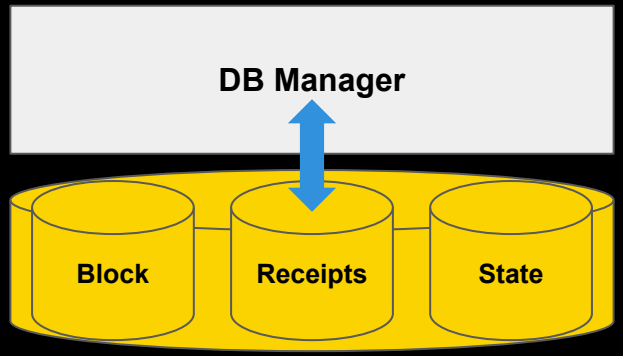
2-2.
파티셔닝

메인넷 이전



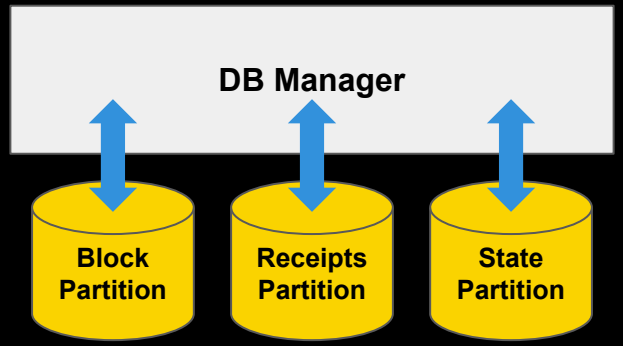
2-2.
파티셔닝

메인넷 이전

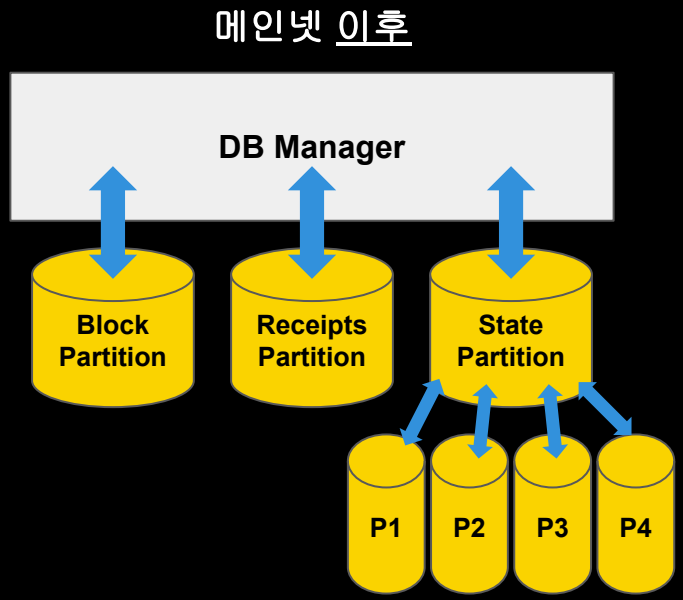


2-2.
파티셔닝

메인넷 이후

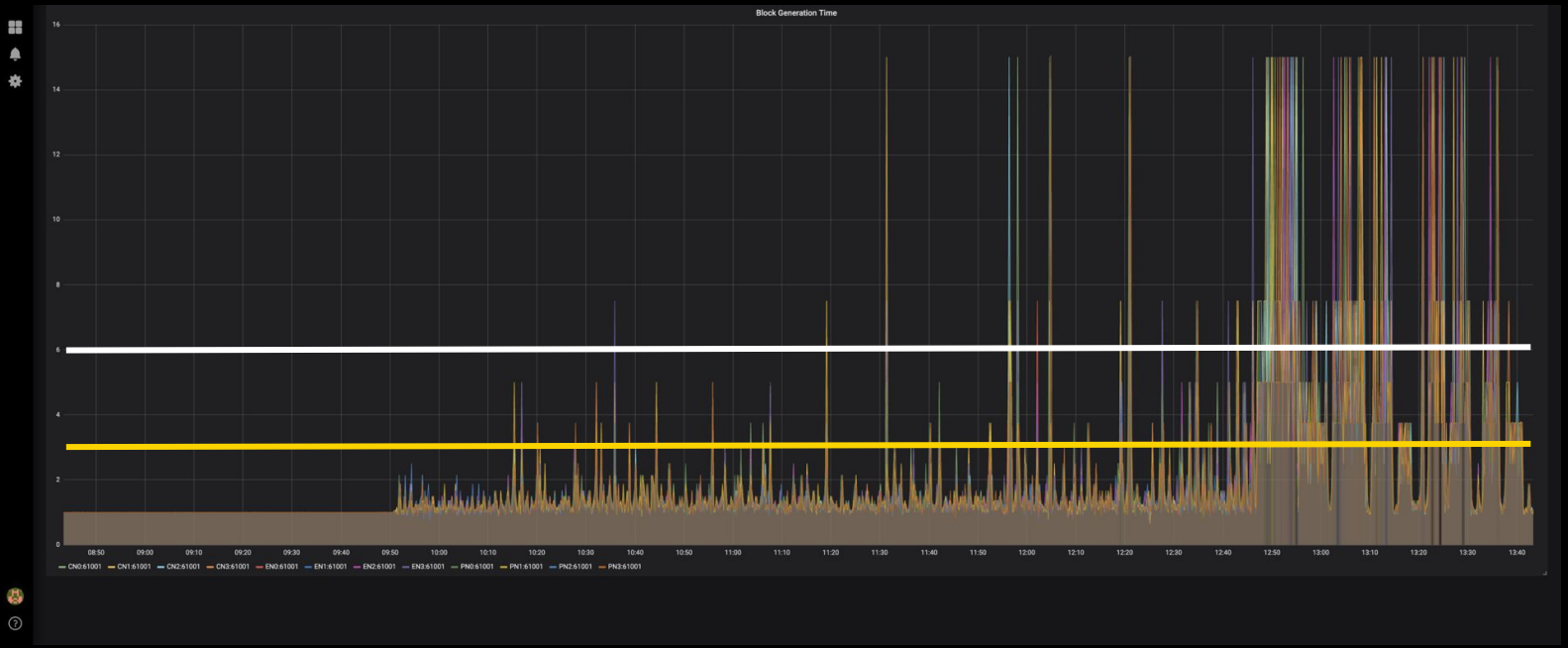


2-2.
파티셔닝



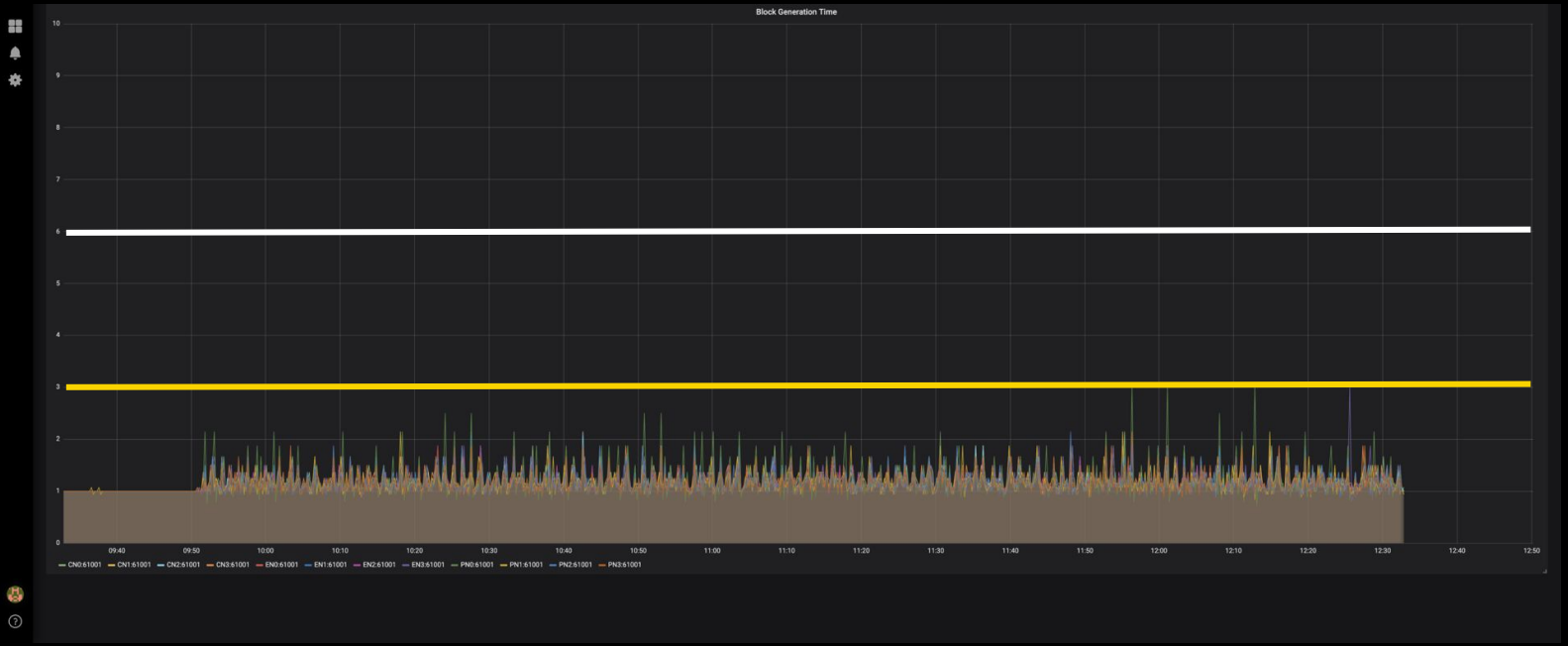
2-2.
파티셔닝

블록 쓰기에 걸린 시간 (**second**) - 파티셔닝 이전



2-2.
파티셔닝

블록 쓰기에 걸린 시간 (second) - 파티셔닝 이후



03.늘어나기만 하는 블록체인 데이터, 줄일 수 있는 방법은?

3-1. 문제

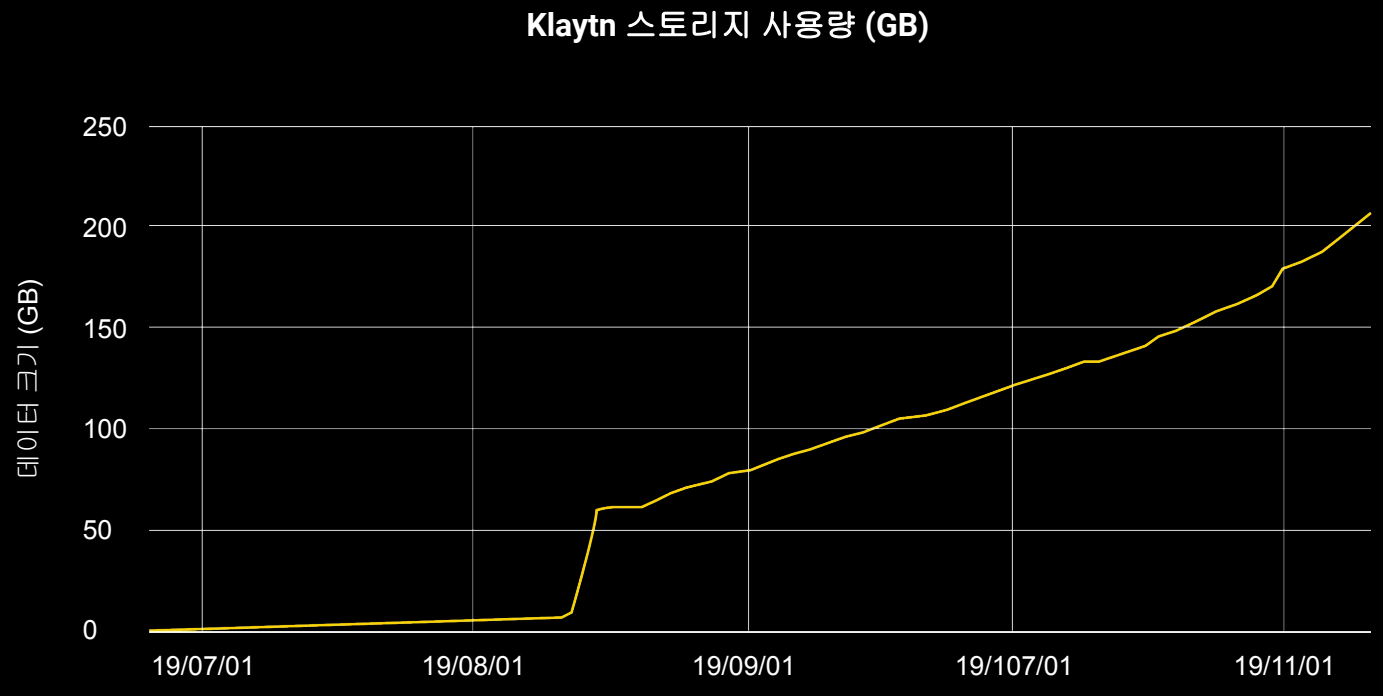
3-2. 해결 방안

3-2-1. 데이터를 줄일 수 있는 방법

3-2-2. 노드가 최신 블록을 따라갈 수 있게 하는 방법

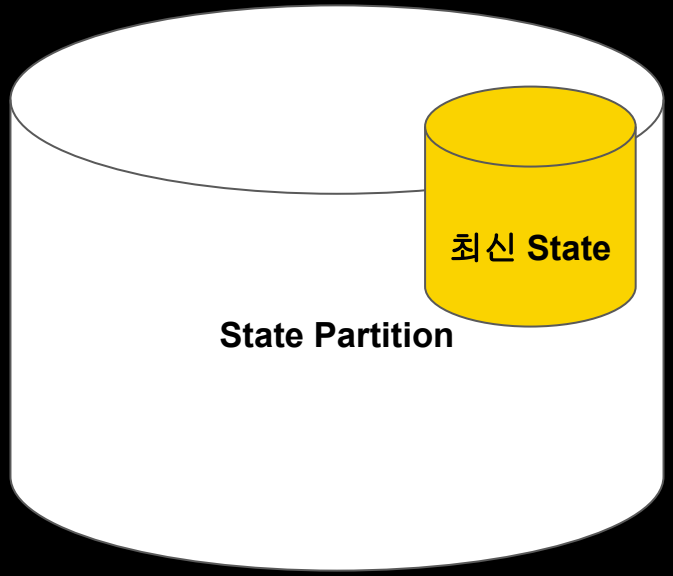
3-3. 앞으로 해결해야 하는 문제

3-1. 문제

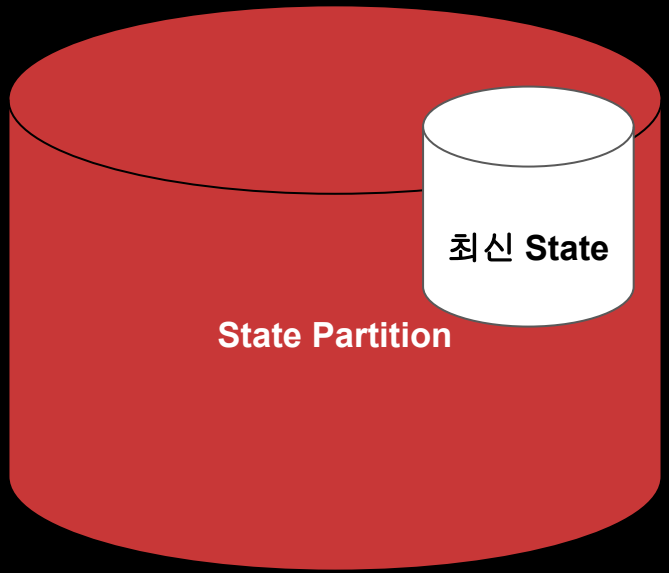


블록체인 특성상, 데이터의 크기는 계속 증가 = 스토리지 비용의 지속적인 증가

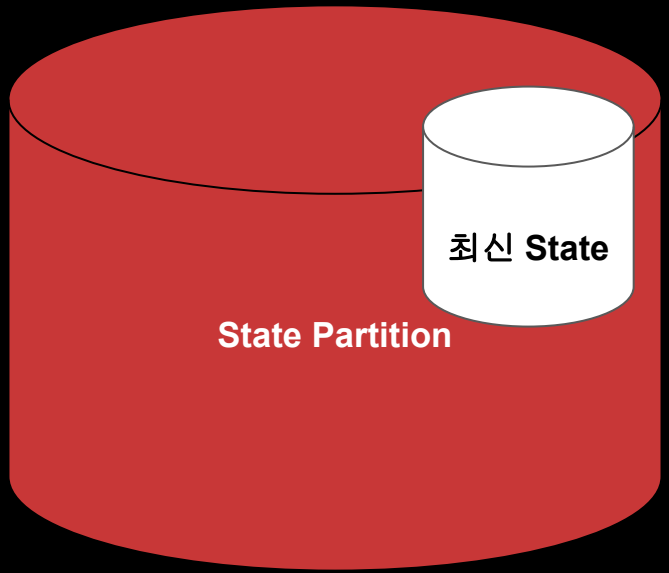
3-1.
문제



3-1.
문제



3-1.
문제



- A의 계정
- 11월 01일 = 100 KLAY
- 11월 08일 = 200 KLAY
- 11월 15일 = 500 KLAY
  = “outdated” State
- 11월 22일 = 250 KLAY
  = “최신” State

3-2.
해결 방안

가정

- “최신” 상태가 아닌 **“지나간” 상태에 대한 요청은 많지 않다**.
- “지나간” 상태에 대한 요청을 처리할 수 있는 노드가 존재한다.

해결해야하는 문제

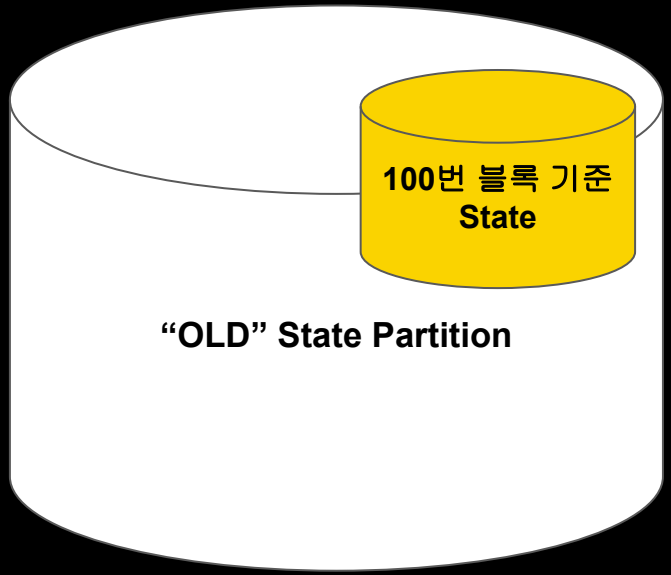
1. **“최신” 상태를 계속 업데이트**하면서 “지나간” 상태 데이터를 삭제해야함.

해결 방안

1. 특정 블록 넘버 기준의 State Trie 를 NEW DB에 쓰고, 이전에 사용하던 OLD DB는 삭제.
2. 최신 상태 업데이트는 NEW DB에만 써줌. (=OLD DB는 read-only)

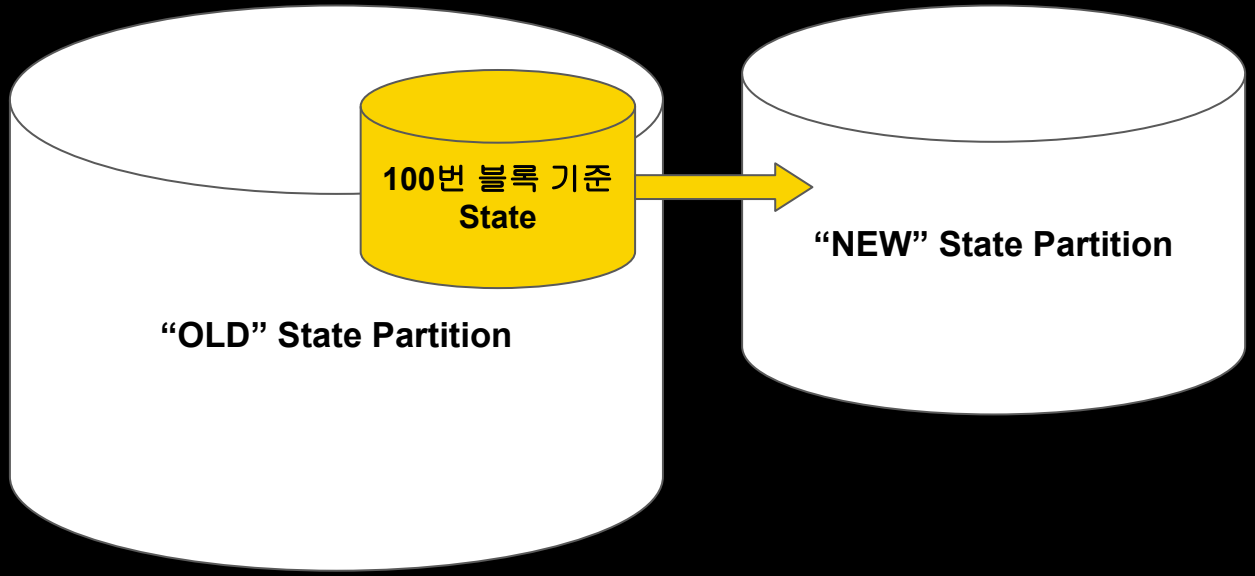
3-2-1.
데이터를
줄일 수 있는 방법

3-2-1.
데이터를
줄일 수 있는 방법

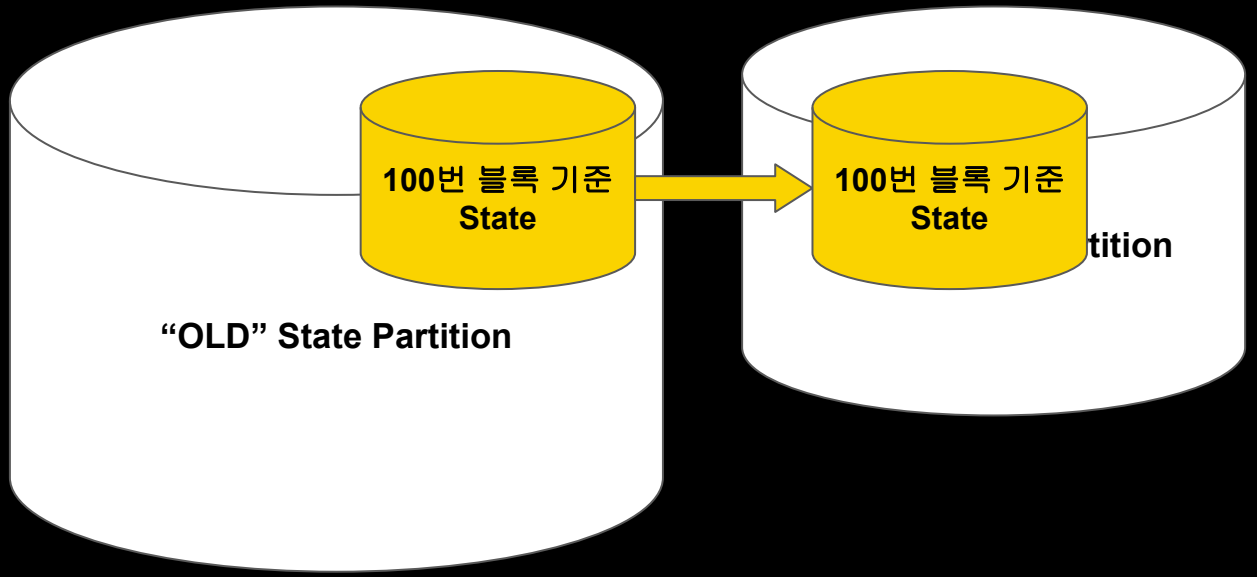


3-2-1.
데이터를
줄일 수 있는 방법

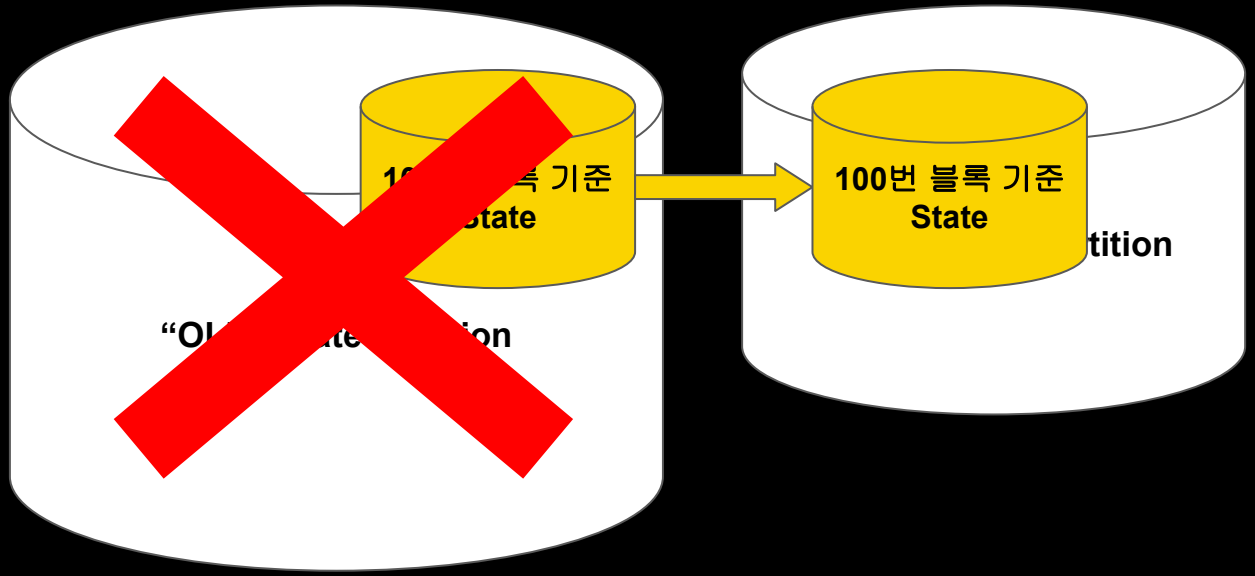
3-2-1.
데이터를
줄일 수 있는 방법



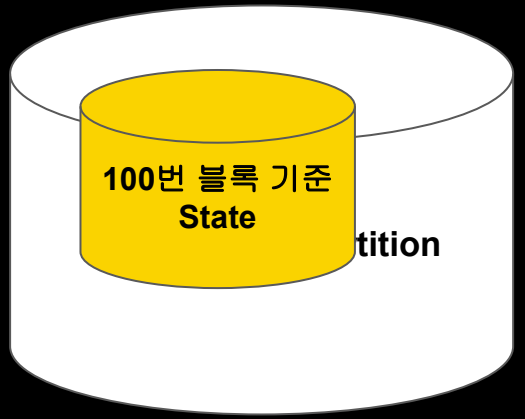
3-2-1.
데이터를
줄일 수 있는 방법



3-2-1.
데이터를
줄일 수 있는 방법

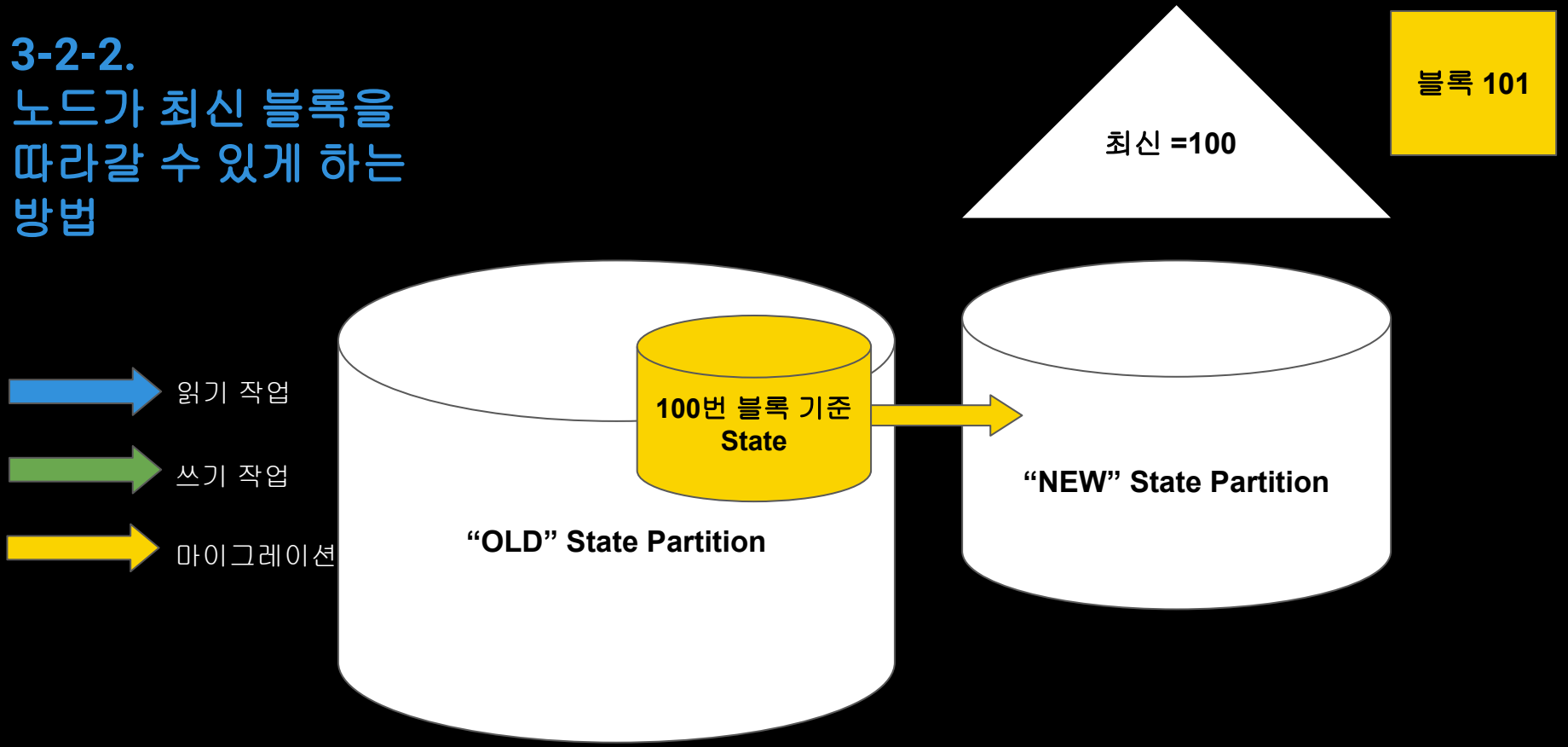


3-2-1.
데이터를
줄일 수 있는 방법

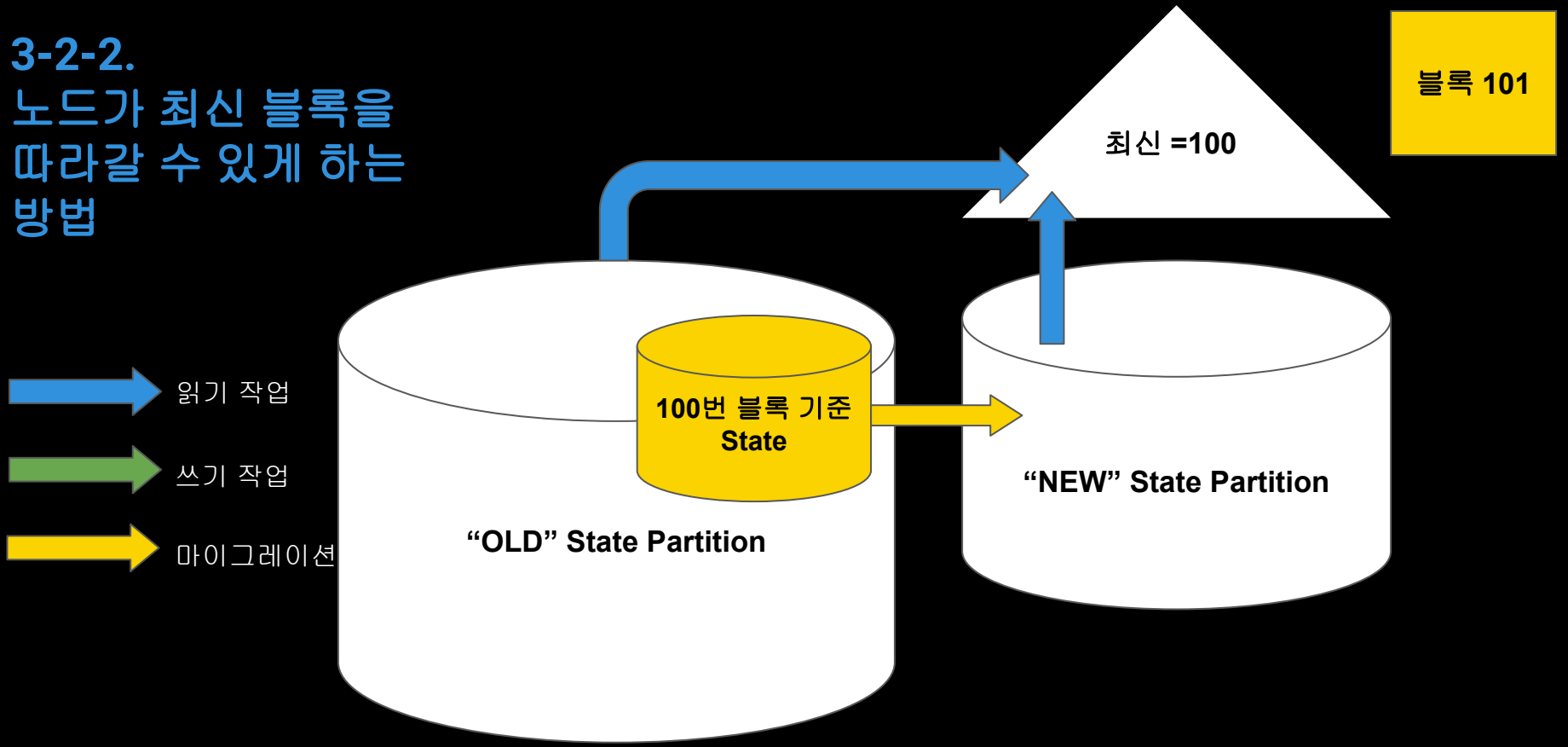


3-2-2.
노드가 최신 블록을 따라갈 수 있게 하는 방법

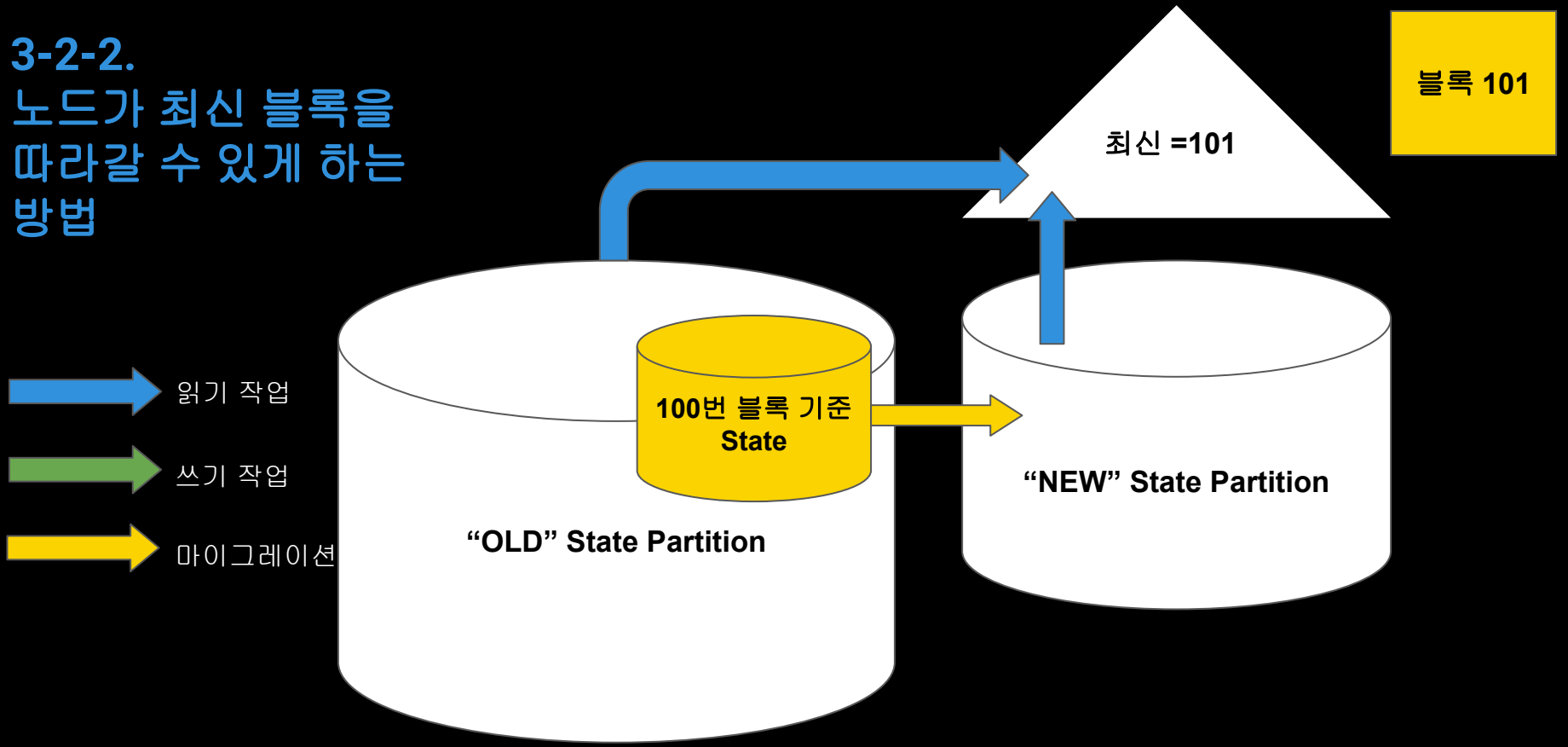
3-2-2.
노드가 최신 블록을 따라갈 수 있게 하는 방법



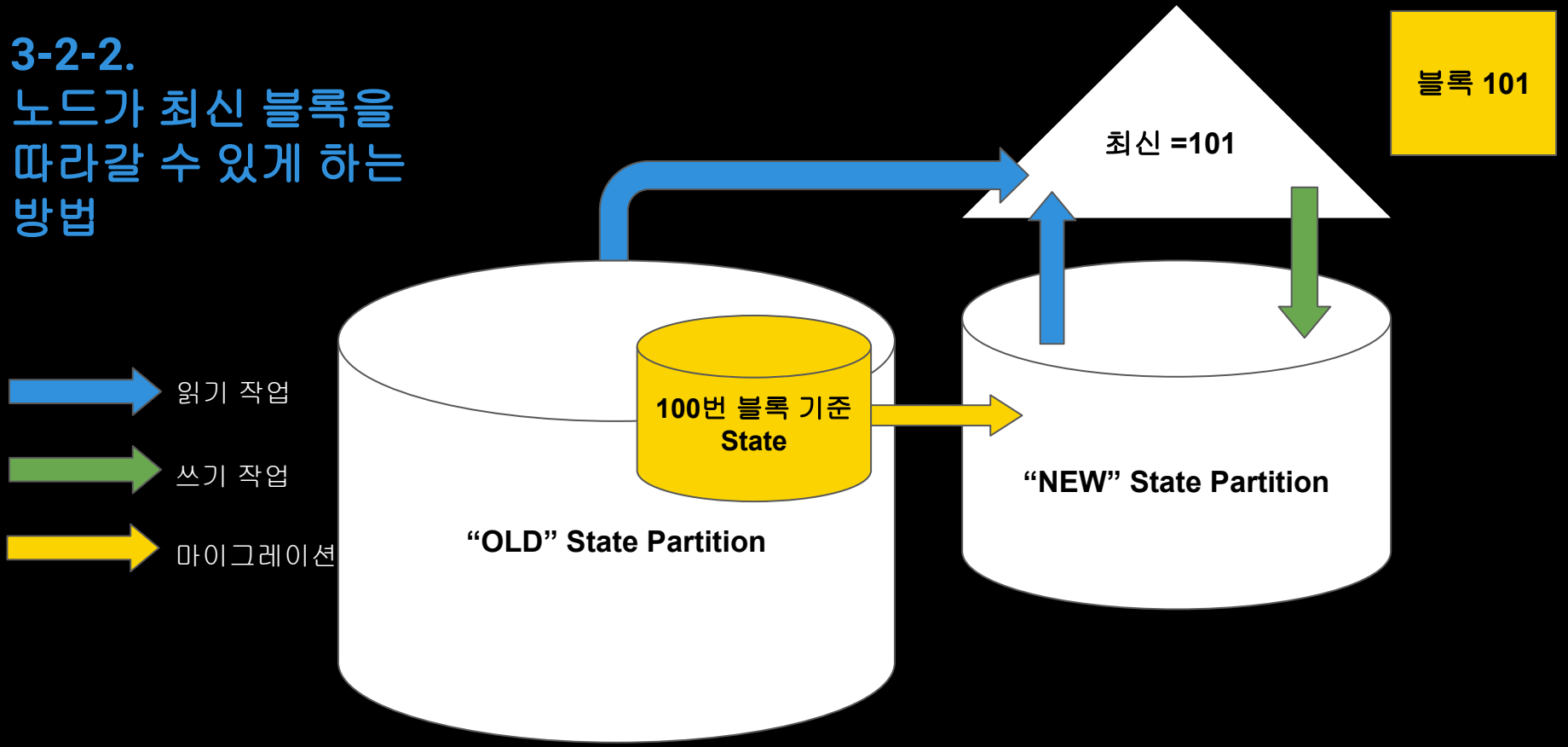
3-2-2.
노드가 최신 블록을 따라갈 수 있게 하는 방법



3-2-2.
노드가 최신 블록을 따라갈 수 있게 하는 방법



3-2-2.
노드가 최신 블록을 따라갈 수 있게 하는 방법



3-3. 앞으로 해결해야하는 문제

리소스 배분

- 마이그레이션에 많은 리소스가 할당되면, 최신 상태 처리에 사용할 수 있는 리소스가 줄어든다.
  반대로 적은 리소스가 할당되면, 마이그레이션 작업에 오랜 시간이 걸리게 된다.
  => 따라서 현재 리소스 사용량을 보고 유연하게 리소스를 관리할 수 있는 시스템 필요.

점점 늘어나는 처리 시간

- 시간이 갈수록 계정 숫자는 증가하기만 하고, 줄어들지 않음.
  = 하나의 마이그레이션 작업에서 처리해야 하는 데이터의 크기는 점점 증가.
- 이를 보완할 수 있는 메커니즘이 필요.

Summary

메인넷 이전에는 데이터 영역의 구조적인 문제가 있었고, 이로 인해서 성능이 낮았음.

- 이전 구조에서는 어카운트 개수가 많을 때, 디스크 영역에 대한 접근이 빈번하게 일어남.
- 데이터가 저장되는 DB가 하나라서, 병렬 처리가 불가능한 구조였음.

메인넷 개발 과정에서 이러한 문제점을 해결할 수 있는 구조를 적용함.

- [1] 캐시와 [2] 파티셔닝을 적용함.
- [1] 캐시를 적용해서, 디스크 영역에 대한 접근을 줄임.
- [2] 파티셔닝을 통해서, 병렬 처리를 가능하게 함.

지속적으로 증가하는 블록체인 데이터의 크기를 줄이기위한 노력을 진행중.

- 과거 데이터에 대한 접근이 필요하지 않은 경우, 이를 삭제할 수 있는 메커니즘을 제공.

THANK YOU

Ground X
27F, 521, Teheran-ro,
Gangnam-gu, Seoul, Republic of Korea