

아비트럼(Arbitrum)

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

소개

아비트럼은 이더리움을 개선하기 위해 설계된 기술 모음이다. 아비트럼 체인을 사용하면 웹 3.0 앱 사용 및 스마트 컨트랙트 배포와 같이 이더리움에서 하는 것과 동일한 작업을 수행할 수 있다. 차이점은 거래가 더 저렴하고 빠르다는 것이다. 주요 제품인 아비트럼 Rollup은 이더리움과 동일한 보안을 제공하는 옵티미스틱 롤업 프로토콜이다.

하지만 왜?

이더리움은 정말 멋지지만, 한계가 있다. 이더리움 블록체인은 모든 사용자에게 대해 초당 20~40건의 거래 (TPS)만 처리할 수 있다. 이 한계에 도달하면 사용자들은 자신의 거래 내역을 저장하기 위해 공간을 확보하기 위해 경쟁해야 하며, 이로 인해 수수료가 증가한다.

아비트럼 Rollup은 이 문제를 해결한다! 작동 방식은 다음과 같다. 아비트럼 Rollup 체인은 이더리움 내에서 하위 모듈 역할을 한다. 일반적인 이더리움 거래와 달리, 모든 아비트럼 거래를 처리하기 위해 이더리움 노드가 필요하지 않다. 대신, 이더리움은 아비트럼에 대해 "무죄 추정 원칙"을 따른다. 처음에 부모 체인은 아비트럼에서의 작업이 규칙을 따른다고 가정한다. 규칙 위반(예: "이제 당신의 모든 돈을 가져갔습니다"라고 주장하는 경우)이 발생하면 부모 체인에서 해당 주장에 대해 이의를 제기할 수 있다. 이 경우, 사기를 입증하고, 무효 주장을 무시하고, 가해자에게 불이익을 줄 수 있다. 부모 체인에서 사기를 조사하고 확인할 수 있는 이러한 기능은 아비트럼의 주요 기능이며, 이더리움의 보안성이 아비트럼에 미치는 이점을 설명한다.

이러한 사기 증거는 어떻게 작동하는가?

부모 체인에서 아비트럼 체인을 관리하는 사람들을 검증자라고 한다. 검증자는 체인 상태에 대한 주장을 하고, 다른 사람의 주장에 이의를 제기하는 등의 활동을 한다. 대부분의 아비트럼 사용자는 검증자를 운영하지 않는 것처럼, 대부분의 이더리움 사용자도 부모 체인의 스테이킹 노드를 운영하지 않는다. 하지만 누구나 검증자가 될 수 있으며, 특별한 허가는 필요하지 않다(허용 목록이 해제되면). 오픈 소스 검증자 소프트웨어를 실행하고 필요한 경우 ETH를 본딩하기만 하면 된다.

최소 한 명의 정직한 검증자가 있으면 체인은 안전하게 유지된다. 신뢰할 수 있는 사기 검증자 한 명만 있어도 많은 악의적인 행위자를 잡을 수 있다. 이러한 특징 덕분에 이 시스템은 "trustless" 시스템이다. 사용자는 자금을 안전하게 보호하기 위해 특정 집단에 의존할 필요가 없다.

누가 사기 방지를 담당하나?

이 단계에서 "롤업"이 시작된다. 아비트럼 Rollup 체인은 사용자 거래 데이터를 이더리움에 직접 게시하여 처리한다. 이 설정은 이더리움이 안전한 한 누구나 아비트럼에서 무슨 일이 일어나는지 볼 수 있음을 의미한다. 또한 발생하는 모든 사기 행위를 적발하고 입증할 수 있다.

검증자는 부모 체인에서 아비트럼 체인 상태를 진행하는 데 도움을 주는 노드이다. 이들은 체인 상태에 대한 클레임을 제기하고 다른 사람의 클레임에 이의를 제기할 수 있다. 대부분의 아비트럼 사용자는 검증자를 운영하고 싶어하지 않는데, 이는 대부분의 이더리움 사용자가 부모 체인의 스테이킹 노드를 운영하지 않는 것과 마찬가지로이다. 하지만 누구나 검증자가 될 수 있다. 허용 목록이 제거되면 사용자는 오픈소스 검증자 소프트웨어를 실행하고 ETH필요한 경우에만 본딩하면 된다.

최소 한 명의 정직한 검증자가 있어야 네트워크의 보안이 유지된다. 즉, 신뢰할 수 있는 사기 검증자 한 명만으로도 수많은 악의적인 행위자를 잡을 수 있다. 이러한 설정 덕분에 시스템은 "trustless" 상태가 되며, 사용자는 자금을 안전하게 보호하기 위해 특정인에게 의존할 필요가 없다.

보기보다 복잡하지 않다. 두 검증자가 의견이 일치하지 않으면 한 명만 진실을 말하는 것이다. 분쟁이 발생하면 두 검증자는 상호 작용 게임에 참여하여 서로에게 호출-응답(call-and-response) 방식으로 응답한다. 이 과정을 통해 두 검증자는 의견 차이를 단일 계산 단계, 즉 두 수를 곱하는 것과 같은 간단한 단계로 좁힐 수 있다. 이 단계는 부모 체인에서 실행되며, 이는 어느 쪽이 정직한지 보여준다. 더 자세한 설명은 상호 작용 사기 방지 및 챌린지 프로토콜에 대한 설명을 참조하십시오.

사용자는 아비트럼에서 이더리움으로 자금을 인출할 때만 지연을 경험한다. 아비트럼에서 이더리움으로 직접 인출하는 경우, 일반적으로 부모 체인에서 자금을 받기까지 일주일 정도 기다려야 한다. 하지만 빠른 브리지 애플리케이션을 사용하면 소액의 수수료만 지불하면 이 대기 시간을 건너뛸 수 있다. 이더리움에서 아비트럼으로 자금을 입금하거나 아비트럼 체인에서 탈중앙화 애플리케이션(dApp)을 사용하는 경우와 같이 다른 활동에서는 이러한 지연이 발생하지 않는다.

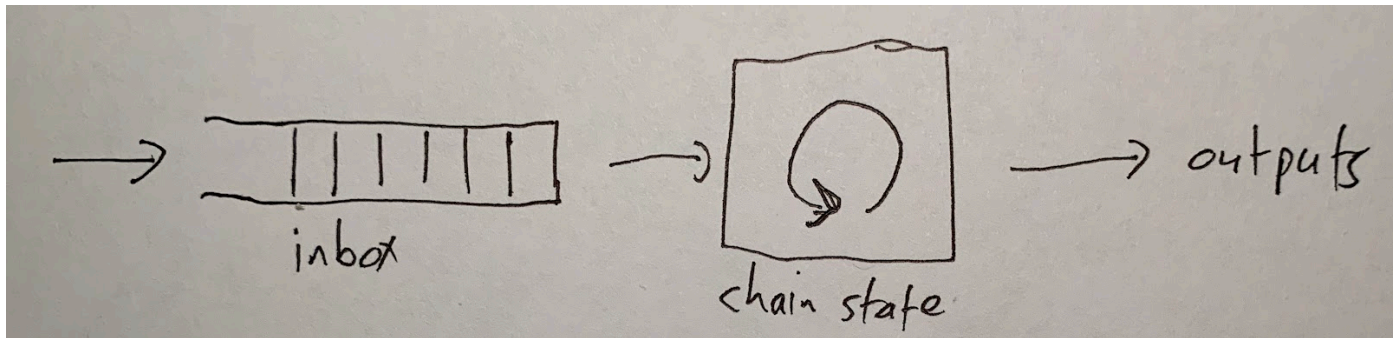
어떻게 더 저렴한가?

아비트럼은 부모 체인의 부담을 줄여 사용자 거래 비용을 절감한다. 아비트럼의 주요 방식은 일괄 처리 방식으로 거래를 처리하는 것이다. 일괄 처리는 수백 개의 자식 체인 거래를 포함할 수 있으며, 하나의 부모 체인 거래로 제출된다. 이러한 일괄 처리 방식은 각 거래를 개별적으로 제출하는 것보다 간접 비용을 절감하여 부모 체인과의 상호 작용을 더욱 저렴하게 만든다.

또한, 아비트럼은 부모 체인에 거래 데이터를 압축된 형식으로 게시한다. 이 데이터는 자식 체인 환경에서만 압축 해제되므로 부모 체인에 저장해야 하는 정보의 양이 줄어든다.

이 모든 것이 어떻게 함께 작동하나?

가장 기본적인 수준에서 아비트럼 체인은 다음과 같이 작동한다.



사용자와 스마트 컨트랙트는 인박스에 메시지를 넣는다. 체인은 메시지를 하나씩 읽고 처리한다. 이 처리 과정은 체인의 상태를 업데이트하고 몇 가지 출력을 생성한다.

아비트럼 체인에서 거래를 처리해 달라고 요청하려면 해당 거래를 체인의 수신함에 넣어야 한다. 그러면 체인이 거래를 확인하고 실행한 후, 거래 영수증과 거래로 발생한 출금 내역 등 몇 가지 출력을 생성한다.

실행은 결정론적이다. 즉, 인박스의 내용이 체인의 동작을 고유하게 결정한다. 따라서 거래 결과는 인박스에 입력되는 즉시 알 수 있다. 모든 아비트럼 노드는 결과를 알려줄 수 있다. (원하는 경우 아비트럼 노드를 직접 실행할 수도 있다.)

이 문서의 모든 기술 세부 사항은 이 다이어그램과 연결된다. 이 다이어그램에서 아비트럼에 대한 완전한 설명을 얻으려면 다음과 같은 질문에 답해야 한다.

- 인박스, 체인 상태, 출력을 추적하는 사람은 누구인가?
- 아비트럼은 체인 상태와 출력이 올바른지 어떻게 확인하나?
- 이더리움 사용자와 스마트 컨트랙트는 아비트럼과 어떻게 상호 작용할 수 있나?
- 아비트럼은 Ethereum과 호환되는 스마트 컨트랙트와 거래를 어떻게 지원하나?
- ETH와 토큰은 아비트럼 체인으로 어떻게 전송되고 체인에서 어떻게 관리되나?
- 어떻게 하면 나만의 아비트럼 노드나 검증기를 실행할 수 있나?

니트로의 디자인: 4가지 핵심

니트로의 본질과 핵심 혁신은 네 가지 핵심 아이디어에 있다. 여기서는 각 아이디어를 간략하게 요약하여 나열하고, 이후 섹션에서 더 자세히 살펴보겠다.

핵심 아이디어: 시퀀싱 후 결정론적 실행 : 니트로는 2단계 전략으로 거래를 처리한다. 먼저, 거래를 단일 순서로 구성하고 니트로가 해당 순서를 준수한다. 그런 다음, 거래는 결정론적 상태 전이 함수(State Transition Function)에 의해 해당 순서에 따라 처리된다.

핵심 아이디어: 핵심 Geth : 니트로는 인기 있는 go-ethereum("Geth") 이더리움 노드 소프트웨어의 핵심 코드를 컴파일하여 이더리움의 데이터 구조, 형식 및 가상 머신을 지원한다. 이렇게 Geth를 라이브러리로 사용하면 이더리움과의 매우 높은 호환성을 보장할 수 있다.

주요 아이디어: 실행과 검증을 분리한다. 니트로는 동일한 소스 코드를 두 번 컴파일한다. 한 번은 속도에 최적화된 니트로 노드에서 실행할 네이티브 코드로 컴파일하고, 두 번째는 이식성과 보안에 최적화된 검증에 사용할 WASM으로 컴파일한다.

핵심 아이디어: 상호작용 사기 방지 기능을 갖춘 낙관적 롤업 : 니트로는 아비트럼이 개척한 상호작용 사기 방지 기능을 포함한 낙관적 롤업 프로토콜을 사용하여 부모 이더리움 체인으로의 거래를 처리한다. 이제 아비트럼 니트로의 기본 개념, 큰 그림, 그리고 네 가지 핵심 아이디어를 살펴보았으니, 아비트럼 프로토콜을 통한 거래 추적 과정을 살펴보겠다. 다음 섹션에서는 거래 수명 주기를 살펴보겠다.

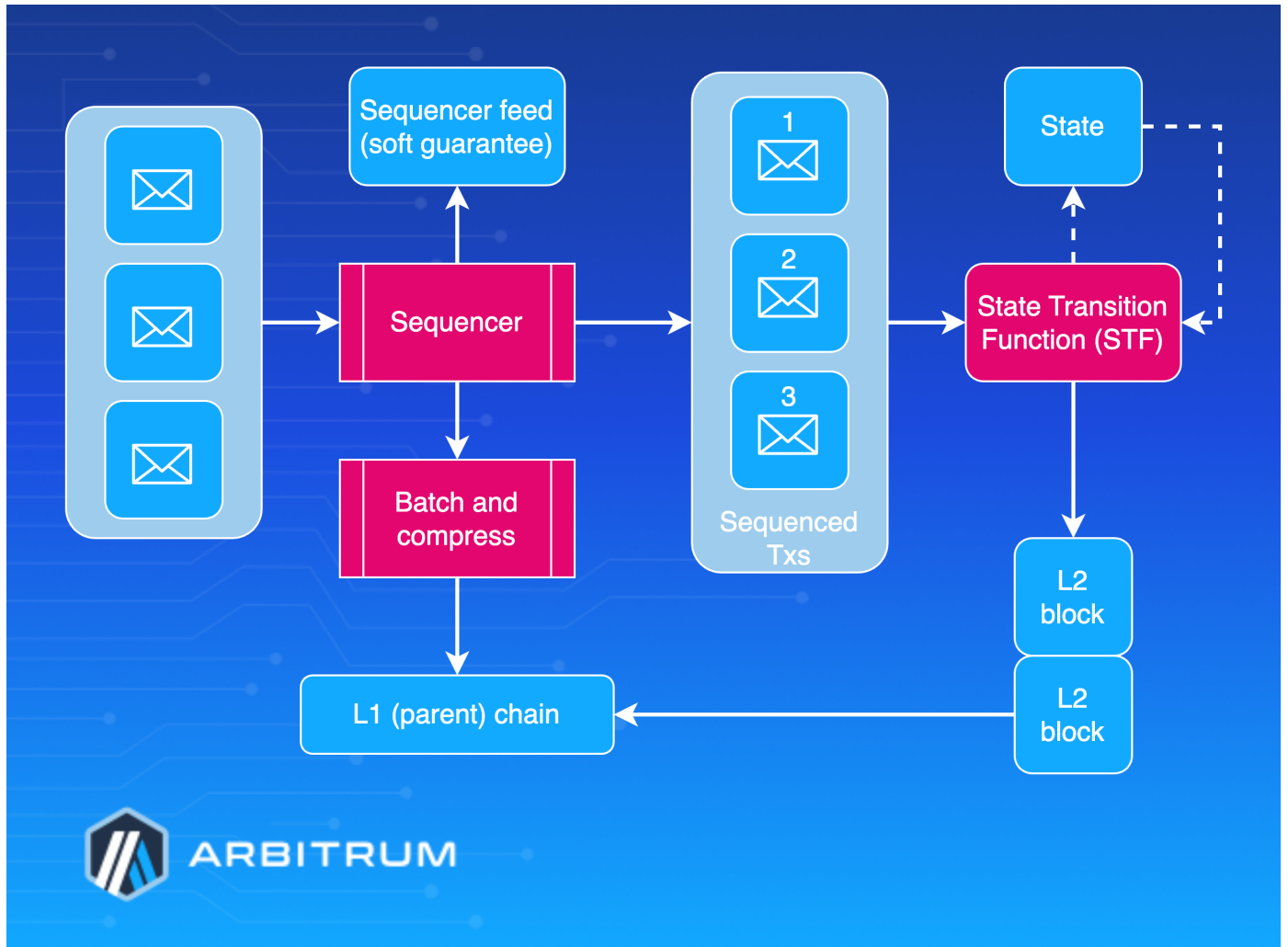
시퀀싱 후 결정적 실행

이 섹션에서는 사용자가 아비트럼 체인에 포함할 거래를 제출하는 데 사용할 수 있는 다양한 방법을 살펴본다. 시퀀서로 거래를 전송하거나, 부모 체인의 지연된 수신함 스마트 컨트랙트를 통해 거래를 제출하여 시퀀서를 우회하는 등 다양한 방법을 살펴본다. 이러한 옵션을 간략하게 설명함으로써 사용자가 네트워크와 상호 작용하는 방식을 명확히 하고, 각 방법에 관련된 프로세스를 자세히 설명하며, 이러한 거래를 처리하는 모듈을 파악하는 것을 목표로 한다. 이 개요는 아비트럼 생태계의 거래 수명 주기 초기 단계에 대한 이해를 높이고, 이후 섹션에서 거래 포함 메커니즘을 자세히 살펴볼 수 있도록 도와준다.

첫 번째 하위 섹션인 "시퀀서에 거래 제출"에서는 사용자가 시퀀서에 거래를 전송하는 데 사용할 수 있는 네 가지 방법을 제시한다. 공개 RPC, 타사 RPC, 아비트럼 노드, 그리고 시퀀서 엔드포인트를 통해 전송된다. 처음 세 가지 경로를 통해 전송된 거래는 시퀀서에 도달하기 전에 로드 밸런서를 거친다. 반면, 시퀀서 엔드포인트를 사용하면 거래가 로드 밸런서를 거치지 않고 시퀀서로 직접 전송된다.

두 번째 하위 섹션인 "시퀀서 우회"에서는 사용자가 시퀀서에 의존하지 않고 아비트럼 체인에 거래를 포함할 수 있는 대체 방법을 설명한다. 부모 체인의 Delayed Inbox 스마트 컨트랙트로 직접 거래를 전송함으로써 사용자는 추가적인 유연성을 확보하여 시퀀서를 사용할 수 없거나 사용하지 않으려는 경우에도 거래를 처리할 수 있다.

이 다이어그램은 아비트럼 체인에 거래를 제출하는 다양한 경로를 보여준다. 시퀀서를 통해 거래를 전송하거나, 시퀀서를 우회하여 부모 체인의 지연된 수신함 스마트 컨트랙트를 사용하는 옵션을 강조한다.



시퀀서에 거래 제출

이 섹션에서는 사용자가 아비트럼 체인의 시퀀서에 거래를 제출하는 다양한 방법을 간략하게 설명한다. 이를 위한 네 가지 주요 방법은 공개 RPC, 타사 RPC, 아비트럼 노드, 그리고 시퀀서 엔드포인트이다. 이러한 방법을 자세히 살펴보고, 각 방법을 선택해야 하는 시점과 각 방법을 사용하여 아비트럼 시퀀서에 거래를 효과적으로 제출하는 방법을 설명한다.

1. 공개

아비트럼은 아비트럼 One , 아비트럼 Nova , 아비트럼 Sepolia 등 주요 체인에 공개 RPC를 제공한다. 이러한 RPC 엔드포인트는 속도 제한적 특성으로 인해 리소스 사용량이 적은 작업에 적합하다. 공개 RPC는 일반적인 사용 사례 및 네트워크와의 간단한 상호작용에 적합한 옵션이다.

각 체인의 특정 RPC 엔드포인트에 대한 자세한 내용은 설명서의 이 섹션을 참조하십시오.

2. 타사

사용자는 타사 노드 제공업체를 통해 아비트럼의 퍼블릭 체인과 상호 작용할 수도 있다. 이러한 제공업체는 이더리움에서 널리 사용되는 것과 동일하므로 리소스 집약적인 작업에 안정적인 선택이 될 수 있다. 성능과 확장성이 중요한 경우 이러한 타사 제공업체를 사용하는 것이 좋다.

지원되는 타사 공급자 목록은 여기에서 확인할 수 있다.

3. 아비트럼

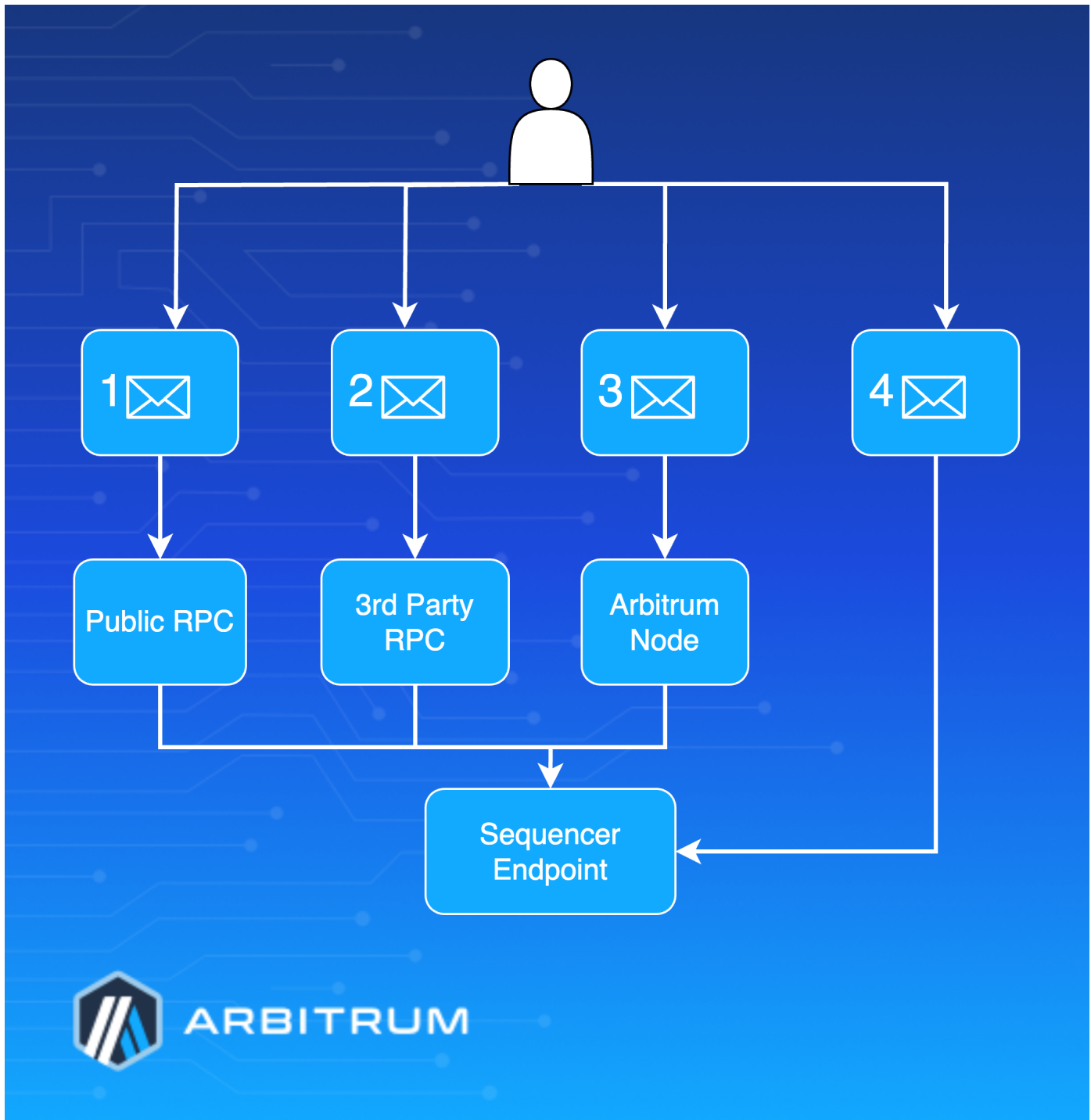
시퀀서로 거래를 전송하는 또 다른 방법은 자체 호스팅 아비트럼 노드를 이용하는 것이다. 노드를 실행하면 거래를 직접 제어할 수 있으며, 거래는 시퀀서 피드를 통해 시퀀서로 전송된다.

노드를 설정하고 실행하는 방법에 대한 자세한 내용은 아비트럼 Node 설명서를 참조하십시오.

4. 시퀀서

시퀀서 엔드포인트는 거래이 시퀀서에 도달하는 지연을 최소화하려는 사용자에게 가장 직접적인 방법이다. 표준 RPC URL과 달리, 시퀀서 엔드포인트는 `eth_sendRawTransaction` 및 `eth_sendRawTransactionConditional` 호출만 지원하며 로드 밸런서를 완전히 우회한다. 이 엔드포인트는 가장 빠른 거래 처리 시간을 필요로 하는 사용자에게 최적의 선택이다.

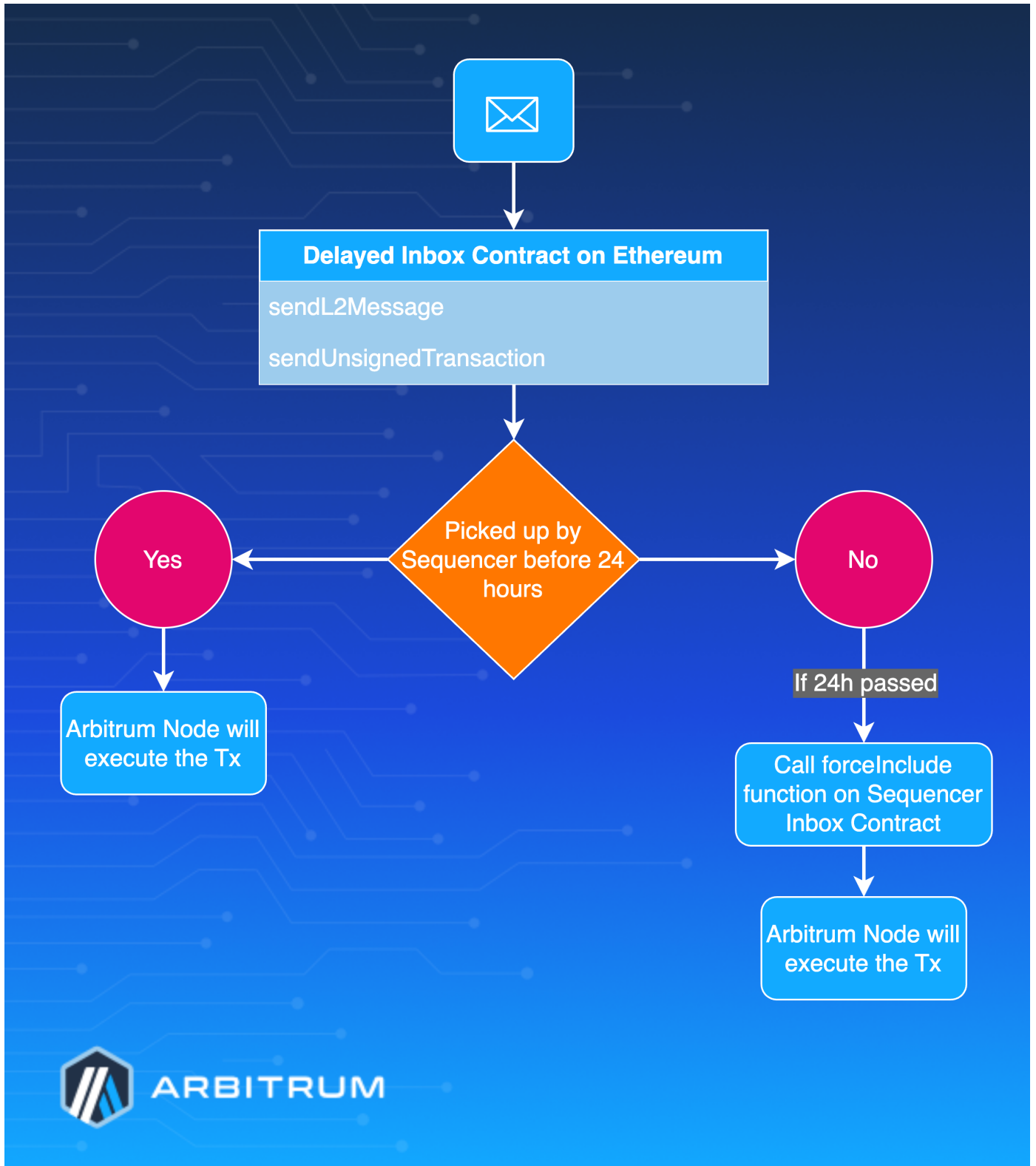
아래 다이어그램은 시퀀서에 거래를 제출하는 다양한 방법을 보여준다.



시퀀서 우회

이 섹션에서는 시퀀서를 우회하여 아비트럼 체인에 거래를 제출하는 대체 방법을 살펴본다. 이 페이지에서는 사용자가 시퀀서를 거치지 않고 부모 체인의 지연된 수신함(Delayed Inbox) 스마트 컨트랙트로 직접 거래를 전송하는 방법에 중점을 둔다. 이 방법은 거래가 두 가지 경로를 통해 이루어질 수 있도록 하며, 각 경로는 네트워크와 서로 다른 방식으로 상호 작용하여 거래 포함을 달성한다. 이 접근 방식은 사용자에게 더 큰 유연성을 제공하고, 시퀀서를 사용할 수 없거나 사용자가 시퀀서에 의존하지 않기로 선택한 경우에도 거래를 계속 처리할 수 있도록 보장한다. 이 섹션에서는 이러한 대체 제출 메커니즘을 강조하고 아비트럼 네트워크의 견고성과 분산화 기능을 강조한다.

다이어그램3에서는 사용자가 Delayed Inbox 스마트 컨트랙트를 사용하여 시퀀서를 우회하여 거래를 제출하는 방법을 보여준다. 다이어그램에서 볼 수 있듯이 거래 처리에는 두 가지 경로가 있다. 거래가 Delayed Inbox에 제출되면 시퀀서는 자동으로 해당 거래를 선택하여 주문된 거래로 포함하고 시퀀서 피드로 전송한다. 그러나 시퀀서가 24시간 이내에 거래를 처리하지 않으면 사용자는 스마트 컨트랙트 `forceInclude`에서 해당 함수를 호출할 수 있다 시퀀서. 이 작업을 통해 시퀀서는 거래를 선택하여 주문된 거래 목록에 포함하므로 사용자는 거래에 대한 보안을 유지할 수 있다.



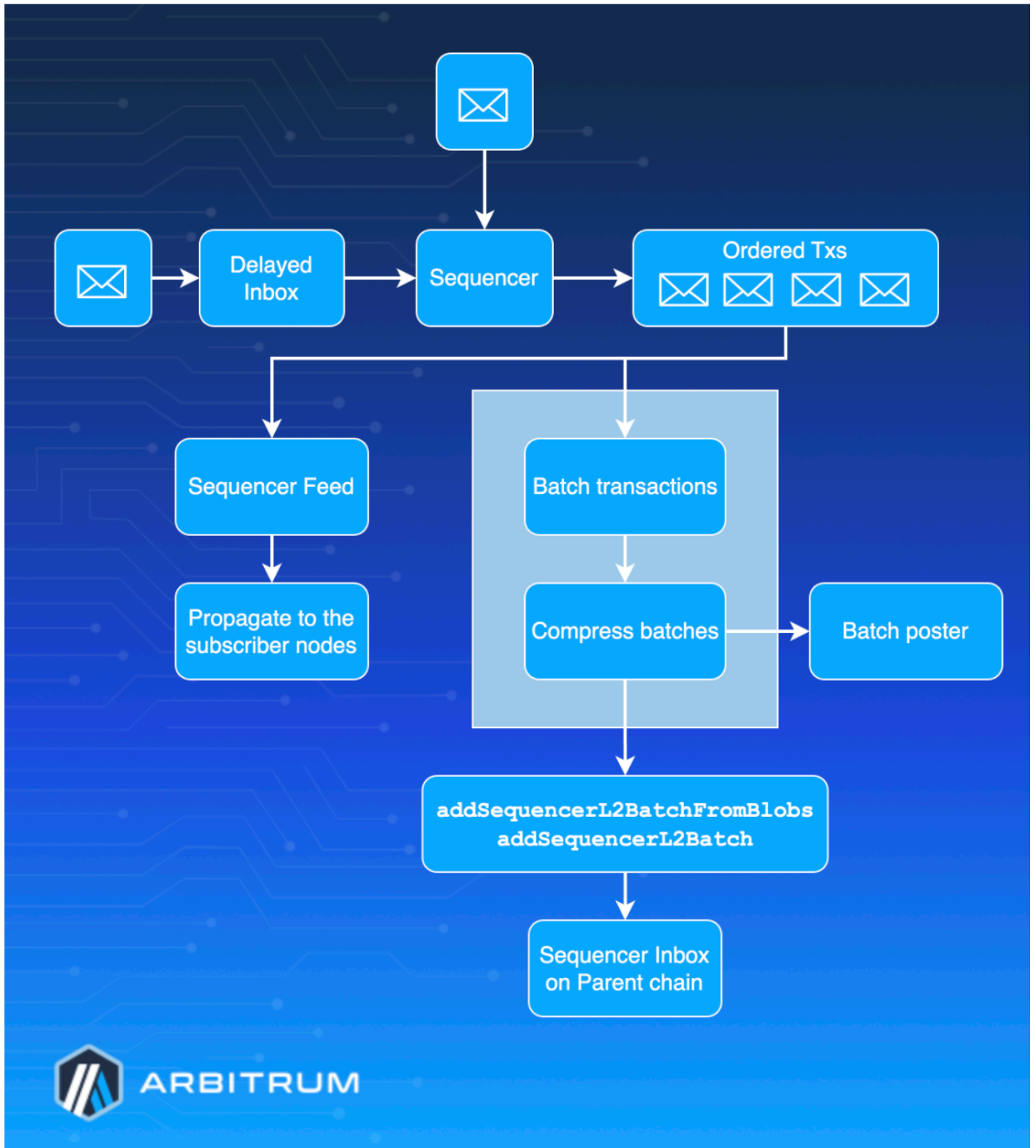
거래를 시퀀서에 제출하는 대신 지연된 수신함으로 전송하려면, 사용자는 거래를 생성한 후 `sendL2Message` 함수를 호출하고 직렬화된 서명된 거래의 데이터를 인수로 전달하면 된다. 이 함수를 사용하면 부모 체인 검증이 필요하지 않은 모든 메시지에 적합한 일반적인 자식 체인 메시지를 체인으로 전송할 수 있다.

시퀀서가 24시간 이내에 온라인 상태가 되지 않거나 거래 검열을 결정할 경우, 사용자는 스마트 컨트랙트에서 해당 함수를 호출할 수 있다. 이 작업을 통해 시퀀서의 역할을 우회하여 해당 거래가 체인에 포함된다.

또한, 아비트럼 SDK는 지연된 수신함에 거래를 제출하는 과정을 간소화하는 클래스를 제공한다. 사용자는 메서드를 사용하여 거래를 전송하고, 해당 메서드가 포함되도록 할 수 있다. SDK는 또한 직렬화된 서명된 거래 16진수 문자열 생성 시 거래 서명을 지원하는 도우미 메서드를 제공하여 전체 프로세스를 간소화한다.

시퀀서와 검열 저항

시퀀서는 아비트럼 네트워크의 핵심 구성 요소로, 거래를 효율적으로 주문하고 처리하는 역할을 한다. 블록 체인의 보안과 무결성을 유지하면서 사용자에게 빠른 거래 확인을 제공하는 데 중요한 역할을 한다. 아비트럼에서 시퀀서는 수신 거래를 주문하고, 거래 데이터를 부모 체인에 일괄 처리, 압축 및 게시하여 비용과 성능을 최적화한다.



이 섹션에서는 시퀀서의 작동 방식을 자세히 살펴보겠다. 다루는 주제는 다음과 같다.

- 시퀀싱 및 브로드캐스팅(시퀀서 피드) : 시퀀서가 제공하는 실시간 거래 피드에 대한 개요로, 노드가 거래 시퀀스에 대한 즉각적인 업데이트를 받을 수 있도록 해준다.
- 일괄 게시 : 시퀀서가 거래를 일괄 그룹화하고, 압축하여 데이터 크기를 줄인 후, 상위 체인의 시퀀서 수신함 스마트 컨트랙트로 전송하는 방식이다. 이 섹션에서는 상위 체인의 가격 책정 모델과 이것이 거래 비용에 미치는 영향도 자세히 살펴본다.
 - 배칭

- 압축
- 시퀀서 인박스 스마트 컨트랙트서 제출
- 파이널리티 : 아비트럼에서 소프트 및 하드 파이널리티 메커니즘을 통해 거래 파이널리티가 어떻게 달성되는지 이해하고, 거래가 안전하고 안정적으로 확인되도록 보장한다. (시퀀서 작업이 아님)
- 검열 시간 초과 : 장기간의 시퀀서 검열 및/또는 예상치 못한 시퀀서 중단으로 인한 부정적인 영향을 제한하는 것을 목표로 하는 특별 기능에 대한 간략한 소개이다.

이러한 측면을 살펴보면 아비트럼 생태계 내에서 시퀀서의 역할을 이해할 수 있다. 여기에는 거래 처리량을 향상하고, 지연 시간을 줄이고, 공정하고 분산된 네트워크를 유지하는 방법이 포함된다.

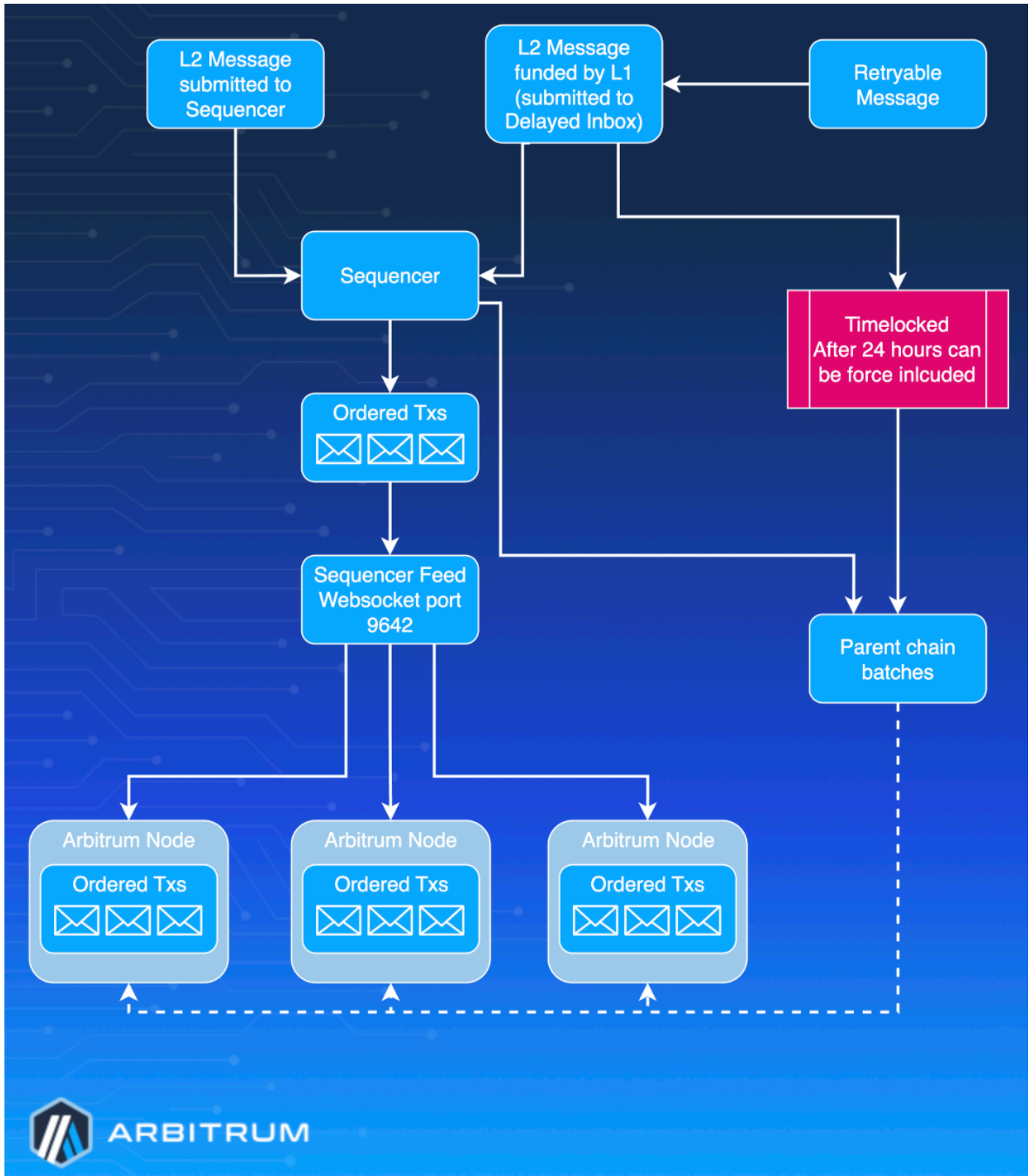
시퀀싱 및 브로드캐스팅

시퀀서 피드는 아비트럼 네트워크의 니트로 아키텍처의 핵심 구성 요소이다. 시퀀서가 데이터를 수용하고 정렬하는 즉시 거래 데이터를 실시간으로 배포할 수 있도록 한다. 사용자와 노드는 이를 통해 거래 시퀀싱에 대한 즉각적인 업데이트를 수신하여 신속한 거래 확인을 촉진하고 네트워크의 전반적인 응답성을 향상시킨다.

시퀀서

시퀀서는 두 가지 기본 채널을 통해 거래 순서를 전달한다.

1. 실시간 시퀀서 피드 : 거래가 시퀀싱되는 즉시 게시되는 라이브 방송이다. 이 피드에 가입한 노드와 클라이언트는 즉시 알림을 받아 지연 없이 거래를 처리할 수 있다.
2. 부모 체인에 배치 게시 : 시퀀서는 정기적으로 거래를 집계하여 최종 확정을 위해 부모 체인에 게시한다. (이 프로세스에 대한 자세한 내용은 배치 게시 섹션을 참조하십시오.)



실시간 시퀀서 피드

실시간 피드는 시퀀서가 특정 순서대로 거래를 처리한다는 약속을 나타낸다. 이 피드를 구독하면 노드와 클라이언트는 다음을 수행할 수 있다.

- 즉각적인 알림 받기 : 새로 순서가 지정된 거래와 그 순서에 대한 즉각적인 정보를 얻는다.
- 거래를 신속하게 처리한다. 순차적인 거래를 활용하여 상태를 로컬로 업데이트하여 신속한 애플리케이션

선 응답과 사용자 상호 작용을 가능하게 한다.

- 소프트 파이널리티의 이점 : 부모 체인이 파이널리티에 도달하기 전에 거래 승인 및 주문에 대한 임시 보장을 얻는다.
이 메커니즘은 분산형 거래소나 실시간 게임 플랫폼과 같이 낮은 지연 시간과 높은 처리량이 필요한 애플리케이션에 특히 유용하다.

소프트 파이널리티 및 신뢰 모델

"소프트 파이널리티"는 시퀀서의 실시간 피드를 기반으로 한 거래의 예비 확정을 의미한다. 주요 특징은 다음과 같다.

- 시퀀서 무결성에 대한 의존성 : 피드의 정확성과 안정성은 시퀀서가 정직하게 작동하고 심각한 가동 중지 시간이 없는 데 달려 있다.
- 즉각적인 사용자 피드백 : 사용자는 거래 확인에 신속하게 대응할 수 있어 사용자 경험이 향상된다.
- 부모 체인과의 최종 일관성 : 실시간 피드는 빠른 업데이트를 제공하지만, 거래가 부모 체인에 게시되고 완료되는 순간 최고의 보안과 파이널리티가 확립된다. (자세한 내용은 파이널리티 섹션을 참조 하세요.)

이 신뢰 모델을 이해하는 것이 필수적이다. 시퀀서가 정상적으로 작동할 것으로 예상되지만, 사용자와 개발자는 소프트 파이널리티(soft finality)이 이러한 가정에 달려 있다는 점을 알아야 한다. 절대적인 확실성이 요구되는 상황에서는 당사자들이 부모 체인에서 거래가 확정될 때까지 기다릴 수 있다.

네트워크에서의 시퀀서 피드의 역할

시퀀서 피드는 아비트럼 생태계 내에서 여러 가지 중요한 기능을 수행한다.

- 상태 동기화 : 노드는 피드를 사용하여 네트워크의 최신 상태와 동기화를 유지하여 분산 플랫폼 전반에서 일관성을 보장한다.
- 애플리케이션 개발 : 개발자는 네트워크 이벤트에 즉시 대응하는 애플리케이션을 구축하여 라이브 업데이트, 즉시 알림, 실시간 분석과 같은 기능을 구현할 수 있다.
- 생태계 투명성 : 피드는 거래 순서와 네트워크 활동에 대한 가시성을 제공하여 커뮤니티 내에서 투명성과 신뢰를 증진한다.

고려사항 및 제한 사항

시퀀서 피드는 상당한 이점을 제공하지만 다음 사항을 고려하십시오.

- 시퀀서 가용성에 대한 의존성 : 실시간 피드의 효율성은 시퀀서의 가동 시간과 응답성에 달려 있다. 네트워크 문제나 시퀀서 다운타임은 거래 가시성을 지연시킬 수 있다.

- 소프트웨어의 임의적 특성 : 부모 체인에서 거래가 파이널리티에 도달할 때까지 피드에서 제공하는 임의 순서가 예외적인 상황에서 변경될 수 있는 약간의 위험이 있다.
- 보안 영향 : 보안이 가장 중요한 고위험 거래(예: 중앙화된 거래소 입금 및 출금)의 경우, 사용자는 지연 시간이 길어지더라도 부모 체인의 확인을 기다리는 것을 선호할 수 있다.

개발자와 사용자는 이러한 요소를 염두에 두고 애플리케이션과 상호작용을 설계하고, 요구 사항에 따라 속도와 확실성 간의 적절한 균형을 선택해야 한다.

시퀀서 피드

다이어그램에서 볼 수 있듯이, 시퀀서 피드는 시퀀서에 직접 게시된 자식 체인 거래를 전송할 뿐만 아니라, 부모 체인에서 제출된 자식 체인 거래도 포함한다. 여기에는 부모 체인에 제출된 자식 체인 메시지와 재시도 가능한 거래가 포함된다. 시퀀서 에이전트는 부모 체인의 지연된 수신함 스마트 컨트랙트에 제출된 확정된 메시지를 모니터링한다. 확정된 메시지는 피드에 수신 메시지로 처리하여 정렬된 거래로 추가되도록 한다.

니트로 노드는 지연된 수신함 거래가 부모 체인에 제출된 직후, 심지어 완료되기 전에도 추가되도록 설정할 수 있다는 점에 유의해야 한다. 그러나 이 방식은 거래가 부모 체인에서 완료되지 않을 경우 자식 체인이 재구성될 위험을 초래한다. 이러한 위험을 완화하기 위해 아비트럼 One 과 Nova의 시퀀서는 이더리움 체인에서 완료되는 거래만 피드에 포함한다.

이 페이지에서 피드가 WebSocket을 통해 들어오는 메시지를 보내는 방법을 알아보고 피드에서 메시지 데이터를 추출하는 방법을 알아볼 수도 있다

일괄 게시

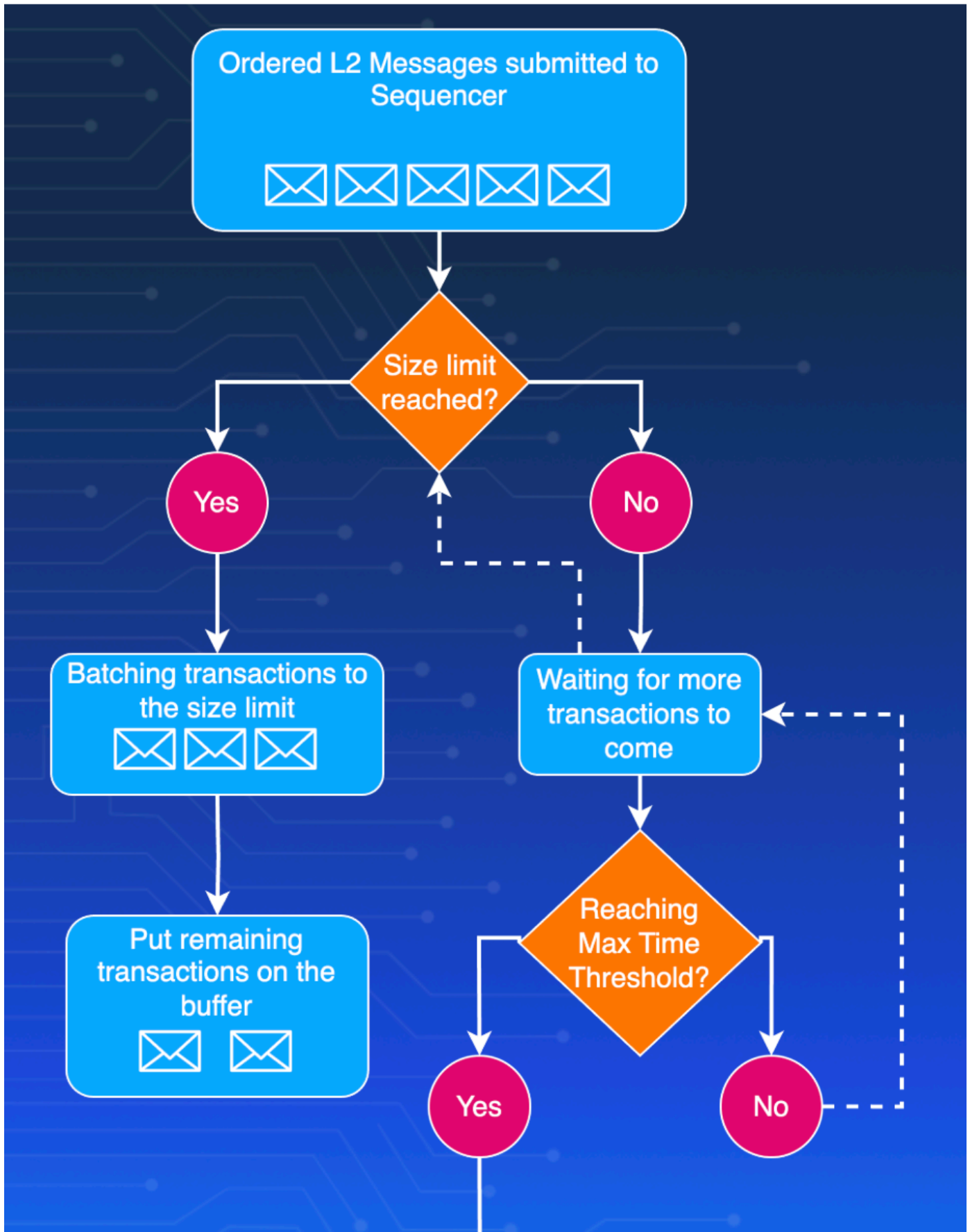
배치 포스팅은 아비트럼 네트워크 내 시퀀서(시퀀서) 운영의 기본 프로세스이다. 여러 자식 체인 거래를 수집하여 배치로 구성하고, 데이터를 압축하여 크기를 줄인 후, 이러한 배치를 부모 체인의 시퀀서 인박스 컨트랙트(시퀀서 Inbox Contract)로 전송하는 과정을 포함한다. 이 메커니즘은 비용과 성능을 최적화하는 동시에 부모 체인 블록체인에 거래가 안전하게 기록되도록 하는 데 매우 중요하다.

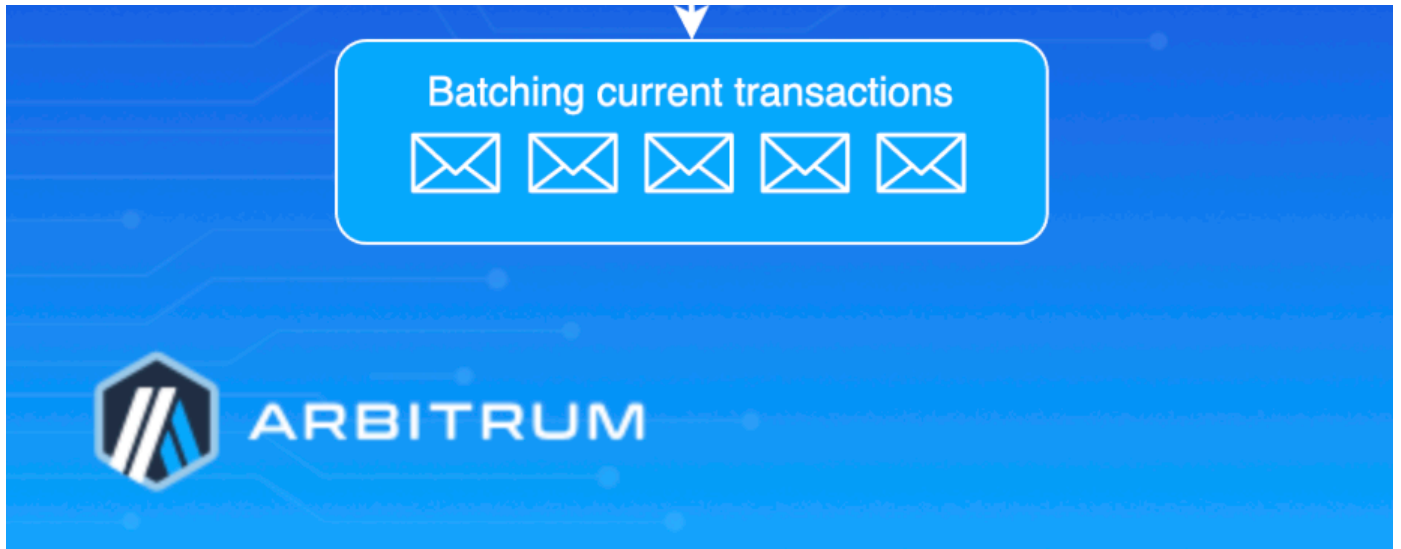
이 섹션에서는 다음 주제를 다루면서 일괄 게시 프로세스를 자세히 살펴보겠다.

- 배치 : 시퀀서가 효율적인 처리 및 게시를 위해 들어오는 거래를 배치로 그룹화하는 방식이다.
 - 압축 : 거래 데이터를 압축하여 부모 체인에 게시해야 하는 데이터 양을 최소화하고 비용을 절감하는데 사용되는 방법이다.
 - 시퀀서 인박스 스마트 컨트랙트로 전송 : 부모 체인의 시퀀서 인박스 스마트 컨트랙트에 압축된 배치를 제출하는 절차로, 안전하고 신뢰할 수 있는 거래 기록을 보장한다.
- 아비트럼이 보안을 저해하지 않으면서 확장성과 비용 효율성을 어떻게 달성하는지 이해하려면 배치 포스팅(Batch-Posting)에 대한 이해가 필수적이다. 이러한 하위 주제를 자세히 살펴보면 거래 처리량 최적화 및 수수료 최소화에서 시퀀서(시퀀서)의 역할과 모체인 데이터 가격 책정의 어려움을 해결하기 위해 구현된 혁신적인 솔루션에 대한 통찰력을 얻을 수 있다.

배칭 및 압축

아비트럼의 시퀀서는 자식 체인의 거래를 부모 체인에 게시하기 전에 수집하고 정리하는 데 중요한 역할을 한다. 배칭 프로세스는 비용 효율성과 적시 거래 포함을 최적화하도록 설계됐다.





거래 수집 및 주문:

- 지속적인 수신: 시퀀서는 사용자가 제출한 거래를 지속적으로 수신한다.
- 주문: 거래는 수신된 순서에 따라 주문되므로 결정적인 거래 순서가 유지된다.
- 버퍼링: 수신된 거래는 일괄 처리를 기다리며 버퍼에 임시로 저장된다.

배치 형성 기준:

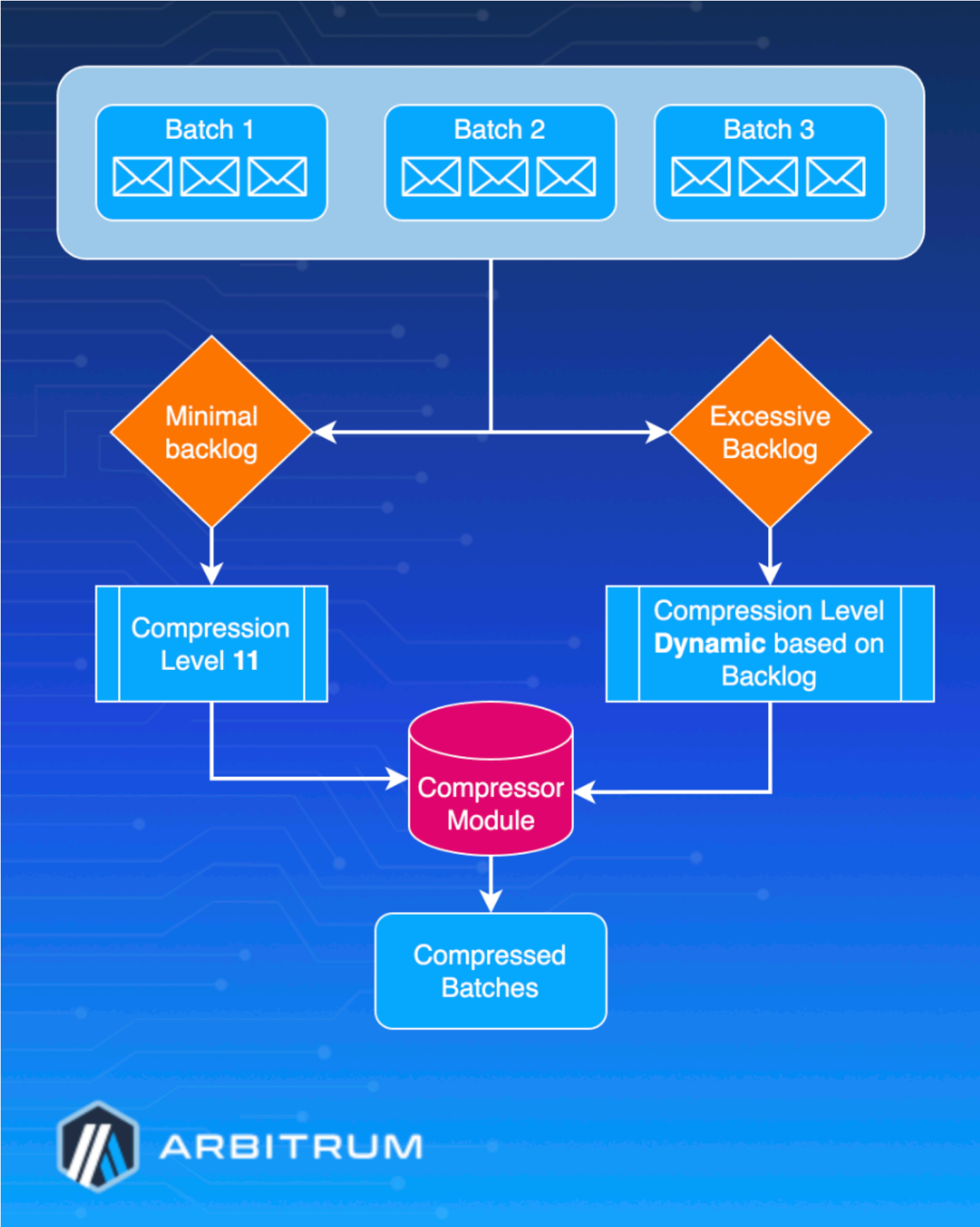
- 크기 임계값: 누적된 거래가 미리 정의된 크기 한도에 도달하면 배치가 생성된다. 이 한도는 부모 체인에 데이터를 게시하는 데 드는 고정 비용을 더 많은 거래에 걸쳐 상각하여 비용 효율성을 향상시킨다.
- 시간 제약: 시퀀서는 불필요한 지연을 방지하기 위해 마지막 배치 게시 이후 경과된 시간도 모니터링한다. 최대 시간 임계값에 도달하면, 배치가 크기 임계값을 충족하지 않더라도 시퀀서는 지금까지 수집된 거래로 배치를 생성한다.

배치 생성 프로세스:

- 집계: 배치 형성 기준(크기 또는 시간 임계값)이 충족되면 시퀀서는 버퍼링된 거래를 단일 배치로 집계한다.
- 메타데이터 포함: 배치에는 모든 거래에 필요한 메타데이터가 포함된다.
- 압축 준비: 압축 단계를 위한 일괄 준비가 시작되며, 이때 부모 체인에 게시하기 전에 기술이 데이터 크기를 최소화한다.

이 배치 메커니즘을 통해 시퀀서는 비용 효율적인 모체인 게시와 신속한 거래 처리라는 두 가지 요건을 균형 있게 충족하여 거래를 효율적으로 관리할 수 있다. 시퀀서는 규모와 시간 기준에 따라 거래를 전략적으로 그룹화함으로써 거래당 비용을 절감하고 아비트럼 네트워크의 전반적인 확장성을 향상시킨다.

시퀀서는 부모 체인에 게시되는 배치의 데이터와 비용을 최적화하기 위해 거래 배치를 생성할 때 압축을 사용한다. 아비트럼은 높은 압축률과 효율성을 갖춘 Brotli 압축 알고리즘을 사용하며, 이는 부모 체인 게시 비용 절감에 필수적이다.



Brotli 알고리즘

Brotli의 압축 알고리즘에는 0에서 11까지의 압축 수준 매개변수가 포함되어 있다. 이 매개변수를 통해 두 가지 핵심 요소의 균형을 맞출 수 있다.

압축 효율성 : 수준이 높을수록 크기 감소가 더 크다.

계산 비용 : 수준이 높을수록 더 많은 처리 능력과 시간이 필요하다.

압축 수준이 높아질수록 압축 비율은 높아지지만 압축 시간은 길어진다.

동적 압축 레벨

아비트럼의 압축 수준은 시퀀서가 부모 체인에 게시하기를 기다리는 배치의 현재 백로그에 따라 동적으로 조정된다. 버퍼에 여러 배치가 대기 중인 경우, 처리량 향상을 위해 압축 수준을 동적으로 조정할 수 있다. 버퍼가 지연된 배치로 인해 과부하 상태가 되면 압축 수준이 감소한다.

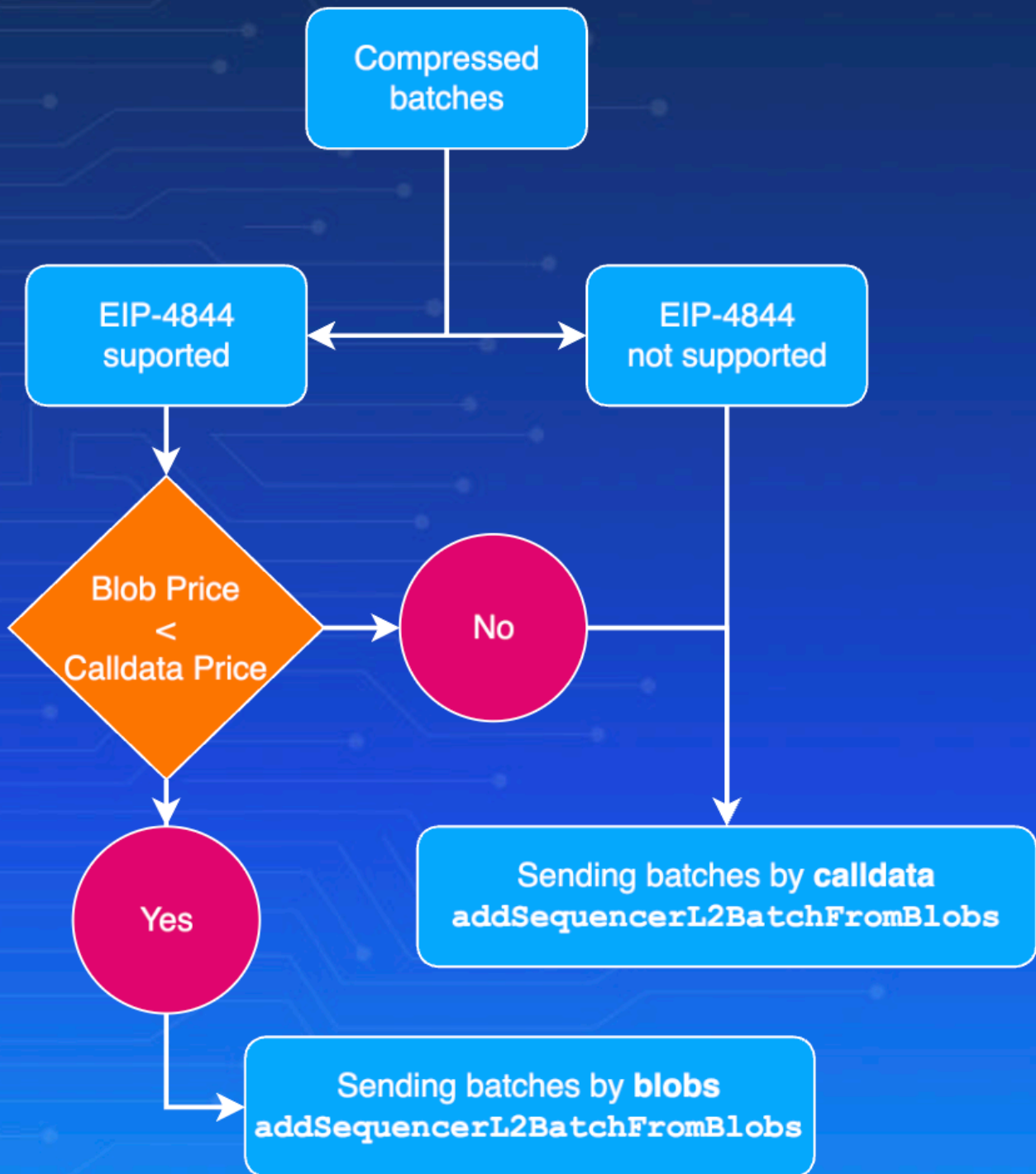
이러한 절충안은 압축 효율성보다 속도를 우선시하여 보류 중인 배치의 처리 및 전송 속도를 높인다. 이를 통해 버퍼가 더 빨리 비워져 전반적인 시스템 성능이 향상된다.

이제 거래가 일괄 처리되고 압축되었으므로 이를 배치 포스터로 전달하여 부모 체인으로 보낼 수 있다.

시퀀서 인박스 스마트 컨트랙트서

시퀀서는 거래를 일괄 처리하고 압축한 후, 보안과 파이널리티를 보장하기 위해 이러한 일괄 처리를 부모 체인에 게시한다. 이 프로세스에는 시퀀서가 관리하는 외부 소유 계정(EOA)인 일괄 처리 포스터가 포함된다. 일괄 처리 포스터는 압축된 거래 일괄 처리를 부모 체인의 시퀀서 수신함 스마트 컨트랙트 에 제출할 책임이 있다.

시퀀서는 체인이 EIP-4844(Proto-Danksharding)를 지원하는지 여부와 현재 네트워크 조건에 따라 부모 체인에 배치를 전송하는 데 두 가지 주요 방법을 사용한다.



ARBITRUM

1. Blob을 사용

- 기본 접근 방식 : 부모 체인이 EIP-4844를 지원하는 경우, 시퀀서는 블롭 거래를 활용하여 배치를 효율적으로 게시한다.

- 방법 : 배치 포스터는 시퀀서 인박스 스마트 컨트랙트의 기능을 호출한다.
- 프로세스 :
 - 배치 데이터는 확장성을 위해 최적화된 대규모 바이너리 데이터 구조인 블롭으로 포함된다.
 - 해당 거래에는 배치에 대한 메타데이터가 포함되지만, 호출 데이터에 배치 데이터 자체는 포함되지 않는다.
- 이익 :
 - 비용 효율성 : Blob을 사용하면 calldata보다 저렴하게 데이터를 포함할 수 있으므로 가스 비용이 줄어든다.
 - 확장성 : 블롭을 활용하면 네트워크가 대량의 거래를 처리하는 능력이 향상된다.

2. Calldata를 사용?

- 대체 접근 방식 : Blob 기반 요금이 상당히 높거나 일괄 게시 중에 Blob 공간이 제한되는 경우 시퀀서는 calldata를 사용하기로 선택할 수 있다.
- 방법 : 배치 포스터는 인박스 스마트 컨트랙트의 기능을 호출한다.
- 프로세스 :
 - 압축된 일괄 거래는 거래의 호출 데이터에 직접 포함된다.
- 고려 사항 :
 - 비용 평가 : 시퀀서는 현재 가스 가격과 Blob 수수료를 기준으로 CallData를 사용하는 것이 Blob을 사용하는 것보다 비용 효율적인지 여부를 동적으로 평가한다.
 - 호환성 : 부모 체인이 EIP-4844를 지원하지 않는 경우 이 방법은 일괄 게시의 기본이자 유일한 옵션이다.

권위와 파이널리티

- 배타적 접근 : 시퀀서만 시퀀서 인박스 스마트 컨트랙트에서 이러한 메서드를 호출할 수 있다. 이러한 배타적 접근을 통해 다른 당사자가 메시지를 직접 포함할 수 없다.
- 소프트 확인 영수증 : 시퀀서의 고유한 기능으로 거래를 즉시 처리하고 포함할 수 있어 사용자에게 즉각적인 "소프트 확인" 영수증을 제공할 수 있다.
- 부모 체인의 확정성 : 배치가 게시되면 거래는 부모 체인 수준의 확정성을 달성하며, 부모 체인의 합의 메커니즘을 통해 보호된다.

시퀀서는 가장 비용 효율적인 방법을 사용하여 압축된 거래 배치를 시퀀서 인박스 스마트 컨트랙트로 효율적으로 전송함으로써 거래가 부모 체인에 안전하게 기록되도록 보장한다. 이 프로세스는 네트워크의 무결성과 안정성을 유지하여 사용자에게 빠르고 안전한 거래 처리를 제공한다.

블록체인 시스템에서 파이널리티란 거래가 되돌릴 수 없게 되고 블록체인 원장에 영구적으로 포함되는 시점을 의미한다. 아비트럼의 니트로 아키텍처 맥락에서, 개발자와 사용자가 거래 확인, 보안 보장 및 애플리케이션 설계에 대한 정보에 기반한 결정을 내리려면 파이널리티를 이해하는 것이 매우 중요하다.

아비트럼은 두 가지 수준의 확정성을 제공한다.

1. 소프트 파이널리티 : 시퀀서의 실시간 피드에서 제공되며 즉각적이지만 임시적인 거래 확인을 제공한다.
2. 하드 확정성 : 거래가 부모 체인에 게시되고 확정되는 일괄 처리에 포함되어 강력한 보안 보장을 제공한다.

이 섹션에서는 소프트 파이널리티와 하드 확정성의 개념, 그 의미, 신뢰 고려 사항, 그리고 아비트럼 네트워크 내에서 이를 효과적으로 활용하기 위한 지침을 살펴본다.

소프트 파이널리티

소프트 파이널리티는 시퀀서의 실시간 피드를 기반으로 한 거래의 예비 확정을 의미한다. 주요 특징은 다음과 같다.

- 즉각적 확인 : 거래는 시퀀서에 의해 승인되고 주문되는 즉시 거의 즉시 확인된다.
- 임시 보장 : 확인은 임시적이며 시퀀서의 무결성과 가용성에 의존한다.
- 고성능 : 애플리케이션이 빠른 응답과 실시간 상호작용을 제공할 수 있도록 하여 사용자 경험을 향상시킨다.

소프트 파이널리티의 장점 :

- 낮은 지연 시간 : 사용자는 거래 상태에 대한 즉각적인 피드백을 받는다.
- 속도에 최적화 : 응답성이 중요한 애플리케이션에 이상적이다.
- 향상된 사용자 경험 : 대기 시간과 불확실성이 줄어든다.

소프트 파이널리티의 하나?

- 신뢰 의존성 : 시퀀서의 정직성과 가동 시간을 유지하는 능력에 의존한다.
- 순서 변경 가능성 : 드물지만 시퀀서가 악의적으로 작동하거나 문제가 발생하면 임시 순서가 변경될 수 있다.
- 고가 거래에는 적합하지 않음 : 강력한 보안 보장이 필요한 거래의 경우, 소프트 파이널리티만으로는 충분하지 않을 수 있다.

하드 확정성

일괄 처리된 거래가 부모 체인에 게시될 때 하드 파이널리티가 발생한다. 주요 특징은 다음과 같다.

- 강력한 보안 보장 : 부모 체인의 블록에 포함되면 거래는 부모 체인의 보안 보장을 상속받는다.
- 비가역성 : 일단 확정되면 거래는 변경 불가능하며 변경이나 취소가 불가능하다.
- 데이터 가용성 : 모든 거래 데이터는 체인에 기록되어 투명성과 검증 가능성을 보장한다.

하드 파이널리티의 장점 :

- 최고 수준의 보안 : 부모 체인의 합의 메커니즘의 견고성으로 보호된다.
- 신뢰 최소화 : 이는 시퀀서에 대한 신뢰가 필요하지 않다. 보안은 기반 블록체인에서 제공된다.
- 고가 거래에 적합 : 보안과 불변성이 가장 중요한 시나리오에 이상적이다.

하드 파이널리티의 한계 :

- 더 높은 지연 시간 : 부모 체인이 배치를 처리하고 마무리하는 데 필요한 시간으로 인해 확실한 확정성을 달성하는 데 더 오랜 시간이 걸린다.
- 비용 고려 사항 : 부모 체인에 배치를 게시하면 수수료가 발생하며, 이는 거래 비용에 영향을 미칠 수 있다.

신뢰 고려사항

각 파이널리티 수준과 관련된 신뢰 가정을 이해하는 것이 필수적이다.

- 소프트 파이널리티 신뢰 모델 :
 - 시퀀서에 대한 의존성 : 사용자는 시퀀서가 정직하게 작동하고, 거래 순서를 올바르게 지정하며, 사용 가능한 상태를 유지한다고 신뢰해야 한다.
 - 잘못된 행동의 위험 : 시퀀서가 악의적으로 행동하는 경우, 거래가 확정되기 전에 특정 거래를 재정렬하거나 검열할 수 있다.
- 하드 파이널리티 신뢰 모델 :
 - 부모 체인에 대한 의존성 : 보안은 부모 체인의 합의와 무결성을 기반으로 한다.
 - 시퀀서의 신뢰 감소 : 시퀀서가 제대로 작동하지 않더라도 게시된 배치에 포함된 거래는 부모 체인에서 마무리되면 보안이 유지된다.

응용 프로그램

개발자와 사용자는 특정 사용 사례에 따라 적절한 파이널리티 수준을 고려해야 한다.

- 소프트 파이널리티를 사용해야 하는 경우 :
 - 저위험 거래 : 재주문이나 지연의 잠재적 영향이 최소화되는 거래에 해당한다.
 - 사용자 경험 우선순위 : 게임이나 소셜 플랫폼과 같이 반응성과 즉각성이 사용자 참여를 강화하

는 애플리케이션이다.

- 빈번한 거래 : 대량의 소규모 거래가 포함되어 확정을 기다리는 것이 비현실적인 시나리오이다.
- 확실한 확정성을 요구하는 경우 :
 - 고가 거래 : 금융 이체, 대규모 거래 또는 보안이 중요한 모든 거래.
 - 규정 준수 : 보안 표준과 감사 가능한 기록을 엄격히 준수해야 하는 상황.
 - 중앙 집중형 거래소(CeX) : 거래의 파이널리티 확실성이 필수적인 입금 및 출금 작업에 사용된다.

검열 시간 초과

아비트럼 BoLD 포럼 게시물 에서 언급했듯이 , 아비트럼 BoLD 의 초기 버전에는 "검열 시간 초과"(원래 "지연 버퍼")라는 기능이 포함될 예정이다. 아비트럼 One과 아비트럼 Nova의 경우, BoLD 업그레이드와 함께 이 기능이 기본적으로 활성화될 예정이다.

검열 타임아웃은 다음과 같은 부정적인 영향을 제한하는 것을 목표로 한다.

- 장기간의 시퀀서 검열 및/또는
- 예상치 못한 시퀀서 중단

검열 시간 초과 기능

이 기능이 아비트럼 One과 아비트럼 Nova에 결제되는 체인의 보안을 어떻게 향상시키는지 설명하기 위해, L3의 부모 체인 시퀀서(L2 시퀀서)가 검열 중이거나 오프라인 상태인 시나리오를 생각해 보겠다. 이 경우, 모든 어설션 및/또는 하위 챌린지 동작은 L2 시퀀서를 우회하기 전에 24시간을 기다려야 한다(여기에 설명된 시퀀서Inbox의 forceInclusion 메서드 사용). 이 시나리오에서 챌린지 해결은 시간만큼 지연된다. 예를 들어, 챌린지가 50번의 연속적인 동작으로 해결된다면 지연 시간은 50일이 된다.

검열 시간 초과 기능은 위에서 언급한 검열 사례 중 하나(또는 모두) 또는 시퀀서 중단으로 인해 지연된 메시지 포함에 예기치 않은 지연이 발생할 때 강제 포함 임계값을 낮춰 이를 완화하고, 엔터티가 이동당 24시간 지연 없이 이동할 수 있도록 한다.

강제 포함 창은 실제로 또는 중 더 작은 값 delayBuffer과 delayBlocks 이며, 여기서 delayBlocks는 상수이고 현재 24시간으로 설정되어 있으며 delayBuffer는 30분에서 48시간 사이에 늘어나거나 줄어든다.

delayBuffer 값은 시퀀서가 오프라인 상태이거나 트랜잭션을 검열하는 시간에 따라 “증가 및 감소”한다. 이 동작을 측정하는 방법으로, delayBuffer는 지연된 메시지의 지연 시간이 임계값을 초과한 정도와 실제 지연된 시간의 차이에 따라 감소한다(즉, 일부 지연된 메시지의 지연 시간이 임계값을 초과할 경우, 해당 메시지들의 지연 시간과 임계값 사이의 차이가 버퍼에서 제거됨). 예를 들어, 임계값이 30분이고 메시지가 32분 지연된 경우 delayBuffer는 2분 감소한다. 아비트럼 One에서는 임계값이 30분으로, 아비트럼 Nova

에서는 1시간으로 설정된다.

지연 버퍼는 시퀀서가 정상적으로 작동할 때 선형 속도로 재충전된다. 즉, 메시지가 임계값을 초과하여 지연되지 않는 20분마다 1분의 명목상 속도로 재충전된다.

아비트럼 One과 아비트럼 Nova의 검열 시간 초과 기능에 대해 제안된 초기 매개변수 값은 다음과 같다.

- delay buffer= 14400개의 부모 체인(이더리움) 블록(2일)
- threshold= 아비트럼 One의 경우 L1 Ethereum 블록 150개(30분) 및 아비트럼 Nova의 경우 상위 체인(Ethereum) 블록 300개(1시간)
- replenish rate= 5% (즉, 20일마다 하루가 보충되거나 가동 시간이 약 95%가 됨)

검열 시간 제한 기능은 아비트럼 체인, 특히 아비트럼 One이나 아비트럼 Nova를 사용하는 체인의 검열 저항성을 더욱 강화한다고 생각한다. 체인 소유자는 항상 그렇듯이 이 기능의 사용 여부를 스스로 결정할 수 있으며, 사용 사례에 맞게 기본 매개변수를 변경할 수도 있다.

분산형 공정 시퀀싱

아비트럼의 장기적인 비전은 중앙집중식 시퀀서에서 탈중앙화되고 공정한 시퀀싱 모델로의 전환을 포함한다. 이 프레임워크에서 서버(또는 검증자)로 구성된 위원회는 거래 순서를 공동으로 결정하여 공정성을 보장하고, 특정 당사자의 영향력을 줄이며, 조작에 대한 저항력을 강화한다. 과반수 합의를 요구함으로써, 이 접근 방식은 여러 정직한 참여자에게 시퀀싱 권한을 분배하고, 선행 거래 또는 검열의 위험을 완화하며, 향상된 보안, 투명성 및 탈중앙화라는 블록체인의 광범위한 원칙을 준수한다.

부모 체인에서 자식 체인으로 브리징

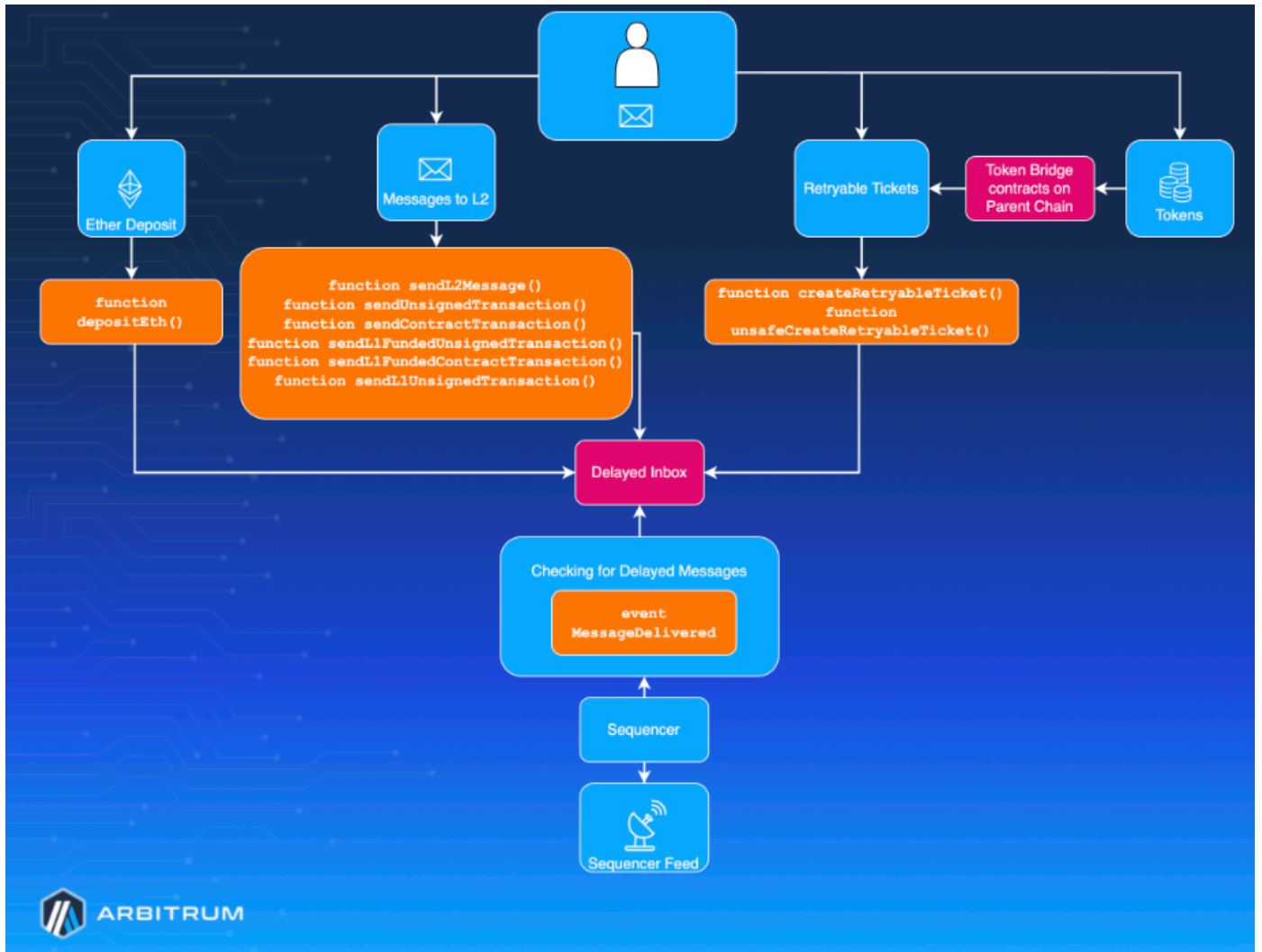
시퀀서 우회 섹션에서는 사용자가 시퀀서로 직접 거래를 전송하는 대신 부모 체인의 지연된 수신함 스마트 컨트랙트를 통해 자식 체인에 거래를 제출하는 대체 방법을 소개했다. 이 방식은 부모-자식 메시징 경로의 한 예이다. 더 광범위하게, 부모-자식 체인 메시징은 다음과 같은 모든 방법을 포함한다.

- 부모 체인에서 자식 체인 바운드 거래 제출
- ETH부모 체인에서 자식 체인으로의 입금 또는 네이티브 토큰
- 부모 체인에서 자식 체인으로 임의의 데이터나 명령어를 보낸다.

일반적으로 이러한 부모-자식 체인 메시징 방법은 다음과 같이 분류된다.

1. 네이티브 토큰 브리징 : 부모 체인에서 자식 체인으로 자식 체인의 네이티브 토큰을 예치하는 것을 말한다. 아비트럼 체인 유형에 따라 다음이 포함될 수 있다.
 - ETH브리징ETH : 가스 토큰으로 아비트럼 체인을 사용하는 경우, 사용자는 ETH지연된 인박스를 통해 자식 체인에 입금할 수 있다.
 - 사용자 정의 가스 토큰 브리징 : 사용자 정의 가스 토큰을 사용하는 아비트럼 체인의 경우, 사용자는 동일한 메커니즘을 사용하여 해당 체인의 네이티브 토큰을 자식 체인에 입금할 수 있다.
2. 지연된 인박스를 통한 거래 : 시퀀서 우회 섹션에서 설명한 대로, 이 방법을 사용하면 사용자가 부모 체인을 통해 거래를 전송할 수 있다. 여기에는 두 가지 하위 유형의 메시지가 포함된다.
 - 서명되지 않은 메시지 : 일반적인 임의의 데이터 또는 함수 호출
 - 서명된 메시지 : 특정 인증 작업을 가능하게 하는 서명이 포함된 메시지
3. 재시도 가능 티켓은 부모-자식 메시지(부모 체인에서 시작된 거래가 자식 체인에서 실행되도록 트리거하는 메시지)를 생성하는 아비트럼의 표준 메커니즘이다. 이 메서드는 다음과 같은 기능을 포함한다.
 - 일반 재시도 가능 메시징 : 부모-자식 체인에서 임의의 데이터나 호출을 전송하는 데 사용된다.
 - 맞춤형 기능 메시징 (예: 토큰 브리징): 부모-자식 체인에서 토큰을 브리징하는 것과 같은 특수한 작업을 위해 재시도 가능한 티켓(및 기타 메시징 구조)을 활용한다.

이 섹션에서는 이러한 범주를 자세히 살펴보고 작동 방식을 설명한다. 아래 다이어그램은 부모-자식 체인 간 통신 및 자산 전송에 사용 가능한 다양한 경로를 보여준다.



상태 전이 함수

상태 전이 함수 (STF)는 블록체인 시스템의 핵심 메커니즘으로, 거래가 처리될 때 시스템이 한 상태에서 다른 상태로 어떻게 변화하는지 정의한다. 본질적으로 STF는 블록체인의 현재 상태(계좌 잔액, 스마트 스마트 계약 데이터 및 기타 원장 정보의 포괄적인 스냅샷)와 입력(예: 거래 또는 거래 블록)을 사용하여 새로운 상태를 결정론적으로 계산한다. 이러한 결정론적 속성은 분산 네트워크의 모든 노드가 거래를 적용할 때 동일한 결과에 도달하도록 보장하여 시스템 전체에서 합의를 유지하므로 매우 중요하다.

아비트럼을 사용하면 STF(Secure Token Transfer Function)가 더욱 중요한 역할을 한다. 오프체인 거래는 이 기능에 따라 일괄 처리되며, STF는 주기적으로 부모 체인에 변경 사항에 대한 간략한 요약물을 제출한다. 이러한 접근 방식은 오프체인 연산을 활용하여 처리량을 높이고 가스 비용을 절감하는 동시에 이 프로세스를 이더리움의 강력한 보안 모델에 고정한다. 부정확하거나 악의적인 오프체인 실행을 방지하기 위해 아비트럼은 사기 방지(fraud proofs)라는 챌린지 메커니즘을 사용한다. 분쟁이 발생할 경우, STF는 온체인에서 단계적으로 재계산되어 네트워크가 오프체인 연산의 유효성을 검증하고 오류 또는 사기 행위를 감지 및 수정할 수 있도록 한다.

이러한 기본적인 측면 외에도, 아비트럼 니트로 스택의 STF는 전반적인 구조 면에서 이더리움의 STF와 매우 유사하지만, 아비트럼 체인의 고유한 요구 사항을 충족하기 위해 몇 가지 주요 수정 사항을 포함하고 있다. 기본적으로 이 함수는 일련의 정렬된 거래를 입력으로 받고, 정의된 규칙 집합을 이러한 입력에 적용하여, 거래 배치의 모든 업데이트를 반영하는 새로운 최종 상태를 생성한다.

스타일러스 도입으로 아비트럼은 WebAssembly(WASM) 기반 스마트 계약 실행 기능을 추가하여 EVM(이더리움 가상 머신)을 넘어 실행 모델을 확장했다. 이러한 기능 향상을 통해 Rust, C, C++로 작성된 고성능 스마트 계약을 기존 EVM 계약과 함께 실행할 수 있다. 스타일러스 통합으로 STF에 다음과 같은 여러 가지 수정 사항이 적용된다.

스타일러스 전용 거래 처리

스타일러스 거래를 인식하고 처리하여 상태 전환에 적절하게 포함되도록 하는 Geth의 수정된 버전이다.

WASM 런타임

스타일러스 거래는 EVM 대신 ArbOS의 WASM 런타임에서 실행되므로 실행 속도가 빨라지고 계산 효율성이 높아진다.

스타일러스 가스 회계 및 가격 책정

표준 EVM 거래와 달리 스타일러스 거래는 명령어 가격 책정, 호스트 I/O 작업, 잉크 사용 비용 과 같은 요소를 고려하는 새로운 가스 가격 책정 모델을 도입한다.

EVM과의 상호 운용성

스타일러스 스마트 컨트랙트는 Solidity 스마트 컨트랙트와 원활하게 상호 작용하여 EVM 및 WASM 실행 환경을 활용하는 하이브리드 애플리케이션을 구현할 수 있다.

스타일러스 관련 변경 사항은 스마트 스마트 컨트랙트 개발을 위한 보다 효율적이고 유연하며 확장 가능한 대안을 도입하는 동시에 Ethereum의 실행 모델과의 호환성을 유지하는 것을 목표로 한다.

다음 섹션에서는 이러한 입력이 무엇이고 아비트럼 노드가 이를 어떻게 수신하는지 설명한다. 또한 아비트럼 니트로 Stack의 STF(Syntax Task Force)에 구현된 규칙에 대해서도 살펴보고, 이더리움, 아비트럼, 스타일러스 기반 실행 환경에서 상태 전환이 발생하는 방식의 차이점을 강조한다. ArbOS 내에서 처리되는 스타일러스 관련 실행 작업은 호스트 I/O 작업, 캐싱, WASM 메모리 관리에 중점을 두고 별도로 다룬다.

상태 전환 함수에 대한 입력

아비트럼 노드는 신뢰성과 일관성을 강화하는 듀얼 채널 방식을 통해 거래 입력을 받는다.

먼저, 노드는 시퀀서 피드를 구독하여 실시간으로 게시되는 주문된 거래를 수신한다. 자세한 내용은 실시간 시퀀서 피드 문서를 참조하십시오.

아비트럼 노드는 시퀀서 피드 외에도 부모 체인의 이벤트를 구독한다. 이 이벤트는 일괄 처리 게시자를 통해 부모 체인에 거래가 전달될 때마다 발생한다. 이 이벤트를 수신한 노드는 부모 체인에 기록된 거래가 시퀀서 피드에서 이전에 제공한 거래와 일치하는지 확인한다. 불일치가 발생하면 노드는 부모 체인을 확실한 진실의 원천으로 사용하여 부모 체인에서 확인된 거래를 채택하도록 재구성한다.

고려 사항 및 제한 사항 섹션 에서 논의한 대로 이러한 방법은 특정 요구 사항에 따라 다양한 응용 프로그램에 더 적합할 수 있다.

아비트럼 노드는 이러한 메커니즘을 통해 정렬된 일련의 거래를 안정적으로 수신하며, 이는 상태 전이 함수 (STF) 의 입력으로 사용된다. STF는 이러한 거래를 처리하고 최종 상태를 출력하여 네트워크 전체의 일관성과 보안을 보장한다.

상태 전환 함수: Ethereum vs. 아비트럼

이 섹션에서는 이더리움과 아비트럼 니트로 스택의 상태 전환 함수 (STF) 아키텍처와 구현을 설명하고, 이들의 유사점과 주요 차이점을 강조한다.

이더리움의 STF

이더리움에서는 STF가 거래를 입력으로 받고, EVM을 통해 처리하고, 최종 상태를 출력으로 생성한다.

이더리움 상태는 수정된 머클 패트리샤 트라이(Merkle Patricia Trie)로 표현되는 방대한 데이터 구조이다. 이 구조는 모든 계정을 보유하고 해시를 통해 연결하며, 전체 상태를 블록체인에 저장된 단일 루트 해시로 축소한다.

이더리움 가상 머신(EVM)은 수학 함수와 유사하게 작동한다. 입력이 주어지면 결정론적 출력을 생성한다. 이더리움의 STF(Single Function Transfer)는 이러한 동작을 캡슐화한다.

$$Y(S, T) = S' \tag{1}$$

여기서 S는 현재 상태를 나타내고, T는 거래를 나타내며, S'는 S의 실행으로 인해 발생하는 새로운 상태를 나타낸다.

EVM은 최대 1024개 항목의 깊이를 가진 스택 머신으로 작동한다. 각 항목은 256비트 암호화(예: Keccak-256 해시 및 secp256k1 서명)와의 호환성을 위해 선택된 256비트 단어이다.

실행 중에 EVM은 거래 기간 동안만 유지되는 임시 메모리 (워드 주소 바이트 배열)를 사용한다. 이와 대조적으로, 각 스마트 컨트랙트는 전역 상태의 일부를 구성하는 영구적인 Merkle Patricia 저장 트라이(워드 주소 지정 가능 워드 배열)를 유지한다.

스마트 컨트랙트 바이트코드는 일련의 EVM 오퍼레이션 코드로 컴파일되며, 이는 표준 스택 연산(예: XOR, AND, ADD, SUB)과 블록체인 특화 연산(예: ADDRESS, BALANCE, BLOCKHASH)을 수행한다.

Geth (go-Ethereum)는 이더리움의 주요 클라이언트 구현 중 하나로, STF와 EVM 실행 엔진의 실질적인 구현체 역할을 한다. 스마트 컨트랙트의 바이트코드를 실행하고 전역 상태를 업데이트하여 거래를 처리하며, 모든 상태 변경이 결정론적이고 안전하게 이루어지도록 보장한다.

본질적으로 Geth는 EVM 내에서 거래 입력을 정확한 연산 단계로 변환하여 이더리움 블록체인의 기반이 되는 복잡한 데이터 구조를 유지한다. 견고한 설계는 이더리움의 핵심 운영을 지원할 뿐만 아니라 아비트럼 다투로 스택과 같은 플랫폼의 고급 수정을 위한 기반을 제공한다.

아비트럼의 STF

아비트럼 다투로 스택은 이더리움 STF의 수정된 버전을 구현한다. 이더리움의 핵심 원칙은 그대로 유지하지만, 아비트럼만의 몇 가지 기능과 프로세스가 이더리움 구현과 차별화된다. 주요 차이점은 다음과 같다.

가스

아비트럼에서 거래 실행에는 두 가지 비용이 발생한다. 하나는 자식 체인에서 실행하는 비용이고, 다른 하나는 부모 체인에 거래 배치를 제출하는 비용이다. 이러한 이중 수수료 구조는 가스 회계에 대한 다른 접근 방식을 요구한다.

기본 수수료

아비트럼의 가스 가격과 기본 수수료는 이더리움과 다르게 변경된다. 아비트럼은 두 가지 기본 수수료를 유지한다. 하나는 자식 체인의 실행 비용을 추적하는 수수료이고, 다른 하나는 부모 체인에 배치를 게시하는 비용을 추정하는 수수료이다. 자식 체인의 기본 수수료를 변경하는 메커니즘은 현재 이더리움의 메커니즘과 다르다.

크로스 체인 브리지

아비트럼은 크로스체인 메시징과 정기적인 거래를 지원하여 자식 체인과 부모 체인 간의 원활한 상호작용을 용이하게 한다.

블록 시간 및 블록

아비트럼 체인은 두 개의 블록 번호를 사용한다. 자식 체인의 블록 번호와 첫 번째 비-아비트럼 조상 체인의 블록 번호이다. 또한, 블록 시간은 블록 번호 및 시간 문서에 설명된 대로 자식 체인의 시퀀서 클럭에 의해 결정된다.

사전 컴파일

아비트럼 체인의 특정 사전 컴파일된 스마트 컨트랙트는 아비트럼 고유 기능을 처리한다. 또한, 개별 아비트럼 체인은 자체적인 맞춤형 사전 컴파일을 구현할 수 있다.

사용자 정의 거래 유형

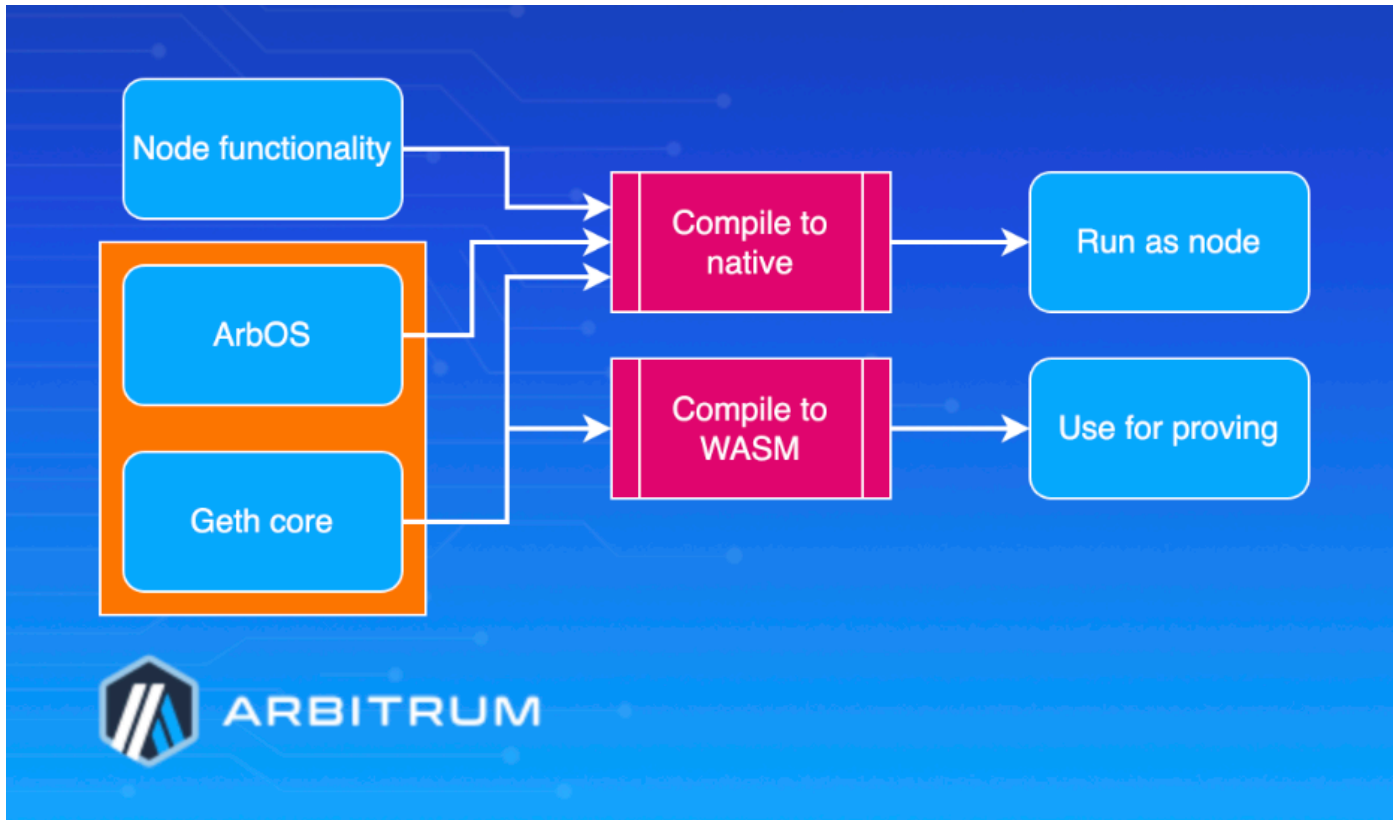
니트로 Geth는 비표준 EIP-2718(Typed Transaction Envelope)을 활용하는 여러 자식 체인별 거래 유형을 도입한다. 이러한 유형은 아비트럼 체인의 다양한 거래(예: 재시도 가능한 거래)를 나타낸다.

STF의 스타일러스

스타일러스의 도입으로 아비트럼 니트로에는 이제 Rust, C, C++로 작성된 스마트 컨트랙트 실행을 지원하여 EVM에 대한 고성능 대안을 제공한다. 표준 이더리움 거래와 달리 스타일러스 컨트랙트는 ArbOS 내의 WASM 기반 런타임에서 실행되어 효율적인 호스트 I/O 작업과 최적화된 캐싱 메커니즘을 활용한다.

구현 개요

아비트럼 STF는 Geth의 수정된 버전을 사용하여 구현되고 ArbOS와 통합된다. 이러한 조합을 통해 아비트럼은 이더리움 핵심 실행 모델과의 호환성을 유지하면서 추가 기능을 지원할 수 있다. 다음 다이어그램은 개요를 간략하게 보여준다.



니트로 노드 소프트웨어의 구성은 세 가지 주요 계층으로 구성된다.

1. 기본 계층(Geth Core) : 이 계층은 EVM 스마트 컨트랙트 실행을 에뮬레이션하고 이더리움의 상태 데이터 구조를 유지하는 Geth의 핵심 구성 요소로 구성된다. 니트로는 이 코드를 라이브러리 형태로 통합하며, 추가적인 후크를 삽입하기 위한 약간의 수정을 거친다.
2. 중간 계층(ArbOS) : ArbOS는 자식 체인 기능을 확장하는 맞춤형 소프트웨어이다. 시퀀서 데이터 배치의 압축을 풀고 파싱하며, 부모 체인 가스 비용(환급 수수료 징수)을 계산하고, 크로스 체인 브리지 작업(예: 이더리움 및 토큰 입출금)을 지원한다. ArbOS에 대한 자세한 내용은 다음 섹션에서 설명한다.
3. 최상위 계층(노드 소프트웨어) : 주로 Geth에서 파생된 이 계층은 네트워크 연결, 들어오는 RPC 요청 및 Ethereum 호환 블록체인 노드를 운영하는 데 필요한 기타 고급 기능을 처리한다.

상위 계층과 하위 계층 모두 Geth 기반이므로 이 아키텍처는 종종 "Geth 샌드위치"라고 불린다. 즉, Geth가 "빵"이고 ArbOS가 "속"에 해당하는 것이다.

상태 전이 함수(STF) 자체는 수정된 Geth 코어(기본 계층)와 ArbOS의 일부(중간 계층)로 구성된다. 실제로 STF는 소스 코드(호출하는 모든 코드 포함) 내에 지정된 함수이며, 다음과 같은 기능을 한다.

- 인박스에서 수신된 거래의 바이트를 입력으로 사용한다.
- 이더리움 상태 트리의 수정 가능한 복사본을 기반으로 작동한다.

- 실행 중에 상태를 수정할 수 있다.
- 니트로 체인에 추가되는 새로운 블록 헤더(이더리움의 블록 헤더 구조로 포맷됨)를 생성한다.

수정된 Geth

니트로의 디자인은 Geth의 견고한 기반을 기반으로, 자식 체인별 기능을 지원하도록 특정 부분을 수정했다. 이러한 수정 사항은 다음과 같다.

사용자 정의 후크 및 인터페이스 적응

니트로는 Geth의 거래 처리 흐름에 일련의 커스텀 후크를 통합한다. 이러한 후크를 통해 니트로에는 이중 가스 회계를 구현하고, 부모 체인의 콜데이터 수수료를 관리하며, 재시도 가능한 거래와 같은 커스텀 거래 유형을 이더리움의 핵심 로직에서 벗어나지 않고 처리할 수 있다.

핵심

니트로는 Geth 내의 주요 데이터 구조와 인터페이스를 조정함으로써 상태 전환 및 거래 실행이 자식 체인 최적화를 원활하게 수용할 수 있도록 보장한다. 이러한 조정은 이더리움의 기존 아키텍처와의 호환성을 유지하는 동시에 향상된 기능을 제공한다.

최소한이지만 필수적인

Geth의 안정성과 보안성을 유지하기 위해 의도적으로 최소한의 수정만 가했다. 이 "수정된 Geth" 계층은 ArbOS와 함께 사용되어 EVM의 결정론적 동작을 손상시키지 않으면서 니트로의 STF(Simple Functional Transfer)에 필요한 기능을 향상시킨다.

스타일러스 특징

Geth는 스타일러스 거래를 라우팅하고, 사용자 정의 가스 회계를 처리하고, 스타일러스 관련 상태 변경을 추적하도록 수정되어 아비트럼 니트로 내에서 원활한 실행을 보장한다.

이러한 수정 사항에 대한 자세한 기술적 설명은 다음 섹션에서 제공되며, 니트로의 사용자 정의 후크, 인터페이스 구현 및 Geth 적응이 어떻게 결합되어 고급 고성능 STF를 지원하는지 자세히 살펴보겠다.

ArbOS

ArbOS는 아비트럼 체인의 실행 환경을 제공하는 자식 체인 EVM 하이퍼바이저이다. STF 내에서 신뢰할 수 있는 "시스템 연결" 구성 요소 역할을 하는 ArbOS는 다음과 같은 역할을 한다.

네트워크

자식 체인에서 거래를 실행하는 데 필요한 리소스를 할당하고 추적한다.

블록

ArbOS는 들어오는 시퀀서 데이터 배치를 처리하여 자식 체인 블록을 생성하고 상태가 올바르게 업데이트되도록 보장한다.

크로스체인

부모 체인과 자식 체인 간의 통신을 원활하게 하고, 이더와 토큰 입금 및 출금과 같은 기능을 지원한다.

향상된 EVM

ArbOS는 스마트 스마트 컨트랙트를 실행하기 위해 Geth의 계층화된 인스턴스를 운영하며, 자식 체인 환경에 특화된 추가 로직을 통합한다.

ArbOS의 스타일러스 관련 작업

ArbOS는 스타일러스 거래에 대한 호스트 I/O 호출, 메모리 작업 및 실행 컨텍스트를 관리하여 WASM 런타임을 통한 효율적이고 결정론적인 처리를 보장한다.

ArbOS는 부모 체인에서 높은 비용으로 실행될 작업을 오프로드함으로써 자식 체인에서 이러한 작업을 빠르고 비용 효율적으로 수행할 수 있도록 한다. 이러한 설계는 컴퓨팅 및 저장 비용을 절감하고 상당한 유연성을 제공하여 자식 체인 코드가 부모 체인 기반 아키텍처보다 더 쉽게 개발되거나 사용자 정의될 수 있도록 한다.

이더리움의 STF(Single-Test Function)가 안전하고 확정적인 상태 업데이트의 기반을 형성하는 반면, 아비트럼의 니트로 스택은 이 기반 위에 이중 가스 어카운팅부터 크로스 체인 메시징까지 다양한 핵심 수정을 통해 성능과 유연성을 최적화한다. 이러한 혁신은 Geth에 대한 최소한의 전략적 수정을 통해 실현되며, ArbOS와 완벽하게 통합되어 "Geth 샌드위치"를 형성한다.

스타일러스 도입으로 아비트럼은 EVM을 넘어 실행 모델을 확장하여 고성능 WASM 기반 스마트 컨트랙트를 구현한다. 이 통합을 통해 Geth에 추가적인 수정 사항이 적용되어 스타일러스 거래과의 호환성을 보장하는 동시에 Ethereum과 유사한 실행 보장을 유지한다. 이러한 변경 사항에는 스타일러스 전용 거래 유형 처리 및 EVM과 WASM 환경 간의 원활한 상호 작용 보장이 포함된다.

다음 섹션에서는 이러한 수정 사항을 자세히 살펴보고, 니트로가 'Geth를 핵심으로' 활용하고 ArbOS가 제공하는 맞춤형 개선 사항을 통해 고급 고성능 STF를 제공하는 방법을 살펴볼 것이다. ArbOS 내에서 스타일러스 관련 작업은 실행, 호스트 I/O 및 메모리 작업 관리에서 스타일러스의 역할을 강조하기 위해 별도로 다룬다.

핵심 Geth: 아비트럼 니트로에서 수정된 Geth

니트로를 Geth의 가정을 위반하지 않기 위해 최소한의 수정만 한다. 이 섹션에서는 Geth와 ArbOS 간의 관계를 살펴볼 것이다. 이 관계는 일련의 후크, 인터페이스 구현, 그리고 Geth 기본 타입의 전략적 재적용으로 구성된다.

우리는 ArbOS의 상태를 Geth 스테이트DB 내부의 주소에 저장한다. 이를 통해 ArbOS는 스테이트DB의 상태 유지성과 수명 특성을 상속받는다. 예를 들어, 트랜잭션이 ArbOS에 직접 가하는 상태 변경은 롤백 시 폐기된다.

0xA4B05FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF는 ArbOS를 나타내는 가상의 계정이다.

ArbOS

ArbOS는 자식 체인 아비트럼의 실행 환경을 원활하게 하는 자식 체인 하이퍼바이저 이다. ArbOS는 상태 전환 함수 (STF)의 일부로 자식 체인에서 실행되는 신뢰할 수 있는 "시스템 연결" 구성 요소이다. 네트워크 리소스를 처리 및 관리하고, 수신 메시지와 크로스 체인 메시징을 통해 블록을 생성하며, 스마트 스마트 컨트랙트 실행을 위해 자체적으로 구현된 Geth 인스턴스를 운영한다.

아비트럼에서는 부모 체인 에서 많은 비용이 드는 작업의 대부분을 ArbOS가 완료한다. ArbOS는 자식 체인의 속도와 낮은 비용으로 이러한 기능을 신뢰할 수 있게 수행한다.

이러한 기능을 이더리움처럼 부모 체인에 적용되는 아키텍처 규칙에 내장하는 대신, 자식 체인 신뢰 소프트웨어에서 지원하는 것은 비용 측면에서 상당한 이점을 제공한다. 부모 체인 스마트 컨트랙트의 일부로 리소스를 관리하는 대신, 자식 체인에서 계산 및 저장 비용이 낮아져 이러한 작업이 가능하기 때문이다. 자식 체인에 신뢰 운영 체제를 구축하면 유연성 측면에서도 상당한 이점이 있다. 부모 체인에 적용되는 아키텍처보다 자식 체인 코드를 특정 체인에 맞게 개발하거나 사용자 정의하기가 더 쉽기 때문이다.

ArbOS는 스마트 컨트랙트가 Solidity 함수를 호출하는 것과 동일한 방식으로 호출할 수 있는 메서드를 포함하는 자식 체인별 프리컴파일을 제공한다. 프리컴파일 의 작동 방식에 대한 자세한 내용은 프리컴파일 개념 페이지를 참조하고, 아비트럼 체인에서 사용 가능한 프리컴파일에 대한 전체 참조는 프리 컴파일 참조 페이지를 참조하십시오.

프리컴파일은 contracts/src/precompiles/에 위치한 솔리디티 인터페이스와 precompiles/에 있는 대응하는 고령 구현체로 구성된다. solgen/gen.go는 Geth의 ABI 생성기를 사용하여 solgen/go/precompiles/gen/precompiles/gen.go를 생성하며, 이 파일은 프리컴파일의 ABI 데이터를 수집한다. 런타임 설치기는 이 생성된 파일을 활용하여 각 프리컴파일 구현자의 타입 안전성을 검증한다.

설치기는 런타임 리플렉션을 활용하여 각 구현자가 올바른 메서드와 시그니처를 모두 갖췄는지 확인한다. 이 리플렉션에는 선언된 순도(purity)에 기반하여 EVM 및 스테이트DB(statedb)와 같은 상태 저장 객체에 대한 접근 제한이 포함된다. 또한 설치기는 이벤트 함수 포인터를 검증하고 채워 각 프리컴파일이 로그를 출력하고 가스 비용을 파악할 수 있도록 한다. 체인 소유자만 프리컴파일 메서드를 호출할 수 있도록 제한하는 등의 추가 구성은 설치 항목에 ownerOnly 및 debugOnly와 같은 프리컴파일 래퍼를 추가하여 가능하다.

프리컴파일 메서드의 호출, 디스패치 및 기록 완료는 런타임 리플렉션을 통해 이루어지며, 이는 바이트를 수동으로 파싱하고 작성할 때 발생할 수 있는 인적 오류를 방지하고, 값의 패킹 및 언패킹을 위해 Geth의 안정적인 API를 사용한다.

자식 체인 전용 프리컴파일의 메서드를 트랜잭션이 호출할 때마다, 소모된 가스를 추적 및 기록하기 위한 호출 컨텍스트가 생성된다. 편의상, 이 컨텍스트는 기본 TxProcessor의 공용 필드에 대한 접근도 제공한다. 하위 트랜잭션이 이 구조체 업데이트 없이 롤백될 수 있으므로, TxProcessor는 최상위 트랜잭션이 지불한 상위 체인 콜데이터 양과 같이 안전한 정보만 공개한다.

검증 및 증명

검증 및 증명 개요

아비트럼의 검증 및 증명 메커니즘은 이더리움의 보안을 유지하면서 오프체인 실행의 무결성을 보장한다. 낙관적 롤업(옵티미스틱 롤업) 방식인 아비트럼은 이의 제기가 없는 한 거래가 정상적으로 실행된다고 가정한다. 아비트럼은 분쟁 발생 시 상호 작용적인 사기 방지(fraud proof)를 활용하여 효율적으로 문제를 해결한다.

아비트럼 검증 모델의 핵심은 실행과 증명의 분리이다. 거래는 낙관적으로 실행되지만, 상태 전환은 필요한 경우 온체인에서 검증될 수 있다. 아비트럼은 증명에 최적화된 수정된 WASM 형식인 WAVM(아비트럼 Virtual Machine용 WebAssembly)과 대화형 다중 라운드 이의 제기 시스템을 통해 의견 불일치를 해결하는 BoLD 분쟁 프로토콜을 통해 이를 구현한다.

롤업 프로토콜 및 검증

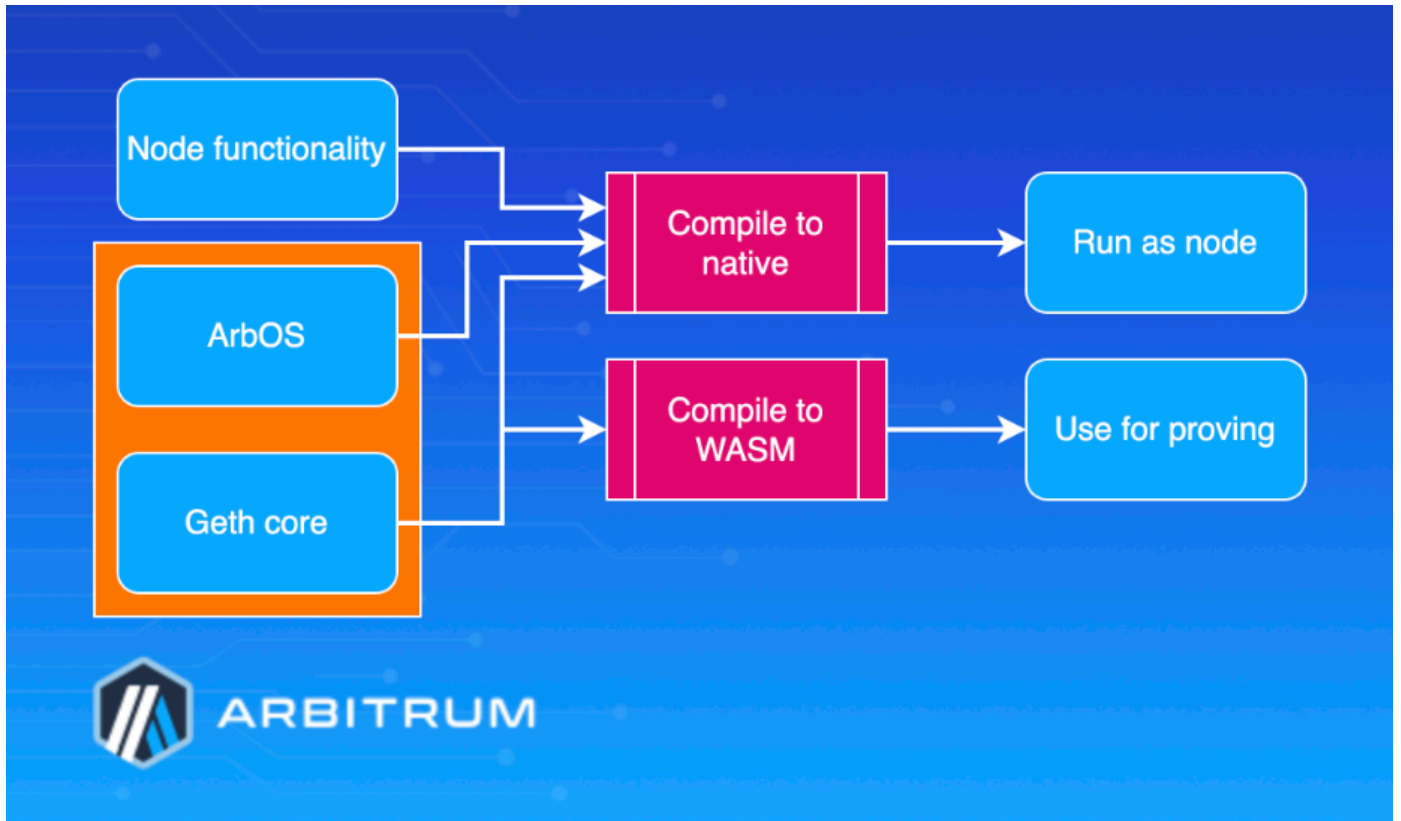
아비트럼은 분쟁이 발생하지 않는 한 거래 주장을 기반으로 체인 상태를 진행하는 낙관적 롤업(옵티미스틱 롤업) 방식으로 작동한다. 이러한 접근 방식은 이더리움의 무신뢰 검증 모델을 통해 보안을 유지하면서 확장성, 비용 절감, 높은 처리량을 제공한다.

낙관적 롤업은 실행과 검증을 분리하여 즉각적인 검증 없이 효율적인 거래 처리를 보장한다. 실행 프로세스는 상태 전환 함수에서 자세히 다룬다. 이 섹션에서는 검증 및 검증 메커니즘에 중점을 둔다. 자식 체인 검증자가 체인의 상태를 확인하는 방법과 확정을 위해 어설션을 부모 체인에 게시하는 방법을 살펴보겠다. 어설션이 잘못된 것으로 판단될 경우 향후 이의 제기 가능성도 함께 고려한다.

증명 및 도전 프로토콜

이 문서는 롤업 프로토콜에서 정확한 실행을 증명하고 분쟁을 해결하는 메커니즘을 자세히 설명한다. 시스템이 일반 실행과 증명을 분리하고, WASM을 특수 증명 형식(WAVM)으로 변환하며, 상호 작용적인 사기 증명을 활용하여 분쟁을 효율적으로 해결하는 방법을 설명한다.

1. 실행과 증명



실용적인 Rollup 시스템을 설계하는 데 있어 핵심 과제 중 하나는 빠르고 일반적인 실행과 그 실행에 대한 신뢰할 수 있는 증명의 필요성 사이에서 균형을 맞추는 것이다. 니트로는 두 가지 목적에 동일한 소스 코드를 사용하지만 서로 다른 대상으로 컴파일함으로써 이 과제를 해결한다.

- 실행 컴파일 :
 - 니트로 노드 소프트웨어는 일반 Go 컴파일러를 사용하여 대상 아키텍처에 대한 네이티브 코드로 컴파일된다.
 - 이 네이티브 바이너리는 소스 코드와 Docker 이미지로 배포되므로 다양한 노드에 배포할 수 있다.
- 컴파일 증명 :
 - 증명을 위해, 상태 전환 함수 (STF) 인 코드 부분만 Go 컴파일러에 의해 WebAssembly(WASM)로 컴파일된다.
 - 생성된 WASM 코드는 WAVM이라는 특수 형식으로 변환된다.
 - STF와 관련된 분쟁이 있는 경우, 올바른 결과를 해결하기 위해 WAVM 코드가 참조된다.

2. WAVM: 전문화된 증명 형식

WAVM은 사기 방지 요구 사항을 충족하도록 설계된 WASM의 수정된 버전이다. WAVM의 특징은 다음과 같다.

- 휴대성 및 구조 :
 - WASM은 휴대성이 뛰어나고, 사양이 명확하며, 강력한 도구로 지원되기 때문에 선택됐다.

- WAVM에서 변경된 사항 :
 - 사용하지 않는 기능 제거 :
 - WAVM은 Go 컴파일러가 생성하지 않는 일부 WASM 기능을 제거한다. 변환 단계에서는 이러한 기능이 제거된다.
 - 특정 기능에 대한 제한 사항 :
 - 부동 소수점 명령어: WAVM에는 부동 소수점 명령어가 없다. 대신 모든 부동 소수점 연산은 Berkeley SoftFloat 라이브러리 호출로 대체된다.
 - 근거: 소프트웨어 부동 소수점 라이브러리를 사용하면 서로 다른 아키텍처 간의 비호환성 위험이 줄어든다.
 - 중첩된 제어 흐름 :
 - WAVM은 제어 흐름 구조를 점프 명령어로 변환하여 평면화한다.
 - 가변 시간 명령어 : WASM의 일부 명령어는 실행하는 데 시간이 가변적으로 걸릴 수 있다. WAVM은 이러한 명령어를 고정 비용 명령어를 포함하는 구조로 변환하여 증명을 간소화한다.
 - 블록체인 상호작용을 위한 추가 명령어 :
 - WAVM은 코드가 블록체인 환경과 상호 작용할 수 있도록 새로운 명령어를 추가한다.
 - 예를 들어, 새로운 명령어를 사용하면 WAVM 코드가 체인의 글로벌 상태를 읽고 쓰고, 체인의 인박스에서 다음 메시지를 검색하거나 STF 실행을 성공적으로 종료한다는 신호를 보낼 수 있다.

3. ReadPreImage 그리고 해시 오라클

WAVM에서 흥미로운 명령어는 "해시 오라클 트릭"이 사용하는 ReadPreImage이다. 이 메커니즘은 다음과 같이 작동한다.

- ReadPreImage의 기능 :
 - 입력 : 해시 H와 오프셋 I.
 - 출력: H의 원본 이미지에서 오프셋 I 위치에 있는 데이터 단어와 함께 기록된 바이트 수(I가 끝 부분에 있거나 그 이후인 경우 0).
- 안전을 위한 제약 사항 :
 - 임의의 해시에 대한 원상(preimage)을 생성하는 것은 불가능하다. 따라서 이 명령은 다음과 같은 경우에만 허용된다.
 - 사전 이미지는 공개적으로 알려져 있다
 - 사전 이미지의 크기는 고정된 상한값(약 110KB)보다 작은 것으로 알려져 있다.
- 니트로에서의 사용 예 :
 - 주립 나무 접근 :

- 니트로의 상태 트리는 머클 해시로 인덱스된 노드와 함께 머클 트리 오프체인에 저장된다.
- STF는 루트 해시만 저장하지만, ReadPreImage를 사용하여 모든 노드의 내용을 검색할 수 있다.
- 블록 헤더 :
 - 이 명령어는 제한된 크기의 최근 공개된 자식 체인 블록 헤더를 가져온다.

4. 상호작용형 사기 증명 방식을 통한 분쟁 해결

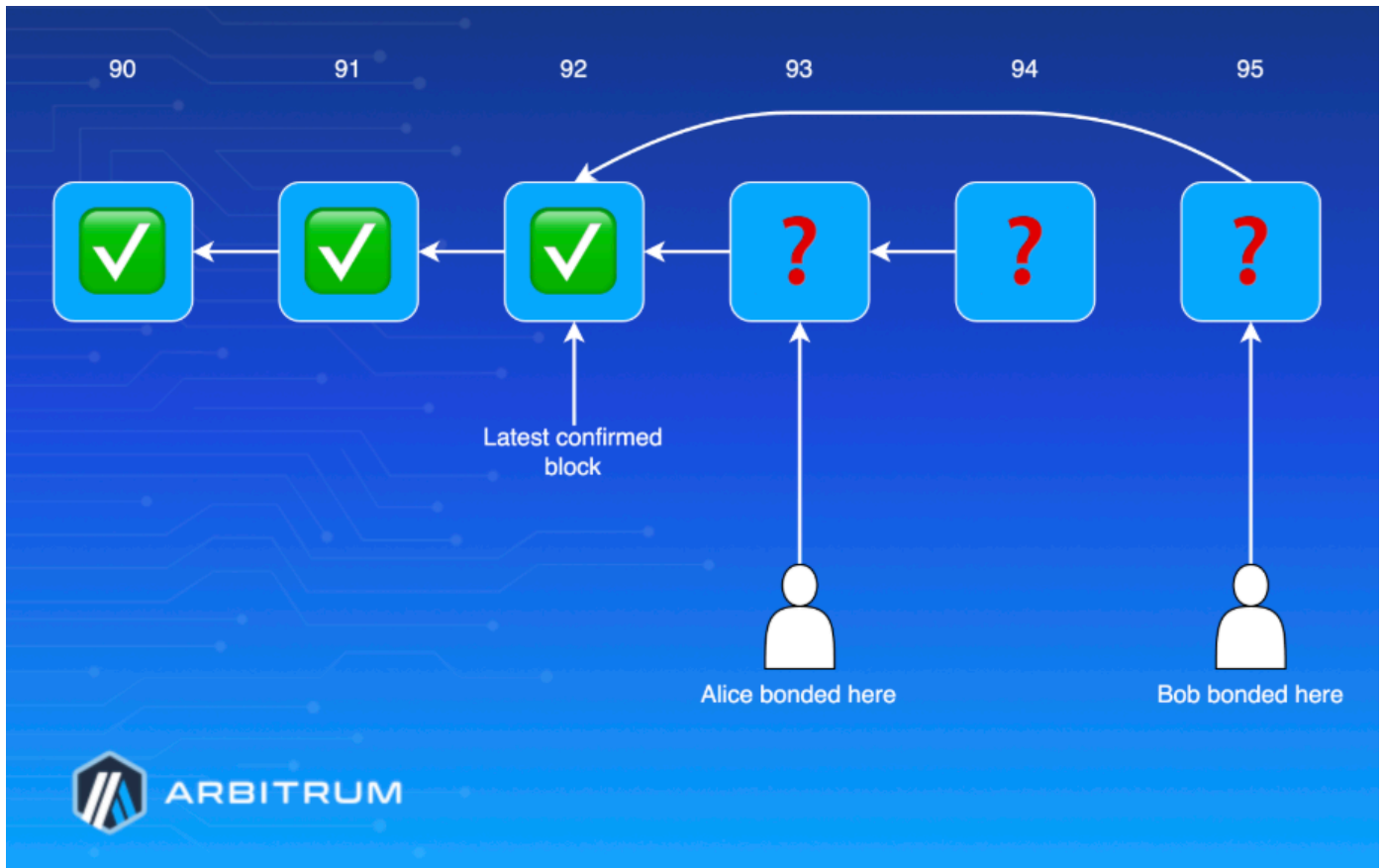
낙관적 롤업에서 분쟁 해결은 중요한 설계 결정이다. 프로토콜은 두 당사자가 의견이 일치하지 않을 때 어떤 실행 버전이 올바른지 결정해야 한다.

4.1 분쟁 해결 개요

- 낙관적 롤업 :
 - 이의가 제기되지 않는 한, 실행에 대한 주장은 옳다고 가정한다.
- 분쟁 해결 옵션 :
 - 대화형 증명 :
 - 당사자들은 분쟁의 실행 단계를 좁히기 위해 도전 게임에 참여한다.
 - 거래 재실행 :
 - 부모 체인은 각 상태 전환을 확인하면서 거래를 다시 실행한다.

4.2 대화형

아비트럼은 효율성과 유연성 때문에 대화형 증명을 선호한다.



대화형 증명 프로세스

1. 최초 청구 및 분쟁 시작 :
 - 앨리스가 이 체인이 특정 결과를 낼 것이라고 주장한다고 가정해 보자. 밥은 이에 동의하지 않는다.
2. 분쟁의 양분 :
 - 첫 번째 라운드 :
 - 앨리스는 단계 N을 다루는 주장을 게시한다.
 - 밥은 그 주장에 이의를 제기한다.
 - 앨리스는 각 $N/2$ 단계를 다루는 두 개의 하위 주장을 제공한다.
 - 이후 라운드 :
 - 밥은 이의를 제기할 절반을 선택하고, 이 과정은 반복된다. 즉, 분쟁 단계가 절반으로 줄어들어 결국 단 하나의 지침만 남게 된다.
3. 최종 1단계 증명 :
 - 분쟁이 단일 실행 단계로 좁혀지면 상위 체인 심판관은 해당 단일 단계 주장을 검증한다.

단순화된 이분법

- 단계별 분석 :

- 앨리스는 전체 계산 기록에 대한 약속을 제공한다.
- 그녀는 자신의 주장을 두 부분으로 나눈다.
- 밥은 자신이 분쟁하는 절반을 선택한다.
- 이러한 이분법의 반복은 분쟁이 단일 지침에 도달할 때까지 계속된다.

1. 앨리스의 주장이 맞다면 :

- 모든 이분법은 진실된 중간 상태를 제공한다.
- 밥의 도전은 결국 올바른 단계적 증명으로 이어져 앨리스가 승리하게 된다.

2. 앨리스의 주장이 틀렸다면 :

- 앨리스의 반쪽 중 하나는 어떤 이분법에서 거짓이 될 것이다.
- 밥은 항상 잘못된 절반을 선택할 수 있으며, 궁극적으로 검증할 수 없는 잘못된
- 단계적 증명을 강요하게 된다.
- 따라서 정직한 당사자는 항상 부정직한 청구에 이의를 제기할 수 있다.

4.3 거래 재실행(대안적 접근 방식)

- 메커니즘 :
 - 각 거래에는 실행 후 상태 해시가 포함된다.
 - 분쟁이 발생하는 경우, 부모 체인은 모든 거래를 다시 실행하여 정확성을 검증한다.
- 단점 :
 - 모든 거래에 대해 국가 청구가 필요하므로 계산 부하가 증가한다.
 - 분쟁이 발생하면 전체 거래를 체인상에서 다시 실행해야 하므로 가스 비용이 높아진다.

5. 대화형 사기 방지 및 챌린지 프로토콜

상호 작용형 사기 증거는 상충되는 주장이 제기될 때 제안자 간 분쟁을 해결하는 기초를 형성한다.

5.1 분쟁 시나리오

- 설정 예시 :
 - Rollup 체인에 어설션 93과 95가 포함되어 있다고 가정해 보겠다. 이 둘은 형제이다(둘 다 어설션 92 뒤에 온다).
 - 앨리스는 주장 93에 따라 보증을 받았고, 밥은 주장 95에 따라 보증을 받았다.
 - 밥이 95에 대해 결속력을 갖는 것은 주장 93이 틀렸음을 의미한다.
- 도전의 시작 :
 - 두 제안자가 형제 주장에 보증을 걸면 자동으로 대화형 챌린지가 시작된다.

- 하나의 이의신청에는 여러 주장이 참여할 수 있다.
- 이 의정서는 이의신청을 기록하고 심판하며, 최종적으로 승자를 선언하고, 패자측의 채권을 몰수하고 제안자 자격을 박탈한다.

5.2 2단계 챌린지 게임

1. 이분 단계 :

- 목표는 분쟁을 단일 실행 단계로 좁히는 것이다.
- 각 단계에서 클레임은 더 작은 세그먼트로 이분된다(먼저 자식 체인 블록을 거쳐, 그다음 2²⁶ 명령어의 "큰 단계"를 거쳐, 마지막으로 단일 명령어로 나뉜다).
- 부모 체인 심판은 동작이 "올바른 형태"(예: 이분법이 올바르게 수행되었는지)인지만 확인하고 실행의 정확성을 평가하지 않는다.

2. 1단계 증명 단계 :

- 분쟁이 하나의 지시로 줄어들면, 모든 플레이어가 단계적 증명을 시작할 수 있다.
- 부모 체인은 제공된 증명 데이터를 사용하여 분쟁 명령을 실행한다.
- 1단계 증명이 정확하면, 분쟁의 여지가 확인되고, 문제가 해결된다.

5.3 효율성

• 낙관적 사례 :

- 대화형 증명을 사용하면 단일 주장으로 블록의 모든 거래를 포괄할 수 있다.

• 비관적인 경우 :

- 분쟁 중에는 체인상에서 단 하나의 명령만 다시 실행된다.

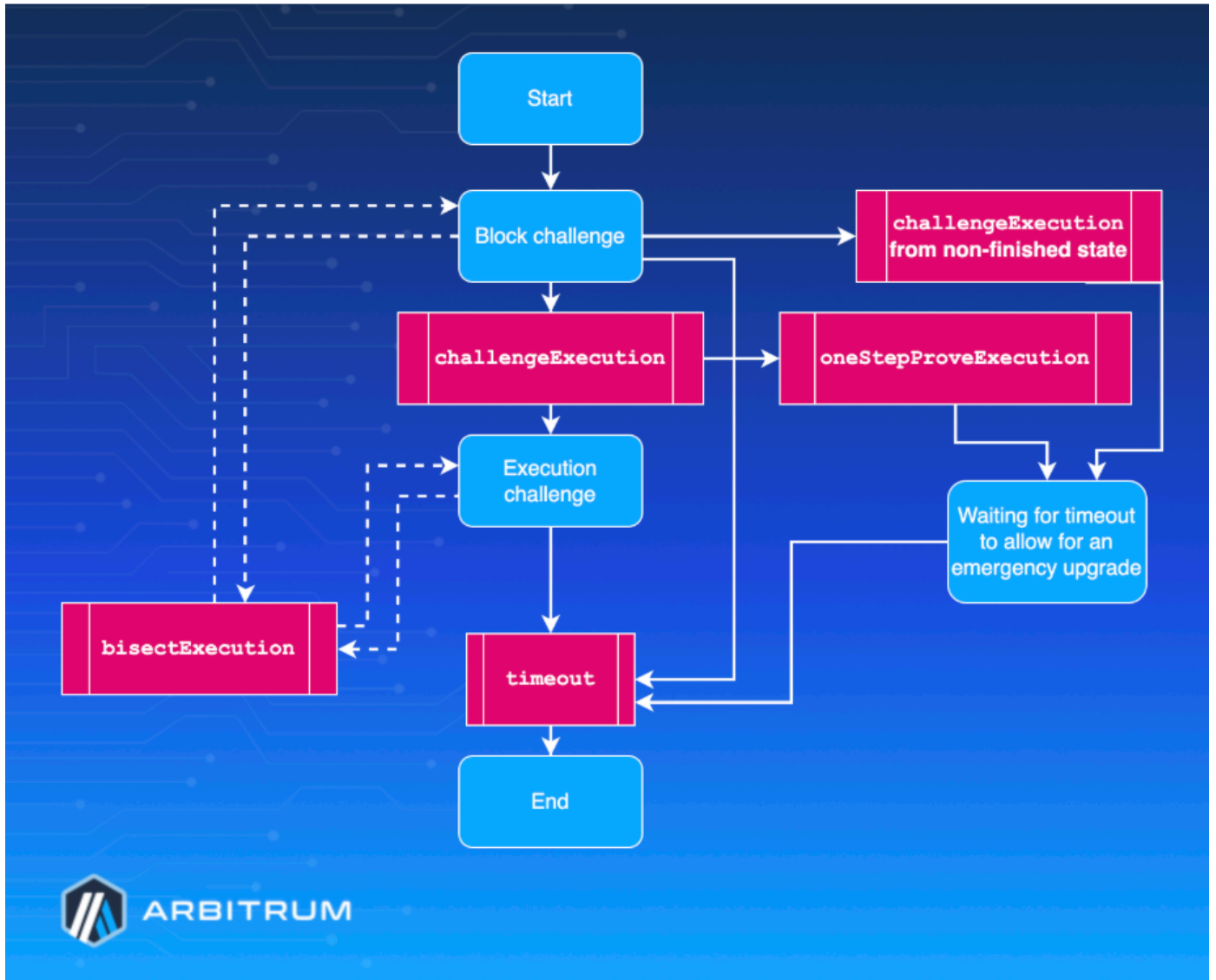
• 가스 제한 :

- 대부분의 작업이 오프체인으로 수행되므로 대화형 증명 모델은 더 높은 거래당 가스 한도로 작동할 수 있다.

• 구현 유연성 :

- 이 시스템은 전체 거래를 다시 실행하는 대신, 체인상에서 단계적으로 증명하는 검증만 요구한다.

5.4 ChallengeManager:



- 역할 :
 - ChallengeManager 스마트 컨트랙트는 챌린지 게임을 중재한다.
 - 이는 주장 기록, 타이머 및 상태 약정을 추적한다.
- 상태 머신 :
 - 상태 머신은 여러 가지 뚜렷한 단계로 과제를 표현한다.
 - 블록 챌린지 :
 - 이 과제는 글로벌 상태(블록 해시 포함)를 양분하여 분쟁을 단일 블록으로 좁히는 것으로 시작된다.
 - 빅스텝 실행 챌린지 :
 - 블록을 식별하면 실행 과제는 2^{26} 개의 명령어 "덩어리"를 양분한다.
 - 소규모 실행 과제 :
 - 이 단계에서는 분쟁이 단일 지침으로 좁혀질 때까지 청크를 양분한다.
 - 1단계 증명 :

- 최종 온체인 1단계 증명은 논란의 여지가 있는 지시를 확인한다.

- 도전에서 승리 :
 - 승리는 즉각적으로 이루어지지 않는다. 내장된 시간 지연 장치가 잘못된 도전 해결을 방지한다.
 - 이러한 지연으로 인해 필요한 경우 스마트 컨트랙트 업그레이드를 통해 오류를 진단하고 수정할 시간이 확보된다.

6. 1단계 증명

1단계 증명(OSP)은 단일 분쟁 명령에 대한 최종 판정자이다. OSP의 설계는 다음과 같은 몇 가지 핵심 가정을 고려한다.

6.1 올바른 실행을 위한 가정

- 올바른 사례 :
 - 적어도 한 명의 정직한 당사자가 관련된 이의신청에서 정직한 측은 항상 접근 가능한 사례를 증명하게 된다.
 - 중재자는 WAVM 코드(유효한 WASM 컴파일에서)를 생성하며, 올바른 실행에서는 도달할 수 없는 사례가 발생하지 않아야 한다.

6.2 접근 불가능한 경우 처리

- 정의 :
 - 도달할 수 없는 사례는 올바른 실행 중에는 결코 발생하지 않을 것으로 가정되는 상황이다.
- 악의적인 도전에서의 행동 :
 - 악의적인 주장 사이의 이의 제기에서는 도달할 수 없는 사례가 발생할 수 있으며, 이는 단계적 증명으로 임의로 처리될 수 있다.
- 정직한 도전에서의 안전 :
 - 정직한 당사자는 도달할 수 없는 사건을 증명할 필요가 없다.
 - 부정직한 당사자가 도달할 수 없는 사건을 강제로 진행하려고 시도하면, 도달 가능한 무효 사건이 먼저 제기되어 결국 부정직한 당사자가 이의를 제기하는 데 실패하게 된다.

6.3 추가 안전 가정

- WAVM 코드 유효성 :
 - 중재자가 생성한 WAVM 코드는 유효한 WASM 컴파일을 통해 생성되며, wasm-validateWebAssembly Binary Toolkit(WABT)을 사용하여 검사된다.
- 인박스 메시지 크기 :
 - 인박스 메시지는 증명에 완전히 사용할 수 있을 만큼 충분히 작아야 한다(117,964바이트를 넘지 않아야 함).

- 사전 이미지 요청 :
 - 요청된 프리이미지는 ReadPreImage알려져야 하며 크기 제한(117,964바이트) 내에 있어야 한다.
 - 예로는 블록 헤더, 상태 트라이 노드, 최근 자식 체인 블록 헤더 등이 있다.

7. WASM에서 WAVM으로의 트랜스포메이션

모든 WASM 명령어가 WAVM 명령어에 1:1로 직접 매핑되는 것은 아니다. 이러한 변환 설계는 증명 실행을 간소화하고 추가적인 블록체인 상호작용을 가능하게 한다.

7.1 변환 프로세스 개요

- 원천 :
 - Go 컴파일러는 WASM 코드를 생성한다.
- 변형 :
 - 간단한 변환 단계를 통해 WASM 코드를 WAVM 코드로 변환한다.
- 목적 :
 - WAVM 코드에 사용되지 않는 기능이 없도록 하고, 필요한 경우 기능을 제한하며, 블록체인 상호작용을 위한 추가 명령어로 보강한다.

7.2 WASM과 WAVM 핵심 차이점들

- 제거된 기능 :
 - Go 컴파일러에서 생성되지 않은 WASM 기능은 모두 제거된다.
- 제한된 기능 :
 - Berkeley SoftFloat 라이브러리 호출은 부동 소수점 명령어를 대체한다.
 - 중첩된 제어 흐름은 점프로 평평해진다.
 - 가변 시간 명령은 고정 비용 명령으로 변환된다.
- 추가된 명령어 :
 - 블록체인과 상호작용하기 위한 새로운 명령어가 추가됐다(예: 전역 상태 읽기, 인박스 메시지 가져오기, 완료 신호 보내기).

7.3 특정 WASM 구조

- 블록 및 루프 :
 - WASM에서 블록에는 명령어와 블록을 종료하기 위한 분기 명령어가 포함된다.
 - WAVM에서는 명령어가 평면적이므로 분기 명령어는 미리 정해진 목적지로의 점프로 대체된다.

- 만약 그리고 그렇지 않다면 :
 - ArbitraryJumpIf 명령어가 포함된 블록으로 변환된다.
 - 구조 예시:
 - 종료점 end로 블록 시작
 - else 블록으로 조건부 점프
 - [if-문 명령어]
 - 무조건 분기 (else 건너뛰기)
 - Else 블록: [else-문 명령어]
 - 블록 종료
- 분기 명령어 (br 및 br_if):
 - 컴파일 시점에 점프 대상이 알려진 ArbitraryJump 및 ArbitraryJumpIf로 변환.
- 분기 테이블(br_table):
 - 가능한 각 분기에 대한 일련의 조건부 검사로 변환되며, 조건이 모두 충족되지 않을 경우 기본 분기(default branch)가 포함된다.
- Local.tee:
 - 로컬 값을 복제(Dup 명령어 사용)한 후 LocalSet을 실행하는 방식으로 구현된다.
- 반환:
 - 함수 시그니처는 반환 값의 개수를 알고 있다:
 - 각 반환 값마다 MoveFromStackToInternal 명령어가 추가된다.
 - 루프는 IsStackBoundary를 사용하여 스택을 정리한다.
 - 마지막으로 각 반환 값에 대해 MoveFromInternalToStack 오퍼코드가 추가되고, 그 뒤에 Return 오퍼코드가 추가된다.
- 부동 소수점 명령어:
 - f32 및 f64 값은 i32 및 i64로 비트 캐스팅된다.
 - 모듈 간 호출은 부동 소수점 라이브러리로 이동한다.
 - 반환 값은 부동 소수점 유형으로 다시 비트 캐스팅된다.

7.4 WASM

WAVM에는 증명을 간소화하는 여러 가지 고유한 연산 코드가 포함되어 있다. 이러한 연산 코드는 다음과 같은 여러 범주로 구분된다.

불변량 및 코드생성 내부 연산코드

- 일반 가정:

- 많은 오퍼코드들은 스택 상의 특정 항목을 가정한다(예: “i32를 팝한다”는 스택 최상위 항목이 반드시 i32여야 함을 의미).
- WASM 검증 및 중재자 코드 생성은 이러한 불변 조건을 유지한다.

8. WAVM 부동 소수점

WAVM의 부동 소수점 연산은 결정성과 일관성을 보장하기 위해 다르게 처리된다.

- Berkeley SoftFloat를 사용한 구현 :
 - WAVM에서 IEEE 754-2019를 직접 구현하는 대신, 부동 소수점 명령어는 Berkeley SoftFloat-3e 라이브러리 호출로 대체됐다.
 - 소프트 float 라이브러리는 링크되어 있으며, 부동 소수점 연산은 모듈 간 호출을 통해 구현된다.
- 알려진 차이점 :
 - 예를 들어, 부동 소수점에서 정수로의 잘림은 오버플로우로 인해 포화 상태가 되며(오류 발생 대신), 이는 더 안전하고 x86 동작과 더 일관성이 있다.
 - 이러한 차이는 아직 널리 채택되지 않은 제안(예: 포화 명령어)에 따른 것이다.

9. WAVM 모듈 및 연결

WASM의 모듈 개념은 WAVM에서 다중 모듈 프로그램을 지원하도록 확장됐다. 이 확장을 통해 다음이 가능하다.

9.1 진입점

- 정의 :
 - 진입점 모듈은 실행의 시작점이다.
- 행동 :
 - 지정된 경우 모든 모듈의 시작 함수를 호출한다.
 - 그러면 메인 모듈의 main함수를 호출한다.
 - 이동을 위해 :
 - argv를 [“js”]로 설정(JS 환경을 모방하기 위해)하고 run을 호출한다
 - Rust의 경우 :
 - 인수 없이 main를 간단히 호출한다.

9.2 내보내기 라이브러리

- 내보내기 명명 규칙 :
 - 라이브러리는 module__name 패턴을 사용하여 함수를 내보낸다.

- 이 규칙에 따라 향후 라이브러리나 메인 모듈에서 이러한 기능을 가져올 수 있다.
- 예 :
 - 라이브러리 wasi-stub는 WASI(WebAssembly System Interface)를 준수하여 Rust가 가져오는 함수를 제공한다.

9.3 WAVM 게스트

- 메커니즘 :
 - 라이브러리는 명명 체계를 사용하여 기본 모듈의 내보내기를 호출할 수 있다. "env" "wasm_guest_call___*".
- 사용 예 :
 - go-stub은 wasm_guest_call_resume()을 통해 비동기 이벤트를 큐에 넣을 때 Go의 resume 함수를 호출하며, wasm_guest_call_getsp()로 새로운 스택 포인터를 가져온다.

9.4 호출자 모듈 내부 호출

- 목적 :
 - 각 스택 프레임은 호출자 모듈과 내부 오프셋에 대한 정보를 보관한다.
- 구현 :
 - WAVM은 각 모듈에 작은 메모리 로드나 저장 작업을 수행하는 4개의 "내부" 함수를 추가한다.
 - 라이브러리는 호출자의 메모리에 접근하기 위해 wasm_caller_{load,store}{8,32}와 같은 오프코드를 사용한다
 - 오직 라이브러리만이 호출자의 메모리에 접근할 수 있으며, 제한 사항으로 인해 메인 모듈은 접근할 수 없다.

10. 스타일러스 증명 및 사기

스타일러스는 EVM과 함께 실행 경로를 도입하여 WebAssembly(WASM)에서 스마트 스마트 컨트랙트 실행을 가능하게 한다. 스타일러스 프로그램은 Geth의 EVM 구현을 통해 해석되는 것이 아니라 직접 사기 방지 검증을 거치므로, 검증 메커니즘은 표준 아비트럼 검증 아키텍처를 수정해야 했다.

10.1 스타일러스

EVM 기반 스마트 컨트랙트과 달리 스타일러스 프로그램은 사기 입증 시 리플레이 머신에 동적으로 연결된다. 이는 두 가지 주요 명령어를 통해 구현된다.

- LinkModule: 스타일러스 스마트 컨트랙트의 WASM 모듈을 리플레이 머신에 로드한다.
- UnlinkModule: 실행이 완료되면 모듈을 제거한다.

이러한 명령어는 스타일러스 스마트 컨트랙트가 재생 머신의 일부로 처리되도록 보장하여 WASM 수준에서 정밀한 사기 방지 이분법을 허용한다.

10.2 실행 증명

스타일러스는 아비트럼의 실행 증명 분리를 활용한다. 여기서 재생 머신은 WAVM에서 실행되고 스타일러스 거래에 대한 분쟁은 표준 대화형 사기 방지 프로토콜(BoLD)을 따른다. 즉, 다음을 의미한다.

1. 검증자가 스타일러스 거래 실행에 이의를 제기 하면 이의가 제기된다.
2. 분쟁은 이분법 프로토콜을 통한 단일 WAVM 명령어로 좁혀졌다.
3. OSP(One-Step Proof) 메커니즘은 스타일러스 관련 작업을 처리하도록 확장되어 명령어 수준에서 WASM 실행의 정확성을 증명한다.

10.3 증명

스타일러스 프로그램은 호스트 I/O를 통해 블록체인과 상호 작용하므로, 사기 방지 기능은 모듈 간 호출의 정확한 실행을 보장해야 한다. 다음과 같은 추가 증명 명령어(opcode)가 스타일러스 프로그램 실행을 용이하게 한다.

- CrossModuleInternalCall: 스타일러스 모듈 내의 내부 기능을 독립적으로 검증할 수 있다.
- CrossModuleForward: 호스트 I/O 호출이 정확하게 전달되도록 보장하고 forward.wat사기 방지의 일관성을 유지한다.

10.4 오류 복구 및 체인

스타일러스는 악성 프로그램이 체인을 중단하는 것을 방지하기 위해 새로운 오류 보호 메커니즘을 도입했다. 스타일러스 프로그램 내에서 검증 중 오류가 발생하면 다음과 같은 문제가 발생한다.

1. 실행은 주 리플레이 머신 컨텍스트로 돌아간다.
2. 리플레이 머신은 오류 가드 스택을 사용하여 프로그램의 실행 상태를 재설정한다.
3. 복구가 불가능한 경우 해당 거래는 무효로 표시되어 스타일러스 사기 방지가 확정적으로 유지된다.

10.5 스타일러스

One-Step Prover는 다음을 포함한 스타일러스 관련 명령어의 유효성을 검사하도록 확장된다.

- 메모리 작업 (예: MemoryGrow, HeapAlloc)
- 호스트 I/O (예: storage_load_bytes32, return_data)
- WASM 실행 내 가스 계량

스타일러스는 이러한 기능을 통합하여 아비트럼의 기존 BoLD 사기 방지 아키텍처를 활용하는 동시에 WASM 스마트 컨트랙트 실행도 EVM 거래처럼 사기 방지가 가능하도록 보장한다.

AnyTrust 프로토콜

AnyTrust는 약간의 신뢰 가정을 수용하여 비용을 낮추는 아비트럼 니트로 기술의 변형이다.

아비트럼 프로토콜은 검증자(체인의 정확성을 검증하고 올바른 결과에 대한 보증을 제공할 준비가 된 노드)를 포함한 모든 아비트럼 노드가 아비트럼 체인의 수신함에 있는 모든 자식 체인 거래의 데이터에 접근할 수 있도록 요구한다. 아비트럼 롤업은 부모 체인인 이더리움에 콜데이터(calldata)라는 이름으로 데이터를 (배치되고 압축된 형태로) 게시하여 데이터 접근을 제공한다. 이를 위해 지불하는 이더리움 가스는 아비트럼 비용에서 가장 큰 부분을 차지한다.

AnyTrust는 외부 데이터 가용성 위원회(DAC)에 의존하여 데이터를 저장하고 필요에 따라 제공한다. DAC에는 N명의 구성원이 있으며, AnyTrust는 그중 최소 두 명은 정직하다고 가정한다. 즉, (N-1)명의 DAC 구성원이 특정 데이터에 대한 접근 권한을 제공할 것이라고 약속하는 경우, 약속 당사자 중 최소 한 명은 정직해야 한다. 정직한 구성원이 두 명인데 약속을 지키지 못한 구성원은 한 명뿐이므로, 약속 당사자 중 최소 한 명은 정직해야 하며, 그 정직한 구성원은 체인의 원활한 작동을 위해 필요할 때 데이터를 제공해야 한다.

키셋은 DAC 구성원의 BLS 공개 키와 데이터 가용성 인증서(DAC)의 유효 여부에 필요한 서명 수를 지정한다. 키셋은 DAC 구성원 자격 변경을 가능하게 하고 DAC 구성원에게 키 변경 권한을 제공한다.

키셋에는 다음이 포함된다.

- DAC 회원국 수 및
- 각 DAC 멤버에 대해 BLS 공개 키 및
- 필요한 DAC 서명의 수.

키셋은 해시로 식별된다.

부모 체인 KeysetManager 스마트 컨트랙트는 현재 유효한 키셋 목록을 유지한다. 자식 체인의 소유자는 이 목록에서 키셋을 추가하거나 제거할 수 있다. 키셋이 유효해지면 KeysetManager 스마트 컨트랙트는 키셋의 해시값과 전체 내용을 포함하는 부모 체인의 이더리움 이벤트를 생성한다. 이를 통해 키셋 해시값만 있으면 누구나 나중에 내용을 복구할 수 있다.

API는 동시에 유효할 수 있는 키셋의 수를 제한하지 않지만, 일반적으로는 하나의 키셋만 유효한다.

데이터 가용성 인증서

AnyTrust의 핵심 개념은 데이터 가용성 인증서(이하 "DACert")이다. DACert에는 다음이 포함된다.

- 데이터 블록의 해시
- 만료 시간

- (N-1)명의 DAC 회원이 (해시, 만료 시간) 쌍에 서명했다는 증거 (다음으로 구성됨)
 - 서명에 사용된 키셋의 해시
 - 어떤 DAC 회원이 서명했는지 알려주는 비트맵
 - 해당 당사자가 서명했다는 것을 증명하는 BLS 집계 서명(BLS12-381 곡선에 대한) 2-of-N 신뢰 가정으로 인해 DACert는 블록의 데이터(즉, DACert의 해시의 사전 이미지)를 적어도 한 명의 정직한 DAC 멤버가 만료 시간까지 사용할 수 있다는 증거를 구성한다.

일반(AnyTrust 기반이 아닌) 니트로에서 아비트럼 시권서는 부모 체인에 데이터 블록을 calldata로 게시한다. 데이터 블록의 해시는 부모 체인의 Inbox 스마트 컨트랙트에 의해 커밋되므로 자식 체인 코드가 데이터를 안정적으로 읽을 수 있다.

AnyTrust는 시권서가 부모 체인에 데이터 블록을 게시하는 두 가지 방법을 제공한다. 위와 같이 전체 데이터를 게시하거나, 데이터 가용성을 증명하는 DACert를 게시할 수 있다. 부모 체인의 수신함 스마트 컨트랙트는 유효하지 않은 키셋을 사용하는 모든 DACert를 거부한다. DACert 유효성의 다른 측면은 자식 체인 코드로 검사된다.

수신함에서 데이터를 읽는 자식 체인 코드는 일반 니트로처럼 전체 데이터 블록을 읽는다. 만약 DACert를 발견하면, DACert에서 지정한 키셋(부모 체인인 수신함이 이를 검증했기 때문에 유효한 것으로 알려짐)을 참조하여 DACert의 유효성을 확인한다. 자식 체인 코드는 다음을 검증한다.

- 서명자 수는 최소한 키셋에 필요한 수 이상이어야 한다.
- 집계된 서명은 주장된 서명자에게 유효한다.
- 만료 시간은 현재 자식 체인 타임스탬프 이후 최소 2주 후이다.

DACert가 유효하지 않으면 자식 체인 코드는 DACert를 삭제하고 다음 데이터 블록으로 이동한다.

DACert가 유효하면 자식 체인 코드는 데이터 블록을 읽는다. DACert가 유효하므로 해당 데이터 블록은 사용 가능하다.

데이터 가용성 서버

DAC 회원사는 데이터 가용성 서버(DAS) 소프트웨어를 운영한다. DAS는 두 가지 API를 제공한다.

- 아비트럼 체인의 시권서에서만 호출되도록 설계된 시권서 API는 시권서가 DAS에 데이터 블록을 제출하여 저장할 수 있도록 하는 JSON-RPC 인터페이스이다. 배포 시 일반적으로 시권서 이외의 호출자가 이 API에 접근하는 것이 차단된다.
- 전 세계에서 사용할 수 있도록 설계된 REST API는 해시를 통해 데이터 블록을 가져올 수 있는 RESTful HTTP(S) 기반 프로토콜이다. 이 API는 완벽하게 캐시 가능하며, 배포 시 캐싱 프록시 또는 CDN을 사용하여 확장성을 높이고 DoS 공격으로부터 보호할 수 있다.
DAC 회원사만이 시권서 API를 지원할 이유가 있다. 다른 회원사들도 REST API를 사용하기를 기대하며, 이는 도움이 된다. (자세한 내용은 아래에서 설명한다.)

DAS 소프트웨어는 구성 옵션에 따라 로컬 파일, Badger 데이터베이스, Amazon S3 또는 여러 백업 저장소에 중복 저장할 수 있다. 또한, Bigcache를 사용한 메모리 캐싱이나 Redis 인스턴스 캐싱을 선택적으로 지원한다.

시권서-DAC 상호작용

아비트럼 시권서가 DAC를 사용하여 게시하려는 데이터 배치를 생성할 때, RPC를 통해 배치 데이터와 만료 시간(일반적으로 3주 후)을 모든 DAC 멤버에게 병렬로 전송한다. 각 DAC 멤버는 데이터 해시값으로 색인된 데이터를 백업 저장소에 저장한다. 그런 다음 멤버는 BLS 키를 사용하여 (해시값, 만료 시간) 쌍에 서명하고, 서명을 성공 표시자와 함께 시권서에 반환한다.

시권서가 충분한 서명을 수집하면, 서명을 집계하여 (해시, 만료 시간) 쌍에 대한 유효한 DACert를 생성할 수 있다. 그런 다음 시권서는 해당 DACert를 부모 체인의 수신함 스마트 컨트랙트에 게시하여 자식 체인의 AnyTrust 체인 소프트웨어에서 사용할 수 있도록 한다.

시권서가 몇 분 안에 충분한 서명을 수집하지 못하면 DAC 사용 시도를 포기하고, AnyTrust가 아닌 체인에 서처럼 전체 데이터를 부모 체인에 직접 게시하여 "롤업"으로 돌아간다. 자식 체인 소프트웨어는 두 가지 데이터 게시 형식(DACert 또는 전체 데이터)을 모두 이해하고 각 형식을 올바르게 처리한다.

자식에서 부모로의 체인 메시징

아비트럼의 아웃박스 시스템은 임의의 자식 체인에서 부모 체인으로의 스마트 컨트랙트 호출, 즉 자식 체인에서 시작된 메시지가 부모 체인에서 실행되어 최종적으로 확인되는 것을 허용한다. 자식 체인에서 부모 체인으로의 메시지(일명 "송신" 메시지)는 아비트럼의 부모 체인에서 자식 체인으로의 메시지 와 몇 가지 공통점을 가지고 있지만, "역방향"으로 동작하며 차이점이 있는데, 이 섹션에서 자세히 살펴보겠다.

프로토콜

자식 노드에서 부모 노드로 전송되는 메시지는 아비트럼의 Rollup 상태에 포함되고 Rollup 프로토콜을 통해 완료된다. 이 프로세스는 다음 단계를 따른다.

1. 자식 체인에서 메시지 생성 :
 - 자식-부모 체인 메시지를 시작하려면 사용자 또는 계약이 ArbSys 프리컴파일러의 `sendTxToL1` 메서드를 호출한다.
2. 단언에 메시지 포함 :
 - 해당 메시지는 다른 거래와 함께 일괄 처리되어 Rollup 어설션에 포함된다.
 - 주장은 부모 체인의 롤업 스마트 컨트랙트에 제출되고 분쟁 창(약 1주일)에 들어간다.
3. 주장 확인 :
 - 분쟁 기간 이후에도 주장에 이의가 제기되지 않으면 롤업 스마트 컨트랙트에 따라 주장이 확정된다.
 - 해당 주장의 머클 루트는 부모 체인의 아웃박스 스마트 컨트랙트에 게시된다.
4. 부모 체인에서 실행 :
 - 주장이 확인되면 누구나 부모 체인에 메시지가 포함되어 있음을 증명하여 메시지를 실행할 수 있다.
 - `OOoutbox.executeTransaction`을 사용하여 실행이 가능하다. 이 메서드는 메시지가 최종 확정된 어설션에 존재함을 증명하는 머클 증명을 인수로 받는다.

클라이언트 흐름(메시지를 보내고 실행하는 방법)

자식 체인에서 부모 체인

스마트 컨트랙트 또는 사용자는 다음을 호출하여 자식 체인에서 메시지를 보낸다.

```
ArbSys(100).sendTxToL1(destAddress, calldata);
```

메시지를 부모 체인에서 실행 중

7일간의 챌린지 기간이 끝난 후, 주장이 확인되면 누구나 부모 체인에서 메시지를 실행할 수 있다.

1. 증명 데이터 검색 :
 - NodeInterface스마트 컨트랙트에서 constructOutboxProof 메서드를 호출하여 증명 데이터를 가져온다.
2. 부모 체인에서 실행
 - 증명 데이터를 Outbox.executeTransaction를 호출하여 부모 체인에서 메시지를 실행한다.

프로토콜 설계

노드

- 롤업 계약에서 confirmNode를 호출하는 데는 어설션 내 메시지 수와 무관하게 일정한 가스 오버헤드가 발생한다.
- 이 확인을 통해 악의적인 행위자가 많은 발신 메시지로 어설션을 제출하여 네트워크를 손상시킬 수 없다.

자식-부모 체인 메시지에 수동

사전 자금 조달된 가스로 자동으로 실행할 수 있는 재시도 가능 티켓 과 달리 , 자식-부모 체인 메시지는 Ethereum(부모 체인)이 예약된 실행을 지원하지 않기 때문에 수동으로 실행해야 한다.

그러나 애플리케이션은 제3자가 수수료를 받고 메시지를 실행할 수 있도록 하는 실행 시장을 구현할 수 있다.

지속성과 만료

- 자식-부모 체인 메시지 : - 아웃박스에 포함되면 부모 체인에 무기한으로 유지된다.

부모-자식 체인 메시지

각 메시지는 세 가지 기본 상태를 거친다.

메시지 처리 단계

단계	설명
자식 체인에 게시됨	메세지는 <code>ArbSys.sendTxToL1</code> 을 통해 전송된다.
마무리를 기다리는 중	해당 메시지가 포함된 주장은 이의 제기 기간(~6.4일)에 있다.
부모 체인에서 확인 및 실행 가능	사기 증거가 제출되지 않으면 주장이 확인되고 해당 메시지는 발신함에서 실행할 수 있다.

이더 인출

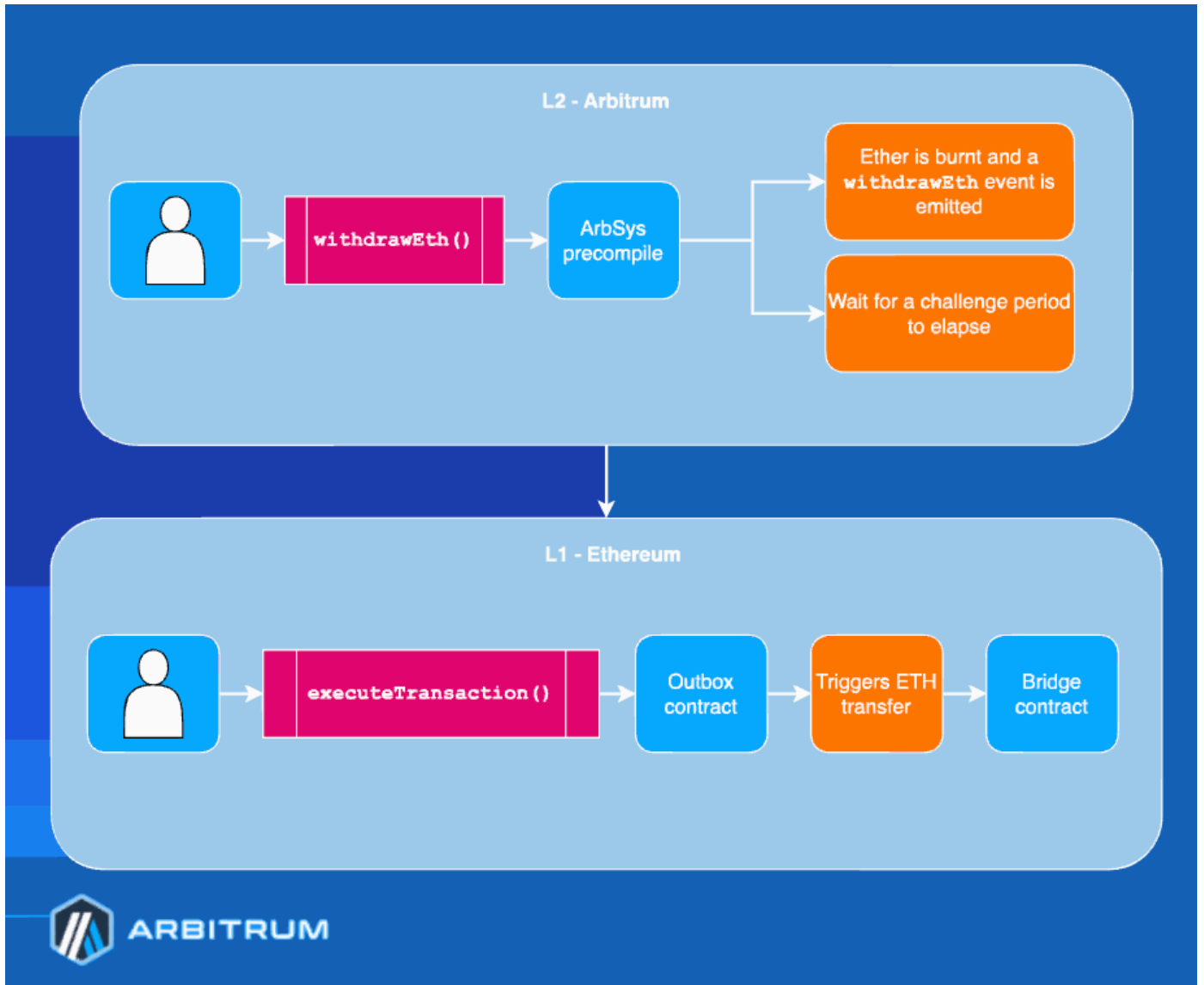
Ether를 인출하려면 ArbSys 사전 컴파일 방법을 사용하면 된다.

```
ArbSys(100).withdrawEth{ value: 2300000 }(destAddress)
```

출금 시, 이더 잔액은 자식 체인 측에서 소각되고, 나중에 부모 체인 측에서 사용 가능하게 된다.

`ArbSys.withdrawEth`는 `calldataForL1`이 빈 상태로 `ArbSys.sendTxToL1`을 호출하는 것과 동등한 편의 함수이다. 다른 `sendTxToL1` 호출과 마찬가지로, 분쟁 기간이 만료된 후 사용자가 부모 체인에서 자금을 최종적으로 청산하기 위해서는 부모 체인에서 `Outbox.executeTransaction`에 대한 추가 호출이 필요하다. 인출이 실행되면(아웃박스에서 제거됨), 사용자의 이더 잔액이 부모 체인에서 입금 처리된다.

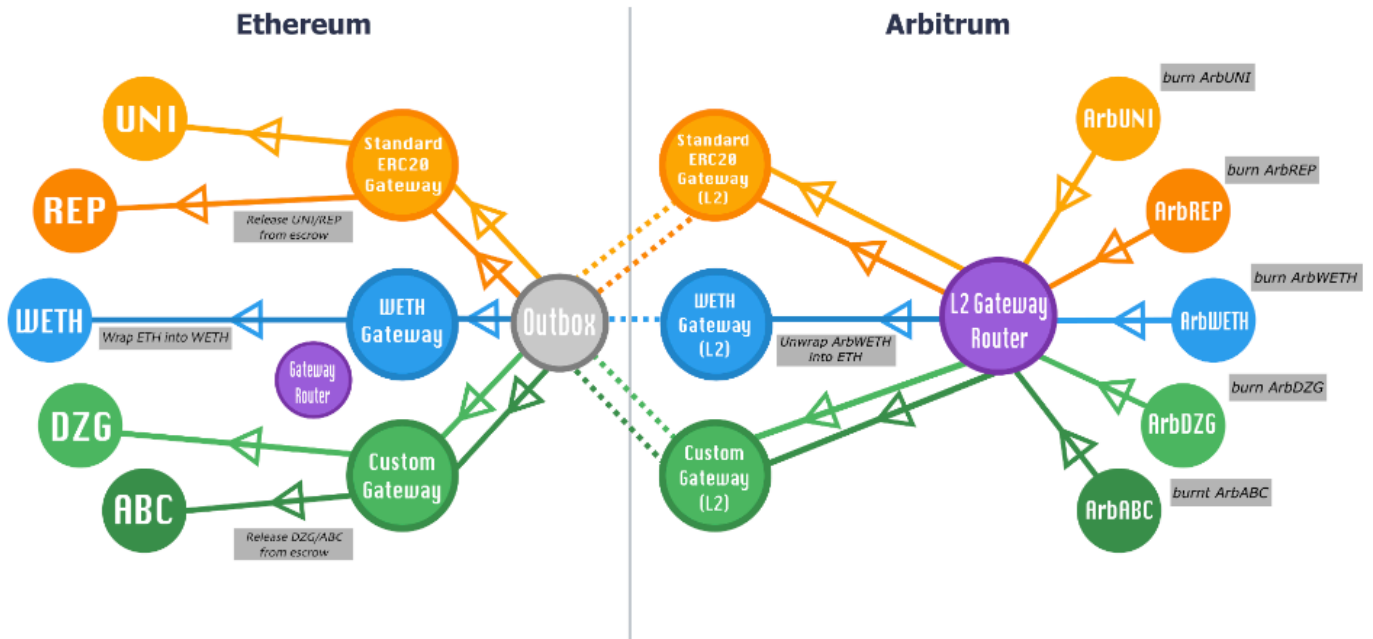
다음 다이어그램은 자금 인출 작업 중에 따르는 프로세스를 보여준다.



ERC-20 토큰 인출

아비트럼은 정식 브리지 설계 및 아키텍처를 가지고 있으며, 이에 대해서는 부모 체인에서 자식 체인으로의 브리징 문서의 토큰 브리징 섹션에서 자세히 설명한다. 이 섹션에서는 아비트럼 정식 브리지가 자식 체인에서 부모 체인으로의 토큰 브리징에서 어떻게 작동하는지 설명한다.

1. 자식-부모 체인 전송은 자식 체인의 L2GatewayRouter 스마트 컨트랙트를 통해 시작된다.
2. 자식 체인의 토큰 게이트웨이 스마트 컨트랙트(L2ArbitrumGateway)이 호출된다.
3. 마지막으로 해당 스마트 컨트랙트를 사용하여 부모 체인의 해당 게이트웨이 스마트 컨트랙트와 통신한다.



주어진 게이트웨이 페어링의 경우, 호출은 해당 라우터(L2GatewayRouter)를 통해 시작되어야 하며, 게이트웨이는 TokenGateway 인터페이스에 부합해야 한다. TokenGateway 인터페이스는 특정 토큰에 필요한 모든 브리징 기능을 지원할 만큼 유연하고 확장 가능해야 한다.

토큰 인출은 자식-부모 체인 메시징에서 가장 일반적으로 사용되는 방식이다. 표준 인출은 다음과 같이 진행된다(표준 게이트웨이의 경우).

1. 자식 체인에서 사용자가 L2GatewayRouter.outBoundTransfer를 호출하면, 이는 arbSomeERC20Token의 게이트웨이(즉, L2ERC20Gateway)에서 outBoundTransfer를 호출한다.
2. 이 호출은 arbSomeERC20Token 토큰을 소각하고 L1ERC20Gateway.finalizeInboundTransfer에 대한 인코딩된 메시지와 함께 ArbSys를 호출하며, 이는 결국 부모 체인에서 실행된다.
3. 분쟁 기간이 만료되고 사용자의 트랜잭션에 대한 어설션이 확인되면, 사용자는 Outbox.executeTransaction을 호출할 수 있다. 이 호출은 인코딩된 L1ERC20Gateway.finalizeInboundTransfer 메시지를 호출하여 L1ERC20Gateway 계약의 에스크로에서 사용자의 토큰을 해제한다.

가스 및 수수료

아비트럼은 니트로 체인의 실행 비용을 추적하기 위해 가스를 사용한다. 이더리움의 gas와 동일하게 작동하는데, 모든 EVM 명령어는 이더리움과 동일한 양의 가스를 소모한다.

사용자가 거래를 제출할 때 비용을 지불하는 당사자는 두 명이다.

- 부모 체인 리소스(예: 거래를 게시하는 데 필요한 부모 체인 호출 데이터)에 대한 포스터(환불 가능한 경우)
- 네트워크 수수료 계정은 자식 체인 리소스에 대한 것으로, 여기에는 자식 체인 노드가 거래를 처리하기 위해 감당해야 하는 계산, 저장 및 기타 부담이 포함된다.

부모 체인 구성 요소는 Brotli를 사용하여 거래 자체에서 계산한 거래의 배치 크기에 대한 추정 기여도와 부모 체인 데이터 가격에 대한 자식 체인의 관점의 결과이며, 이 값은 배치 게시자가 궁극적으로 공정하게 보상을 받을 수 있도록 시간이 지남에 따라 동적으로 조정된다.

자식 체인 구성 요소는 Geth가 바닐라 부모 체인의 본더에게 지불하는 기존 수수료(예: 상태 전환 함수 적용 시 발생하는 계산 및 저장 비용)로 구성된다. ArbOS는 자식 체인별 프리컴파일 실행에 대해 추가 수수료를 부과하며, 해당 수수료는 호출 실행 시 사용된 특정 리소스에 따라 동적으로 책정된다.

다음 섹션에서는 부모 체인과 자식 체인 수수료를 계산하는 방법을 자세히 설명한다. 정확한 계산이나 기술적 이해가 필요하지 않으면 다음 섹션인 부모 체인에서 자식 체인으로의 메시징으로 넘어가세요.

모회사 체인 가스 가격

ArbOS는 부모 체인 가스의 가격을 동적으로 책정하며, 시간이 지남에 따라 부모 체인 가스 수수료로 수집된 금액이 충당해야 할 비용에 최대한 가깝도록 가격을 조정한다.

모회사 체인 비용

부모 체인 비용에는 일괄 게시 비용과 보상이라는 두 가지 유형이 있다.

일괄 게시 비용은 일괄 게시자가 부모 체인에 일괄 데이터를 게시하는 데 지불하는 실제 비용을 반영한다. 일괄 게시가 이루어질 때마다, 해당 일괄 게시를 기록하는 부모 체인 스마트 컨트랙트는 자식 체인 ArbOS에 특별 "일괄 게시 보고서" 메시지를 전송하여 누가 일괄 비용을 지불했는지, 그리고 당시 부모 체인의 기본 수수료는 얼마였는지 보고한다. 이 메시지는 체인의 "지연된 인박스"에 저장되므로, 일정 시간 후에 자식 체인 ArbOS에 전달된다.

배치 게시 보고 메시지가 자식 체인에 도착하면 ArbOS는 보고된 기본 수수료에 배치의 데이터 비용을 곱하여 참조된 배치의 비용을 계산한다. (ArbOS는 인박스 상태에서 배치의 데이터를 검색하고, 배치에서 0바이트와 0이 아닌 바이트의 수를 세어 배치가 사용했을 부모 체인 가스를 계산한다.) 결과 비용은 가격 책정자에 의해 배치를 제출한 것으로 보고된 당사자에게 지급해야 할 금액으로 기록된다.

두 번째 유형의 부모 체인 비용은 거래 호출 데이터 처리에 대한 선택적(체인당) 단위당 보상이다. 일반적으로 보상은 시퀀서 (시퀀서) , AnyTrust 체인의 데이터 가용성 위원회(Data Availability Committee) 구성원, 또는 체인을 대신하여 호출 데이터 바이트당 비용을 부담하는 다른 사람에게 지급될 수 있다. 보상은 데이터 단위당 wei 고정된 개수이며, 단일 주소로 지급된다.

부모 체인 가격 책정자는 지금까지 처리된 데이터 단위 수를 기반으로 보상 주소로 지급되는 금액을 추적한다. 이 금액은 자식 체인에 일괄 게시 보고서가 도착할 때마다 업데이트된다.

부모 체인 콜데이터 수수료

부모 체인의 콜데이터 수수료는 시퀀서, 즉 시퀀서의 거래 배치를 이더리움에 게시하는 배치 게시자가 부모 체인의 가스 비용을 발생시켜 콜데이터로 이더리움에 거래를 게시하기 때문에 발생한다. 부모 체인의 콜데이터 수수료에서 모인 자금은 배치 게시자에게 입금되어 비용을 충당한다.

시퀀서를 통해 들어오는 모든 거래는 부모 체인의 콜데이터 수수료를 지불한다. 지연된 인박스를 통해 들어오는 거래는 일괄 게시 비용에 추가되지 않으므로 이 수수료가 발생하지 않는다. 하지만 이러한 거래는 지연된 인박스에 입력될 때 이더리움에 가스 수수료를 지불한다.

부모 체인 가격 책정 알고리즘은 각 시퀀서 거래에 부모 체인 콜데이터 수수료를 할당한다. 먼저, 거래 크기를 계산하는데, 이는 거래이 압축된 배치에 얼마나 많은 바이트를 추가할 것인지에 대한 추정치이다. 이 계산 공식에는 거래의 압축 가능성에 대한 추정치가 포함된다. 둘째, 계산된 크기 추정치에 현재 추정 바이트당 가격을 곱하여 거래의 부모 체인 콜데이터를 계산한다. 마지막으로, 이 비용을 현재 자식 체인의 기본 수수료로 나누어 수수료를 자식 체인의 가스 단위로 환산한다. 그 결과는 거래의 "포스터 수수료"로 보고된다.

예상 바이트당 가격은 부모 체인에서 징수된 총 콜데이터 수수료와 배치 게시자가 실제로 지불한 총 수수료를 비교하여 두 수수료가 최대한 동일해지도록 하는 동적 알고리즘에 의해 설정된다. 배치 게시자의 비용이 수수료 수입보다 적으면 가격이 상승하고, 배치 게시자의 비용이 수수료 수입을 초과하면 가격이 하락한다.

모회사 체인 수수료 징수

부모 체인 가스는 시퀀서 배치의 일부로 도착한 경우에만 거래에 대한 요금이 부과된다. 즉, 누군가 부모 체인에 거래를 게시하기 위해 부모 체인 가스 비용을 지불했을 것이다.

부모 체인에 거래를 게시하는 데 드는 예상 비용은 거래의 예상 규모와 현재 부모 체인의 가스 기본 수수료를 곱한 값이다. 이 예상 비용을 현재 자식 체인의 가스 기본 수수료로 나누면 부모 체인 운영에 필요한 자식 체인 가스량을 구할 수 있다(자세한 내용은 이 문서 에서 확인할 수 있다).

추정 크기는 부모 체인 가스에서 측정되며 다음과 같이 계산된다. 먼저 brotli-zero 알고리즘을 사용하여 거래 데이터를 압축한 다음 결과 크기에 16을 곱한다. (16은 부모 체인이 바이트당 16가스 요금을 부과하기 때문이다. 부모 체인은 0인 바이트에 대해 더 적은 요금을 부과하지만 여기서는 의미가 없다.) Brotli-zero는 압축 가능한 거래를 게시하는 사용자에게 보상하기 위해 사용된다. 이상적으로는 전체 배치의 압축 가능성(brotli 압축기 사용)에 기여하는 거래를 게시하는 사용자에게 보상을 제공하고 싶지만 정의하기 어려운 개념이며 어쨌든 자식 체인에서 계산하기에는 너무 비싸다. Brotli-zero는 계산하기에 충분히 저렴한 근사치이다.

거래에서 징수된 부모 체인 가스 수수료 자금은 특별 계좌로 이체되므로 해당 계좌의 잔액은 징수된 자금의 양을 나타내며 비용을 지불하는 데 사용할 수 있다.

모체 체인 가격 책정자는 수신된 "데이터 단위"(추정 크기의 합계에 16을 곱한 값)의 총 수도 기록한다.

자식 체인 가스 가격 책정

특정 Arbitrum 체인의 자식 체인 가스 가격에는 설정된 최저값이 존재하며, 이는 ArbGasInfo의 getMinimumGasPrice 메서드를 통해 조회할 수 있다.(현재 Arbitrum One에서는 0.01 gwei, Nova에서는 0.01 gwei).

자식 체인 가스 추정

Arbitrum 노드의 eth_estimateGas RPC를 호출하면 주어진 자식 체인의 가스 가격에서 전체 트랜잭션 수수료를 충당하기에 충분한 값을 반환한다. 즉, eth_estimateGas에서 반환된 값에 자식 체인의 가스 가격을 곱하면 트랜잭션 성공에 필요한 총 이더리움(ETH) 양을 알 수 있다. 이는 특정 작업에 대해 eth_estimateGas가 반환하는 값이 시간에 따라 변동함을 의미한다(부모 체인의 콜데이터 가격 변동에 따라). (자세한 내용은 2차원 수수료 및 Arbitrum에서 가스 추정 방법 참조)

자식 체인 가스

차일드 체인 가스 수수료는 이더리움의 가스 수수료와 매우 유사하게 작동한다. 거래는 일정량의 가스를 사용하며, 이 가스에 현재 기본 수수료를 곱하여 해당 거래에 부과되는 차일드 체인 가스 수수료가 계산된다.

자식 체인 기본 수수료는 이더리움 커뮤니티에서 폭넓게 논의된 "지수 메커니즘"의 한 버전으로 설정되며, 이더리움의 EIP-1559 가스 가격 책정 메커니즘과 동일한 것으로 나타났다.

이 알고리즘은 가스 사용량을 속도 제한이라는 매개변수와 비교한다. 속도 제한은 체인이 시간 경과에 따라 지속 가능하게 처리할 수 있는 초당 가스 목표량이다. (현재 아비트럼 One의 속도 제한은 초당 7,000,000 가스이다.) 이 알고리즘은 가스 백로그를 추적한다. 거래가 가스를 소비할 때마다 해당 가스는 백로그에 추가된다. 시계가 1초씩 움직일 때마다 속도 제한은 백로그에서 차감되지만, 백로그는 0보다 낮아질 수 없다.

직관적으로, 백로그가 증가하면 알고리즘은 가스 사용량이 지속 가능한 수준을 초과하므로 가스 가격을 인상하여 가스 사용량을 줄여야 한다. 백로그가 감소하면 가스 사용량이 지속 가능한 한도보다 낮으므로 가격이 다시 하락하여 가스 사용량 증가를 허용해야 한다.

이를 더 정확하게 설명하자면, 기본 수수료는 백로그의 지수 함수이다. $F = \exp(-a(Bb))$ 여기서 a 와 b 는 적절히 선택된 상수이다. a 는 백로그와 함께 가격이 얼마나 빨리 상승하는지를 제어하고, b 는 기본 수수료 상승이 시작되기 전에 약간의 백로그를 허용한다.

자식 체인 팁

시퀀서는 선착순으로 거래의 우선순위를 정한다. 이 모델에서는 팁이 의미가 없으므로 무시된다. 아비트럼 사용자는 선택한 팁과 관계없이 항상 기본 수수료만 지불한다.

가스 추정 재시도 기능

한 거래이 다른 거래를 스케줄링할 때, 노드의 RPC를 통해 가스비를 추정할 때 후속 거래의 실행이 포함된다. 따라서 모든 거래이 주어진 가스 한도에서 성공해야만 거래의 가스비를 추정할 수 있다. 이는 특히 재시도 가능 거래를 처리하고 리딤 시도를 스케줄링할 때 중요하다.

redeem 호출은 호출의 모든 가스를 제공하기 때문에, 여러 번 호출하려면 각 하위 호출에 제공되는 가스량을 제한해야 한다. 그렇지 않으면 첫 번째 호출이 모든 가스를 가져가고 두 번째 호출은 추정 가스 한도와 관계없이 필연적으로 실패하게 된다.

재시도 가능한 제출에 대한 가스 추정은 NodeInterface를 통해 가능하며, 마찬가지로 자동 상환 시도가 성공해야 한다.

속도 제한

니트로 체인의 보안은 한 검증자가 어설션을 생성하면 다른 검증자들이 이를 확인하고, 올바른 어설션은 응답하고, 잘못된 어설션은 챌린지 로 응답한다는 가정에 달려 있다. 이를 위해서는 다른 검증자들이 각 어설션을 신속하게 확인하여 시의적절한 챌린지를 발행할 시간과 자원을 확보해야 한다. 아비트럼 프로토콜은 어설션의 마감일을 설정할 때 이러한 점을 고려한다.

이는 니트로 체인 실행에 실질적인 속도 제한을 설정한다. 장기적으로 체인은 검증자가 실행을 에뮬레이션할 수 있는 속도보다 더 빠르게 진행될 수 없다. 어설션이 속도 제한보다 빠른 속도로 게시되면, 마감일은 미래에 점점 더 멀어질 것이다. 롤업 프로토콜 스마트 컨트랙트에 의해 적용되는 마감일의 미래 한계로 인해, 결국 새로운 어설션의 속도가 느려지고, 결과적으로 실질적인 속도 제한이 적용된다.

제한 속도를 정확하게 설정하려면 어설션 검증에 필요한 시간을 어느 정도 정확하게 예측할 수 있어야 한다. 검증 시간 예측에 불확실성이 있으면 안전을 위해 제한 속도를 낮게 설정해야 한다. 제한 속도를 낮추고 싶지 않기 때문에 정확한 예측을 가능하게 하려고 노력한다.

총 비용 및 가스 건적

거래에 부과되는 총 수수료는 자식 체인 기본 수수료에 사용된 자식 체인 가스비와 부모 체인의 콜데이터 요금을 곱한 값이다. 이더리움과 마찬가지로, 거래가 충분한 가스비를 공급하지 못하거나 현재 기본 수수료보다 낮은 기본 수수료 한도를 지정하면 거래가 실패한다. 이더리움은 "팁"도 허용하지만, 니트로에는 이 필드를 무시하고 팁을 수집하지 않는다.

자금 할당 및 빗 지불

하위 체인에서 일괄 게시 보고서가 처리될 때, 가격 책정자는 수집된 자금의 일부를 발생한 비용 지불에 할당한다. 자금을 할당하기 위해 가격 책정자는 다음 세 가지 타임스탬프를 고려한다.

- `currentTime`은 일괄 포스팅 보고 메시지가 자식 체인에 도착하는 현재 시간이다.
- `updateTime`은 보고된 배치가 제출된 시간이다(일반적으로 약 20분 전 `currentTime`)
- `lastUpdateTime`은 이전에 보고된 배치가 제출된 시간이다.

가격 산정기는 할당 비율 $F = (\text{업데이트 시간} - \text{마지막 업데이트 시간}) / (\text{현재 시간} - \text{마지막 업데이트 시간})$ 을 계산하고 `L1PricerFundsPool` 내 자금의 비율 F 를 현재 보고서에 할당한다. 핵심 개념은 가격 결정자가 `lastUpdateTime`과 `currentTime` 사이에 모인 자금 규모를 알고 있으며, 이 자금 중 `lastUpdateTime`과 `updateTime` 사이 구간에 할당할 비율을 계산한다는 점이다. 주어진 공식은 `lastUpdateTime`과 `currentTime` 사이 구간에서 자금이 균일한 속도로 유입되었다고 가정할 때 올바른 할당 방식이다. 가격 결정자는 마찬가지로 총 데이터 단위 중 일부를 현재 보고서에 할당한다.

이제 가격 결정자가 할당된 자금을 지급하여 지급 예정인 보상과 배치 게시자에게 지급 예정인 금액을 충당함으로써 각 당사자에 대한 미지급 잔액을 감소시킨다. 할당된 자금이 모든 지급 예정 금액을 충당하기에 충분하지 않은 경우 일부 미지급 금액이 남게 된다. 할당된 자금으로 모든 지급 예정 금액을 충당할 수 있는 경우, 남은 할당 자금은 `L1PricerFundsPool`로 반환된다.

부모 체인 수수료 정보

부모 체인 가스 기본 수수료는 `ArbGasInfo.getL1BaseFeeEstimate`를 통해 쿼리할 수 있다. 거래에 사용될 부모 체인 수수료를 추정하려면 `NodeInterface.gasEstimateComponents()` 또는 `NodeInterface.gasEstimateL1Component()` 메서드를 사용할 수 있다.

아비트럼 거래 영수증에는 부모 체인에서 사용된 가스량을 자식 체인 가스 단위로 표시하는 `gasUsedForL1` 필드가 포함되어 있다.

부모 체인 가스 기본 수수료

자금을 할당하고 미납 금액을 납부한 후, 모체인 Pricer는 모체인 Gas Basefee를 조정한다. 이 과정의 목표는 시간이 지남에 따라 징수 금액이 미납 금액과 같아지도록 하는 값을 찾는 것이다.

알고리즘은 먼저 잉여금(L1PricerFundsPool의 자금에서 총 미지급 자금을 뺀 금액)을 계산하는데, 이 금액은 음수일 수 있다. 잉여금이 양수이면 모체인 가스 기본 수수료가 감소하여 고정된 미래 기간 동안 징수되는 금액이 잉여금만큼 정확히 감소한다. 잉여금이 음수이면 기본 수수료가 증가하여 동일한 고정된 미래 기간 동안 부족분이 해소된다.

두 번째 항은 잉여금(현재 잉여금에서 이전 일괄 게시 보고서 처리 후 잉여금을 뺀 금액)의 미분을 기반으로 모체인 가스 기본 수수료에 추가된다. 변동을 줄이기 위해 평활화 계수를 곱한 이 항은 잉여금이 증가하면 기본 수수료를 줄이고, 잉여금이 감소하면 기본 수수료를 높인다.

데이터 가용성

아비트럼 데이터 흐름에 대한 전반적인 견해는 무엇입니까

아비트럼은 현재 두 가지 주요 데이터 가용성 메커니즘을 지원한다.

롤업

이 모드에서는 모든 거래 데이터가 부모 체인(예: 아비트럼 One의 이더리움 메인넷)에 제출된 거래의 콜 데이터(calldata) 또는 거래에서 제출된 블롭(blob)에 포함된다. 이렇게 포함되면 모든 데이터가 온체인에서 누구나 다운로드하여 확인할 수 있도록 쉽게 제공된다.

AnyTrust

AnyTrust 모드에서는 거래 데이터가 처음에 데이터 가용성 위원회(DAC)라는 노드 그룹에 제출된다. DAC는 데이터를 저장하고 배포한다. 전체 데이터 세트를 체인에 포함하는 대신, DAC에 데이터가 저장되었음을 증명하는 암호화된 증명(데이터 가용성 인증서, 또는 DACert)만 부모 체인에 제출된다. 이 증명은 체인에 저장되는 데이터 양을 크게 줄여 비용을 절감한다.

이러한 데이터 가용성 메커니즘으로 인해 아비트럼 니트로 노드는 이더리움 노드나 다른 1계층 네트워크 노드와는 다른 방식으로 데이터를 동기화한다. Go-Ethereum 노드는 정교한 P2P 네트워크를 활용하여 다른 노드를 검색하고, 데이터를 교환하고, 합의 메커니즘에 참여함으로써 이더리움 블록체인과 동기화하는 반면, 아비트럼 노드는 이러한 기존 방식과 달리 신뢰할 필요가 없는 프로세스를 사용한다.

아비트럼 데이터 흐름은 다음과 같이 작동한다.

1. 배치 및 제출:
 1. 시퀀서는 거래를 대기열에 넣고 일괄 처리한다.
 2. 이러한 배치는 부모 체인에 제출된다.
 - a. 롤업 모드에서는 시퀀서가 거래 배치를 부모 체인의 시퀀서 인박스 스마트 컨트랙트에 직접 제출한다. (Blob 또는 calldata를 통해 직접 제출)
 - b. AnyTrust 모드에서는 시퀀서가 배치를 DAC(데이터 가용성 위원회)로 보낸 다음, DAS에서 생성하여 부모 체인에 반환하는 데이터 가용성 인증서(DACert)를 제출한다.
2. 노드 동기화:
 1. 네트워크에 가입하면 전체 노드는 다음과 같다.
 - a. 롤업 모드에서는 데이터가 부모 체인 calldata 또는 blob(시퀀서가 데이터를 게시하는 방법에 따라 다름)에서 직접 읽힌다.

b.AnyTrust 모드에서는 DACert를 검사하여 데이터 가용성을 확인하고 DAC에서 데이터를 쿼리한다.

2. 노드는 최신 체인 높이를 따라잡기 위해 이 프로세스를 계속 따른다.
3. 따라잡으면 노드는 시퀀서 피드에서 직접 새로운 시퀀서 큐 메시지에 대한 업데이트를 받는다 (이 프로세스에 대한 자세한 내용은 마지막 섹션에서 제공한다).

3. 따라잡기:

1. 노드가 체인 뒤처지면 2단계에서 설명한 프로세스로 돌아가 최신 상태로 다시 동기화된다. 본질적으로 아비트럼 노드는 부모 체인에서 데이터를 검색하는 것을 우선시하고 실시간 업데이트를 위해 시퀀서에 의존하며, 이는 Ethereum 노드에서 사용하는 기존 P2P 동기화 방식과 다르다.

전체 노드가 부모 체인

아비트럼 전체 노드는 부모 체인(AnyTrust 체인의 경우 DAC)에서 수신한 데이터를 디코딩하여 로컬 상태를 업데이트한다. 이 과정에는 이벤트 모니터링, 데이터 파싱, 메시지 처리가 포함된다.

1. 이벤트 쿼리:

1. 풀 노드는 상위 체인의 인박스 계약이 발행하는 SequencerBatchDelivered 이벤트를 구독한다. 이 이벤트는 새로운 트랜잭션 배치의 도착을 의미한다.

2. 이벤트 구문 분석:

1. SequencerBatchDelivered 이벤트를 수신하면 노드는 이벤트 데이터를 SequencerInboxBatch 구조체로 파싱한다. 이 구조체는 일반적으로 다음을 포함한다:
 - a. BlockHash: 배치를 포함하는 부모 체인 블록의 해시이다.
 - b. ParentChainBlockNumber: 부모 체인 블록의 블록 번호이다.
 - c. SequenceNumber: 배치의 순서 번호.
 - d. TimeBounds: 배치에 대한 시간적 제약.
 - e. AfterDelayedAcc: 지연된 메시지를 처리한 후의 누적 해시이다.
 - f. AfterDelayedCount: 지연된 메시지 수.
 - g. rawLog: 원시 이벤트 로그 데이터.

3. 데이터 직렬화:

1. SequencerInboxBatch 구조체는 바이트 배열로 직렬화된다.
2. 직렬화된 데이터는 특정 형식을 따른다.
 - a. TimeBounds.MinTimestamp(8바이트)
 - b. TimeBounds.MaxTimestamp(8바이트)
 - c. TimeBounds.MinBlockNumber(8바이트)
 - d. TimeBounds.MaxBlockNumber(8바이트)
 - e. AfterDelayedCount(8바이트)
 - f. payload(가변 길이)

- a. 해당 payload 필드에는 다음 내용이 추가로 포함된다.
 - a. 유형: 페이로드의 헤더를 나타낸다(예: DACert, blob 메시지).
 - b. 내용: 페이로드 헤더와 관련된 실제 데이터(예: DACert, BlobHashes, brotli 압축 데이터).

4. 데이터 디코딩 및 검색:

1. payload 헤더를 기준으로:
 - a. DAS 메시지 헤더: 노드는 원시 데이터를 검색하기 위해 데이터 가용성 서버(DAS)에 쿼리를 보낸다.
 - b. Blob 메시지 헤더: 노드는 Blob 메시지를 디코딩하여 원시 데이터를 얻는다.
 - c. Brotli 메시지 헤더: 여기서는 추가 단계가 필요하지 않다. 다음 단계로 넘어가세요.
2. 데이터 압축 해제: 원시 데이터가 Brotli 압축된 경우, 노드는 해당 데이터를 압축 해제한다. 위 i와 ii에서 얻은 원시 데이터도 Brotli 압축 데이터일 수 있다는 점에 유의해야 한다.

5. 메시지 처리:

1. 데이터를 디코딩하고 압축 해제한 후, 노드는 일련의 일괄 세그먼트 메시지를 얻는다.
 - a. 메시지 유형:

배치 세그먼트 메시지 유형	이 메시지의 용도는 무엇입니까?
BatchSegmentKindL2Message	이 메시지에는 일련의 거래에 대한 원시 데이터가 포함된다. 일반적으로 이는 단일 블록이다.
BatchSegmentKindL2MessageBrotli	메시지는 위의 메시지와 동일하지만, 이것은 압축된 데이터이다.
BatchSegmentKindDelayedMessages	이 메시지에는 부모 체인인 지연된 인박스에서 읽은 새로운 지연된 메시지가 포함되어 있다.
BatchSegmentKindAdvanceTimestamp	이 메시지는 상태 전환 기능(STF)에 타임스탬프 상태를 1초 앞당기도록 알린다.
BatchSegmentKindAdvanceL1BlockNumber	이 메시지는 STF에 새로운 부모 체인 블록 번호를 알리는 메시지이다.

2. 상태 전환: 마지막으로 상태 전환 기능(STF)이 이러한 메시지를 처리하고 STF는 규칙에 따라 아비트럼 노드의 로컬 상태를 실행하고 업데이트한다.

전체 노드가 시퀀서 피드

아비트럼 전체 노드가 체인을 따라잡으면 초기 동기화 모드에서 실시간 업데이트 모드로 전환된다. 이 전환은 시퀀서 피드로부터 데이터를 수신하는 것을 포함하며, 시퀀서 피드는 새로 대기열에 추가된 거래에 대한 업데이트를 지속적으로 브로드캐스트한다.

1. 데이터 수집:

1. 전체 노드는 시퀀서 피드 또는 개인 피드에 대한 연결을 유지한다. 개인 피드를 실행하는 방법은 피드 릴레이 실행 방법 을 참조하십시오.
2. 시퀀서 피드는 최신 대기 중인 거래에 대한 정보가 포함된 데이터 패킷을 전송한다.

2. 데이터 디코딩:

1. 전체 노드는 시퀀서 피드를 읽는 방법 에 설명된 방법을 사용하여 수신된 데이터 패킷을 디코딩한다.

3. 메시지 처리:

1. 디코딩이 성공적으로 완료되면 전체 노드는 이전 섹션의 5단계에서 설명한 것과 동일한 유형의 데이터를 얻는다.
2. 메시지를 상태 전환 함수(STF)로 보내고 실행한다.
(이 단계는 이전 섹션의 5단계와 동일하다)

BoLD 분쟁 프로토콜

이 소개는 BoLD에 대해 알아보고자 하는 분들을 위한 것이다. BoLD는 아비트럼 체인에 대한 허가 없는 검증을 가능하게 하는 낙관적 롤업을 위한 새로운 분쟁 프로토콜이다. BoLD는 Bounded Liquidity Delay의 약자로, 아비트럼 One, 아비트럼 Nova, 아비트럼 Sepolia에서 활성화되어 있다.

BoLD는 정확히 무엇인가?

BoLD는 아비트럼의 이전 분쟁 프로토콜을 업그레이드했다. 특히, BoLD는 검증자가 아비트럼의 상태에 대한 분쟁을 개시하고 해결하는 데 사용하는 일부 규칙을 변경하여, 유효한 상태만 아비트럼 체인의 부모 체인(이더리움)에서 확인을 받도록 했다.

이전 분쟁 프로토콜은 사기 방지를 통해 이의를 제기했으며, 허용 목록에 있는 검증자 집합으로 제한됐다. BoLD는 누구나 체인 상태(이의 제기 포함) 검증에 참여할 수 있도록 하고, 자식-부모 체인 메시징(인출 포함)의 보안을 강화한다.

BoLD에 따라 보증 검증자의 책임은 다음과 같다.

- 아비트럼 체인 상태에 대한 클레임을 부모 체인(예: Ethereum)에 게시한다.
- 다른 검증자가 제기한 무효 청구에 대해 이의를 제기하고,
- 도전에 참여하고 승리하여 유효한 청구를 확인하시오.

BoLD는 허가 없는 검증을 해제하여 분쟁이 정해진 기간(현재는 두 번의 이의 제기 기간과 필요 시 안전보장이사회의 개입을 위한 이틀의 유예 기간, 그리고 계산을 위한 약간의 델타) 내에 해결되도록 보장한다. 이를 통해 지연 공격 위험을 효과적으로 제거하고 부모 체인으로의 인출을 더욱 안전하게 만든다. BoLD는 모든 단일 주체가 악의적인 당사자로부터 아비트럼을 방어할 수 있도록 하는 새로운 분쟁 시스템을 도입하여 누구든 허가 없이 아비트럼의 체인 상태를 검증, 제안 및 방어할 수 있도록 했다.

아비트럼에 새로운 분쟁 해결 프로토콜이 필요한 이유는 무엇인가?

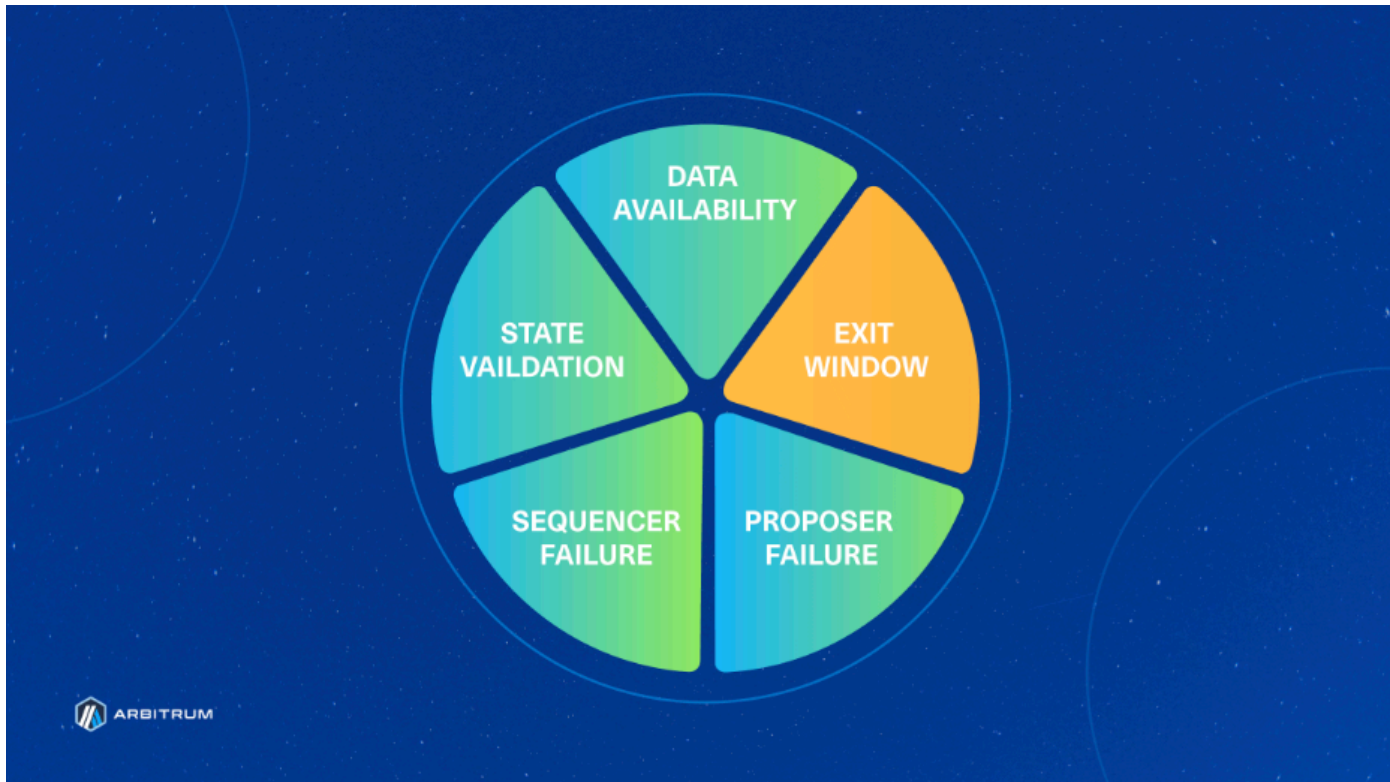
과거에는 작업 사기 증명이 허용 목록에 있는 검증자에게만 제한되어 체인 상태를 주장할 수 있었다. BoLD는 누구나 정해진 시간 내에 분쟁에 이의를 제기하고 이길 수 있도록 허용함으로써 프로토콜을 더욱 분산화한다. 아비트럼 체인은 보안을 위해 검증자와 사기 증명 간의 상호작용 증명 게임을 계속 사용할 예정이지만, 이 게임은 완전히 허가 없이 진행되며 한 번의 이의 제기 기간(기본 6.4일)과 동일한 시간 제한을 갖는다는 추가적인 이점이 있다.

BoLD는 정확한 상태 주장이 자본을 채권에 예치하는 검증자와 연계되지 않기 때문에 시간 제한적이고 허가 없는 검증을 제공하는 독보적인 방식을 제공한다. 이러한 특징과 자식 체인 상태가 완전히 결정론적이라는 사실은 이더리움에서 입증될 수 있으며, 이는 다수의 정직한 당사자가 BoLD를 통해 자신의 주장이 정확함을 입증할 수 있음을 의미한다. 마지막으로, BoLD는 아비트럼을 방어하는 데 단 하나의 정직한 당사자만 필요하다는 사실을 변경하지 않는다.

BoLD를 통해 아비트럼은 2단계

Vitalik이 제안한 이정표 에서 영감을 받아 L2BEAT 팀은 이더리움 롤업 개발 평가를 위한 널리 인정받는 프레임워크를 구축했다. Vitalik과 L2BEAT 프레임워크는 모두 롤업 개발의 마지막 단계를 " 2단계 - 보조 바퀴 없음 " 이라고 부른다. 2단계 롤업으로 간주되기 위한 중요한 기준은 누구나 제한 없이 자식 체인 상태를 검증하고 사기 방지를 이더리움에 게시할 수 있도록 허용하는 것이다. 이 단계는 2단계 롤업의 핵심 요건인데, "시스템이 제한된 집단에 의해 통제되는 것이 아니라 전체 커뮤니티의 공동 감시를 받는다"는 것을 보장하기 때문이다.

BoLD는 누구나 잘못된 아비트럼 상태 주장에 이의를 제기할 수 있도록 허용함으로써 허가 없는 검증을 가능하게 했다. 이는 네트워크 보안 참여를 위한 새로운 길을 열어 포용성과 복원력을 강화한다. BoLD는 올바른 아비트럼 상태 주장에 자본이 예치된 단일하고 정직한 주체가 악의적인 적대 세력에 맞서 항상 승리할 수 있도록 보장함으로써 이를 달성한다.



BoLD를 핵심으로 하는 아비트럼은 L2BEAT 위험 분석 파이 차트에서 현재 노란색(위)으로 표시된 상태 검증(State Validation) 문제를 해결하여 2단계 롤업으로 인정받기 위한 방향을 제시한다. BoLD는 더욱 허가 없이, 효율적이며, 강력한 롤업 생태계를 구축하는 데 기여한다.

BoLD는 모체인으로

과거에는 상태 주장(state assertion) 이후 "챌린지 기간(challenge period)"이라고 알려진 기간이 있었는데, 이 기간 동안 모든 검증자는 주어진 자식 체인 상태 루트의 유효성에 대해 이의를 제기할 수 있었다. 이의 제기 기간 동안 이의 제기가 발생하지 않으면 프로토콜은 상태 루트를 확인하고 유효성을 검증한다.

이 챌린지 기간 때문에 아비트럼 One에서 자산을 인출하려면 약 1주일(6.4일)을 기다려야 한다. 이 설계는 보안을 위해 작동하는 사기 방지 기능을 사용하지만, 지연 공격에 취약하다. 악의적인 행위자가 채권을 희생하더라도 챌린지 기간을 연장하기 위해 지속적으로 분쟁을 제기하는 것이다. 즉, 각 분쟁을 하나씩 해결하는데 걸리는 시간만큼 챌린지 기간이 무기한 연장되는 것이다. 이러한 위험은 이상적이지도 안전하지도 않으며, 아비트럼 One과 Nova의 검증이 아비트럼 DAO가 감독하는 허가된 개체 집합으로 제한되는 이유이다.



BoLD는 기존 Rollup 프로토콜에 분쟁 해결 기한을 도입하여 이러한 과제를 정면으로 해결하고, 6.4일 이내에 분쟁이 종결되도록 보장한다(DAO는 아비트럼 One과 Nova의 경우 이 기간을 변경할 수 있다). 이러한 종결은 두 가지 이유로 가능하다.

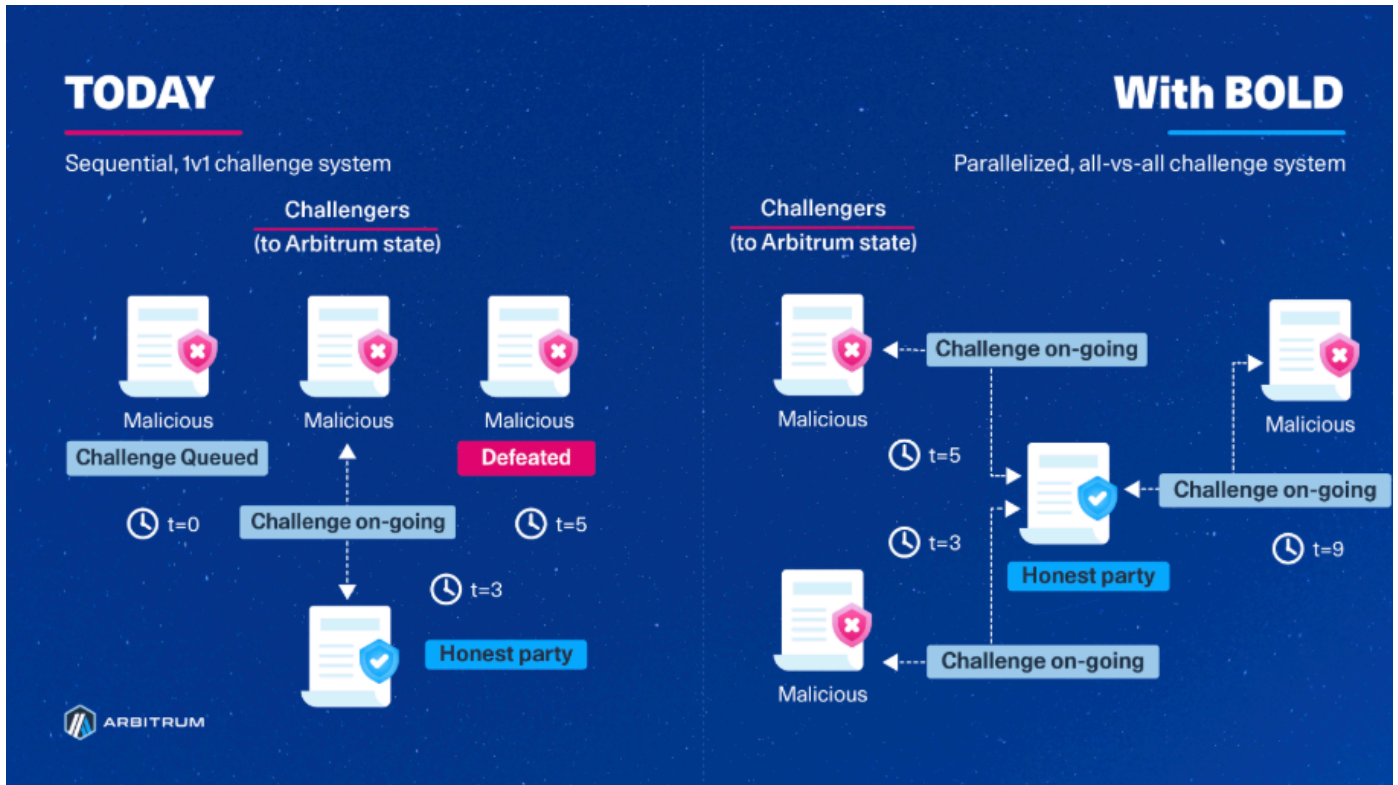
BoLD의 디자인은 정직한 당사자와 악의적인 적대자 간의 도전이 동시에 발생할 수 있도록 허용한다.

이의신청자가 응답하지 않을 경우 정직한 당사자의 주장을 자동으로 확인하는 기간 제한을 사용한다.

BoLD의 디자인은 정직한 당사자와 악의적인 적대자 간의 도전이 동시에 발생할 수 있도록 허용한다.

이의신청자가 응답하지 않을 경우 정직한 당사자의 주장을 자동으로 확인하는 기간 제한을 사용한다.

아래 그림과 비유를 통해 요약해 보겠다. 아비트럼의 이전 분쟁 해결 프로토콜은 이의가 제기된 모든 주장은 "1대1 토너먼트"처럼 각 도전자에게 순차적으로 방어되어야 한다고 가정했다. 반면 BoLD는 정직한 당사자 한 명이라도 올바른 주장을 방어하고 승리를 보장할 수 있도록 했다. 마치 "전면 대 전면 배틀 로얄"처럼, 결국에는 항상 단 한 명의 승자가 나올 수밖에 없는 상황과 유사한다.



어떻게 이런 일이 가능할까요

BoLD 프로토콜은 검증자가 아비트럼 체인 상태에 대한 주장에 이의를 제기하는 방식에 대한 가드레일과 규칙을 제공한다. 아비트럼의 상태는 결정론적이므로, 주어진 온체인 작업 및 거래 입력에 대해 항상 하나의 정확한 상태만 존재한다. BoLD 설계의 장점은 분쟁이 정해진 시간 내에 해결되도록 보장하여 지연 공격의 위험을 제거하고 궁극적으로 누구나 아비트럼의 단일 정확한 상태에 자금을 예치하고 성공적으로 방어할 수 있도록 한다.

BoLD가 실제로 어떻게 작동하는지 개요를 살펴보겠다.

1. 주장이 제기된다: 검증자는 가장 최근에 확인된 주장(블록 A)을 시작으로, Nitro의 결정론적 상태 전환 함수(STF)를 사용하여 이후 일정 수의 트랜잭션이 최종 상태인 블록 Z로 이어질 것이라고 주장한다. 검증자가 블록 Z가 나타내는 최종 상태가 정확하다고 주장할 경우, 자금을 블록 Z에 보증하고 해당 상태를 상위 체인에 제안한다. (본딩 작동 방식에 대한 자세한 내용은 대담한 기술 심층 분석 참조). 일정 시간(도전 기간)이 지난 후 아무도 이의를 제기하지 않으면, 주장이 나타내는 상태 블록 Z가 Arbitrum 체인의 올바른 상태로 확정된다. 그러나 누군가 최종 상태 블록 Z에 동의하지 않을 경우, 도전을 제출할 수 있다.
2. 도전장이 열린다: 다른 검증자가 블록 Z가 나타내는 최종 상태를 관찰하고 이에 동의하지 않을 경우, 자본을 담보로 제공하며 다른 최종 상태(주장 블록 Y로 표현됨)에 대한 주장을 제기함으로써 허가 없이 도전장을 열 수 있다. 이 시점에서 두 가지 주장 상태가 존재한다: 블록 A → 블록 Z와 블록 A → 블

록 Y. 이 시점에서 각 주장된 상태는 '에지(edge)'로 지칭되며, 특정 시작점부터 종점(예: 블록 A → 블록 Z)까지의 주장된 상태들로 구성된 머클 트리는 보다 공식적으로 '역사 커밋먼트(history commitment)'로 알려져 있다. _이 시점에서 이더리움은 어떤 에지가 옳고 그른지에 대한 개념이 없다는 점을 유의해야 한다. 에지는 단순히 검증자가 체인의 역사에 대해 주장하는 일부일 뿐이며, 이는 특정 최종 상태에서 초기 상태까지 거슬러 올라가는 과정에 대한 주장이다. 또한 검증자가 제시한 보증금은 해당 보증금을 제시한 당사자가 아닌 주장에 대한 것이므로, 잘못된 주장에 대해 이의를 제기할 수 있는 정직한 익명의 당사자는 무수히 많을 수 있다. 이의를 제기하기 위해 제시된 보증금은 롤업 계약에 보관된다는 점을 유의해야 한다. 이러한 자금 사용에 관한 Arbitrum 재단의 기대 사항을 설명하는 규정된 절차가 있으며, 요약은 아래 5단계를 참조하십시오.

3. 다단계 상호작용적 해부가 시작된다: 분쟁을 해결하려면 의견이 다른 주체들이 실제 올바른 주장 상태가 무엇이어서 하는지에 합의해야 한다. 블록 A → 블록 Z와 블록 A → 블록 Y 사이의 모든 것을 재실행하고 비교하는 것은 엄청난 비용이 들 것이다. 특히 A, Z, Y 사이에 잠재적으로 수백만 건의 트랜잭션이 존재할 수 있기 때문이다. 대신, 주체들은 각자의 역사적 약속을 반으로 나누는 작업을 번갈아 가며 수행하여 단일 명령 단계에 도달할 때까지 진행한다. 이 단계에서 이더리움과 같은 중재자가 승자를 선언할 수 있다. 이 시스템은 현재 Arbitrum 체인에서 도전이 해결되는 방식과 매우 유사하다.

BoLD는 해결 과정의 사소하지만 중요한 세부 사항만 변경한다. 다음 단계로 넘어가 보겠다:

- 블록 챌린지 : 챌린지가 열리면, 에지는 아비트럼 블록 단위의 세분성을 가지므로 레벨 제로 에지라고 한다. 분쟁 당사자들은 서로 의견이 다른 특정 블록을 찾을 때까지 과거 약속을 번갈아 가며 반으로 나눈다.
- 빅스텝 챌린지: 이제 당사자들이 분쟁 범위를 단일 블록으로 좁혔으므로, 해당 블록 내에서 앞으로 이분하는 연습이 계속된다. 이 블록은 모든 당사자가 초기 상태 다음이지만 최종 상태 이전인 상태로 합의한 것이다. 그러나 이번에는 당사자들이 블록 내 상태 전이 함수(State Transition Function)에 대한 특정 명령어 범위를 좁혀, 본질적으로 의견 불일치가 있는 명령어 집합을 파악하는 데 주력한다. 현재 이 범위는 2^20 단계의 WASM 명령어로, 아비트럼 체인 검증을 위한 최적의 조합이다.
- 한 단계 과제: 2^20개의 명령어 범위 내에서, 모든 당사자가 동의하지 않는 단일 명령어 단계에 도달할 때까지 앞뒤로 이등분하는 과정이 계속된다. 이 시점에서 당사자들은 해당 단계 이전의 아비트럼의 초기 상태에는 동의하지만, 바로 다음 단계의 최종 상태에는 동의하지 않는다. 아비트럼의 상태는 완전히 결정론적이므로 올바른 최종 상태는 단 하나라는 점을 기억하십시오.

4. 1단계 증명: 문제가 단일 단계에 대한 분쟁으로 분리되면, 양측은 해당 단계를 실행하여 1단계 증명을 생성하고 부모 체인(예: 이더리움)의 OneStepProof 스마트 스마트 컨트랙트에 제출한다. 1단계 증명은 단일 계산 단계가 특정 상태를 초래한다는 증명이다. 부모 체인의 스마트 스마트 컨트랙트는 분쟁 단계를 실행하여 양측이 제출한 증명의 정확성을 검증한다. 이 시점에서 정직한 당사자의 증명은 유효한 것으로 간주되고, 해당 에지 트리는 시간 경과에 따라 확인되지만, 부정직한 당사자의 에지는 시간 초과로 거부된다.

5. 확인: 정직한 1단계 에지가 확인되면, 프로토콜은 정직한 당사자의 레벨 0 에지에 도달할 때까지 부모 에지를 확인 또는 거부한다. 정직한 당사자의 레벨 0 에지가 확인되었으므로, 해당 당사자의 주장 보증은 환불된다. 한편, 부정직한 당사자는 부정직이 항상 처벌받도록 보증을 박탈당한다.

- 레벨 0 에지가 확인되는 또 다른 방법은 바로 시간이다. 챌린지의 각 미니 단계(블록 챌린지, 빅스텝 챌린지, 원스텝 챌린지)에서 타이머는 BoLD에서 정의한 챌린지 기간 T 까지 증가한다. 이 타이머는 파티가 이등분된 이력 커밋먼트를 제출하면 시작되고, 도전자가 이등분된 이력 커밋먼트를 응답으로 제출할 때까지 계속된다. 타이머가 T 에 도달하면 에지가 자동으로 확인된다.

6. 정직한 당사자의 모회사 체인 가스 비용과 다른 도전 레벨에서 발생한 미니 본드에 대한 상환은 아비트럼 Foundation에서 처리한다.

여기까지이다! 이제 검증자가 BoLD 프로토콜을 통해 이의 제기에 이의를 제기하는 각 단계를 살펴보았다. 마지막으로, 위에서 설명한 각 단계는 동시에 실행될 수 있으며, 이것이 BoLD가 정해진 시간 내에 분쟁 해결을 보장할 수 있는 이유 중 하나이다.

타임부스트

아비트럼 Timeboost 는 아비트럼 체인을 위한 새로운 거래 주문 정책으로 , 체인 소유자가 체인에서 최대 추출 가능 가치(MEV)를 확보하고, 스팸을 줄이며, 빠른 블록 시간을 유지하는 동시에 샌드위치 공격 및 프런트러닝과 같은 유해한 유형의 MEV로부터 사용자를 보호할 수 있도록 해준다.

Timeboost 는 Offchain Labs 팀이 1년 넘게 연구 개발 에 매진한 결과물이다. 현재 아비트럼 One과 아비트럼 Nova에서 사용 가능하며, 아비트럼 Orbit 체인에서도 사용 가능하다. 각 체인 소유자는 원하는 대로 채택하고 맞춤 설정할 수 있다.

아비트럼 체인에 Timeboost가 필요한 이유는 무엇입니까

과거 아비트럼 체인은 수신 거래를 "선착순(FCFS)" 방식으로 정렬했다. 이 정렬 정책은 이해하고 구현하기 쉬우며, 빠른 블록 생성 시간(250ms에서 필요에 따라 100ms까지)을 가능하게 하고, 프런트러닝 및 샌드위치 공격과 같은 유해한 유형의 MEV로부터 사용자를 보호한다.

그러나 FCFS 주문 정책에는 몇 가지 단점이 있다. FCFS 하에서 검색자는 오프체인 하드웨어에 투자하여 지연 시간 경쟁에 참여하고 이를 이기기 위해 노력할 인센티브를 갖는다. 이러한 "경쟁"은 아비트럼 체인의 검색자가 차익거래 및 청산 기회를 통해 수익을 창출하는 데 많은 스팸이 포함되어 체인 인프라에 부담을 주고 혼잡을 유발한다는 것을 의미한다. 또한, FCFS 하에서 아비트럼 체인에서 확보된 모든 MEV는 검색자에게 할당되므로, 사용 가능한 MEV는 체인 소유자나 체인의 애플리케이션에 반환되지 않는다.

타임부스트란 무엇인가요?

Timeboost는 FCFS의 한계를 해결하면서 대부분의 FCFS 이점을 유지한다.

타임부스트는 거래 순서 정책 이다. 이는 아비트럼 체인의 시퀀서가 사용자가 제출한 거래를 순서 지정할 때 준수해야 하는 일련의 규칙이다. 가까운 미래에는 여러 시퀀서가 분산형 타임부스트를 통해 이러한 규칙을 적용할 수 있게 될 것이다.

아비트럼 체인의 경우, 시퀀서의 유일한 역할은 사용자로부터 수신된 유효한 거래를 받아 거래 순서 정책에 따라 정렬한 다음, 최종 시퀀스를 실시간 피드와 압축된 배치로 체인의 데이터 가용성 계층에 게시하는 것이다. Timeboost 이전에는 거래 순서 정책이 FCFS(선입선출 방식)였으며, Timeboost는 이를 수정한 FCFS 순서 정책이다.

Timeboost는 아비트럼 체인이

- 아비트럼 체인의 기본 블록 시간은 Timeboost를 적용하더라도 업계 최고 수준인 250ms를 유지한다. Timeboost를 적용하면, 빠른 처리 속도(express lane)에 속하지 않는 일부 거래는 다음 블록

까지 지연될 수 있다.

Timeboost를 사용하면 아비트럼 체인은 사용자를 유해한 유형의

- 타임부스트는 경매 낙찰자에게 일시적인 시간적 이점 만 부여할 뿐 , 들어오는 거래를 확인하거나 순서를 변경하거나 모든 블록의 첫 번째 블록을 생성할 수 있는 권한은 부여하지 않는다. 또한, 거래의 메모리풀은 계속 비공개로 유지되므로 타임부스트를 활성화한 사용자는 유해한 MEV 유사 선행 공격 및 샌드위치 공격으로부터 안전하게 보호받을 수 있다.

Timeboost는 체인

- 체인 소유자는 Timeboost를 사용하여 체인에서 사용 가능한 MEV의 일부를 확보할 수 있다. 이 금액은 검색자에게 전액 지급되었을 것이다. 여기에는 맞춤형 가스 토큰을 제공하거나, 이러한 수익을 체인의 애플리케이션과 사용자에게 재분배하는 등 다양한 방식이 있다.

Timeboost는 네트워크

- 타임부스트 경매를 통해 "시간 우위 구매" 기능을 도입함으로써, 합리적이고 이윤을 추구하는 행위자들은 지연 시간 경쟁에서 승리하기 위해 하드웨어나 인프라에 투자하는 대신 경매에 투자할 것으로 예상된다. 이러한 자원 분산을 통해 네트워크에서 FCFS MEV 기반 스팸이 감소할 것으로 예상된다.

어떻게 작동하나요

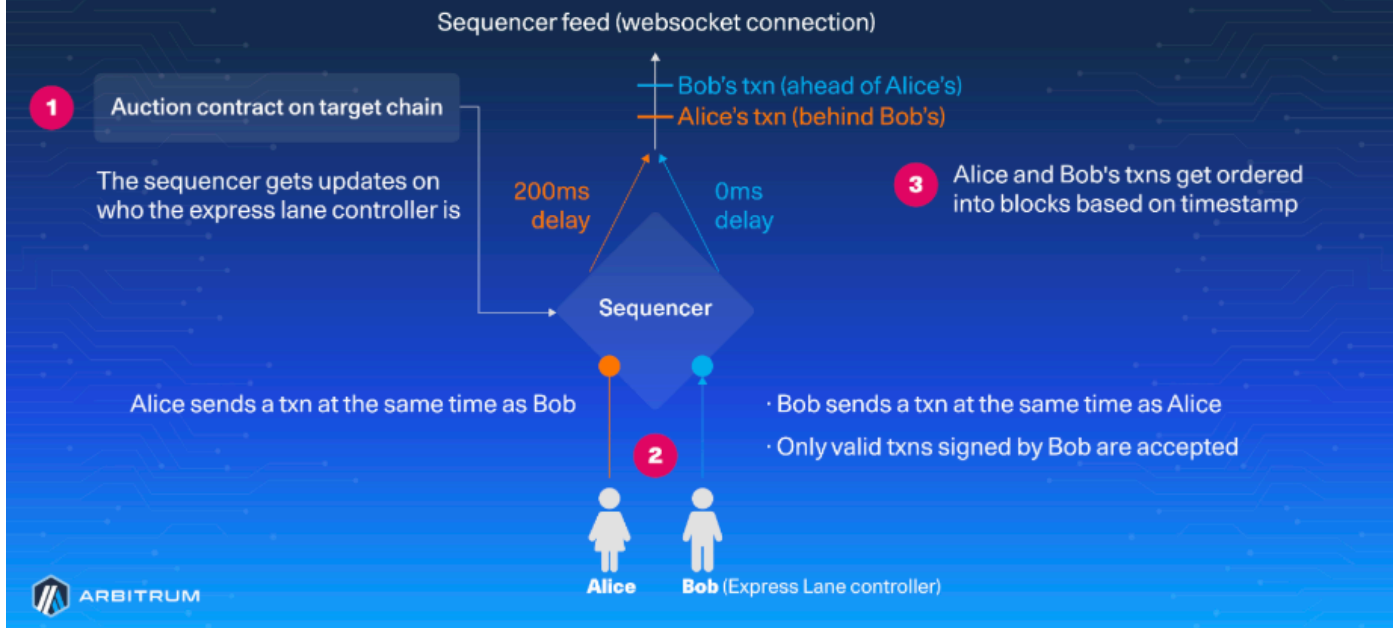
Timeboost는 함께 작동하는 세 가지 개별 구성 요소를 사용한다.

- 시퀀서가 주어진 라운드에 대한 유효한 거래를 수신하자마자 시퀀싱할 수 있는 특별한 "고속 레인"이다.
- 특정 라운드의 익스프레스 레인 관리자를 결정하기 위한 오프체인 경매이다. 이 경매는 자율 경매인 이 관리한다.
- 경매 결과와 경매 수익금 처리에 대한 정식 진실의 출처 역할을 하기 위해 대상 체인에 배포된 경매 스마트 컨트랙트이다.

우선, 라운드의 기본 지속 시간은 60초이다. 익스프레스 레인에 속하지 않는 거래는 시퀀싱되기 전에 도착 타임스탬프에 기본적으로 200밀리초의 인위적인 지연이 적용된다. 즉, 익스프레스 레인에 속하지 않는 일부 거래는 다음 블록까지 지연될 수 있다. 아비트럼의 기본 블록 시간은 250밀리초로 유지된다는 점에 유의해야 한다(원하는 경우 100밀리초로 조정 가능). 각 구성 요소의 작동 방식을 자세히 살펴보겠다.

익스프레스 레인

Using The Express Lane Workflow



익스프레스 레인 구현은 시퀀서에 공식적으로 라는 이름의 특수 엔드포인트를 사용한다. 이 엔드포인트가 특별한 이유는 여기에 제출된 거래가 시퀀서에 의해 즉시 시퀀싱되기 때문이다. 따라서 "익스프레스 레인"이라는 이름이 붙었다. 시퀀서는 현재 라운드의 익스프레스 레인 컨트롤러 `timeboost_sendExpressLaneTransaction` 가 올바르게 서명한 경우에만 이 엔드포인트로 유효한 거래 페이로드를 허용한다.

시퀀서에 제출된 거래는 일반적으로 비익스프레스 레인 거래로 간주되어 도착 타임스탬프가 200밀리초 지연된다. 익스프레스 레인과 비익스프레스 레인 모두에서 전송된 거래는 결국 시퀀서가 데이터 가용성 계층에 게시하고, 노드 작업이 체인 상태를 구성할 수 있도록 단일의 순서화된 거래 스트림으로 시퀀싱된다는 점에 유의해야 한다. 익스프레스 레인 컨트롤러는 다음을 수행하지 않는다.

- 거래를 재주문할 권리가 있다.
- 그들의 거래가 항상 "블록의 최상위"에서 먼저 이루어질 것이라는 보장을 받으세요.
- 전혀 이익을 보장하지 않는다.

익스프레스 레인의 가치는 익스프레스 레인 관리자가 다음 라운드에서 추출할 수 있을 것으로 예측하는 MEV(경매 마감 전 MEV 기회 추정치)와 익스프레스 레인 관리자가 통제하는 동안 추출할 수 있는 MEV(예상치 못한)의 합계이다. 익스프레스 레인의 가치가 어떻게 결정되는지 이해하는 것은 인위적인 지연과 경매 마감 전 시간에 적응할 때 체인 소유주에게 유용할 수 있다.

Timeboost 경매

각 라운드(기본: 60초)에서 익스프레스 레인의 통제권 결정은 라운드별 경매를 통해 이루어진다. 이 경매는 봉인 입찰, 차가 입찰 방식이다. 이 경매는 다음 라운드의 익스프레스 레인 통제권을 결정하기 위해 진행된다. 즉, 익스프레스 레인 통제권 결정은 이전 경매 라운드의 어느 시점에서나 이루어질 수 있다. 경매 입찰은 모든 ERC-20토큰, 모든 금액, 모든 주소로 가능하며, 이는 체인 소유자의 전적인 재량에 따라 결정된다.

Auction Workflow



라운드 경매는 auctionClosingSeconds라운드 시작 전(기본값: 15초)에 종료 시간을 갖는다. 이 종료 시간은 기본 매개변수에서 당사자들이 입찰을 제출할 수 있는 45초의 시간이 주어지며, 이 시간이 지나면 경매는 더 이상 입찰을 수락하지 않는다. 입찰 종료 후 새 라운드가 시작되기까지의 15초 동안 자율 경매인은 모든 입찰을 검증하고, 낙찰자를 결정하며, 온체인 경매 스마트 컨트랙트에 호출하여 경매를 공식적으로 완료한다.

경매 스마트 컨트랙트

경매에 입찰하기 전에 당사자는 경매 스마트 컨트랙트에 자금을 입금해야 한다. 언제든지 입금하거나 기존 입금에 추가 자금을 추가할 수 있다. 이러한 입금은 전액 출금 가능하며, 기존 라운드의 결과에 영향을 미치지 않도록 약간의 지연 시간(기본 2라운드 또는 2분)이 발생한다. 최소 입금 금액은 없지만, WETH "최소 입찰가"라고 하는 0.001(기본 금액 및 토큰)의 시작 최소 입찰가가 필요하다.

체인 소유자는 최소 예약 가격을 설정하며, 이 가격은 다음 라운드 시작 전 최대 30초(기본값)까지 언제든지 업데이트될 수 있다. 이를 통해 경매 참가자는 입찰을 제출하기 최소 30초 전에 예약 가격을 항상 알 수 있다. 또한, 체인 소유자(또는 체인 소유자가 지정한 주소)는 최소 입찰가를 높이기 위해 예약 가격을 설정할 수 있다. 경매 스마트 컨트랙트에 따라 예약 가격은 최소 예약 가격보다 낮을 수 없다.

자율 경매자가 낙찰자를 결정하면, 경매 계약은 최고 입찰자의 계정에서 두 번째로 높은 입찰 금액을 차감하고 해당 자금을 기본적으로 체인 소유자가 지정한 수혜자 계정으로 이체한다. 최고 입찰에 명시된 `expressLaneControllerAddress`가 해당 라운드의 익스프레스 레인 컨트롤러가 된다.

기본 매개변수

앞서 언급한 기본 Timeboost 매개변수 중 몇 가지를 아래에 소개한다. 이러한 매개변수를 비롯한 모든 매개변수는 체인 소유자가 설정할 수 있다.

매개변수 설명 및 권장 기본값

매개변수 이름	설명	권장 기본값
roundDurationSeconds	현재 라운드의 익스프레스 레인 컨트롤러가 서명한 거래에 대해 시퀀서가 익스프레스 레인 권한을 존중하는 시간이다.	60초
auctionClosingSeconds	다음 라운드 시작 전 시간이다. 자율 경매사는 이 시간 동안 입찰을 접수하지 않는다.	15초
beneficiary	flushBeneficiaryBalance() 경매 스마트 컨트랙트 이 체결될 때 Timeboost 경매 수익금이 보내지는 주소이다.	체인점 소유자가 관리하는 주소
_biddingToken	Timeboost 경매에서 입찰에 사용되는 토큰의 주소이다. ERC-20(선택한 토큰에 이체 수수료, 리베이징, 이체 후크 또는 기타 비표준 ERC-20 로직이 없는 경우) 어떤 토큰이든 가능하다.	WETH
nonExpressDelayMsec	익스프레스 레인 거래의 시퀀싱이 이루어지기 전에 익스프레스 레인 거래의 도착 타임스탬프에 적용되는 인위적인 지연이다.	0.2초 또는 200밀리초
reservePrice	현재 Timeboost 경매에 대한 경매 스마트 컨트랙트에서 허용되는 최소 입찰 금액을 _biddingToken으로 표시된다.	없음
_minReservePrice	Timeboost 입찰의 "최저가"로 작용하려면 최소가 같거나 이하이어야 한다. reservePrice 경매 스마트 컨트랙트에 따라 적용된다.	0.001 WETH

Timeboost는 누구를 위한 제품이며, 어떻게 사용하나요?

Timeboost는 아비트럼 체인 인프라에 선택적으로 추가할 수 있는 기능으로, Timeboost 활성화 여부는 체인 소유자의 재량에 달려 있으며, 아비트럼 체인은 Timeboost 없이도 정상적으로 작동할 수 있다.

Timeboost를 활성화하면 다양한 집단에 각기 다른 수준의 효과와 이점을 제공한다. 아래에서 자세히 살펴 보겠다.

일반 사용자의 경우

타임부스트의 영향은 미미할 것이다. 익스프레스 레인이 아닌 거래는 200ms의 지연 시간을 경험하게 되며, 이는 일반 사용자의 경우 거래가 시퀀싱되어 블록에 포함되는 데 약 450ms(기존 약 200ms)가 소요됨을 의미한다. 모든 사용자는 프라이빗 메모를 계속 사용함으로써 샌드위치 공격 및 선행 공격과 같은 유해한 MEV 활동으로부터 보호받을 수 있다.

체인점 소유자를 위한

타임부스트는 토큰 가치를 축적하고 체인 수익을 창출하는 독특한 방식을 제시한다. 체인 소유자는 타임부스트 경매를 설정하여 네트워크에서 가스비로 사용되는 토큰과 동일한 토큰으로 입찰 수익을 수집한 후, 이 수익금을 어떻게 사용할지 선택할 수 있다.

검색자/중재자를 위한

타임부스트는 기존 또는 향후 MEV 전략에 독특한 변화를 주어 이전보다 수익성을 높일 수 있다. 예를 들어, 타임부스트 경매를 통해 제공되는 시간적 이점을 구매하는 것이 하드웨어 투자 및 지연 경쟁에서 승리하는 비용보다 저렴할 수 있다. 또 다른 예로, 익스프레스 레인 이용권을 시간대 또는 거래 건당으로 다른 당사자에게 재판매하는 새로운 비즈니스 모델이 있다.

체인 소유자를 위한 타임부스트 특별 안내

아비트럼 니트로의 많은 새로운 기능 및 업그레이드와 마찬가지로, Timeboost는 체인 소유자가 원하는 대로 배포하고 사용자 지정할 수 있는 선택적 기능이다. 활성 아비트럼 체인에 Timeboost를 배포하고 활성화/비활성화하더라도 체인이 중단되거나 영향을 받지 않지만, 체인의 거래 순서 정책에는 영향을 미친다. Timeboost가 배포되었지만 비활성화된 경우 또는 특정 라운드에 대한 고속 레인 컨트롤러가 없는 경우 아비트럼 체인은 기본적으로 FCFS(선불식별) 방식으로 전환된다.

아비트럼 체인은 Timeboost를 배포하고 활성화하기 전에 해당 체인에 대한 Timeboost의 적용 가능성과 사용 사례를 전체적으로 평가할 것을 권장한다. 일부 아비트럼 체인은 애초에 충분한 MEV(예: 차익거래)를 확보하지 못할 수 있으므로 이러한 평가가 필수적이다. 또한, 아비트럼 체인은 Offchain Labs에서 권장하는 기본 매개변수를 기반으로 시작하여, 매개변수 조정을 고려하기 전에 시간 경과에 따른 결과와 체인 생태계에 미치는 영향을 면밀히 모니터링할 것을 권장한다.