

리네아(LINEA)

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 위해
서는 원문을
참고하시거나 원문 작성 주체 측에 문의하시기 바랍니다. 두나무는 해당 번역본 내용의 진실성 및 정확
성을 보장하지 않고, 제공된 정보에 의한 투자 결과에 대하여 어떠한 책임도 지지 않으며, 투자 손실은
투자자에게 귀속됩니다.

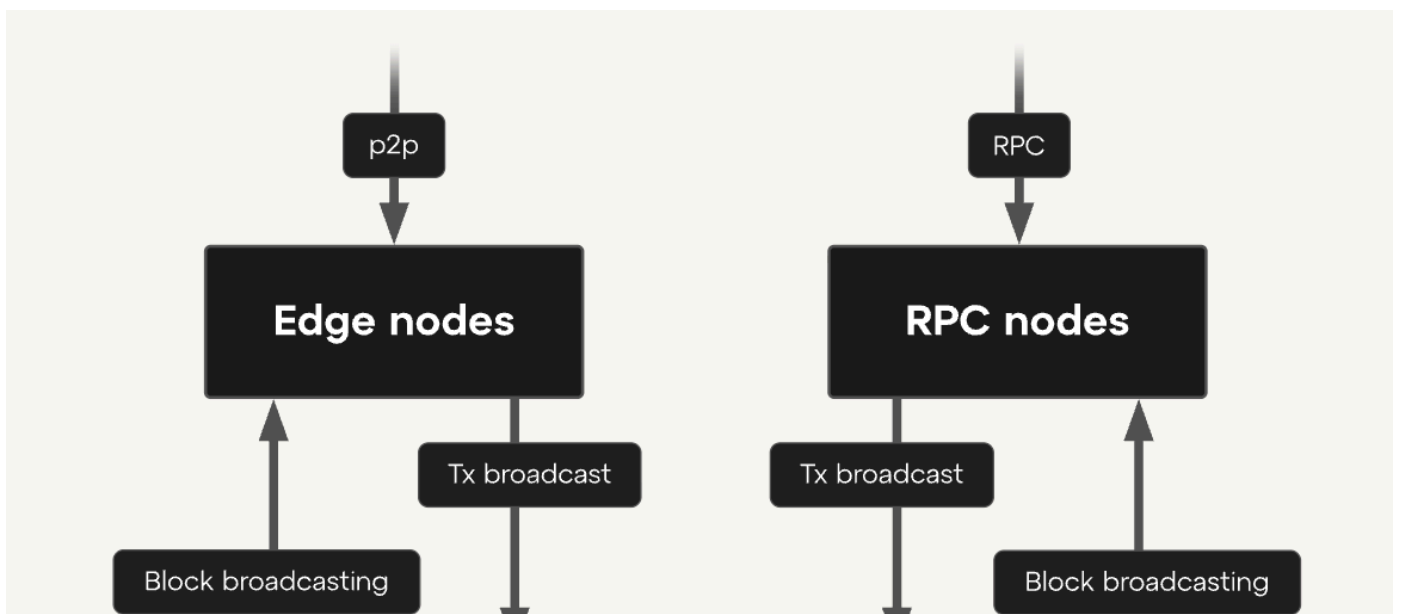
아키텍처

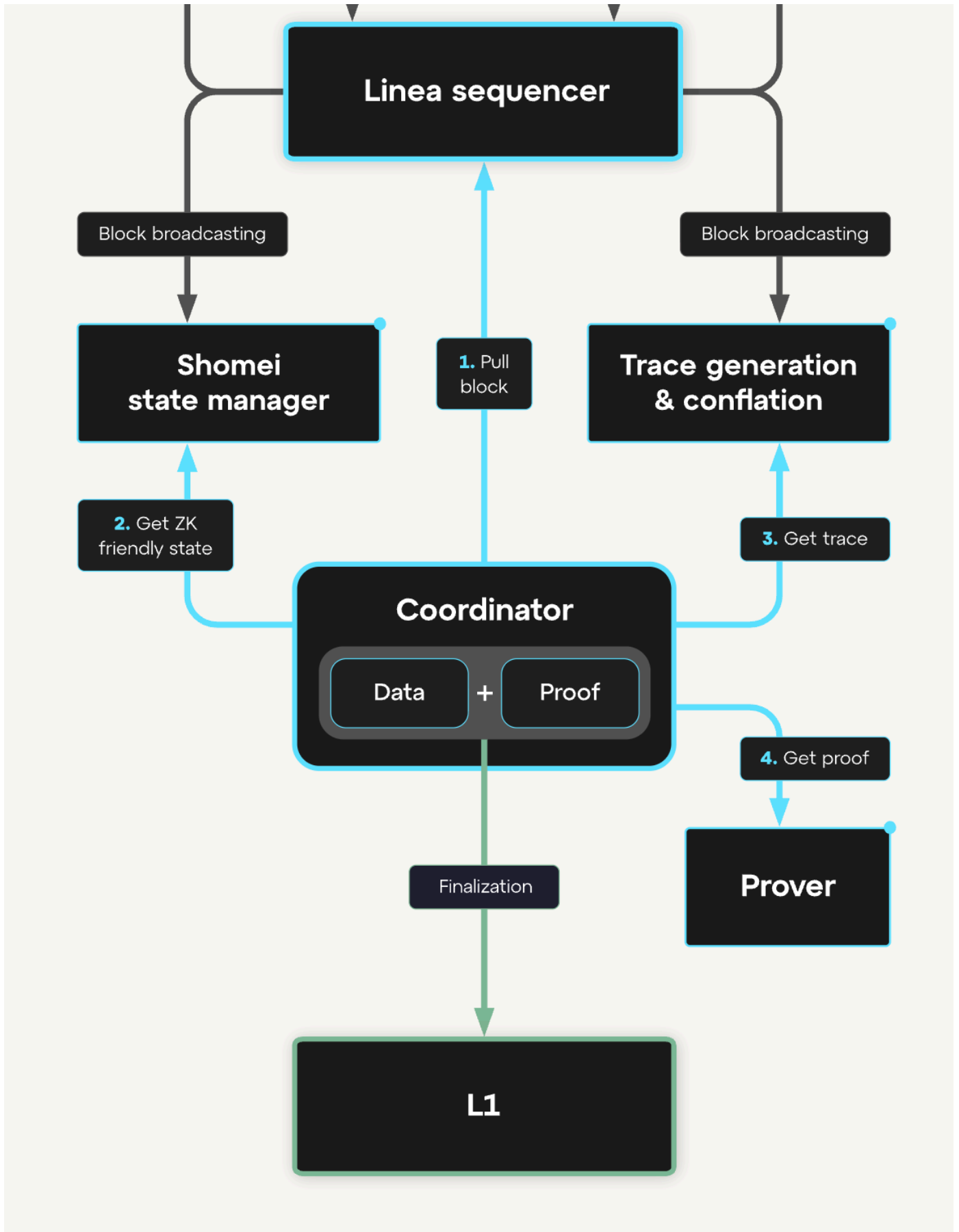
Linea는 완전히 탈중앙화되고 퍼미션리스(permissionless) 네트워크를 지향합니다. 이 목표를 지원하
기 위해, Linea 아키텍처는 세 가지 주요 요소로 구성됩니다:

- 시퀀서(Sequencer)
- 프로버(Prover)
- 브릿지 릴레이어(Bridge relayer)

현재 상태

Linea는 메인넷(mainnet) 상태에 있으며, 팀은 완전한 탈중앙화를 향해 열정적으로 작업하고 있습니다.
아래는 Linea의 주요 구성 요소와 그 상호작용을 잘 나타낸 그림입니다:





우선: Linea는 무엇입니까? zkEVM L2란 무엇입니까?

사람들은 서로 다른 네트워크와 그 관계를 설명하기 위해 여러 가지 사고 모델을 사용합니다. 어떤 이들은 L2를 "롤업 네트워크(rollup networks)" 또는 "확장 솔루션(scaling solutions)"이라고 부르기를 선호합니다. "레이어(layer)"라는 용어도 인기 있는 사고 모델인데, 하나의 네트워크가 "다른 네트워크 위에 구축된다"는 개념입니다.

명확히 해보자면:

이더리움 네트워크는 몇 가지 기능 영역을 가집니다

- **블록체인(Blockchain)**: 주소를 추적하고, 어떤 토큰이 어떤 주소에 할당되어 있는지를 기록합니다.
- **합의 메커니즘(Consensus mechanism)**: 많은 노드가 한 주소에서 다른 주소로의 토큰 이동에 대해 통신하며, 각 노드가 자신의 원장 사본을 최신 상태로 유지합니다.
- **실행 환경(Execution environment)**: 프로그램을 실행할 수 있는 EVM(Ethereum Virtual Machine)입니다.

이 세 가지 영역은 밀접하게 상호 연결되어 있으며, 이는 단순화된 설명이지만 모든 네트워크가 감수해야 하는 트레이드오프를 이해하는 데 유용합니다.

이더리움은 보안을 최우선으로 합니다. 합의 메커니즘은 누구도 네트워크 상태를 단독으로 변경할 수 없도록 설계되어 있습니다. 이는 매우 인상적인 공학적 성취이며, 동시에 트레이드오프를 수반합니다. 실행 환경은 수행할 수 있는 작업량이 크게 제한되는데, 이는 합의 메커니즘이 의도적으로 느리게 동작하여 모든 것을 안전하게 유지하기 때문입니다.

Linea와 다른 L2들은 실행을 최적화하면서도 이더리움의 보안 메커니즘에 참여하도록 설계되었습니다. 다시 말해, L2는 사용자가 많은 트랜잭션을 수행하고, 수많은 프로그램을 실행하며, 컨트랙트를 배포하고, NFT를 발행하는 등 다양한 활동을 빠르고 저렴하게 할 수 있도록 합니다. 그리고 이 데이터를 압축하여 이더리움 블록체인에 포함될 수 있도록 이더리움에 보고서를 제출합니다. Linea에서의 활동을 정기적으로 이더리움에 보고함으로써, 네트워크는 보안에 의해 심하게 제한되지 않고 실행 최적화를 달성할 수 있습니다.

이러한 동작은 "롤업(rolling up)"이라고 불립니다. Linea가 "트랜잭션을 이더리움으로 롤업한다"라고 말할 수 있습니다. 그리고 이더리움의 보안에 의존한다는 사실 때문에, 사람들은 이것을 "이더리움 위에 구축되었다"고 표현하며, 따라서 "세컨드 레이어 네트워크(second-layer network)", 즉 L2라고 부릅니다. 그리고 미리 대답하자면, 맞습니다. 이더리움은 L1이고, L3도 존재합니다. Linea에 롤업되는 네트워크는 결과적으로 이더리움에도 롤업됩니다.

핵심은 **롤업**이 어떻게 이루어지는가에 있습니다. Linea는 특별합니다. 수학과 컴퓨터 과학의 최첨단 연구 분야 중 하나인 영지식(zero-knowledge), 또는 *영지식 암호학(zero-knowledge cryptography)*을 사용하여, Linea 네트워크에서 발생하는 모든 일이 실제로 발생했음을 이더리움 네트워크에 증명합니다. 이 과정에서 모든 개별 트랜잭션의 전체 기록을 제출하고 하나하나 확인할 필요가 없습니다. 대신, Linea는 각 트랜잭션에 대한 검증 가능한 증명을 이더리움에 제출합니다. 이것이 바로 'zk' 부분입니다.

따라서, 지금까지의 개념을 정리하면, Linea는 zkEVM L2 네트워크입니다.



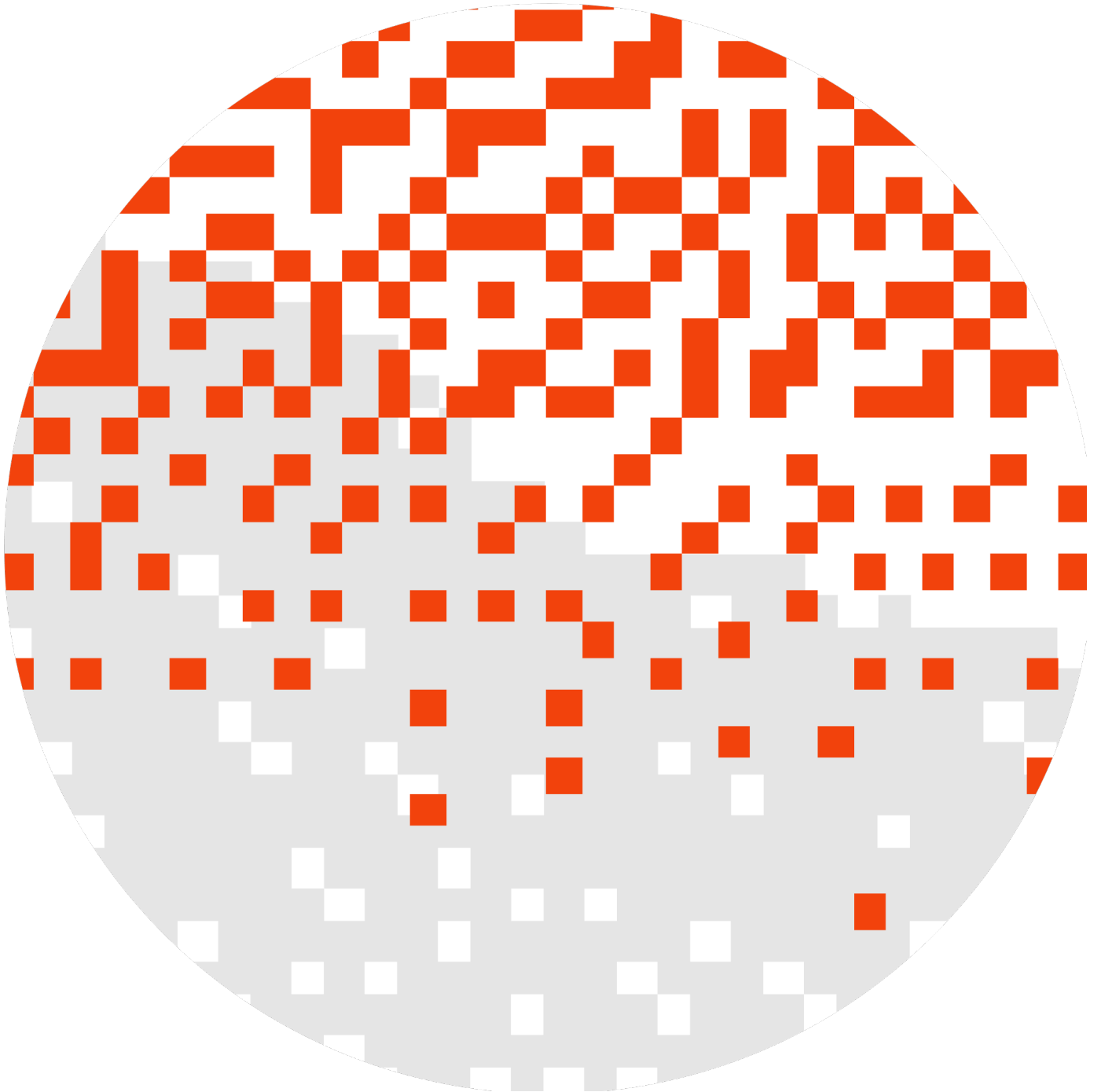
그렇다면... *시퀀서(sequencer)*와 *프로버(prover)*란 무엇입니까? 실제로는 어떻게 동작합니까?

상위 수준에서, 이더리움에서 Linea를 거쳐 다시 이더리움으로 돌아오는 플로우를 따라가 보면 다음과 같습니다:

1. 이더리움 브릿지 컨트랙트 >
2. Linea 브릿지 컨트랙트 >
3. 코디네이터(Coordinator) >
4. 시퀀서(블록 생성 > 실행 > 트래이스 데이터 생성) >
5. 코디네이터 >

6. 트레이스 컨플레이션(Trace Conflation) >
7. EVM 상태 관리자(State Manager) >
8. 트레이스 확장 및 증명(Trace Expansion and Proving: Corset > gnark) >
9. 코디네이터 >
10. zk-증명과 업데이트된 머클 트리 >
11. Linea 브릿지 컨트랙트 >
12. 이더리움 브릿지 컨트랙트 >
13. 이더리움 블록체인

...즉, 많은 과정이 포함됩니다.

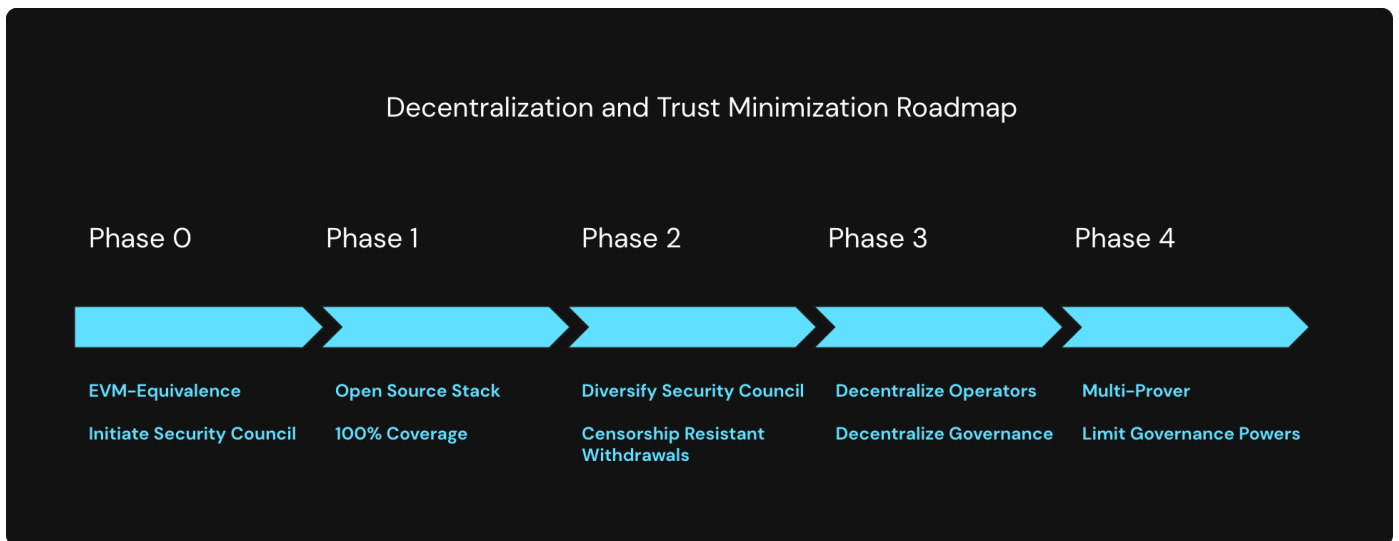


다음 단계

- [Linea 아키텍처](#)를 구성하는 다양한 컴포넌트에 대해 더 알아보십시오.
- 많은 신규 L2 사용자들의 첫 접점인 브릿지를 이해하십시오. Linea에는 여러 개의 브릿지가 있으며, 그 수는 계속 증가할 가능성이 있습니다. [Linea 캐노니컬 메시지 서비스](#)는 이러한 브릿지와 그 작동 방식의 핵심입니다.

탈중앙화 로드맵

Linea 로드맵은 현재 다섯 개의 단계로 범위가 설정되어 있으며, 팀은 더 넓은 Linea 생태계 및 커뮤니티와 협력하여 순차적으로 이를 실행할 계획입니다.



Phase 0 — 완료

Phase 0은 Linea 메인넷 알파(Mainnet Alpha)의 출시와 함께 시작되었습니다. 첫날부터 Linea의 EVM-호환 zkEVM은 이더리움 재단이 명시한 EVM 사양에 따라 구현됨으로써 기본적인 이식성을 제공했습니다. Linea 위에서 dapp을 개발하는 개발자는 Linea 네트워크에 만족하지 못할 경우, 스마트 컨트랙트를 다시 작성하거나 도구를 재구성하거나 새로운 보안 감사를 받지 않고도 언제든지 이더리움과 같은 다른 EVM 지원 체인으로 dApp을 옮길 수 있습니다.

이 단계에서 새로운 프로덕션 시스템과 관련된 위험을 완화하고, 네트워크를 모니터링하며, 사용자를 보호하기 위해 Linea 보안 위원회(Security Council)의 첫 번째 이터레이션을 소집했습니다. 보안 위원회는 멀티시그 세이프(multi-sig Safe, 이전 Gnosis Safe) 컨트랙트를 통해 관리되며, 특정 수의 서명이 제출되어야만 행동이 실행될 수 있습니다. 자세한 내용은 [위험 공시](#)에서 확인할 수 있습니다.

마지막으로, Linea 네트워크에서 로컬 노드를 실행하는 데 사용되는 클라이언트 소프트웨어가 공개되어, 누구나 Linea 상태를 직접 검증하고 Linea 운영자에 대한 신뢰를 최소화할 수 있는 선택권을 갖게 되었습니다. 시작하려면 [노드 실행 가이드](#)를 참조하십시오.

Phase 1 — 현재

이 단계에는 두 가지 중요한 요소가 있습니다: **오픈소스 스택(open-source stack)** 과 **100% EVM 커버리지 보장**입니다.

Linea의 소프트웨어 스택을 오픈소스로 공개하는 것은 로드맵의 필수적인 요소로, 투명성을 촉진하고 웹3 가치와 정렬됩니다. Linea 소프트웨어 스택은 AGPL-2.0 라이선스 하에 공개되어 있으며, 사용자가 코드를 열람하고, 포크하며, 수정할 수 있는 자유를 보장합니다. Linea 리포지토리의 개요는 [아키텍처](#) 섹션에서 확인할 수 있습니다.

이 단계의 두 번째 핵심은 zkEVM 산술화(arithmetization)에서 **100% EVM 커버리지**를 구현하는 것입니다. Linea는 모든 EVM 오프코드(opcodes)와 프리컴파일(precompiles)에 대한 계산 증명을 생성하여, 모든 사용 사례에 대해 신뢰 없는 실행을 가능하게 합니다. 산술화 사양은 공개되고, 감사를 거치며, 버그 바운티에 개방되어 시스템의 보안성과 신뢰성을 강화할 것입니다.

Phase 2

이 단계에는 두 가지 중요한 요소가 있습니다: **보안 위원회의 다각화와 검열 저항적 출금(censorship-resistant withdrawals)** 활성화입니다.

보안 위원회의 다각화는 이 단계의 첫 번째 부분을 구성합니다. Linea 메인넷 알파(Mainnet Alpha) 출시 당시 설립된 보안 위원회는 네트워크의 감독 및 통제에서 중요한 역할을 수행합니다. 그러나 권력 집중과 잠재적 편향을 방지하기 위해, 위원회의 참여를 확대할 계획이며, 동시에 즉각적인 업그레이드를 위해 6-of-8 멀티시그 임계값(또는 더 엄격한 기준)을 유지할 것입니다. 이러한 변경은 균형 잡힌 대표성을 보장하고 보다 견고한 집단적 의사결정을 촉진하며, 시스템에 추가적인 견제와 균형의 장치를 마련합니다.

사용자 주권(user sovereignty) 원칙과 밀접하게 연관된 **검열 저항적 출금 활성화**는 로드맵의 근본적인 요소입니다. 아키텍처는 롤업 운영자가 사용자 출금을 방해하지 못하도록 개선될 것입니다. 사용자가 독립적으로 자산 출금을 시작할 수 있도록 보장하여, Linea에서 자신의 자산에 대한 지속적인 접근과 통제를 가능하게 할 것입니다. 이러한 보장은 사용자 신뢰를 강화하고, 사용자가 자산에 대해 주권을 가진다는 약속을 확고히 합니다.

이러한 이니셔티브를 통해 Phase 2는 권력의 추가적인 탈중앙화, 투명성 강화, 그리고 사용자 경험 향상을 목표로 하며, 동시에 강력한 보안 및 통제 메커니즘을 유지합니다.

Phase 3

이 단계에서는 Linea 생태계의 기능 수행에 중요한 역할과 거버넌스 시스템 자체를 탈중앙화하는 데 초점을 맞추며, 이를 통해 신뢰 최소화(trust minimization)와 탈중앙화에 대한 의지를 더욱 강화합니다.

이 단계의 첫 번째 부분은 **운영자(operator) 탈중앙화**입니다. 현재 영지식 증명을 생성하는 프로버(provers)와 L2 트랜잭션을 관리하는 시퀀서(sequencers)는 생태계 내에서 핵심적인 역할을 맡고 있습니다. 이러한 역할을 탈중앙화함으로써, 네트워크 참여자들의 신뢰를 크게 높이고 시스템의 전반적인 무결성을 강화하려 합니다. 이 과정에서 잠재적인 지연(latency), 거래 비용 증가, MEV(maximum

extractable value) 문제와 같은 기술적 복잡성이 발생할 수 있음을 인식하고 있습니다. 그러나 이러한 도전을 극복하여 커뮤니티를 위한 신뢰할 수 있고 효율적인 플랫폼을 제공하기 위해 최선을 다할 것입니다.

이 단계의 두 번째 요소는 **거버넌스 탈중앙화**입니다. Linea의 거버넌스를 민주화하는 것이 지속적인 진화의 핵심이라고 믿습니다. 이러한 접근법은 모든 이해관계자가 참여할 수 있고, Linea의 미래가 사용자 커뮤니티에 의해 이끌어지는 공정하고 투명한 시스템을 촉진합니다. 공유 거버넌스는 단일 의사결정이나 단일 실패 지점(single point of failure)의 위험을 완화할 뿐만 아니라, 네트워크의 회복력을 강화합니다. 결과적으로, 이는 네트워크 참여자 간의 집단적 소유권과 신뢰를 증진시켜 생태계의 지속 가능한 성장을 촉진합니다.

Phase 4

로드맵의 마지막 단계는 **멀티 프로버(multi-prover)** 구현과 **거버넌스 권한 제한**을 통해 생태계의 견고함을 더욱 보장하고 신뢰 관련 위험을 완화하는 것을 목표로 합니다.

EVM은 이더리움 로드맵에서 설명된 대로 계속 진화하며, 더 큰 이더리움 커뮤니티와 이더리움 재단에 의해 주도될 것입니다. 따라서, Linea 네트워크에 변화를 적절한 속도로 반영하고 구현 위험을 방지하기 위해, zkEVM 프로버의 다양한 구현을 통합할 계획이며, 이를 **멀티 프로버 롤업(multi-prover rollup)**이라고 부릅니다. Linea에서 발생하는 각 트랜잭션은 서로 다른 이질적 구현의 zkEVM 프로버에 의해 검증되며, 그 복잡성으로 인해 특정 프로버가 버그를 겪거나 이용 불가능해지는 상황이 발생하더라도, 시스템은 중단 없이 계속 운영될 수 있습니다. 이 접근법은 EVM 사양이 지속적으로 진화하는 시기에 플랫폼의 신뢰성을 강화할 뿐 아니라, 혁신적인 발전을 탐구하도록 장려하며, 탁월한 수준의 시스템 가동 시간을 유지합니다.

마지막으로, 강력하고 탈중앙화된 거버넌스 모델에 대한 약속의 일환으로, **거버넌스 권한을 필요한 영역으로만 제한**할 계획입니다. 롤업 로직 업그레이드는 불변적(immutable)으로 유지되며, 업그레이드는 오직 EVM 사양 변경에 따른 검증기(verifier) 변경에만 허용될 것입니다. 이 경우 사용자에게는 롤업에서 탈출할 수 있도록 합리적인 사전 통보가 제공됩니다.

보안 위원회의 역할은 프로버 시스템이 불일치하거나 멈춘 것으로 보이는 등 시스템 문제나 불일치가 발생했을 때만 개입하는 것으로 한정됩니다. 이러한 조치를 통해, 필요한 곳에서는 거버넌스를 유지하면서도 탈중앙화 원칙을 지키는 균형을 이루고자 합니다.

토크노믹스(Tokenomics)

LINEA 토큰은 이더리움과 Linea의 성장과 지원을 위해 설계되었으며, 대부분의 레이어 2 토큰과 차별화되는 몇 가지 혁신적인 특징을 가지고 있습니다:

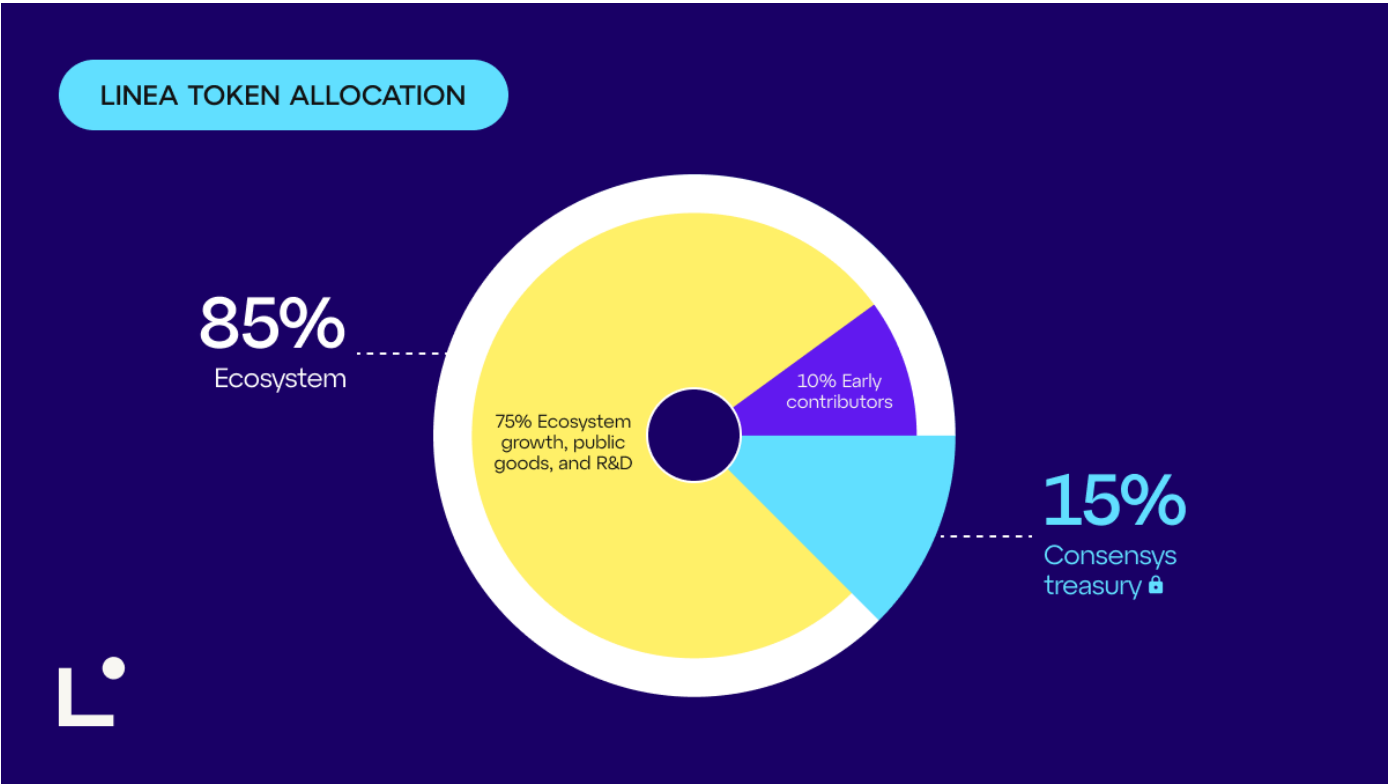
- 가스 토큰은 ETH입니다.
- LINEA는 어떠한 거버넌스 권한도 제공하지 않습니다.
- 내부자, 투자자, 팀 멤버에게 할당된 토큰이 없습니다.

대신, LINEA는 사용자가 Linea에서 앱과 프로토콜을 사용한 대가로 보상을 받으며 체인이 유기적으로 성장할 수 있도록 돕습니다.

할당(Allocation)

LINEA의 총 공급량은 72,009,990,000개입니다. 이는 이더리움의 제네시스 공급량 대비 1,000배에 해당합니다.

LINEA 공급량의 85%는 커뮤니티와 생태계에 투자되며, 이는 생태계 펀드(Ecosystem Fund)로 구성됩니다. 나머지 15%는 Consensys가 장기 보유합니다.



공급량은 아래 표와 같이 할당됩니다:

카테고리	공급 비율	세부 내용
생태계 (Ecosystem)	85%	75%는 장기적인 생태계 펀드에, 10%는 초기 기여자에게 배분됩니다.
Consensys 트레저리	15%	5년간 잠금(양도 불가) 상태로 보관됩니다. Consensys는 Linea 및 그 생태계를 지원하고 정렬하기 위해 토큰을 장기적으로 보유합니다.

Consensys에 대한 초기 할당(5년간 잠금)을 제외하고는 **개인 투자자, 내부자, 직원 및 팀 멤버에게는 어떠한 할당도 없습니다.** 대신, 가치는 곧바로 생태계, 이를 실현하는 빌더들, 그리고 여기에 참여하는 커뮤니티 멤버들에게 전달됩니다.

토큰 생성 이벤트(TGE) 시점에는 LINEA 공급량의 22%가 유통되며, 이는 사용자 및 빌더 에어드롭, 생태계 활성화, 유동성 공급으로 분배됩니다.

기타 모든 할당은 TGE 시점에 잠기거나 다양한 베스팅 일정에 따라 분배됩니다.

생태계 (85%)

초기 기여자 (10%)

초기 기여자에 대한 최초의 에어드롭은 전체 공급량의 10%로 구성되며, 다음과 같이 배분됩니다:

- 9%는 초기 기여자에게
- 1%는 전략적 빌더에게

사용자에 대한 초기 기여자 에어드롭은 LXP를 포함한 장기적인 온체인 활동을 기반으로 진행됩니다. TGE 전에 자격 확인 도구가 제공될 예정입니다. 빌더에 대한 1% 할당은 최대 효과를 내기 위해 큐레이션된, 목표 지향적인 과정을 통해 분배됩니다.

위 에어드롭을 통해 분배되는 모든 토큰은 완전히 언락된 상태입니다.

생태계 펀드 (75%)

생태계 펀드의 장기적 부분은 Linea 컨소시엄(Linea Consortium)에 의해 관리됩니다.

이 75%는 10년에 걸쳐 언락되며, 조기 활성화와 채택을 장려하기 위해 초기에 더 높은 비중으로 언락됩니다. 초기에는 매년 10%씩 언락되며, 10년 차에는 2%까지 점진적으로 줄어듭니다.

이 자금은 다음과 같은 용도로 사용됩니다:

- 이더리움 연구 및 개발(R&D) 자금 지원
- 감사(audits), 개발자 도구, 노드 인프라 등 공유 생태계 인프라 유지
- 오픈소스 소프트웨어, 연구, 커뮤니티 프로그램과 같은 퍼블릭 굿(public goods) 자금 지원
- 연계된 조직이나 신흥 프로토콜과의 전략적 공동 개발

Consensys 트레저리 (15%)

총 토큰 공급량의 15%는 Consensys 트레저리에 포함됩니다. 이 할당은 5년간 잠금되며, 베스팅 일정이 완료될 때까지 양도가 불가능합니다.

소각 메커니즘(Burning mechanism)

Linea와 LINEA 토큰은 이더리움과 ETH의 강점을 강화하도록 특별히 설계되었으며, 이는 자동화된 이중 소각 메커니즘(dual-burn mechanism)을 통해 달성됩니다:

- Linea에서 모든 가스 수수료는 ETH로 지불됩니다.
- 순 ETH 수익(Linea 수익 - 운영 비용 및 보조금)의 20%는 소각되어 ETH 공급을 줄입니다.
- 순 ETH 수익의 80%는 LINEA를 소각하는 데 사용되어 LINEA 공급을 줄입니다.

이 결과, Linea의 네트워크 활동은 ETH와 LINEA 두 자산의 가치를 직접적으로 지원합니다.

트랜잭션 생애주기(Transaction lifecycle)

파이널리티(Finality)

파이널리티는 Linea에서 두 가지 정의를 가집니다:

- **소프트 파이널리티(Soft finality):** 트랜잭션이 Linea에서 확인된 상태입니다. 이는 2초, 즉 Linea의 블록 시간에 해당합니다. 3단계를 참조하십시오. 단순화를 위해, Linea는 L1에서 리오그(reorg)가 발생하더라도 리오그—경쟁하는 블록체인 기록을 제거하고 정식 기록을 선택하는 행위—가 발생하지 않음을 보장합니다.
- **하드 파이널리티(Hard finality):** L1에 제출된 증거가 검증되고 두 개의 에포크(epoch)가 경과한 상태입니다. 일반적으로 하드 파이널리티에 도달하기까지의 시간은 6~32시간이며, 향후 최소 6시간은 0으로 줄어들 예정입니다. 6단계를 참조하십시오.

Linea에서의 트랜잭션은 다음 단계들을 거칩니다:

Step 1: 제출(Submission)

모든 트랜잭션은 여기서 시작됩니다. 사용자가 지갑에서 트랜잭션에 서명하고 이를 Linea로 브로드캐스트합니다. 이후, 트랜잭션은 이더리움과 마찬가지로 **메모리풀(mempool)**로 바로 전달되며, 이곳에서 대기(pending) 상태의 트랜잭션이 됩니다.

Step 2: 블록 생성(Block building)

Linea의 **시퀀서(sequencer)**는 블록의 정렬, 생성, 실행을 담당합니다. 메모리풀(mempool)에 추가되는 각 트랜잭션에 대해 시퀀서는 유효성을 검사하며, 필요한 경우 트랜잭션을 거부합니다. 트랜잭션 유효성 조건은 Linea에 특화되어 있으며, 이더리움을 포함한 다른 네트워크와는 약간 다릅니다.

시퀀서는 각 트랜잭션이 지불한 우선순위 수수료(priority fee)에 따라 트랜잭션을 정렬하는데, 이를 우선 순위 가스 경매(priority gas auction)라고 합니다. 위 검증을 통과한 유효한 트랜잭션은 올바른 순서대로 블록에 배치되어 실행됩니다.

이 시점에서 트랜잭션의 생애주기는 사용자 관점에서 사실상 완료됩니다. 트랜잭션이 포함된 블록은 Linea 블록체인의 "헤드(head)" — 즉, 가장 최근 블록 —에 추가되며, 사용자 지갑에는 확인용으로 트랜잭션 영수증(transaction receipt)이 반환됩니다.

그러나 트랜잭션과 관련 데이터는 ZK 증명을 생성하기 위해 계속 처리됩니다. 계속 살펴보겠습니다.

Step 3: 트랜잭션 데이터가 상태 관리자(state manager)로 전송됨

트랜잭션과 실행 시점의 네트워크 상태에 관한 데이터는 *트레이스(trace)*에 기록되며, 이는 시퀀서의 일부인 *트레이스 제너레이터(trace generator)*의 출력물입니다.

트레이스는 블록 단위로 상태 관리자(state manager)에 전달되고, 네트워크 상태를 업데이트하는 데 사용됩니다. 상태가 최신으로 업데이트되면, 사용자의 지갑에서 해당 트랜잭션이 반영되고 확인되는 것을 볼 수 있습니다.

트랜잭션이 실행되고 상태가 업데이트되면, 트랜잭션은 **소프트 파이널리티(soft finality)**에 도달합니다. Linea 체인만 놓고 보면, 트랜잭션은 완료된 것입니다. 하지만 알다시피, Linea와 같은 L2는 독립적으로 작동하지 않으므로, 진정한 파이널리티에 도달하기까지 더 많은 작업이 필요합니다.

Step 4: 컨플레이션(Conflation)

이후 트랜잭션이 포함된 블록은 *컨플레이션(conflation)* 과정을 거치게 되며, 이 과정에서 두 개 이상의 블록 트랜잭션 데이터가 하나의 데이터 셋(배치, batch)으로 결합됩니다. 이 데이터 셋은 이더리움으로 전달되는 데이터 패키지의 일부를 구성합니다. 여러 블록의 트랜잭션 데이터를 결합하면, 단일 증명으로 대량의 트랜잭션을 검증할 수 있어 L1에 증명을 제출하는 비용을 최소화할 수 있습니다.

때때로, 배치가 하나의 블록만 포함할 수 있으며, 이 경우 컨플레이션이 발생하지 않습니다. 이는 체인 활동 수준이 특히 높아 블록 크기가 평소보다 큰 경우에 발생합니다.

이더리움의 L2 네트워크 중에서 Linea는 유일하게 배치 컨플레이션(batch conflation)을 사용하는 네트워크입니다.

Step 5: 트랜잭션 데이터를 사용한 ZK 증명 생성

트랜잭션의 트레이스(trace) 데이터가 포함된 블록이 하나 이상의 블록과 함께 배치(batch)로 컨플레이트(conflate)된 이후, **하드 파이널리티(hard finality)**에 도달하기 위해 남은 유일한 작업은 해당 트랜잭션의 데이터(트레이스에 포함된)를 사용하여 증명을 생성하는 것입니다.

코디네이터(Coordinator)의 지시에 따라, Linea의 프로버(prover)는 먼저 트레이스를 **확장(expand)** 하여 증명에 포함될 준비를 합니다. Linea의 프로버는 두 단계 방식을 사용하여 L1로 전달되는 증명을 개발합니다. 먼저 **내부 증명(inner proof)**을 생성한 후, **외부 증명(outer proof)**을 생성합니다.

내부 증명은 Arcane과 Vortex를 포함한 여러 도구를 사용하여 증명 크기를 재귀적으로 줄입니다. Linea의 내부 증명 시스템에 대한 자세한 내용은 [이 문서](#)를 참조하십시오.

그 다음, Consensys에서 관리하는 라이브러리 gnark을 사용하여 외부 증명을 생성하며, 증명 크기를 더욱 압축합니다. 결과적으로 생성된 증명이 바로 zk-SNARK이며, 이는 최종적으로 이더리움에 제출되는 증명입니다.

모든 블록의 모든 트랜잭션의 트레이스 데이터가 최종 증명 생성에 사용되므로, 단일 트랜잭션은 3단계에서 소프트 파이널리티에 도달한 이후에도 Linea의 기능에 여전히 중요한 역할을 합니다.

Step 6: 배치 파이널라이제이션(Batch finalization)

프로세스의 마지막 단계는 배치를 이더리움 메인넷에 제출하여, 그 계산적 무결성을 증명하는 것입니다. 컨플레이트된 블록들의 배치는 트랜잭션으로 구성되어 있으므로, Linea가 다루던 트랜잭션도 이 과정에 포함됩니다.

L1에 제출되는 두 가지 요소를 구체적으로 살펴보겠습니다:

- 위에서 설명한 증명(proof)
- **블롭(blob) 데이터**: L2 트랜잭션 및 메시징 데이터를 포함합니다. 블롭 데이터가 L1에 공유되면, 이는 Linea의 상태를 재구성하고 증명의 내용을 검증하는 데 사용됩니다. 블롭 데이터는 약 4,096 에포크(약 18일)가 지나면 사라집니다. L1에 있는 Linea 롤업 컨트랙트는 블롭 데이터를 사용하여 이더리움 검증기(verifier) 컨트랙트를 호출하며, 해당 배치를 유효한 것으로 수락할지 여부를 결정합니다.

CALLDATA

Linea Alpha v3 이전까지, L2 트랜잭션 데이터는 트랜잭션에 포함된 `calldata`를 통해 L1로 전송되었습니다. 이 방식은 효과적이었지만 비용이 많이 들었으며, 이제 이더리움의 Dencun 업그레이드에서 추가된 EIP-4844를 통해 가능해진 블롭(blob)-전송 트랜잭션으로 대체되었습니다.

최종화된 배치는 Lineascan에서 [여기](#)서 확인할 수도 있습니다.

증명이 검증되고 두 개의 에포크가 지나면, 해당 트랜잭션은 불변의 기록이 되어 **하드 파이널리티(hard finality)**에 도달합니다. 이로써 트랜잭션의 생애주기는 완전히 종료됩니다.

네트워크 데이터(Network data)

무엇입니까?

퍼블릭 블록체인 네트워크의 주요 가치 제안 중 하나는 그것이 *공개적(public)*이라는 점입니다. 이는 네트워크에서 무슨 일이 일어나고 있는지에 대한 정보가 쉽게 접근 가능해야 함을 의미합니다. 이더리움과 같은 네트워크는 이러한 기능이 소프트웨어에 내장되어 있습니다. 각 노드는 API를 통해 요청한 모든 정보를 반환할 수 있으며, 요청량이 많다면 단순히 더 많은 노드를 실행하면 됩니다.

분명히 말하자면, Linea도 동일한 방식으로 작동합니다. 단, 이더리움은 탈중앙화되어 있으며 누구나 노드를 실행할 수 있습니다. 반면, Linea는 아직 테스트넷 상태이며, 노드는 Consensys 팀에 의해만 운영되고 있습니다. 따라서 네트워크가 만들어낸 막대한 관심을 지원하기 위해 많은 수의 노드가 운영되어야 합니다. 이를 통해 dapp이 네트워크 상태에 대한 정보를 요청하더라도 클라이언트를 과부하시키지 않고 실제 트랜잭션 실행을 방해하지 않을 수 있습니다.

무엇을 합니까?

사용자 및 dApp의 요청을 Ethereum JSON-RPC API 표준에 따라 수신하고 응답합니다. 여기에는 MetaMask 사용자에게 Linea에서 자신의 계정에 대한 정보를 제공하는 것도 포함됩니다.

어떻게 합니까?

Infura, Consensys, 그리고 전체 이더리움 생태계의 전문성과 자원을 활용합니다. Infura는 이 서비스를 제공하기 위해 여러 개의 노드를 운영하고 있으며, 이는 두 가지 주요 역할을 수행합니다:

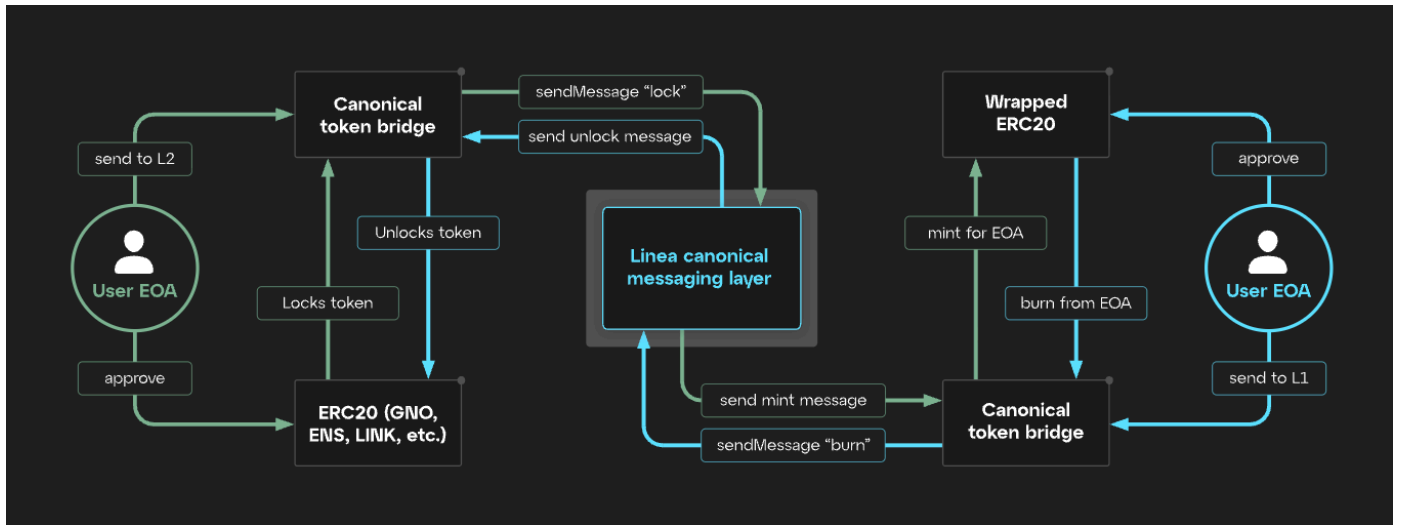
클라이언트 지향 RPC-API 노드(Client-facing RPC-API nodes)

- 이 노드들은 EVM 노드의 “전통적인” 역할을 수행합니다. 한편으로는 시퀀서와 상태 관리자(state manager)로부터 업데이트된 네트워크 상태 정보를 수신하고, 사용자와 dapp이 요청할 때 해당 상태에 대한 정보를 제공합니다. 다른 한편으로는 들어오는 트랜잭션을 수신합니다.
- 이러한 트래픽을 처리하기 위해 Infura는 로드 밸런서(load balancer) 뒤에서 노드를 운영하며, 블록체인 네트워크 확장에 대한 전문성을 운영 전반에 적용합니다. 따라서 사용자가 제출한 트랜잭션은 노드들에 균형 있게 분산되며, 이후 메인풀(mempool)로 라우팅되어 Coordinator와 linea-besu의 시퀀서가 처리하게 됩니다.

아카이브 노드(Archive nodes)

- 점점 더 많은 dapp과 연구 활동이 현재 트랜잭션 데이터뿐 아니라 과거 데이터(어제, 지난달, 혹은 3년 전의 데이터 — 물론 Linea에서는 아직 그렇게 오래되지 않았습니다)까지 다루고 있습니다.
- 이러한 유형의 데이터 요청은 상당한 자원이 소모될 수 있으며, 실시간 네트워크 노드의 성능에 영향을 주어 네트워크 전체의 건전성을 위협할 수 있습니다. 이러한 이유로 Infura는 확장 가능한 아키텍처로 아카이브 노드를 배치했습니다. 이러한 요청이 *항상* 발생하는 것은 아니지만, 발생할 때에는 수요를 감당하기 위해 더 많은 노드를 필요에 따라 확장해 가동합니다.

캐노니컬 토큰 브릿지(Canonical token bridge)



캐노니컬 토큰 브릿지는 모든 ERC-20 토큰의 브리징을 가능하게 하는 “락(lock) & 민트(mint)” 컨트랙트 쌍입니다. 이 브릿지는 크로스체인 상호작용을 위해 [메시지 서비스](#)에 의존합니다.

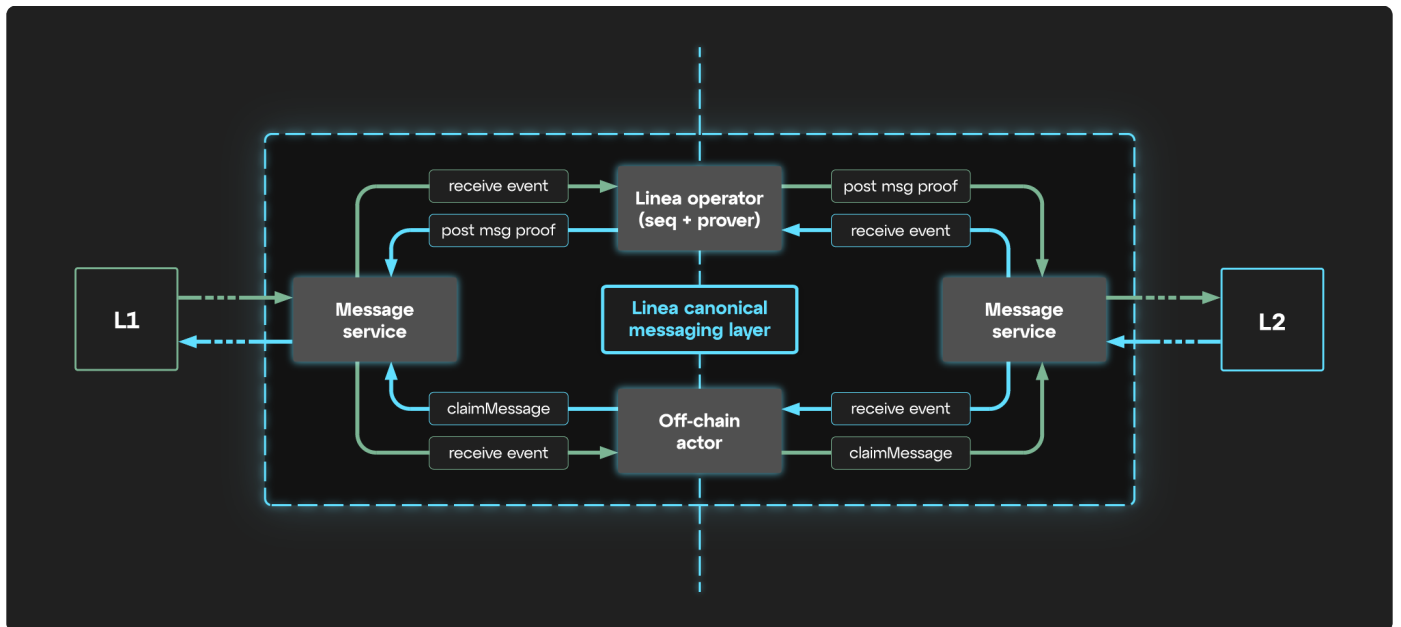
Linea 팀은 [공식 Linea 브릿지](#)의 UI를 운영합니다.

캐노니컬 토큰 브릿지는 Linea에 배포하는 기술적 파트너를 위해 최적화되어 있습니다. 개인 토큰을 네트워크 간에 브릿지하려는 일반 사용자는 네트워크에 배포된 여러 브릿지 중 하나를 활용할 것을 권장합니다.

Linea는 권한 없는(permissionless), 탄력적(resilient), 탈중앙화된 환경을 육성하는 것을 목표로 하며, 브릿지가 중앙화된 중재자나 단일 실패 지점(point of failure)이 되는 것을 원하지 않습니다.

현재 Linea에서 운영 중인 브릿지를 확인하려면 [생태계 포털](#)로 이동하여 **Bridge** 버튼을 클릭하면 사용 가능한 모든 브릿지를 볼 수 있습니다.

캐노니컬 메시지 서비스(Canonical message service)



무엇입니까?

캐노니컬 메시지 서비스는 스마트 컨트랙트와 기타 프로토콜을 조합하여, Linea와 다른 네트워크 간에 “임의의 메시지(arbitrary messages)” — 즉, 사용자가 지정한 데이터 —를 전달하는 역할을 합니다.

무엇을 합니까?

두 블록체인 간의 브릿지를 사용해 본 적이 있다면, 다소 제한적인 경험을 떠올릴 수 있습니다. 예를 들어 특정 토큰만 전송할 수 있는 경우가 많습니다. 그러나 캐노니컬 메시지 서비스 자체는 최종 사용자 브릿지 인터페이스와는 다릅니다. 이는 데이터와 자산을 하나의 블록체인에서 다른 블록체인으로 권한 없이(permissionlessly) 그리고 신뢰성 있게 전송할 수 있는 시스템입니다. 서비스 전체는 한 네트워크에서 다른 네트워크로 무언가를 옮기려는 요청을 수신하고, 해당 요청을 실행하여 도착지 네트워크의 기존 스마트 컨트랙트에 제출된 메시지를 전달합니다.

메시지 서비스가 전송하는 가장 중요한 정보 중 하나는 이더리움 네트워크의 현재 상태에 관한 정보입니다. 이더리움에서 Linea로, 그리고 반대로 Linea에서 이더리움으로는 네트워크 활동을 보고할 때마다 업데이트된 머클 트리(Merkle tree)와 zk-증명이 전송됩니다. 다시 말해, 캐노니컬 메시지 서비스는 롤업 데이터를 전송하는 역할을 합니다.

그러나 이 서비스는 Linea의 핵심 기능에만 제한되거나 국한되지 않습니다. 이는 일반 목적의 퍼블릭 인프라로, 개발자가 사용할 수 있고, dapp에 통합될 수 있으며, 최종 사용자가 직접 실행할 수도 있습니다.

어떻게 합니까?

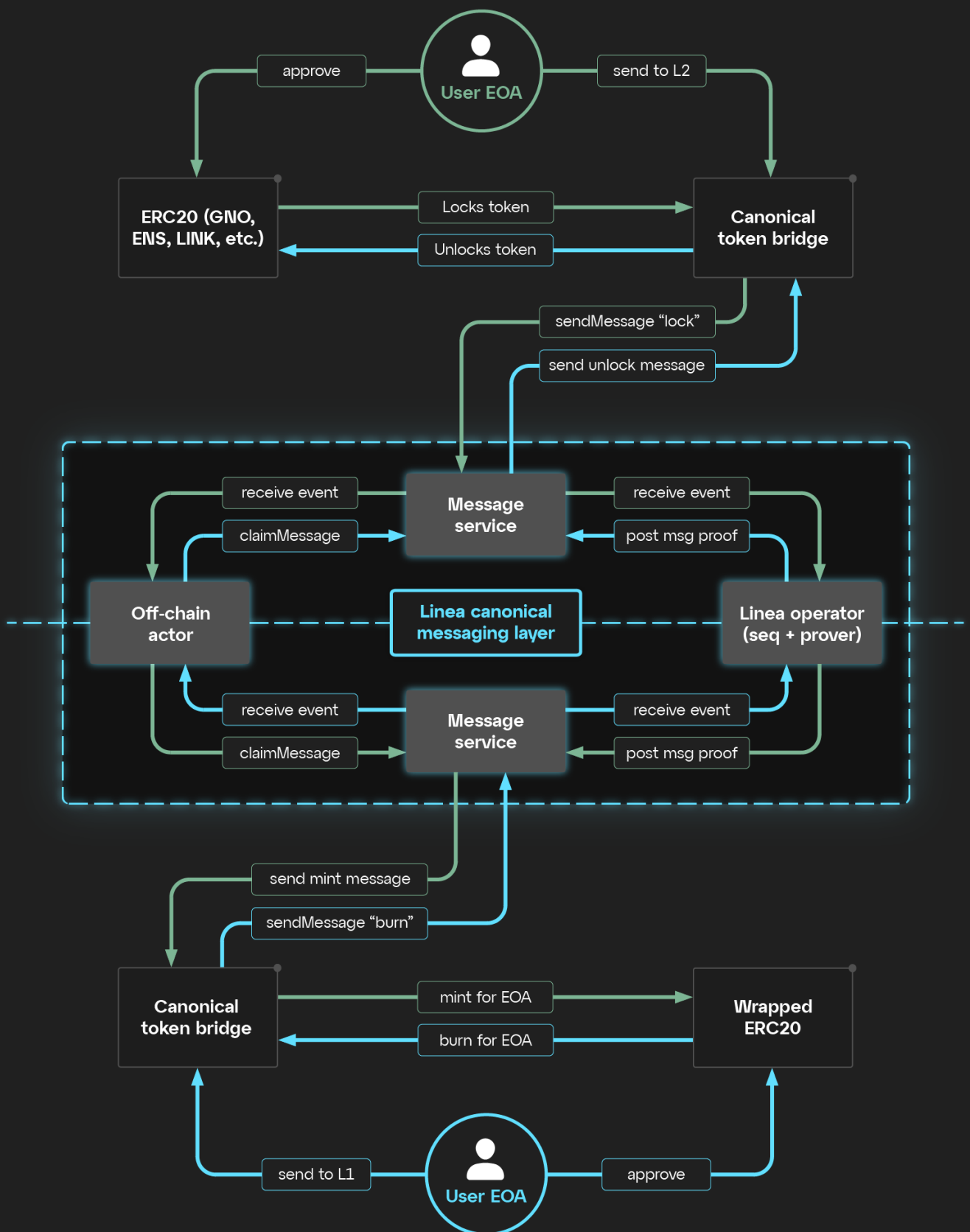
캐노니컬 메시지 서비스는 세 가지 주요 요소로 구성됩니다: 두 개의 스마트 컨트랙트와 그 사이에서 동작하는 포스트맨(postman) 서비스입니다. 이미 예상했듯이, 스마트 컨트랙트는 Linea와 이더리움에 존재하며 거의 동일합니다. 예를 들어, 이 컨트랙트들은 대상 네트워크에서 ETH를 발행(mint)할 수 있도록 하지만, 그 기능은 이에만 국한되지 않습니다.

사용자는 컨트랙트의 메서드 중 하나를 실행하여 네트워크 간 전송을 시작합니다. 즉, 스마트 컨트랙트에 내장된 함수를 호출하는 것입니다. 사용자는 스마트 컨트랙트와 직접 상호작용하는 방법을 알고 있다면 이를 직접 수행할 수 있고, 프론트엔드를 통해서도 수행할 수 있습니다. 요청이 올바르게 작성되면 스마트 컨트랙트는 사용자의 요청을 수락하고 이를 포스트맨에게 전달합니다.

포스트맨(postman) 서비스는 현재 중앙화되어 있으나, 향후 탈중앙화될 예정입니다. 포스트맨은 Linea 또는 이더리움의 컨트랙트 중 하나에 호출이 발생하는 것을 “청취(listen)”하고, 제출된 정보를 다른 네트워크로 전달합니다.

정보가 목적지 스마트 컨트랙트에 도착하면, 요청에 포함된 코드가 실행됩니다. 만약 전송된 메시지가 토큰 발행 명령을 포함하고 있다면, 사용자는 전송된 자산을 브릿지의 목적지 쪽에서 수동으로 *끌어내기(pull)*하거나, 선불 결제를 통해 자산을 목적지 주소로 직접 *밀어넣기(push)* 할 수 있습니다.

여기에 추가적인 로직 계층이 존재하며, 이는 L2로 전달된 메시지가 유효한지 검증하는 역할을 합니다. 기본적으로, 메시지는 이더리움에서 전송되어 메시지 서비스를 통해 릴레이되며, Linea 스마트 컨트랙트에 전달됩니다. 해당 스마트 컨트랙트는 수신된 메시지를 이더리움 측에서 전송된 메시지 목록과 대조하여, 그것이 L1에 존재함을 검증한 뒤에만 이를 유효한 메시지로 받아들입니다:



코디네이터(Coordinator)

무엇입니까?

코디네이터는 Linea의 여러 기능을 조율하는 모듈입니다. 또한, 이더리움의 상태 정보가 Linea로 들어오는 통로이자, Linea의 상태 정보가 다시 이더리움으로 돌아가는 통로이기도 합니다.

무엇을 합니까?

코디네이터는 Linea 실행 클라이언트의 서로 다른 부분들 사이에서 내부적으로, 그리고 다른 블록체인, Linea의 데이터 가용성 스택, 그리고 네트워크 상태를 동기화하는 노드들과 외부적으로 정보를 이동시킵니다. 이러한 방식으로, 코디네이터는 **Linea의 합의 클라이언트(consensus client)** 역할을 수행합니다.

어떻게 합니까?

코디네이터는 고도로 모듈화된 내부 아키텍처를 통해 이러한 역할을 수행합니다. 다양한 시스템과 요구사항이 각각의 독립된 환경으로 나누어져 있으며, 각 환경은 코디네이터로부터 입력을 받아 처리한 후 그에 상응하는 출력을 다시 코디네이터로 반환합니다.

시퀀서(Sequencer)

무엇입니까?

시퀀서는 Linea 실행 클라이언트의 핵심으로, **블록을 정렬, 생성, 실행**하여 이후의 영지식 증명(zero-knowledge proof)이 만들어질 수 있도록 하는 역할을 담당합니다.

무엇을 합니까?

시퀀서는 여러 가지 기능을 수행합니다:

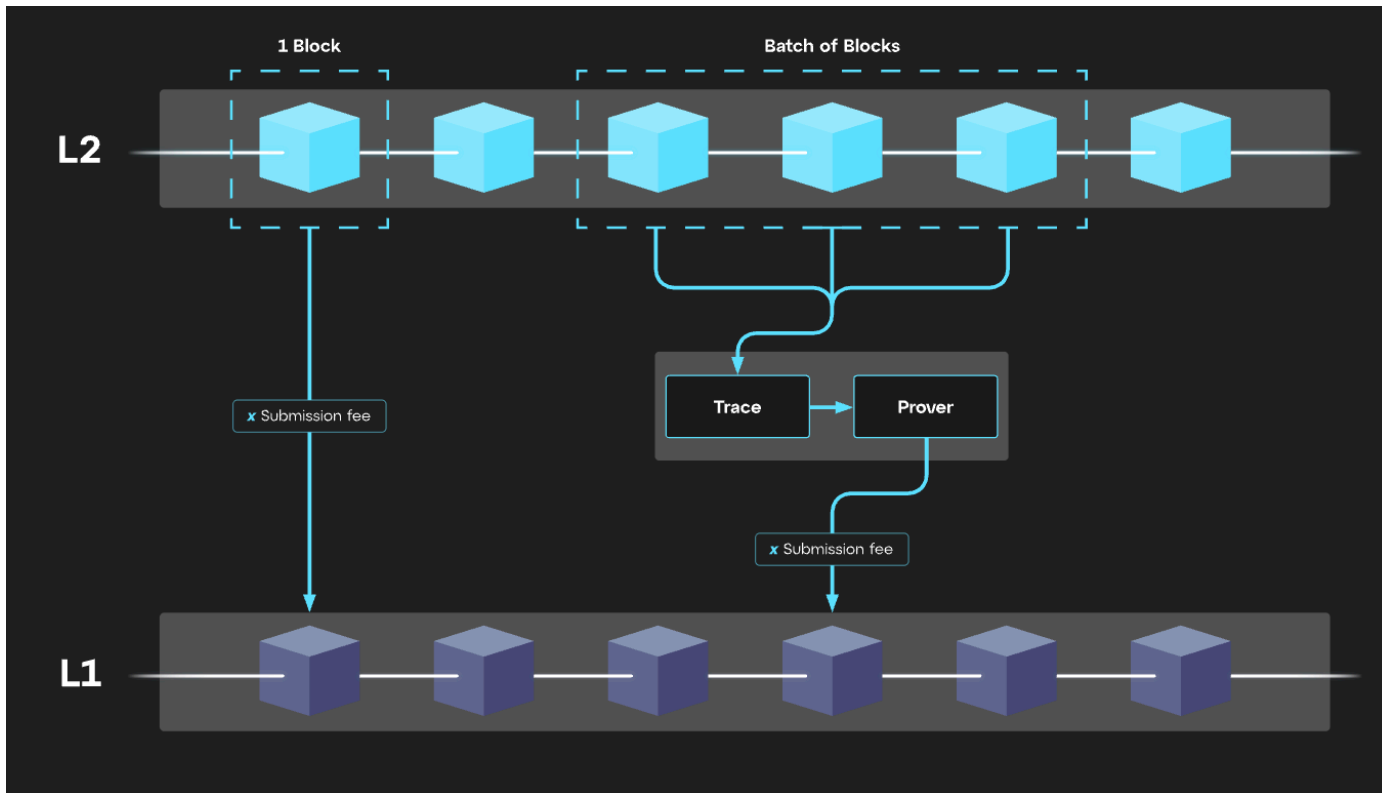
- 트랜잭션이 실행될 순서 — 즉, **시퀀스(sequence)** — 를 결정합니다.
- 트랜잭션을 블록으로 구성합니다.
- 블록을 실행합니다.
- 실행의 **트레이스(trace)**와 관련된 특정 데이터를 영지식 프로버(prover)를 위해 준비합니다.

어떻게 그것을 합니까?

Linea의 실행 클라이언트는 Linea Besu이며, 이는 Besu 실행 클라이언트를 기반으로 Linea와 그 요구 사항을 지원하기 위한 추가 기능을 확장한 구현체입니다. 직접 Linea Besu 노드를 실행할 수도 있습니다.

Linea의 시퀀서는 메모리 풀(memory pool)에서 트랜잭션을 가져와 블록으로 구성합니다. 그러나 Linea에서는 추가적인 작업과 코디네이터(coordinator)와의 통신을 수행하여, 블록이 영지식 프로버에 의해 증명될 수 있는 방식으로 생성되도록 하고, 가능한 한 압축되도록 합니다. 이는 모든 데이터가 이더리움 메인넷에 작은 단위로, 높은 비용을 들여 기록되어야 하는 상황에서 두 배로 중요한 요소입니다. 이러한 작업은 시퀀서 내의 하위 시스템, 즉 트레이스 제너레이터(trace generator)와 컨플레이터(conflator)에 의해 수행됩니다.

컨플레이션(Conflation)



무엇입니까?

컨플레이션은 두 개 이상의 블록에 포함된 트랜잭션을 하나의 데이터 셋으로 결합하는 과정입니다. 이렇게 결합된 데이터 셋은 네트워크 상태의 ‘이전과 이후(before and after)’ 맵(기술적으로는 머클 트리, Merkle tree)과 이더리움에 게시되는 영지식 증명을 생성하는 데 사용됩니다.

무엇을 합니까?

컨플레이션은 일반적으로 메인넷 이더리움 환경에서는 볼 수 없습니다. 메인넷에서는 각 블록이 순차적으로 하나씩 별도로 게시되어야 하기 때문입니다. 반면 zkEVM 환경에서 이더리움의 관점에서 ‘진실의 원천 (source of truth)’은 제출된 데이터 — 즉, ZK 증명, 그 증명이 검증한 트랜잭션 목록, 그리고 머클 트리 — 입니다. 이는 “하나의 블록에 몇 개의 트랜잭션을 담을 수 있는가”의 문제가 아니라, “하나의 증명에 몇 개의 트랜잭션을 담을 수 있는가”의 문제로 바뀝니다. 여러 블록을 하나로 컨플레이트함으로써, Linea의 증명 시스템은 훨씬 더 효율적으로 작동할 수 있습니다.

어떻게 합니까?

컨플레이션은 실행 클라이언트 내부에서 발생하지만, 코디네이터(Coordinator)와의 통신 과정을 통해 수행됩니다:

- 컨플레이터(conflator)는 트레이스 파일(traces file)이 나타나기를 기다립니다.
- 해당 블록의 트레이스를 “병합됨(merged)”으로 표시합니다.
- 더 많은 블록 데이터가 들어올 가능성에 대비해 일정 시간을 기다립니다.
 - 추가 데이터가 들어오면, 블록 내 라인의 수와 데이터 길이가 제한을 초과하는지 확인합니다.
 - 제한 내라면, 컨플레이터는 해당 블록 역시 병합된 것으로 표시합니다.

컨플레이터는 이 사이클을 시간 제한에 도달할 때까지 반복하며, 이후 컨플레이트된 트레이스 데이터를 코디네이터에 전달합니다. 이 데이터는 이후 머클 트리(Merkle tree) 및 증명 생성에 사용됩니다.

트레이스 생성(Traces generation)

무엇입니까?

Linea의 시퀀서에서 프로버(prover)가 ZK 증명을 생성하는 데 사용하는 데이터를 만들어내고, 이를 가능한 한 압축하는 역할을 담당하는 부분입니다.

무엇을 합니까?

트레이스 제너레이터(trace generator)는 시퀀서가 생성한 블록을 실행하고, 각 트랜잭션의 트레이스와 관련된 데이터를 보존합니다.

어떻게 합니까?

시퀀서가 블록을 생성하면, 이 블록들은 실행되며 그 과정에서 EVM은 *트레이스(traces)*라고 불리는 데이터를 생성합니다. 이 트레이스는 각 트랜잭션 실행의 세부 단계마다 네트워크의 상태와 트랜잭션에 관련된 계정의 상태를 지정합니다. 이것이 바로 Linea와 그 영지식 기술의 핵심에 자리 잡은 “무한 불가능성 드라이브(Infinite Improbability Drive)”입니다. 이 트레이스들이야말로 프로버가 증명을 생성하는 데 사용할 데이터입니다. 그 증명을 통해 이더리움은 Linea에서 발생한 모든 일이 실제로 일어났음을 알 수 있으며, 그 구체적인 내용 자체는 알 필요가 없습니다.

Linea의 시퀀서는 이러한 트레이스를 추가적인 과정에 통과시킵니다: **트레이스 컨플레이션(trace conflation)**.

EVM 상태 관리자(EVM state manager)

상태 관리자는 실행 클라이언트의 일부로, 네트워크의 전역 상태를 업데이트하고 각 계정의 개별 상태를 업데이트하는 역할을 담당합니다. 또한 상태 관리자는 EVM에서 수행되는 “읽기(read)” 접근을 감사(audit)하며, 즉 EVM이 블록체인 상태에서 데이터를 읽어야 하는 모든 작업을 모니터링, 검증, 기록합니다.

INFO

“상태(state)”란 특정 시점에 블록체인에 저장된 데이터를 의미합니다. 상태를 업데이트한다는 것은, 변경된 계정들의 내용을 기록에 반영한다는 의미입니다.

상태 관리자의 주요 임무는 시퀀서가 실행한 블록을 수신하고, 그 실행에서 나온 트레이스 데이터를 활용해 네트워크 상태를 업데이트하는 것입니다. Linea는 상태를 관리하기 위해 두 가지 데이터 구조를 사용합니다:

1. 세계 상태(world state)를 기록하고, 합의를 유지하며, 블록을 처리하기 위해 머클-패트리시아 트라이(Merkle-Patricia Trie)를 사용합니다. 이는 이더리움 메인넷에서 합의와 상태를 관리하는 방식과 유사합니다.
2. 일반적인 머클 트리의 변형인 스파스 머클 트리(Sparse Merkle Tree, SMT)를 사용하여, 계정을 나타내는 스토리지 슬롯을 보다 효율적으로 추적, 관리, 업데이트합니다.

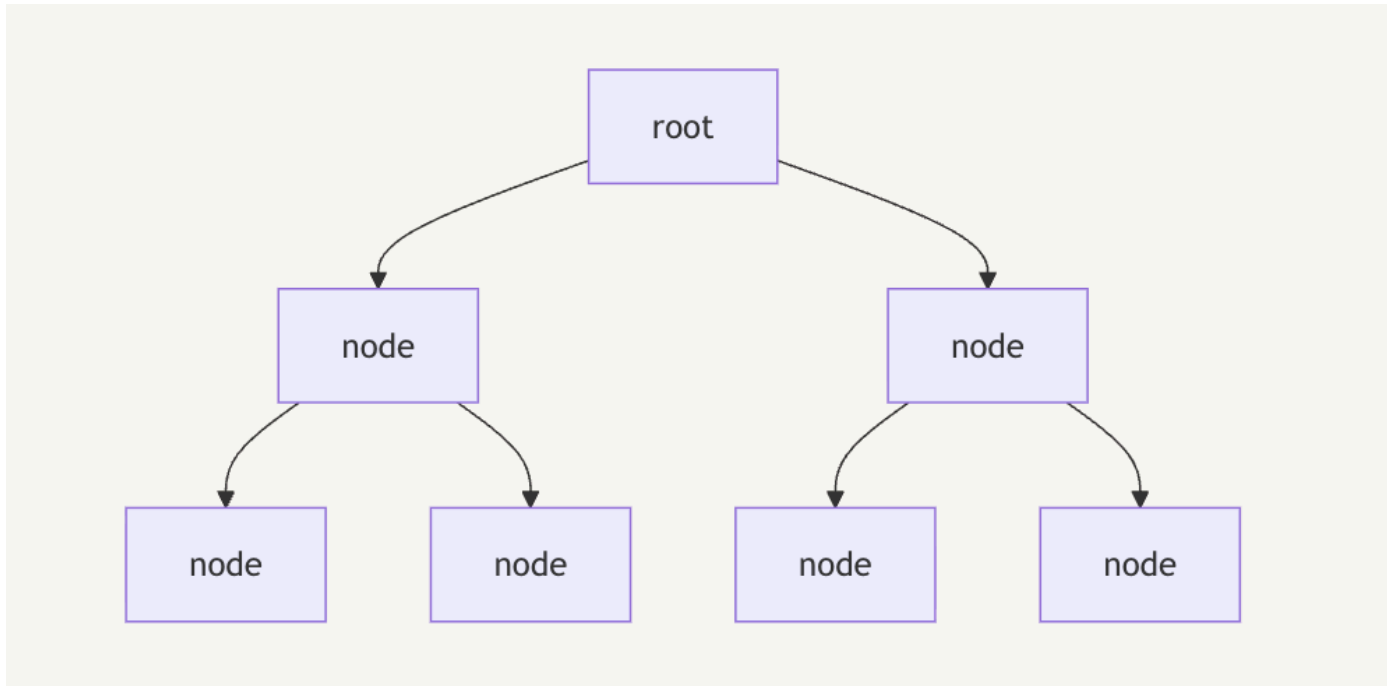
이후 상태 관리자는 업데이트된 네트워크 상태 정보를 머클 증명(Merkle proofs)의 형태로 프로버에게 전달하여, 이더리움 메인넷(L1)에 제출하도록 합니다.

아래에서는 Linea의 상태 관리 요소를 더 자세히 설명하며, 특히 Linea를 차별화하는 SMT 구성에 초점을 맞추겠습니다.

머클 트리(Merkle trees)

머클 트리와 그 변형들은 블록체인에서 각 계정의 상태에 관한 데이터를 저장하고 검색하는 데 EVM 체인 전반에서 일반적으로 사용됩니다.

머클 트리는 서로 가지(branch)로 뿔어 나가는 ‘노드(nodes)’로 구성됩니다. 그 기초(base)에는 ‘루트(root)’ 또는 상태 루트(state root)가 있으며, 여기서 가지들이 뿔어 나가고, 그 가지들에서 다시 잎(leaf)이 파생됩니다.



각 노드는 종류와 관계없이 암호학적 해시(cryptographic hash)로 표현되며, 이는 계정의 잔액과 같은 해당 노드의 속성 데이터를 인코딩합니다. 각 해시는 자식 노드들의 해시를 포함합니다. 이 계단식(cascading) 시스템을 전체적으로 확장하면, 루트는 블록체인 상의 모든 계정 상태 데이터를 인코딩하게 됩니다.

암호학적 해시는 결정적(deterministic)이므로, 해시 함수의 결과로부터 원래 데이터를 역으로 추론할 수 있습니다. 부모가 없는 유일한 노드인 루트의 해시를 보유하고 있다면, 이론적으로 트리 전체의 어떤 노드 데이터라도 도출할 수 있습니다.

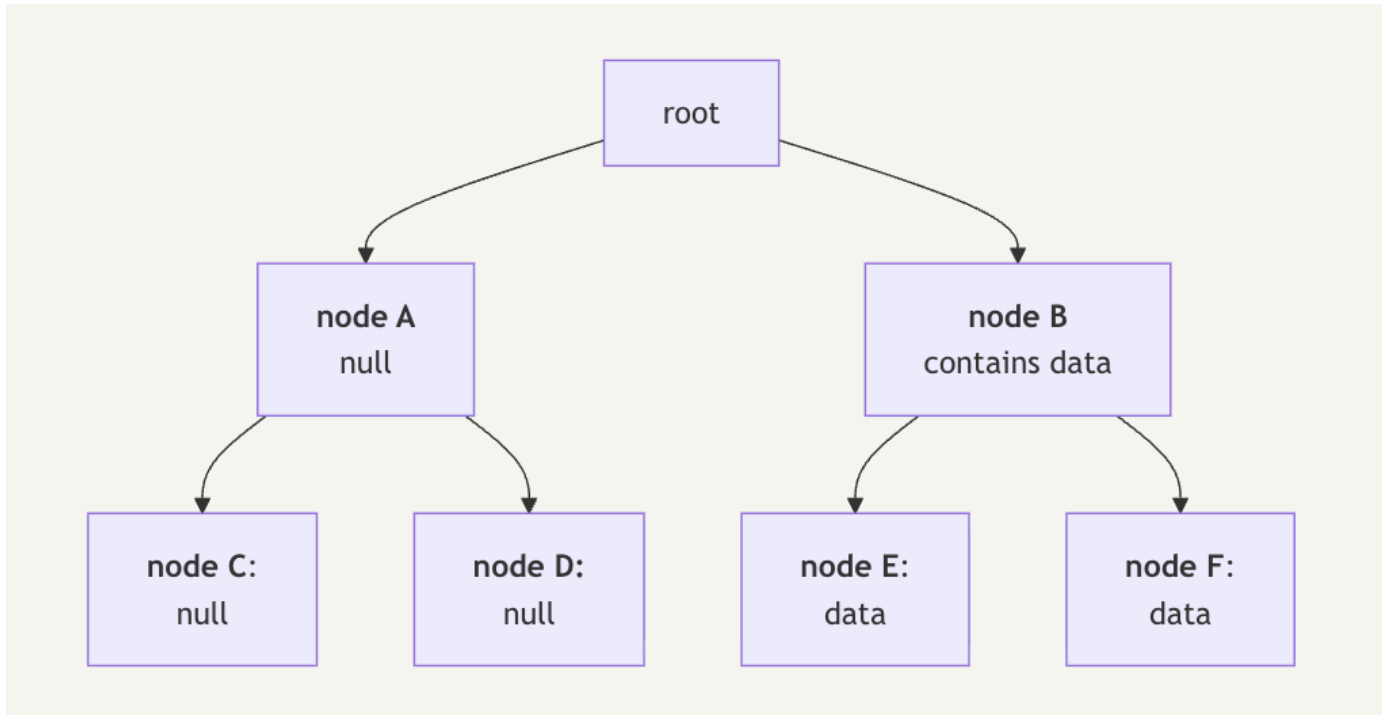
레이어 2(L2) 네트워크로서 Linea는 트랜잭션을 더 빠르고 효율적으로 처리하는 데 초점을 맞추고 있습니다. Linea는 계정 상태를 추적하고 증명을 생성 및 저장하기 위해 스파스 머클 트리(sparse Merkle tree)를 구현합니다. 이는 매 블록마다 재계산이 필요해 과도한 계산 부담을 유발하는 표준 머클 트리에 비해 더 높은 효율성을 제공합니다.

스파스 머클 트리(Sparse Merkle trees)

Linea의 상태 관리는 스파스 머클 트리를 사용하여 연산량을 최소화하고 블록체인의 효율성을 높입니다.

스파스 머클 트리는 표준 머클 트리의 변형으로, 모든 리프 노드(leaf node)가 데이터로 채워지는 것이 아니라 필요한 노드에만 데이터가 저장됩니다. 이는 고정된 깊이를 가지는 완전 이진 트리로, 트리의 모든 가지(branch)는 동일한 길이, 즉 동일한 리프 수를 가집니다.

초기화 시점 — 즉, 체인의 기록이 시작될 때 — 모든 리프 노드는 기본값으로 설정되며, 일반적으로 0과 같은 특정 값의 해시입니다. 모든 리프 노드가 동일한 해시 값을 가지기 때문에, 부모 노드 및 더 상위 노드 역시 동일한 해시 값을 갖습니다. 따라서 특정 레벨에서 해시가 기본값인 노드는 *비어 있는(empty)* 서브트리를 나타내는 것으로 간주됩니다.



위 예시에서 노드 A의 자식 노드(리프)들은 null 값을 포함하고 있으므로, 노드 A 역시 null을 나타냅니다. 반면, 노드 B는 그 자식 노드들이 값을 포함하고 있음을 반영합니다.

이러한 구조 덕분에 모든 개별 노드의 해시를 추적할 필요가 없습니다. 대신, 기본값을 반영하는 해시는 비어 있는 것으로 간주할 수 있으며, 노드 체인에서 더 아래에 있는 서브트리나 노드는 무시할 수 있습니다. *비어 있지 않은(non-empty)* 서브트리에 해당하는 노드들만 주목하면 됩니다.

암호학적 누산기(Cryptographic accumulator)

이 맥락에서, Linea의 스파스 머클 트리를 일종의 “암호학적 누산기(cryptographic accumulator)”로 볼 수 있습니다. 암호학적 누산기는 아이템들의 집합을 매우 짧은 문자열로 인코딩하고, 읽기/쓰기 연산이 증명될 수 있도록 하는 암호학적 프리미티브(primitive)의 한 종류입니다. 머클 트리와 스파스 머클 트리는 기본적인 누산기의 예시이며, 이보다 더 강력한 기능을 가진 다른 누산기도 존재합니다.

Linea의 상태 관리자는 키-주소 기반 데이터베이스의 모든 CRUD(create, read, update, delete) 연산을 증명할 수 있도록 확장된 버전의 스파스 머클 트리를 사용합니다. 개략적으로, 이 구조는 상태의 모든 비영(非0) 항목을 인코딩하는 정렬된 이중 연결 리스트(sorted doubly-linked list)의 노드를 저장하기 위해 스파스 머클 트리를 사용합니다.

Linea의 상태 관리자는 누산기를 사용하여 Linea의 계정 트라이(account trie)를 추적할 뿐 아니라, 각 컨트랙트의 스토리지 또한 개별적으로 추적합니다.

트리의 리프 노드들은 다음과 같은 구조를 가집니다:

```
prev || next || hKey || hVal
```

- hKey와 hVal은 각각 저장된 상태 항목의 키와 값의 해시입니다.
- prev와 next는 사전식 순서(lexicographic order) 상에서 바로 이전과 이후에 위치한 리프들의 위치를 저장하는 포인터입니다.

SMT의 처음 두 리프는 헤드(head)와 테일(tail)이라 불리며, 저장된 튜플을 인코딩하지 않는다는 점에서 특별합니다. 헤드는 가능한 가장 낮은 hKey, 테일은 가능한 가장 높은 hKey를 나타냅니다. 따라서 헤드는 연결 리스트의 시작에, 테일은 끝에 위치합니다.

헤드에서 시작하여 head.next에 저장된 SMT 리프에 접근하면 실제로 저장된 가장 낮은 항목을 얻을 수 있습니다. next 값을 계속 증가시키면 두 번째로 낮은 항목, 그다음 항목을 순차적으로 얻을 수 있습니다. 이 과정을 반복하면 저장된 모든 항목을 순회하게 되고, 마지막에 테일 노드에 도달하며 최종 단계가 완료됩니다.

리프 노드들은 계정의 내용을 담고 있다는 점에서 스토리지 슬롯(storage slots)이라고도 불립니다.

빈 리프 추적(Tracking empty leaves)

트리의 모든 리프는 초기화 시 기본값/제로 값으로 채워집니다. 결정적 해싱 함수(deterministic hashing function)를 사용하기 때문에 이러한 리프들은 항상 동일한 해시로 표현되며, 누산기(accumulator)는 빈 리프를 쉽게 인식할 수 있습니다.

그러나 상태 관리자가 Linea 계정의 내용을 스토리지 슬롯에 업데이트하려면, 어떤 빈 리프를 덮어쓸지, 그리고 이러한 빈 리프들이 정확히 어디에 위치하는지를 알아야 합니다. 또 하나 고려해야 할 점은, ‘새로운’ 리프 — 즉, 데이터를 저장하기 위해 업데이트되는 빈 리프 — 의 인덱스를 결정적으로(deterministically) 덮어써야 한다는 것입니다. 이러한 요구사항 덕분에 누구나 단순히 트랜잭션 기록만 보고도 트리를 재구성할 수 있습니다.

리프의 위치 일관성을 보장하기 위해, 상태 관리자는 항상 트리의 이전 리프 왼쪽에만 ‘새로운’ 리프를 삽입합니다. 만약 상태 관리자가 임의의 노드를 임의의 위치에 삽입할 수 있었다면, 트리를 동일한 구성으로 재구성하는 것이 불가능해지고, L1에서 제공된 머클 증명을 검증하는 능력이 심각하게 훼손되었을 것입니다.

누산기 적용(Applying the accumulator)

이더리움 가상 머신(Ethereum Virtual Machine, EVM)은 머클-패트리시아 트라이(Merkle-Patricia Trie)라는 머클 트리의 변형을 사용하여 다음을 추적합니다:

- 전역 상태(world state): 계정 상태를 추적
- 계정 스토리지 상태(account storage state, 또는 단순히 '스토리지'): 각 계정의 내용을 추적

Linea에서는 이 구조를 변형하여 사용합니다. 전역 상태에는 여전히 머클-패트리시아 트라이가 사용되지만, 계정 스토리지 상태에는 앞서 설명한 맞춤형 암호학적 누산기가 사용됩니다.

누산기는 다음과 같은 연산을 수행할 수 있습니다:

- **삽입(Insertion)**: 이전에 값이 0이었던 슬롯에 비영 값(non-zero value)을 저장할 때, 새로운 스토리지 슬롯을 트리에 추가합니다.
- **업데이트(Update)**: 이전에 비영 값이었던 슬롯에 새로운 비영 값을 저장할 때, 기존 스토리지 슬롯의 값을 변경합니다.
- **삭제(Deletion)**: 이전에 비영 값이었던 슬롯에 0 값을 저장할 때, 스토리지 슬롯을 트리에서 제거합니다.
- **제로 읽기(Read zero)**: 비멤버십(non-membership)을 증명하며, 스토리지 슬롯에 접근했으나 업데이트되지 않았고 값이 0일 때 발생합니다.
- **비제로 읽기(Read non-zero)**: 멤버십(membership)을 증명하며, 스토리지 슬롯에 접근했으나 업데이트되지 않았고 값이 비영일 때 발생합니다.

이러한 연산들은 두 가지 트리에 적용됩니다: 전역 상태(world state)와 계정 스토리지 상태(account storage state).

전역 상태(World state)

전역 상태 트리는 블록체인에 존재하는 모든 계정 — 컨트랙트 계정과 외부 소유 계정(EOA) — 을 매핑하고, 각 계정의 스토리지 상태(account storage state)를 가리킵니다. 이더리움 메인넷에서는 이 데이터가 표준 트라이(trie)에 저장되지만, Linea는 누산기(accumulator)를 사용하여 계정을 키:값(key:value) 쌍으로 매핑합니다. 이외의 구현은 EVM과 유사합니다.

그 구조는 다음과 같습니다:

- **HKey**: Hash(address)
- **Val**: Hash(nonce, balance, storageRoot, codeHash, keccakCodeHash, CodeSize)

중요한 점은, **Val**(value) 해시 함수에 입력되는 모든 데이터가 유한체(finite field) 해석을 가져야 한다는 것입니다. 이러한 방식으로 데이터가 포맷되어야 Linea 프로버가 증명 검증 시 전역 상태에 올바르게 접근할 수 있습니다. 각 요소는 다음과 같이 포맷됩니다 (keccakCodeHash를 제외한 모든 요소는 하나의 필드가 필요합니다):

- **nonce**: 논스는 빅엔디언(big-endian) 형태로 byte32에 기록됩니다. 예를 들어 논스가 10이면, 다음과 같이 인코딩됩니다:
`0x0000000000000000000000000000000000000000000000000000000000000000000a`
- **balance**: 논스와 동일한 방식으로 포맷되며, 빅엔디언 byte32입니다.
- **storageRoot**: 스토리지 루트는 EVM에서처럼 패트리시아 트라이 루트의 Keccak 값이 아니라, [다음 섹션](#)에서 설명하는 계정 스토리지 상태의 “커스텀 머클 트리” 루트여야 합니다.
- **codeHash**: 코드 해시는 코드의 Keccak 값이 아니라, 다음 섹션에서 설명하는 방식으로 얻은 값이어야 합니다.
- **keccakCodeHash**: 2개의 필드 요소로 표현되며, 각각 최상위 128비트와 최하위 128비트를 담습니다. Keccak 코드 해시는 EVM에서 지정된 그대로 Keccak 해시(EXTCODEHASH의 출력)에 해당합니다. 실용적인 이유로 Keccak 버전과 “커스텀” 버전을 모두 유지합니다.
- **codeSize**: 코드 크기는 CODESIZE/EXTCODESIZE 오프코드가 반환하는 값과 동일해야 합니다.

계정 스토리지 상태(Account storage state)

스토리지 트라이(storage trie)라고도 불리는 계정 스토리지 상태는 상태 관리자가 계정의 내용을 조회하기 위해 접근하는 데이터베이스입니다. 계정 스토리지는 주로 컨트랙트 계정과 관련이 있으며, EOAs의 경우 자산 및 트랜잭션에 대한 데이터는 전역 상태 **Val**에 저장되고, **codeHash** 및 그 변형 값들은 비어 있습니다.

계정 스토리지의 주요 기능은 컨트랙트를 쉽게 조회하고 처리할 수 있도록 기록하는 것이므로, 컨트랙트를 효율적으로 인코딩해야 합니다. 이를 위해 다음과 같은 형식을 사용합니다:

- **HKey:** Hash(StorageKeyMSB, StorageKeyLSB)
- **Val:** Hash(StorageValueMSB, StorageValueLSB)

여기서 **MSB**는 '워드(word)'의 첫 16바이트를, **LSB**는 마지막 16바이트를 의미합니다. '워드'는 EVM에서 사용되는 자연 데이터 단위로, 256비트(32바이트) 청크를 가리킵니다.

예를 들어, 컨트랙트 코드에 관한 데이터가 `byte32`로 인코딩되었다면 (EVM 및 Linea에서 워드의 표준 데이터 타입), 다음과 같이 표현될 수 있습니다:

[a0, a1, a2, ..., a15, b0, b1, ..., b15]

이 byte32는 다음과 같이 **MSB**와 **LSB**로 나누어집니다:

- **MSB**: {0, 0, ..., 0, a0, a1, a2, a3, ..., a15}
- **LSB**: {0, 0, ..., 0, b0, b1, b2, b3, ..., b15}

즉, **MSB**는 첫 16바이트를, **LSB**는 두 번째 16바이트를 차지합니다.

상태 루트 전환 증인 생성(Generating state-root-transition witnesses)

스파스 머클 트리(sparse Merkle tree)로 구축된 누산기(accumulator)는 동시에 다음과 같은 역할을 합니다:

- 연산을 수행할 수 있는 데이터 구조
- 언제든지 짧은 문자열(즉, 루트 해시)로 요약할 수 있는 데이터셋
- 특정 연산이 해시 A에서 해시 B로의 전환을 유발했음을 Linea 프로토콜이 검증할 수 있는 도구

누산기가 새로운 블록에 관한 트레이스 정보를 처리하고 상태를 이에 맞게 업데이트하면, 새로운 상태 루트 해시(state root hash)를 코디네이터를 통해 프로버(prover)에게 전달할 수 있습니다. 이 상태 루트 해시는 이후 프로버에 의해 *증인(witness)*으로 사용됩니다. 즉, 블록 내 트랜잭션이 실제로 발생했음을 증명하면서도, 그 트랜잭션의 구체적인 내용을 공개하지 않는 검증 가능한 방법이 됩니다.

트레이스 확장 및 증명(Trace expansion and proving)

무엇입니까?

영지식 증명(zero-knowledge proof)을 위한 최종 데이터 셋을 생성하고, 그 증명을 산출하는 모듈입니다.

무엇을 합니까?

코디네이터와 linea-besu로부터 여러 정보 셋을 수신하고, 이를 사용해 간결하고 비대화형 지식 증명(zkSNARK)을 생성합니다.

어떻게 합니까?

여러 연산이 관련되며, 이 모듈의 첫 부분은 **Corset**입니다. Corset은 코디네이터로부터 컨플레이트된 트레이스 데이터를 입력받아, 두 번째 부분인 **gnark**에 전달할 수 있도록 준비합니다. gnark는 준비된, 즉 “확장된(expanded)” 트레이스 데이터를 입력받아, 이더리움에 제출될 수 있는 유형의 증명을 생성합니다.

증명: 회로 실행과 런타임(Proving: Circuit execution and runtime)

gnark zk-SNARKs

무엇입니까?

Linea에서 **gnark**는 프로버(prover)의 마지막 부분입니다. 또한 독립형 소프트웨어로서, 다른 프로젝트에서도 암호학적 회로(cryptographic circuits)를 생성하는 데 사용할 수 있습니다.

무엇을 합니까?

Corset과 유사하게, gnark는 두 가지 기능을 수행합니다. 첫째, zk-SNARK 증명을 생성하는 제약 시스템(constraint system), 즉 “회로(circuits)” 집합을 준비합니다. 이 증명은 L1 이더리움 환경에서 검증될 수 있습니다. 둘째, 실제로 그 증명을 생성합니다.

어떻게 합니까?

gnark의 코드베이스는 프론트엔드(frontend)와 백엔드(backend), 두 개의 독립된 API로 나뉩니다.

- 프론트엔드 API의 메서드는 개발자가 암호학적 회로를 생성할 수 있도록 합니다. 즉, gnark는 Corset에서 사전 컴파일된 Go 제약 시스템을 최종적인 “회로(circuits)” 또는 제약으로 변환하여, L1 네트워크에 제출될 SNARK 증명을 산출합니다.
- 백엔드 API의 메서드는 런타임에서 사용되며, 프론트엔드에서 생성된 증명 시스템을 통해 데이터를 처리하고 결과적으로 증명을 생성합니다.

즉, gnark는 Corset으로부터 각 블록(또는 블록 집합)의 “확장된(expanded)” 트레이스 데이터를 입력받아, 준비된 회로를 통해 이를 처리합니다. 이 시점에서 새로운 트랜잭션 집합을 확보하게 되며, 이는 EVM 환경에서 실행된 것이고, 실제로 발생했음을 증명할 수 있으며, L1로 전송할 준비가 되어 있습니다. 그렇다면 프로버는 이것을 어떻게 할까요? 바로 코디네이터(Coordinator)로 다시 보내는 것입니다.

모듈 한계(Module limits)

Linea는 증명 복잡성을 관리 가능한 수준으로 유지하고, 프로버가 인프라를 과부하시키지 않으면서 효율적으로 증명을 생성할 수 있도록 모듈 한계를 사용합니다.

Linea의 산술화(arithmetization) 사양은 모듈 단위로 나뉘며, 각 모듈은 EVM 또는 Linea zkEVM 연산(opcodes)의 집합을 나타냅니다. 모듈은 다음과 같은 기능을 할 수 있습니다:

- 트랜잭션 데이터를 처리
- zkEVM 메모리를 관리
- 모듈 간 상호작용을 조율

각 모듈은 그 목적과 관계없이, 연산이 생성할 수 있는 데이터 라인의 수에 대해 특정 한도를 가집니다. 이 한도는 [Linea 소스 코드](#)에서 확인할 수 있습니다. 모듈의 연산에 고유한 명령이 제공될 때마다 새로운 라인이 생성되지만, 동일한 인자를 가진 반복 호출은 새로운 라인을 생성하지 않습니다.

복잡도가 높은 트랜잭션은 대량의 연산을 반복적으로 사용하여 너무 많은 데이터 라인을 생성할 수 있으며, 이 경우 모듈 한도를 초과할 수 있습니다. 한도에 도달한 트랜잭션은 [시퀀서](#)에 의해 거부되어, 프로버에 전달되는 트레이스 데이터가 관리 가능한 범위 내에 있도록 하고, 증명을 생성할 수 있게 합니다.

모듈 한도를 초과하여 거부된 트랜잭션은 다음 메서드를 통해 확인할 수 있으며, 오류를 반환합니다:

- `linea_estimateGas`
- `eth_sendRawTransaction`

만약 트랜잭션이 전송되었지만 직접 거부되지 않았고 블록에 포함되지도 않았다면, `linea_getTransactionExclusionStatusV1`을 사용하여 트랜잭션 해시의 거부 상태를 조회할 수 있습니다.

이 API 메서드는 `linea_estimateGas` 또는 `eth_sendRawTransaction`이 오류를 반환하지 않은 경우, 즉 RPC 노드와 시퀀서 간 트랜잭션 실행 결과가 다른 특수 사례에서 트랜잭션 거부 사유를 조회하기 위해 설계되었습니다.