

그라비티(Gravity)

해당 번역본은 이용자가 디지털자산 관련 정보를 편리하게 확인할 수 있도록 단순 참고용으로 번역된 것이며, 투자 권유를 목적으로 하지 않습니다.

그 내용이 정확하지 않을 수 있으므로 정확한 정보 습득을 위해서는 원문을 참고하시거나 원문 작성 주체 측에 문의하시기 바랍니다.

두나무는 해당 번역본 내용의 진실성 및 정확성을 보장하지 않고, 제공된 정보에 의한 투자 결과에 대하여 어떠한 책임도 지지 않으며, 투자 손실은 투자자에게 귀속됩니다.

Gravity 라이트페이퍼

초록

Gravity Chain은 대중적 채택과 옴니체인 미래를 목표로 Galxe가 개발한 고성능 EVM 호환 레이어 1 블록체인입니다. 초당 1기가가스 처리량, 1초 미만의 최종성, 리스테이킹 기반 PoS 보안성을 제공하며, Gravity는 두 가지 주요 오픈소스 구성 요소를 기반으로 구축되었습니다: (1) 리스테이킹 기반 **파이프라인형 AptosBFT PoS** 합의 엔진인 **Gravity SDK**, (2) 병렬 EVM인 **Grevm(Gravity EVM)**으로 구동되는 Block Stream Reactor (BSR) 실행 계층인 **Gravity reth**. 이들은 웹3 애플리케이션이 자체 대안 L1 또는 빠른 최종성을 제공하는 L2를 론칭할 수 있도록 설계되었으며, 특히 EVM 체인에 최적화되어 있습니다.

본 문서는 Gravity의 엔지니어링 설계와 기술적 혁신을 소개하며, Gravity가 어떻게 파이프라인 아키텍처, 최첨단 합의 알고리즘, 병렬 실행, 고도로 최적화된 스토리지 계층의 조합을 통해 고성능 요구를 충족시키는지를 설명합니다. 이는 특히 reth를 확장하고 Aptos의 오픈소스 합의 엔진을 개선함으로써 구현되었습니다.

소개

Gravity의 개발은 Galxe에서 우리가 직면한 과제들로 인해 처음 시작되었습니다. Galxe는 로열티 포인트, 캠페인 NFT, 토큰 보상, zk-아이덴티티, 옴니체인 스마트 저축과 같은 다양한 서비스를 제공하는 선도적인 웹3 애플리케이션입니다. Galxe의 빠른 성장은 거래량의 폭발적인 증가를 초래하였으며, 로열티 포인트 시스템은 초당 51.2건 이상의 거래를, 토큰 보상 캠페인은 평균 초당 32.1건 이상의 거래를 처리하고 있습니다. Galxe의 백엔드를 탈중앙화하는 과정에서, 모든 사용 사례를 EVM 블록체인으로 전환하면서도 최적의 사용자 경험을 유지하는 것은 도전 과제가 되었습니다. 이러한 변화는 (1) 높은 거래 처리량, (2) 거의 즉각적인 최종성을 지원할 수 있는 고성능 EVM 블록체인의 필요성을 부각시켰습니다.

이러한 성능 요구를 고려할 때, 기존 L2 솔루션을 채택할지 아니면 새로운 레이어 1 체인을 개발할지에 대한 결정은 매우 중요합니다. 핵심적인 트레이드오프는 거래 최종성에 도달하는 방식에 있습니다: 합의 알고리즘을 사용하는 경우 이는 L1로 정의되며, 롤업 프로토콜을 사용하는 경우는 L2로 분류됩니다. 이 트레이드오프는 명확합니다—L1은 일반적으로 합의 알고리즘의 비용으로 인해 이론적인 처리량이 낮지만, L2에 비해 훨씬 빠른 최종성 도달 시간을 제공합니다. 예를 들어, AptosBFT와 같은 합의 알고리즘을 사용할 경우 최종성은 1초 미만으로 달성될 수 있지만, 옵티미스틱 롤업의 경우 도전 기간 때문에 최대 7일까지 소요될 수 있습니다. 제로 지식 증명을 사용하여 이 과정을 가속화하더라도 최종성까지의 시간은 여전히 수 시간 단위입니다. Galxe 사용자 경험과 특히 Gravity의 옴니체인 인텐트 프로토콜에 필수적인 거의 즉각적인 최종성이 필요하다는 점에서, 우리는 L1을 개발하기로 결정했습니다.

또한 L2는 Ethereum과의 메시징을 네이티브하게 지원하지만, Gravity와 같은 L1 체인은 Gravity 인텐트 프로토콜과 Ethereum 생태계 참여자들이 구축한 크로스체인 브리지를 통해 이에 상응하거나 더 확장된 상호운용성을 제공할 수 있습니다. 이 접근 방식은 Ethereum과의 원활한 통신을 가능하게 할 뿐만 아니라, 다른 블록체인 네트워크와의 상호운용성도 확장하여 전체 웹3 생태계의 연결성을 강화합니다.

더불어, 리스테이킹 프로토콜의 도입으로 인해 PoS 레이어 1 블록체인의 부트스트래핑은 과거처럼 어렵지 않습니다. EigenLayer 및 Babylon과 같은 리스테이킹 프로토콜을 통합함으로써, L1은 Ethereum 및 Bitcoin에 존재하는 방대한 스테이킹 가치와 그들의 폭넓은 검증자 네트워크를 활용할 수 있습니다. 이러한 리스테이킹 프로토콜이 초기부터 제공하는 경제적 신뢰는 PoS 합의의 확고한 기반이 되며, Gravity가 Ethereum과 유사한 수준의 탈중앙화와 보안을 달성할 수 있도록 합니다.

이러한 여러 고려 사항에 비추어, 우리는 현대 웹3 애플리케이션의 확장성과 성능 요구를 충족시키기 위해 Gravity라는 고성능 EVM 호환 레이어 1 블록체인을 구축하기로 결정했습니다. 비록 Gravity의 개발은 Galxe의 필요에 의해 시작되었지만, Gravity SDK와 Grevm(Gravity EVM)은 Tendermint/Cosmos SDK처럼 대안 L1 및 빠른 최종성을 지닌 L2를 구축할 수 있는 유연한 프레임워크를 제공합니다.

초당 1기가가스 처리량 요구

블록체인의 가장 핵심적인 성능 요건은 처리량이며, 이는 일반적으로 초당 트랜잭션 수(TPS) 또는 초당 가스(gas/s)로 측정됩니다. Galxe의 로열티 포인트 시스템을 예로 들면, 이 시스템이 효과적으로 작동하기 위해서는 최소 **초당 400만 가스**의 처리량이 요구됩니다. 이 추정치는 로열티 포인트 트랜잭션당 평균 가스 사용량인 80,000 가스와, 관측된 초당 트랜잭션 수인 51.2 TPS를 기반으로 하며, 두 수치를 곱하면 총 400만 gas/s가 됩니다.

이 추정치는 실험적 레이어 2인 Gravity Alpha Mainnet에서의 실제 데이터를 통해 추가로 검증됩니다. Gravity Alpha Mainnet 상의 모든 로열티 포인트 트랜잭션은 지속적으로 약 400만 gas/s의 처리량을 보여주었으며, 이는 위의 계산을 확증합니다.

Network	Block	TPS	Mgas/s	KB/s	Stack	DA	Settlement
Gravity	7277685	60.1	4.48	25.05	arbitrum-nitro	anytrust	ethereum
Base	20233922	47.6	10.16	37.75	op-stack	blobs	ethereum
Taiko	410488	26.1	2.11	4.61	taiko	blobs	ethereum
Xai	41502057	20.5	1.00	6.24	arbitrum-nitro	anytrust	arbitrum
ArbitrumOne	257151747	20.1	2.52	10.59	arbitrum-nitro	blobs	ethereum
ProofOfPlay - Apex	48876401	18.5	16.46	12.60	arbitrum-nitro	anytrust	arbitrum
WINR Chain	10509232	9.9	2.90	9.05	arbitrum-nitro	anytrust	arbitrum
OP Mainnet	125829207	8.4	2.34	6.37	op-stack	blobs	ethereum
ProofOfPlay - Boss	8103539	8.2	5.65	5.16	arbitrum-nitro	anytrust	arbitrum
Blast	9223688	8.2	1.57	6.99	op-stack	blobs	ethereum
Ethereum	20826785	7.3	0.74	3.02	ethereum	ethereum	ethereum

이 수요는 온체인 작업에 수반되는 더 높은 비용으로 인해 다소 감소할 수 있지만, Galxe의 성장 궤적을 고려하면 피크 시점에는 현재의 두세 배 수준까지 수요가 증가할 가능성이 충분합니다. 또한, NFT, 토큰 보상, 그리고 제로 지식 증명 기반의 완전 온체인 퀘스트 기능과 같은 향후 온체인 기능들을 포함할 경우, Galxe가 완전히 온체인 DApp이 된다면 블록체인은 **최소 초당 5천만 가스 처리량**을 지속적으로 감당할 수 있어야 합니다. 만약 Gravity 체인 상의 애플리케이션 가스 사용량이 파레토 분포를 따른다고 가정하면—이는 Uniswap이 지속적으로 Ethereum 전체 가스 사용량의 10%를 소비하는 방식과 유사합니다—체인은 전체 생태계 애플리케이션을 수용하기 위해 **이상적으로는 초당 5억 가스 처리량**을 지원해야 합니다.

이러한 애플리케이션에는 스마트 세이빙 LSD의 크로스체인 결제, 로열티 포인트 거래용 DEX, NFT 마켓플레이스 등이 포함됩니다. 이러한 추정을 종합하면, 생태계 내 애플리케이션의 수요를 충족하기 위해서는 **초당 1기가가스(gigagas)의 처리량을 갖춘 EVM 블록체인**이 필요함이 명확해집니다. 이러한 성능 상한선을 설정함으로써, 더 많은 자원을 요구하는 애플리케이션도 체인의 성능 한계에 제약받지 않고 확장할 수 있게 됩니다.

이러한 처리량을 달성하기 위한 핵심 요소 중 하나는 **병렬 EVM**입니다. 우리는 **Grevm**이라는 병렬 EVM 실행 런타임을 개발했으며, 이는 현재까지의 벤치마크 기준으로 **가장 빠른 오픈소스 병렬 EVM 구현체**입니다 (자세한 내용은 아래 Grevm 섹션 참고).

1초 미만의 최종성

처리량과 더불어, 블록체인의 **최종성 시간**은 긍정적인 사용자 경험을 유지하는 데 있어 핵심 요소입니다. 사용자들은 웹2 애플리케이션에서 경험하는 것처럼, 중앙화된 백엔드 수준의 거의 즉각적인 반응을 기대합니다. 이러한 점에서 Galxe의 요구사항은 완전히 온체인으로 운영되는 게임들과 유사하며, 낮은 지연(latency)이 중요하다는 점에서 일치하지만 그 정도는 덜 극단적입니다. 현재의 EVM 블록체인들은 수 초에서 수 일에 이르는 **최종성 도달 시간(time-to-finality)**을 가지고 있으며, 이는 이러한 기대를 충족시키기에 턱없이 부족합니다. 우리는 이러한 **1초 미만의 최종성**을 달성하기 위한 합의 알고리즘으로 **AptosBFT**를 선택했습니다.

L2 롤업은 이론적으로는 합의를 제거함으로써 더 높은 처리량을 달성할 수 있지만, 일반적으로 도전 기간(challenge period)으로 인해 트랜잭션의 최종화에 상당한 지연이 발생합니다. 이러한 지연은 Galxe와 같이 거의 즉각적인 최종성이 요구되는 애플리케이션에는 문제가 됩니다. 일부 DApp은 이러한 문제를 완화하기 위해 “신뢰(trust) 모드”를 사용하거나 외부 모니터링 시스템(예: 시퀀서 복제본을 실행하여 불변성

검증)을 구현하고 도전 기간을 우회하려 시도합니다. 하지만 이러한 접근은 **추가적인 위험성과 복잡성**을 초래하며, 보안이 중요한 애플리케이션에는 이상적인 방식이 아닙니다. L2 롤업과 L1 사이의 처리량 격차를 줄이기 위해, Gravity SDK는 합의 알고리즘 도입 이후 **5단계 파이프라인**을 구현하여, 다음 블록의 합의와 현재 블록의 실행을 병렬로 수행합니다 (자세한 내용은 파이프라이닝 섹션 참조).

리스테이킹 기반 PoS 보안

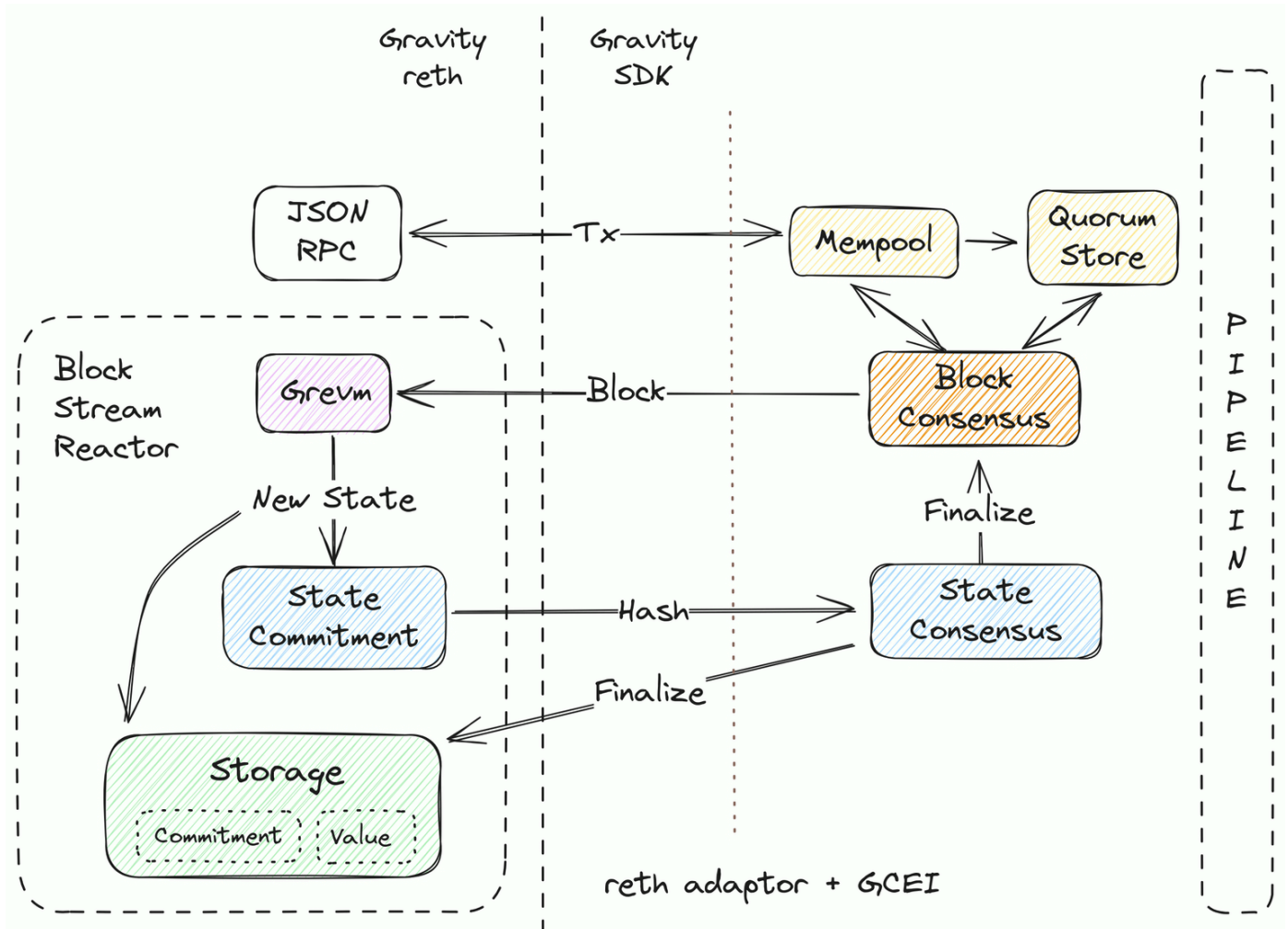
Ethereum의 확장은 L2 롤업에만 국한되지 않습니다. Gravity SDK는 **보안성, 처리량, 최종성 도달 시간, 상호운용성** 간의 균형을 고려하여, 의도적으로 **리스테이킹 기반 보안의 L1 아키텍처**를 선택합니다. 이 전략의 핵심은 EigenLayer 및 Babylon과 같은 여러 리스테이킹 프로토콜을 통합하는 **리스테이킹 모듈**입니다. 이들 프로토콜은 강력한 PoS 합의 체계를 부트스트랩하고 지속적으로 운영하는 데 필요한 경제적 신뢰 (economic trust)를 제공합니다.

Gravity와 Gravity SDK 기반 체인들은 이미 확립된 네트워크에서 **프로그래머블한 경제적 신뢰**를 활용함으로써 다음을 가능하게 합니다:

- (1) EigenLayer 기반의 Ethereum 위에서 **AVS(Actively Validated Service)**를 구축함으로써, Ethereum의 450억 달러 규모의 스테이킹 자산 및 85만 명의 검증자 네트워크에 접근하고,
- (2) Babylon 통합을 통해 **EOTS(추출 가능한 일회용 서명)**과 같은 암호학적 원시 구성요소를 사용하여 **Bitcoin의 6천억 달러 규모 자산**을 활용할 수 있습니다.

이러한 접근은 **Ethereum 및 Bitcoin과 같은 확립된 네트워크의 보안을 확장**하여, PoS 합의를 위한 안전한 기반을 초기에 제공하고, 이는 새로운 PoS 블록체인이 흔히 직면하는 부트스트래핑 문제, 장기 공격, 단일 자산 의존으로 인한 지속 가능성 리스크 등을 효과적으로 해결합니다.

Gravity 체인 아키텍처



Gravity 체인은 두 가지 주요 구성 요소인 **Gravity SDK**와 **Gravity reth**를 기반으로 구축되었습니다. Gravity SDK는 Aptos 체인에서 개선된 블록체인 프레임워크로, 이는 **PBFT** 계열 합의 알고리즘을 사용하는 최첨단 PoS 블록체인입니다. 이 프레임워크는 파이프라인 아키텍처를 통해 처리량과 자원 활용도를 극대화합니다. Gravity reth는 reth를 기반으로 한 파이프라인형 실행 계층으로, **Block Stream Reactor (BSR)**로 동작하며, 합의 계층에서 제안된 블록들을 소비합니다.

reth를 혁신적으로 확장함으로써, 병렬 실행, 일괄 및 비동기 병렬 상태 커밋 계산, 스토리지 최적화에 최적화되어 있습니다.

이 두 구성 요소는 reth 어댑터를 사용하는 **Gravity Consensus Engine Interface (GCEI)**를 통해서 연결되며, 파이프라인 컨트롤러에 의해 원활하게 관리됩니다. 이 컨트롤러는 실행 계층과 합의 계층으로부터 오는 다양한 **역압(backpressure)** 신호에 따라 각 단계의 속도를 동적으로 조절합니다.

이 설계에서는 **블록 실행**과 **블록 합의**가 분리되어 있으며, 실행 계층은 제안된 블록을 소비하는 구조입니다. 우리는 reth를 파이프라인 블록 제안 프로세스에 적합하도록 최적화하였으며, 이 과정은 Block Stream Reactor (BSR)에 의해 관리됩니다.

Gravity 체인에서의 트랜잭션 흐름은 다음과 같습니다:

1. 트랜잭션은 먼저 Gravity reth의 **JSON RPC**, 즉 완전히 Ethereum 호환 JSON-RPC 엔드포인트에 도달합니다.
2. 해당 트랜잭션은 Gravity SDK의 **mempool**로 전달되며 네트워크 전반에 전파됩니다. 검증자들은 트랜잭션을 배치하고 **Quorum Store (QS)** 인증서를 생성하려 시도합니다.

3. 해당 라운드의 **리더(Leader)**는 mempool과 Quorum Store에서 선택된 정렬된 트랜잭션 및 블록 메타데이터를 포함한 **새로운 블록 제안**을 제출합니다.
4. 블록 제안의 상태가 **정렬됨(ordered)**으로 전환되면, 이는 **실행 계층으로 전달**됩니다.
5. 실행 계층에서는 **Grevm이 트랜잭션을 병렬로 실행**하고, 실행 결과를 생성합니다. 생성된 새로운 상태는 **상태 커밋 모듈(state commitment)** 및 **다중 버전 상태 관리자**로 전달됩니다.
6. **상태 커밋 박스**는 **상태 루트(state root)**를 계산하고, 이를 **상태 합의 엔진**에 전달하며, 해당 엔진은 상태 루트에 대해 합의에 도달하려 시도합니다.
7. 상태 루트가 **최종화(finalized)**되면, 실행 스토리지는 **해당 상태 루트 및 블록 데이터를 영속 저장**하라는 알림을 받습니다.

우리는 다음 섹션들에서 각 구성 요소의 세부사항을 자세히 살펴볼 것입니다.

Gravity SDK: 세계 최초의 오픈소스 파이프라인 블록체인 SDK

Gravity SDK는 세계에서 가장 실전 배치된 블록체인인 Aptos를 기반으로 구축된 **오픈소스 모듈형 블록체인 프레임워크**입니다. 이 SDK는 Aptos 블록체인의 기존 아키텍처를 모듈화하도록 설계되었으며, **Quorum Store가 포함된 mempool, AptosBFT 합의 엔진 등 실전 검증된 구성 요소**를 차용하여 세계 최초의 **파이프라인 블록체인 SDK**를 구현했습니다.

Gravity SDK를 Aptos 기반으로 설계한 결정에는 다음과 같은 주요 이유들이 있습니다:

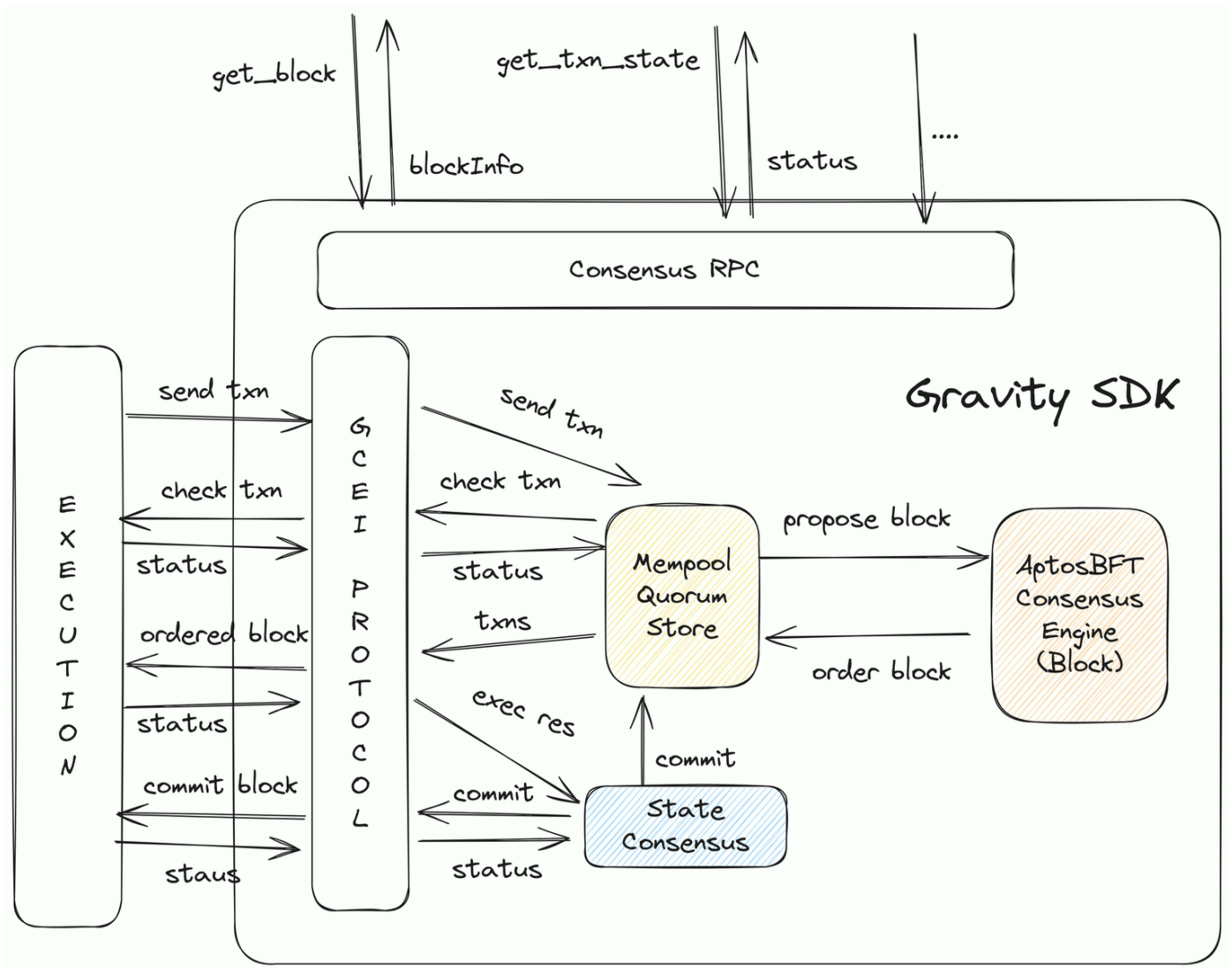
- **최첨단 블록체인 기반:** Aptos는 **PBFT 계열 합의 알고리즘**을 사용하는 최첨단 PoS 블록체인입니다. [Order Votes \(AIP-89\)](#)을 도입함으로써, AptosBFT는 **합의 지연(latency)**을 **3 홉(hop)**으로 줄였으며, 이는 **BFT 기반 합의 프로토콜이 이론적으로 달성 가능한 최적 한계**입니다.
- **극한의 성능 요구에 최적화된 퍼포먼스:** Aptos는 성능 최적화를 위해 정밀하게 설계되어, **초당 약 160,000 트랜잭션의 처리량**과 **1초 미만의 최종성**을 달성했습니다.
- **실전 검증된 신뢰성:** Aptos는 실제 프로덕션 환경에서 이미 배포되어 **신뢰성과 강인성**을 입증했으며, **높은 부하를 안정적으로 처리**할 수 있는 능력을 보여주었습니다.
- **빠르고 지속적인 혁신:** Aptos는 탁월한 속도로 지속적으로 진화하고 있습니다. 지난 2년 동안 **100개 이상의 Aptos 개선 제안서(AIPs)**가 제안되고, 논의되며, 구현되어 Aptos 메인넷에 배포되었습니다. 이러한 **끊임없는 개선 노력**은 Aptos를 블록체인 기술의 최전선에 위치하게 하며, 이는 Gravity SDK 및 그 위에 구축된 체인들의 개발을 **지속적으로 이끄는 기준점(North Star)**으로 작용합니다.
- **재발명의 회피:** Aptos를 기반으로 구축함으로써, 우리는 **성숙하고 잘 검증된 기반을 그대로 활용**할 수 있으며, 처음부터 새로 시작함으로써 발생하는 **불필요한 복잡성과 리스크**를 **피할 수** 있습니다. 일부 프로젝트들은 Aptos를 능가하기 위해 '바퀴를 새로 발명'하려 하지만, 이들은 **이론적 근거도 부족하고 설득력 있는 혁신도 결여**되어 있습니다.
- **시너지 기반의 진화:** Aptos는 **랜덤니스 API** 등 최첨단 기능을 지속적으로 도입하며 진화하고 있는 프로젝트입니다. Gravity SDK는 Aptos와 긴밀히 통합되어 이러한 혁신이 자연스럽게 반영되며, 예를 들어 Gravity Chain과 같이 **Gravity SDK 기반 체인들이 Aptos와 시너지를 형성**하게 됩니다. 반대로, Gravity SDK 역시 Aptos의 구조를 모듈화하고 **리스테이킹 기반 PoS 보안 모듈**을 도입함으로써 **Aptos에 기여**합니다.

Gravity SDK 기반으로 구축된 블록체인들은 **파이프라인 합의 엔진**과 상호작용하기 위해 **Gravity Consensus Engine Interface (GCEI)** 를 사용합니다. 이 인터페이스는 어떠한 실행 계층과도 호환 가능하도록 설계되었으며, Gravity SDK는 기본적으로 **Gravity reth** 중심의 지원을 제공합니다. 다음 섹션에서는 GCEI에 대해 더 자세히 살펴보겠습니다.

Gravity Consensus Engine Interface (GCEI)

GCEI(Gravity Consensus Execution Interface) 프로토콜은 합의 계층과 실행 계층 간의 통신 브리지 역할을 합니다.

이 프로토콜은 두 계층 간의 상호작용을 표준화하여, **파이프라인 컨트롤러**에 의해 합의 및 실행 프로세스가 적절하게 동기화되도록 보장합니다.



전통적인 블록체인 SDK와 Gravity SDK의 가장 큰 차이점은 **파이프라인형 합의 엔진**입니다. 실행 계층은 **Block Stream Reactor** 형태로 구현되어야 하며, 이는 곧 **제안된 블록 스트림을 지속적으로 소비할 수 있어야** 함을 의미합니다. 또한, 상태 커밋(state commitment)은 **트랜잭션 실행과는 비동기적으로** 계산되어야 합니다. 추가로, 실행 계층은 **합의 계층으로 역압(backpressure) 신호를** 제공할 수 있어야 하며, 이에 따라 블록 제안 속도가 동적으로 조절됩니다.

또한 Gravity SDK의 파이프라인 특성으로 인해, 실행 계층은 **제안된 블록 내 실행 불가능한 트랜잭션들을 처리할 수 있어야** 합니다. 이는 mempool이 최신 월드 상태(world state)에 접근할 수 없기 때문에, **어떠한 트랜잭션의 유효성도 완전히 확인할 수 없기** 때문입니다. 실행이 아직 완료되지 않았을 수 있기 때문입니다. 또한 실행 결과는 **향후 블록 생성 과정 자체를 차단해서는 안 됩니다**. 이는 Gravity SDK가 블록 합의와 상태 합의를 병렬화함에 따라, 실행 계층은 **제안된 블록 스트림에 반응하는 리액터(reactor)**가 되어, **실행 결과를 더 나중에 반환할 자유**를 가지게 되었기 때문입니다.

GCEI 프로토콜 명세는 두 가지 API 집합을 정의합니다:

- **합의 계층 API (Consensus Layer APIs):** 이 API는 Gravity SDK에 의해 구현되며, 실행 계층이 합의 엔진이 제안한 블록에 반응하고, **상태 커밋을 제출하는 데** 사용됩니다.
- **실행 계층 API (Execution Layer APIs):** 이 API는 실행 계층이 구현해야 합니다. 합의 엔진은 이 API들을 사용하여, 블록에 트랜잭션을 포함시키기 전에 **가능한 수준의 사전 유효성 검사(best-effort validation)**를 수행하고, **제안된 블록을 스트리밍하며, 최종 상태 커밋을 실행 계층에 알립니다**.

트랜잭션의 라이프사이클 관점에서 볼 때, GCEI 프로토콜은 다음 사항들을 정의합니다:

1. **check_txn** (실행 계층 API)

- **입력(Input):** 트랜잭션(GTxn)을 입력으로 받습니다.
- **출력(Output):** 트랜잭션의 **보낸이 주소, nonce, 가스 한도**를 반환합니다.
- **용도(Usage):** 이 메서드는 **합의 엔진이 블록에 트랜잭션을 제안하기 전, 최대한의 유효성 검사(best-effort validation)**를 수행하기 위해 사용됩니다. 이 메서드는 동일한 트랜잭션에 대해 **여러 번 호출될 수 있으며**, 예를 들어 트랜잭션이 mempool에 들어올 때, 블록에 제안되기 직전, 상태 커밋이 최종화될 때 호출될 수 있습니다.

2. **submit_txn** (합의 계층 API)

- **입력(Input):** 실행 계층으로부터 트랜잭션(GTxn)을 입력으로 받습니다.
- **출력(Output):** **Result<()>**를 반환하며, 트랜잭션이 **mempool에 성공적으로 추가되었는지 여부**를 나타냅니다.
- **용도(Usage):** 실행 계층은 이 메서드를 사용하여 **트랜잭션을 mempool에 제출할 수** 있습니다. 이후 합의 엔진은 트랜잭션을 네트워크 전반에 **gossip 방식으로 전파**하고, 일정량의 트랜잭션을 수신하면 **Quorum Store를 형성**합니다.

3. **recv_ordered_block** (실행 계층 API)

- **입력(Input):** 정렬된 트랜잭션과 블록 메타데이터를 포함하는 **ordered_block(BlockBatch 타입)**을 입력으로 받습니다.
- **출력(Output):** **Result<()>**를 반환하며, **실행 계층이 블록을 수신하고 수락했는지 여부**를 나타냅니다.
- **용도(Usage):** 합의 엔진이 블록을 제안한 후, 해당 블록은 **트랜잭션 실행을 위해 실행 계층으로 전송**됩니다. 이 메서드는 실행 계층이 **정렬된 블록을 수신하고 처리할 수 있도록** 합니다.

4. **update_state_commitment** (합의 계층 API)

- **입력(Input):** 특정 블록 번호의 상태 커밋(StateCommitment)을 입력으로 받습니다.
- **출력(Output):** **Result<()>**를 반환하며, **상태 커밋이 로컬 합의 엔진에 성공적으로 수락되었는지 여부**를 나타냅니다.

- **용도(Usage):** 실행 계층이 상태 커밋을 계산한 후, 이는 합의 계층으로 전송되어 **최종화 과정**, 즉 다른 검증자들과의 **$2f+1$ 경량 합의**에 도달하게 됩니다. 만약 상태 커밋 합의가 제안된 블록과 점점 괴리된다면, 파이프라인 컨트롤러는 **블록 제안 속도를 조정**하게 됩니다.

5. **commit_block_hash** (실행 계층 API)

- **입력(Input):** 커밋 대상인 블록들을 나타내는 `block_ids` 벡터를 입력으로 받습니다.
- **출력(Output):** `Result<>` 를 반환하며, **작업의 성공 또는 실패 여부**를 나타냅니다.
- **용도(Usage):** 상태 커밋이 **최종화된 후**, 합의 계층은 실행 계층에 **블록 해시를 블록체인 스토리지에 커밋하라고 알림**을 보냅니다.

블록체인 파이프라인

Gravity SDK는 더 높은 처리량과 더 낮은 지연 시간을 위해, **5단계 파이프라인 아키텍처**를 활용하여 **하드웨어 자원 활용도를 극대화**합니다. 이 파이프라인은 서로 다른 블록에 걸쳐 작업 실행을 **상호 교차(interleave)** 하며, 파이프라인 매니저는 **피드백 메커니즘**을 활용해 체인이 **일정한 속도로 진행되도록** 보장합니다. 첫 세 단계는 **합의 계층(consensus layer)**에 속하고, 마지막 두 단계는 **실행 계층(execution layer)**에 속합니다.

각 단계는 다음과 같이 설명됩니다:

- **1단계: 트랜잭션 전파(Transaction Dissemination):** 이 단계에서는 트랜잭션을 검증자들 사이에 효율적으로 전파하여, 블록 생성 시 **시의적절하고 신뢰할 수 있는 포함**이 가능하도록 합니다. 해당 설계는 트랜잭션 전파와 합의 메커니즘을 분리하며, [Narwhal & Tusk](#) 및 [Aptos](#)에서 제안된 아이디어를 따릅니다. 즉, 검증자들은 **트랜잭션 배치(batch)**를 지속적으로 공유하며, **모든 네트워크 자원을 병렬로 활용**합니다. 하나의 배치가 **$2f + 1$ 의 지분 가중 서명(stake-weighted signatures)**을 획득하면 **가용성 증명(PoAv, 즉 Proof-of-Store 및 Quorum Store)**이 형성되며, 이는 해당 배치가 **최소 $f + 1$ 개의 정직한 검증자에 의해 저장됨**을 보장하고, 모든 정직한 검증자들이 실행 시 이를 검색(retrieve)할 수 있도록 합니다.
- **2단계: 블록 메타데이터 정렬(Block Metadata Ordering):** 이 단계에서는 네트워크 내에서 트랜잭션 및 블록 메타데이터의 **일관된 순서**를 설정하고 합의합니다. **AptosBFT 합의 메커니즘**은 Byzantine Fault Tolerance(BFT)를 제공하기 위해 **2-체인 룰(2-chain rule)**을 따릅니다. 이때 블록은 실행 단계로 **스트리밍되어 병렬 처리 준비**가 완료됩니다.
- **3단계 (BSR): 병렬 트랜잭션 실행(Parallel Transaction Execution):** 이 단계는 실행 계층에 속하며, **트랜잭션이 병렬로 실행**됩니다. 실행 결과는 이후 **상태 커밋 단계**로 전달됩니다.
- **4단계: 상태 커밋(State Commitment):** 이 단계에서는 트랜잭션 실행 결과로 생성된 **상태 변경**을 최종화하고, **블록 최종화 준비**를 수행합니다. **상태 커밋은 트랜잭션 실행과 비동기적으로** 계산되며, 이는 현재 블록의 상태 커밋이 다음 블록의 실행을 **차단하지 않도록** 보장합니다.
- **5단계: 상태 영속화(State Persistence):** 이 단계에서는 커밋된 상태 변경을 **블록체인의 스토리지**에 영구 저장합니다. 최종화된 **State Root** 및 관련 데이터는 **Gravity Store**에 저장되며, 이는 빠른 접근성과 신뢰성을 위해 **고도로 최적화된 스토리지 엔진**을 사용합니다. 또한 mempool 및 quorum store에 알림을 보내, 더 이상 **향후 블록에 포함될 수 없는 트랜잭션들을 정리(clear)**하도록 합니다.

스테이킹 및 리스테이킹 모듈

보안성 높은 **지분증명(PoS)** 기반의 **레이어 1 블록체인**을 부트스트래핑하는 것은 매우 복잡한 과제입니다. 특히 체인 고유 토큰만을 활용한 스테이킹 방식에만 의존할 경우, **토큰 가치의 변동성**이나 **검증자 참여의 제한**으로 인해 네트워크를 보호하기에 **충분한 경제적 보안성**을 제공하지 못할 수 있습니다. 이 문제를 해결하기 위해, Gravity SDK는 **네이티브 및 외부 스테이킹 메커니즘**을 아우르는 **유연한 스테이킹 및 리스테이킹 모듈**을 도입했습니다.

Gravity SDK가 채택한 핵심 전략은 EigenLayer 및 Babylon과 같은 **리스테이킹 프로토콜 통합**입니다. 이러한 프로토콜은 **Ethereum** 및 **Bitcoin**과 같은 확립된 네트워크에서 **기존 자산을 리스테이킹**할 수 있도록 하며, 결과적으로 해당 체인들의 **보안 보증(security guarantee)**를 활용할 수 있게 해줍니다. 검증자들은 이러한 외부 체인 자산을 담보화(collateralize)하여 참여할 수 있으며, 이를 통해 Gravity SDK는 **자체 토큰에만 의존하지 않고도 경제적 보안성을 강화**할 수 있습니다. 이러한 접근은 체인의 **회복탄력성(resilience)**을 높이는 동시에, **보다 포괄적인 검증자 생태계**를 조성합니다. 모듈성(modularity)은 스테이킹 모듈 설계의 핵심 원칙이며, 리스테이킹 컴포넌트는 **블록체인 생태계 진화에 따라 새로운 리스테이킹 프로토콜을 손쉽게 통합**할 수 있도록 **높은 유연성**을 갖고 설계되었습니다.

리스테이킹 자산 지원 외에도, 본 모듈은 **지원되는 체인에서 커스텀 ERC20 토큰을 통한 스테이킹도 허용**합니다. 예를 들어, Ethereum 상의 G 토큰 등을 검증자가 스테이킹할 수 있으며, 이를 통해 합의에 참여하고 네트워크 보안에 기여할 수 있습니다. 각 검증자의 **투표 권한(voting power)**은 **전체 스테이킹 가치**를 기준으로 계산되며, 이는 **커스텀 토큰 스테이킹 + 리스테이킹 자산** 모두를 포함합니다. 이 계산은 체인별 설정(chain-specific configurations)을 따르며, 각 체인이 자신만의 요구에 맞게 **스테이킹 및 리스테이킹 파라미터를 정의**할 수 있도록 보장합니다.

합의 엔진 내 **에포크 관리자(epoch manager)**는 스테이킹 모듈과 직접 상호작용하여 **다음 검증자 집합의 가중치(weight)**를 계산합니다. 이때 실행 계층으로부터 스테이킹 값을 읽어오며, **합의 과정이 최신 스테이킹 활동을 정확히 반영**하도록 합니다. 이 설계에서는, 예를 들어 Ethereum에서 리스테이킹된 자산과 같은 **크로스체인 값(cross-chain value)**은 먼저 **실행 계층으로 브릿징**되어야 하며, 그 이후에 각 검증자의 총 스테이킹 값 계산에 사용됩니다. 이 브릿징 메커니즘은 **실행 계층이 자체적으로 처리**하도록 위임되어 있으며, 이는 **다양한 방식의 크로스체인 통신을 유연하게 지원**할 수 있게 합니다. 예를 들어, **PoS-브릿지, 체인 상태의 영지식 증명(zero-knowledge proof), 자체 부트스트래핑되는 크로스체인 메시징** 등의 방법이 모두 가능한 솔루션입니다.

스테이킹 및 리스테이킹 메커니즘에 대한 **기술 명세, API 설계, 포괄적인 세부사항**은 향후 배포될 문서에서 추가로 제공될 예정입니다.

Gravity Reth: 블록 스트림 리액터 기반 EVM 실행 계층

Ethereum Virtual Machine(EVM) 실행 계층을 Gravity SDK 아키텍처에 통합하는 것은 **특유의 과제를 동반**합니다. 특히, **파이프라인형 합의 엔진의 기능을 완전히 활용**하려는 경우 그러합니다. 이러한 아키텍처와의 원활한 통합을 달성하고 그 **성능 잠재력을 최대한 이끌어내기 위해**, 우리는 오픈소스 Ethereum 클라이언트인 **reth**에 여러 핵심 최적화를 수행해야 합니다. 이러한 최적화는 reth를 **Gravity reth**로 근본적으로 변형시키며, 이는 **파이프라인 합의 엔진에 최적화된 EVM 실행 계층**으로 재설계됩니다.

전통적인 블록체인 아키텍처는 블록을 순차적으로 처리합니다. 즉, 각 블록은 완전히 검증되고 실행된 후에야 다음 블록이 제안될 수 있습니다. 반면, Gravity SDK는 블록 처리의 다양한 단계를 분리하는 파이프라인형 합의 메커니즘을 채택하여 성능을 향상시킵니다. 이러한 패러다임 전환은 다음과 같은 복잡성을 초래합니다:

1. 예상치 못한 트랜잭션들:

파이프라인 체인에서는 **mempool**이 최신 월드 상태에 접근할 수 없습니다. 이는 이전 블록의 실행이 아직 완료되지 않았을 수 있기 때문입니다. 그 결과, 제안된 블록에 포함된 트랜잭션은 제안 당시 실행 불가능할 수 있으며, 최신 상태 없이는 유효성을 엄밀히 검증할 수 없습니다.

2. 비차단형 실행 결과(Non-Blocking Execution Results):

파이프라인 병목 현상(stall)을 방지하기 위해, 실행 결과는 다음 블록 생성을 멈추게 해서는 안 됩니다. 실행 계층은 제안된 블록을 비동기적으로 처리할 수 있어야 하며, 실행 결과를 합의 프로세스를 지연시키지 않고 후속 단계에서 반환할 수 있어야 합니다. EVM의 경우, 이를 위해서는 블록 헤더의 **stateRoot** 필드에 대한 의존성을 제거하고, 대체 **blockhash** 정의가 필요합니다. 이러한 과제들을 해결하기 위해, 우리는 다음의 네 가지 핵심 최적화를 도입하였습니다:

- **Block Stream Reactor (BSR):**

BSR은 Gravity SDK의 파이프라인형 블록 제안 프로세스에 **reth**를 적응시키기 위해 설계되었습니다. 이 구성 요소는 실행 계층이 제안된 블록 스트림을 지속적으로 소비할 수 있도록 하며, 블록을 비동기적으로 처리하는 리액터(reactor)로 동작합니다. BSR은 합의 엔진과 동적 피드백 루프를 형성하며, 여기에 적절한 역압(backpressure) 신호를 포함합니다. 이러한 신호는 실행 계층의 처리량과 지연(latency)에 따라 블록 제안 및 상태 커밋 속도를 실시간으로 조절합니다. 복잡한 트랜잭션이나 자원 제약으로 인해 실행 계층이 지연될 경우, 역압 메커니즘은 블록 제안 속도를 자동으로 제한(throttle)하여 시스템의 안정성을 유지합니다.

- **트랜잭션 실행과 상태 커밋의 분리:**

두 번째 최적화는 트랜잭션 실행과 상태 커밋 계산 과정을 분리하는 것입니다. 이 분리를 통해 상태 커밋은 비동기적으로 계산될 수 있으며, 현재 블록의 상태 커밋이 완료되기를 기다리지 않고도 다음 블록의 실행을 계속 진행할 수 있게 됩니다. 이를 위해 우리는 **blockhash**의 정의를 변경하여, 블록 헤더의 **stateRoot** 필드에 대한 의존성을 제거했습니다. 그 결과, 상태 루트 계산이 블록 생성 흐름을 차단하지 않도록 보장합니다.

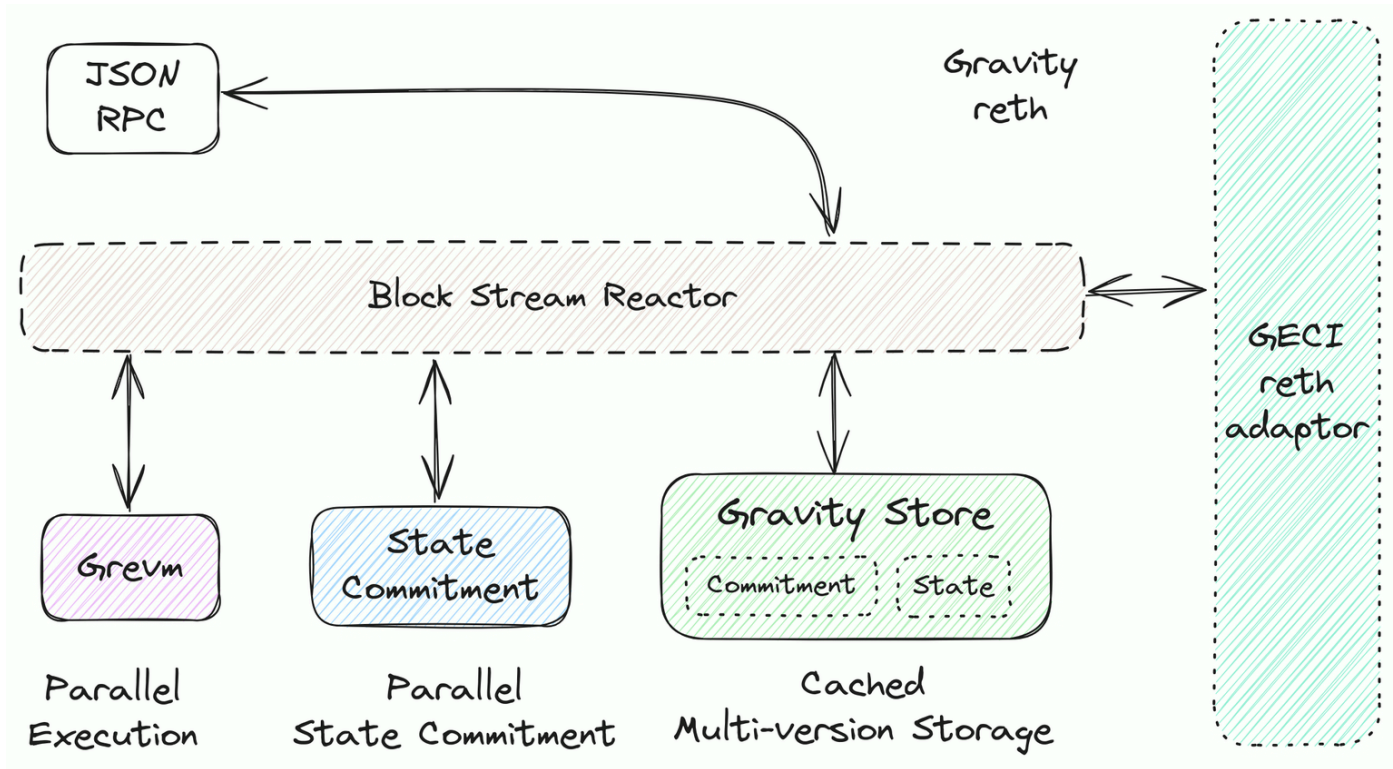
- **스토리지 계층 최적화:**

파이프라인 아키텍처에서는 다중 버전 상태 값 및 상태 커밋을 효율적으로 캐싱하고 영속화(persist)하는 것이 매우 중요합니다. 세 번째 최적화는 이러한 요구사항을 병목 없이 처리할 수 있도록 스토리지 계층을 강화하는 데 집중합니다. 스토리지 메커니즘을 미세 조정함으로써, 상태 데이터를 빠르게 쓰고 고도로 동시적으로 조회할 수 있도록 보장합니다.

이에는 다중 버전 스토리지 엔진 구축, 데이터베이스에서 스토리지 API까지의 비동기 I/O 지원이 포함됩니다.

- **병렬 EVM:**

마지막 최적화는 EVM 내에서 트랜잭션 실행을 병렬화하는 것입니다. 우리는 **Grevm**이라는 병렬 EVM 런타임을 개발하였으며, 이를 통해 트랜잭션 처리 속도를 크게 향상시켰습니다. Grevm은 트랜잭션 시뮬레이션으로부터 파생된 데이터 의존성 힌트를 활용하여 병렬 실행을 최적화합니다. 이를 통해 트랜잭션 재실행을 최소화하고 처리량을 극대화합니다.



Grevm (Gravity EVM) - 병렬 EVM 실행

Grevm은 GitHub에 오픈소스로 공개되어 있습니다 (또는 곧 공개될 예정입니다). 자세한 내용은 [README](#)를 참고하세요.

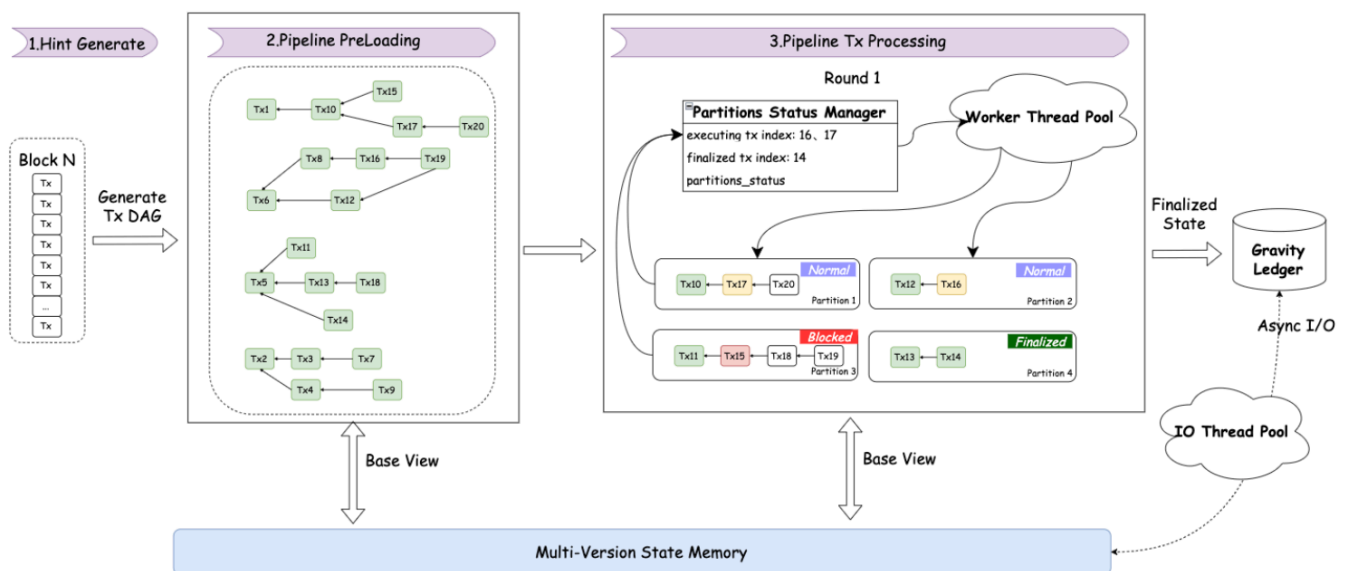
Grevm (Gravity EVM)은 revm을 기반으로 구축된 **오픈소스 병렬 EVM 런타임**입니다. Grevm의 알고리즘은 BlockSTM에서 영감을 얻었으며, 여기에 **트랜잭션 시뮬레이션 결과로부터 파생된 데이터 의존성 그래프**를 통합함으로써 향상되었습니다. 이 메커니즘은 병렬 실행을 위한 스케줄링을 보다 효과적으로 수행할 수 있게 하며, **트랜잭션의 재실행(re-execution)**을 최소화합니다.

우리의 벤치마크 결과에 따르면, **Grevm은 현재까지 가장 빠른 오픈소스 병렬 EVM 구현체**입니다. 충돌이 없는(conflict-free) 트랜잭션의 경우, Grevm은 순차 실행 대비 **4.13배** 빠르며, 초당 **26.50 기가가스(gigagas/s)**를 기록합니다. 실제 환경에서의 I/O 지연(예: **100 마이크로초(μ s)**)을 시뮬레이션할 경우, Grevm은 순차 실행 대비 **50.84배** 빠르며, 초당 **6.80 기가가스**의 처리량을 달성합니다. 이러한 성능 향상은 병렬 실행 자체뿐만 아니라, 병렬성 덕분에 가능한 **비동기 I/O 연산의 통합** 덕분입니다. 이로 인해 I/O 작업이 병렬로 겹쳐 실행되며, 전체 처리 속도가 더욱 증폭됩니다.

Grevm의 핵심 아이디어는, **트랜잭션 간 데이터 의존성(data dependency)**을 활용하여 병렬 실행을 최적화하는 데 있습니다. 구체적으로는, 트랜잭션의 **read/write 집합**을 추론하여 **사전 실행된 시뮬레이션 기반의 힌트**를 이용합니다. 모든 힌트가 완벽히 정확한 것은 아니지만, 실전에서 사용하기에 충분히 정확합니다. 예를 들어, Ethereum 메인넷 기준 **과거 가스 사용 기록에 따르면**, 전체 트랜잭션 중 약 30%는 **단순한 이더 전송**, 그리고 25~30%는 **ERC20 토큰 전송**으로, 일반적으로 **소수의 계정과 저장 슬롯에만 접근**합니다. 이러한 유형의 트랜잭션에서는 시뮬레이션 결과가 **일관되게 정확**합니다.

이러한 통찰을 바탕으로, 우리는 **BlockSTM 모델을 확장**하여 시뮬레이션 기반 데이터 의존성 힌트를 도입한 **Grevm 전용의 3단계 병렬 실행 프레임워크**를 개발하였습니다:

- **1단계: 힌트 생성 및 상태 프리로딩 (Hint Generation & State Preloading)** — 트랜잭션을 시뮬레이션하여 **의존성 힌트**를 수집하고 **메모리 캐시를 예열(warm-up)** 합니다. 이 단계는 블록체인의 설계에 따라 다양한 시점에 수행될 수 있으며, 예를 들어 mempool에 새 트랜잭션이 도착했을 때 즉시 시뮬레이션을 수행하여 **의존성 힌트를 사전에 준비**할 수 있습니다.
- **2단계: 의존성 분석 (Dependency Analysis)** — 시뮬레이션 단계에서 수집한 의존성 힌트를 기반으로 트랜잭션 간 의존 관계를 모델링한 **DAG(방향 비순환 그래프)**를 생성합니다. 이 DAG는 이후 병렬 실행 시 **트랜잭션 스케줄링의 로드맵 역할**을 합니다.
- **3단계: 충돌 해결을 포함한 병렬 실행 (Concurrent Execution with Conflict Resolution)** — 2단계에서 생성된 DAG를 활용하여, 수정된 **BlockSTM 알고리즘**으로 트랜잭션을 병렬로 실행합니다. 이때 스케줄러는 블록 내 트랜잭션 순번(예: 1, 2, 3, ..., n)을 기준으로 선택하는 것이 아니라, **DAG를 기반으로 충돌을 최소화하는 우선순위를 설정하여 재실행을 줄이고 처리량을 높입니다.**



비동기 일괄 상태 커밋 (Asynchronous Batched State Commitment)

상태 커밋(state commitment) 생성은 블록체인 파이프라인에서 여전히 **중대한 병목 지점**입니다. 그 원인은 머클화(merklization)의 본질적인 **순차적 특성**에 있으며, **각 서브트리의 계산이 완료되어야만 최종 상태 커밋이 생성될 수 있기 때문**입니다. 기존 솔루션 —예: reth의 계정 단위 병렬화(account-level parallelization)—는 어느 정도 병렬성을 도입했지만, **최적화 여지가 여전히 많이 존재**합니다. 이는 특히 Gravity reth의 컨텍스트, 즉 **Block Stream Reactor (BSR)** 실행 계층에서 더욱 중요합니다. 해당 구조에서는 **상태 커밋 합의가 트랜잭션 실행과 분리되어 있으며, 지연(deferred) 및 일괄 처리(batched)** 방식의 상태 커밋 계산이 **비동기적으로 수행**될 수 있어, 실행을 차단하지 않습니다.

이러한 과제를 해결하기 위해, 본 프레임워크는 다음과 같은 핵심 혁신을 도입합니다:

비동기 일괄 해시 계산 (Asynchronous Batched Hash Computation):

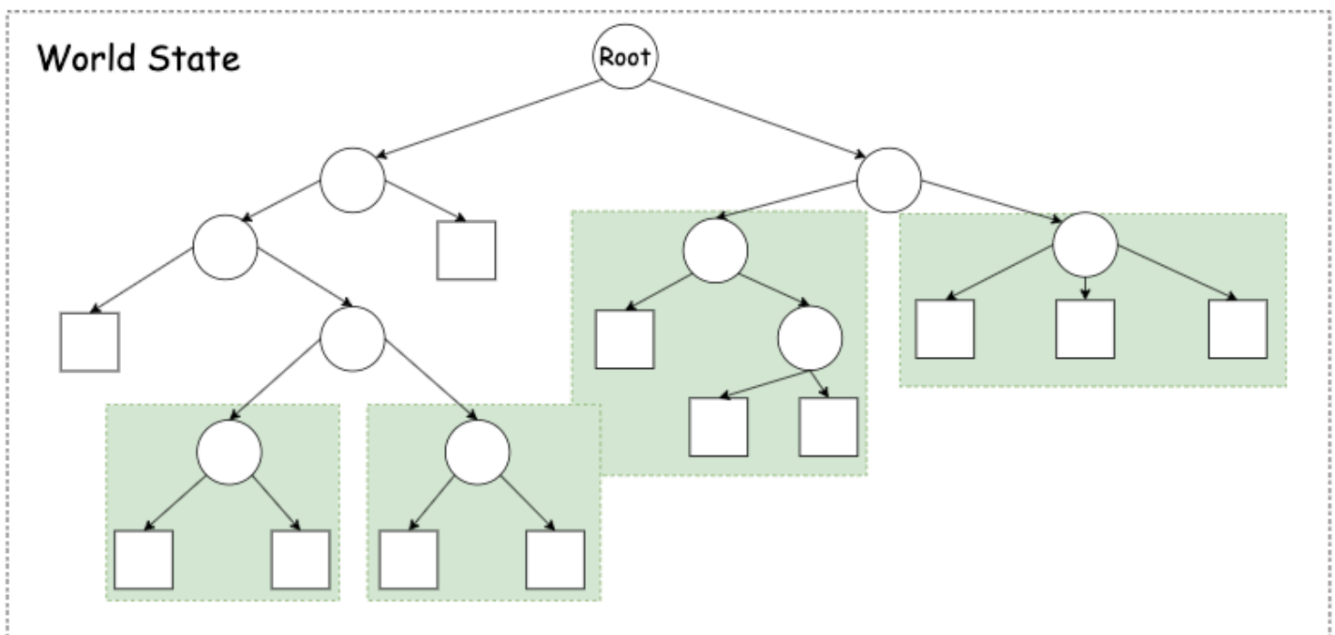
상태 커밋 합의와 트랜잭션 실행의 분리를 활용하여, **상태 커밋을 비동기적으로 계산**할 수 있도록 합니다. 예를 들어, 상태 루트는 **10개 블록 단위로 일괄 계산**되며, 상태 루트 계산 빈도를 줄입니다. 이와 같은 **배칭 (batching)** 접근 방식은 빈번한 업데이트로 인한 오버헤드를 줄이고, **공통으로 더럽혀진 노드 (dirty nodes)**를 집계하여 효율적인 해시 계산을 가능하게 하여 **전체 계산 비용을 절감**합니다. 소규모 블록의 경우 병렬성이 증가하며, 대규모 블록의 경우 전체 계산 비용이 절감됩니다.

전체 병렬화 (Full Parallelization):

이 프레임워크는 병렬화의 범위를 개별 계정 트리 수준에서 **전체 상태 트리 (state tree) 전체**로 확장합니다. "더럽혀진 (dirty)" 노드에 대해, 프레임워크는 **병렬 상태 계산 알고리즘 (Parallel State Calculation)**을 적용하여 상태 트리를 **독립적인 서브트리들로 분할**합니다. 이러한 서브트리는 **동시에 병렬로 처리**되며, 결과는 최상위 레벨에서 집계되어 **최종 루트를 효율적으로 계산**합니다. 이 방식은 수많은 트랜잭션과 상태 변경이 있는 **대규모 블록이 멀티스레딩을 최대한 활용**하여 처리량을 극대화할 수 있도록 보장합니다.

대안적 고속 상태 루트 정의 (Alternative Fast State Root):

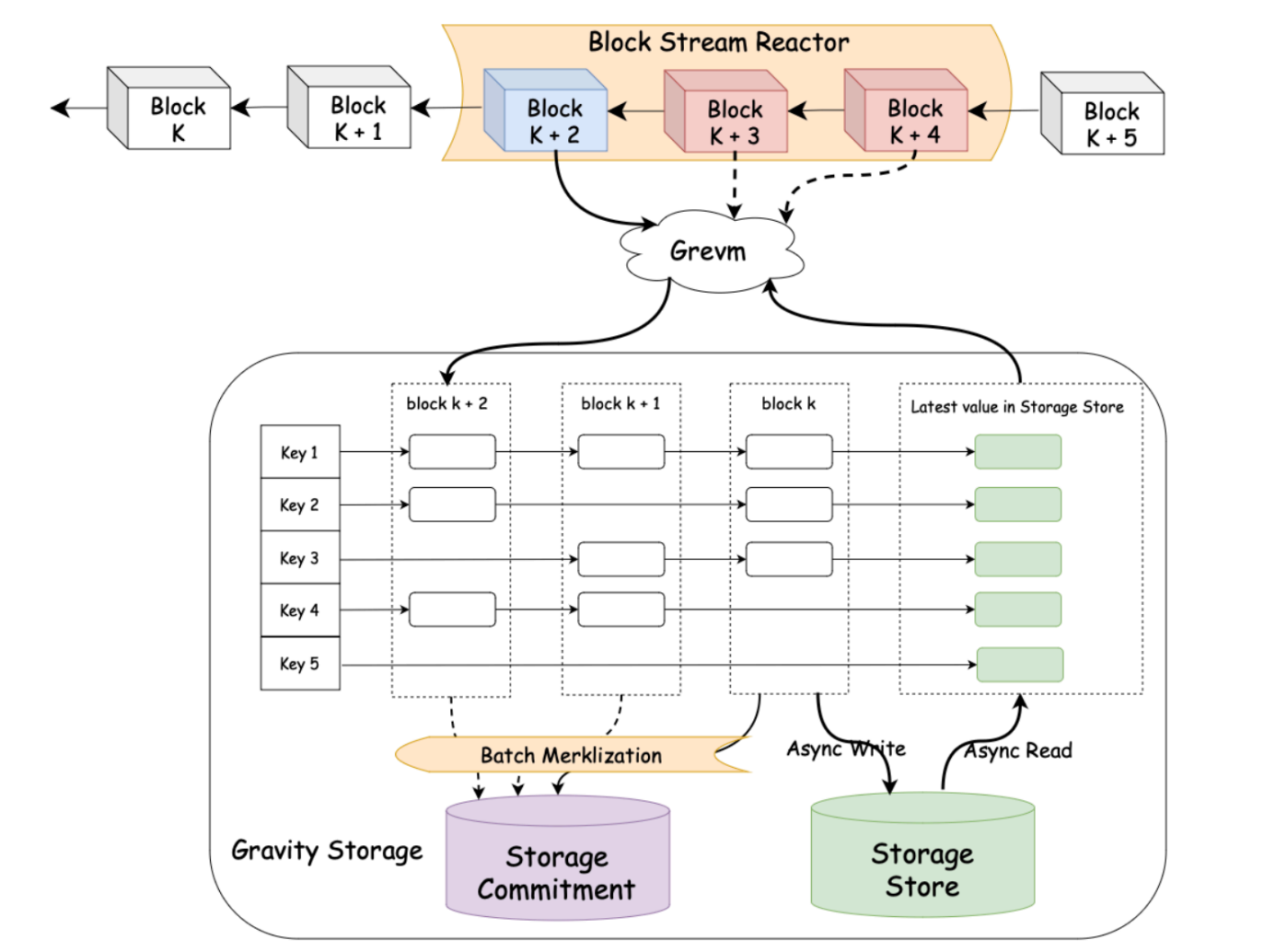
Ethereum의 블록 헤더 및 BLOCKHASH 오퍼코드(opcode)는 **이전 256개의 블록 상태 루트**에 접근해야 하므로, 우리는 **상태 루트의 정의를 재구성**했습니다. 트랜잭션 실행 중에는 **최종 상태 커밋이 존재하지 않기 때문에**, 기존 방식에 의존할 수 없습니다. 대신, 우리는 상태 루트를 **현재 블록의 변경 집합 (change set)**과 **이전 상태 루트를 해시하여 계산**합니다. 이 방식은 **전체 상태 커밋의 최종화 없이도 빠른 상태 루트 계산이 가능**하도록 하며, BLOCKHASH 요구 사항을 충족합니다.



Gravity Store

고성능 블록체인에서는 **트랜잭션 실행 및 상태 커밋**으로 생성되는 방대한 양의 데이터를 효율적으로 관리할 수 있는 **스토리지 계층**이 필요합니다. Gravity Store는 **정밀하게 조정된 다중 버전 스토리지 계층 (multi-version storage layer)**으로, 이미 **상태 커밋 스토어와 상태 저장 스토어를 분리**하여 상태 비대화 (state bloating)를 완화하고 읽기/쓰기 증폭 (read/write amplification)을 줄인 reth 기반의

구조를 확장합니다. 하지만 Gravity reth 실행 계층의 **병렬 실행 및 비동기 상태 커밋**을 지원하기 위해서는 **추가적인 요구 사항**이 발생합니다.



이러한 요구를 해결하기 위해, Gravity Store는 BSR 아키텍처의 특수한 요구를 지원하는 **고성능 다중 버전 트리(multi-version tree)**를 도입합니다. 이는 **여러 버전에 걸친 상태 업데이트 관리**에 사용됩니다. 수정이 발생할 때마다 즉시 해시를 재계산하는 대신, 변경된 노드는 **“더럽혀진(dirty)”** 상태로 표시되며, 특정 트리 버전에 대해 **지연(deferred)** 및 **일괄(batched)** 해시 계산을 가능하게 합니다. 이 구조는 다음과 같은 효율적인 작업을 지원합니다:

- 새로운 버전 생성
- 특정 버전에서 값 조회
- 특정 높이 이하의 버전 삭제

이를 통해 **블록체인 상태를 효율적으로 관리**하는 데 있어 큰 성능 이점을 제공합니다.

우리는 또한 **블록체인에 특화된 스토리지 계층 최적화** 및 **완전한 비동기 I/O 지원**을 목표로 자체 스토리지 엔진인 **Gravity DB** 개발도 검토하고 있습니다. 이 설계는 고성능 블록체인 데이터베이스 엔진인 [LETUS](#)와 유사한 방향을 따르게 되며, LETUS는 **로그 기반 고성능 블록체인 저장소 (Log-structured Efficient Trusted Universal Blockchain Storage)**로 설계된 엔진입니다. Gravity DB 개발은 **LETUS**의 공동 저자이자 Gravity의 리드 개발자 **Richard**가 주도하고 있으며, 곧 블로그 포스트를 통해 더 많은 기

술적 세부사항을 공개할 예정입니다.

결론 (Conclusion)

Gravity Chain은 현대 웹3 애플리케이션의 확장성 및 성능 요구를 충족하기 위해 설계된 고성능 EVM 호환 레이어 1 블록체인입니다. Gravity SDK(파이프라인 구조의 AptosBFT PoS 합의 엔진)와 Gravity reth(Grevm이 구동하는 Block Stream Reactor 실행 계층)를 결합함으로써, Gravity는 다음과 같은 특성을 제공합니다:

- 초당 1기가가스(gigagas/s) 처리량
- 1초 미만의 최종성(sub-second finality)
- 리스테이킹 기반의 PoS 보안성

이러한 구성 요소들은 웹3 애플리케이션이 자체적으로 대안적인 L1 또는 더 빠른 최종성을 지닌 L2를 구축할 수 있도록 지원하며, 특히 EVM 체인에 최적화된 설계를 따릅니다.