DEX 차원에서의 MEV 해결책

1팀
배서준(디사이퍼), 문보설(자유전공학부), 김재현(경제학부)
유형준(경제학부), 이민지(컴퓨터공학부), 김지아(보건학과)

목차

DEX에서
MEV SOLUTION 필요성

– *MEV*와 *DEX*의 개념

MEV

트랜잭션 순서를 임의로 뼌경하여 추출할 수 있는 수익

유저에게 금전적인 피해를 끼치고 네트워크 부하를 증가시킬 위험 O

피해 사례) Sandwich attack
트랜잭션을 보낸 유저가 생각한 금액보다 더 비싼 가격에 자산을 구매하게 됨

피해사례2) MEV 추출을 노리는 봇들의 가스비 경쟁
네트워크 부하 증가 → 사용자 시간적, 금전적 손해 발생

어플리케이션 레벨에서의 MEV 해결책으로 선제적인 대응이 필요함

crList, PBS와 같은 네트워크 레이어에서의 제안이 있었지만 반영 시기 미지수

DEX

분산형 거래소(Decentralized Exchange)

중앙 집중화된 시스템을 사용하지 않고 블록체인 기술을 통해 거래를 처리하는 플랫폼

사용자 본인이 자신의 키를 소유하는 거래소

본인이 자신의 암호 화폐를 온전히 제어하며, 보안 방식을 결정

단점

사용자가 온전히 보안에 책임을 져야 하는 위험 있음

DeFi MEV Solution

- flashbots

MEV-GETH

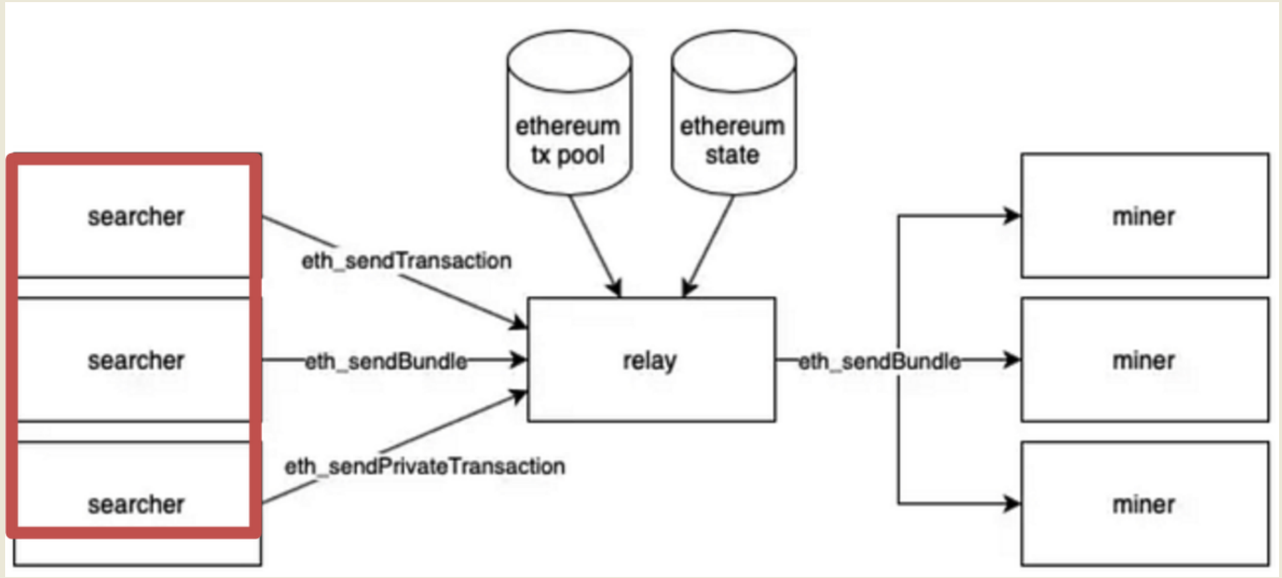
Idea

Private transaction mempool + Sealed bid blockspace auction

- Private Transaction
  - User가 Miner에게 tx를 직접 전송
  - Miner는 tx를 다른 일반 tx보다 앞에 놓아 블록을 생성

MEV-GETH

Architecture

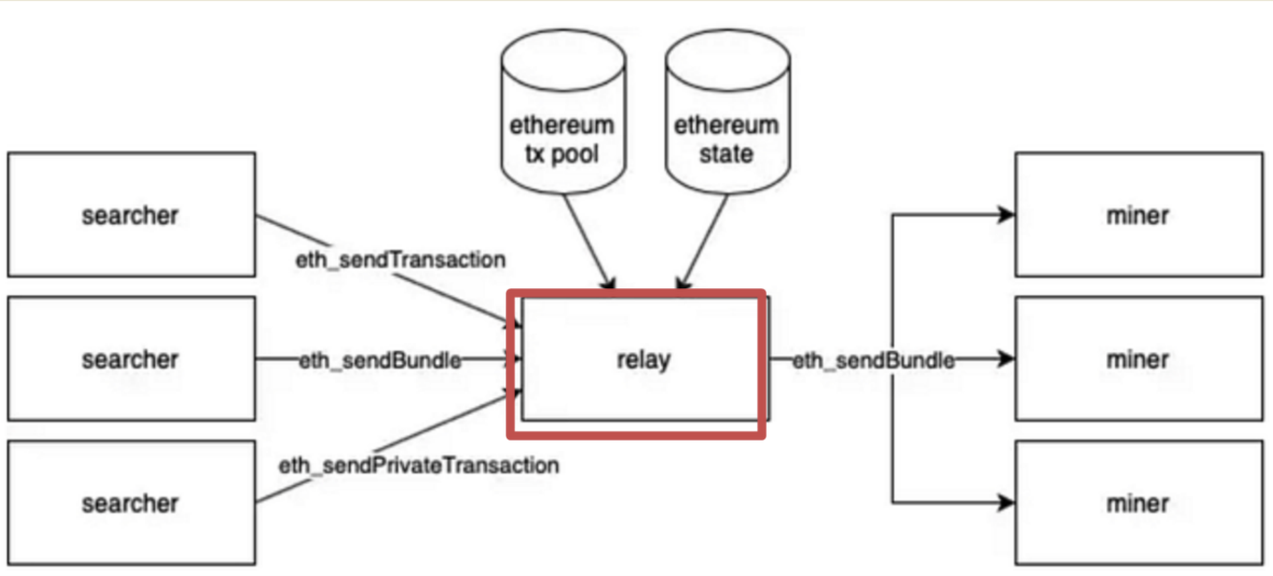


Searcher

- MEV를 피하고 싶은 일반 사용자 + MEV 추출 봇
- 트랜젝션 번들을 flashbot의 private mempool로 bid와 함께 전송
- private mempool은 MEV-geth의 relay, miner 외에는 내용과 bid를 확인할 수 없음
- bid가 비공개이므로 가스비 경쟁 완화
  → 네트워크 혼잡도 감소 효과

MEV-GETH

Architecture

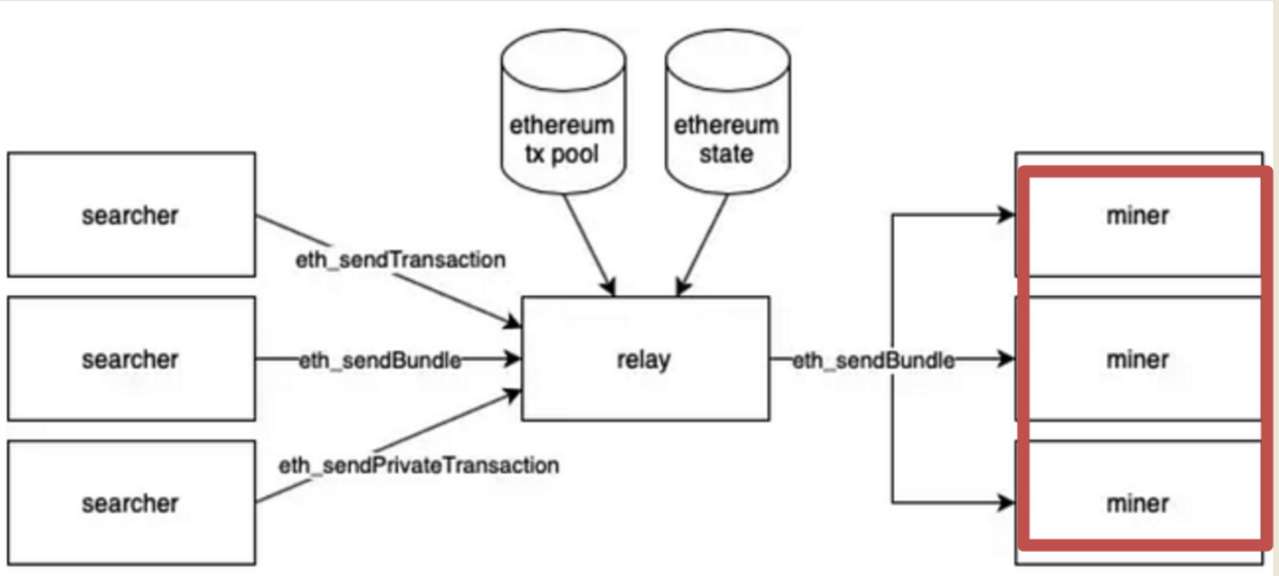


Relay

- private transaction pool의 transaction 및 transaction bundle을 검증해 miner에게 전달

MEV-GETH

Architecture



Miner

- Seald bid blockspace 경매
  relay로부터 전달받은 transaction 중 bid가 높은 transaction을 선택, 해당 transaction bundle을 블록에 포함하여 블록 생성

MEV-BOOST

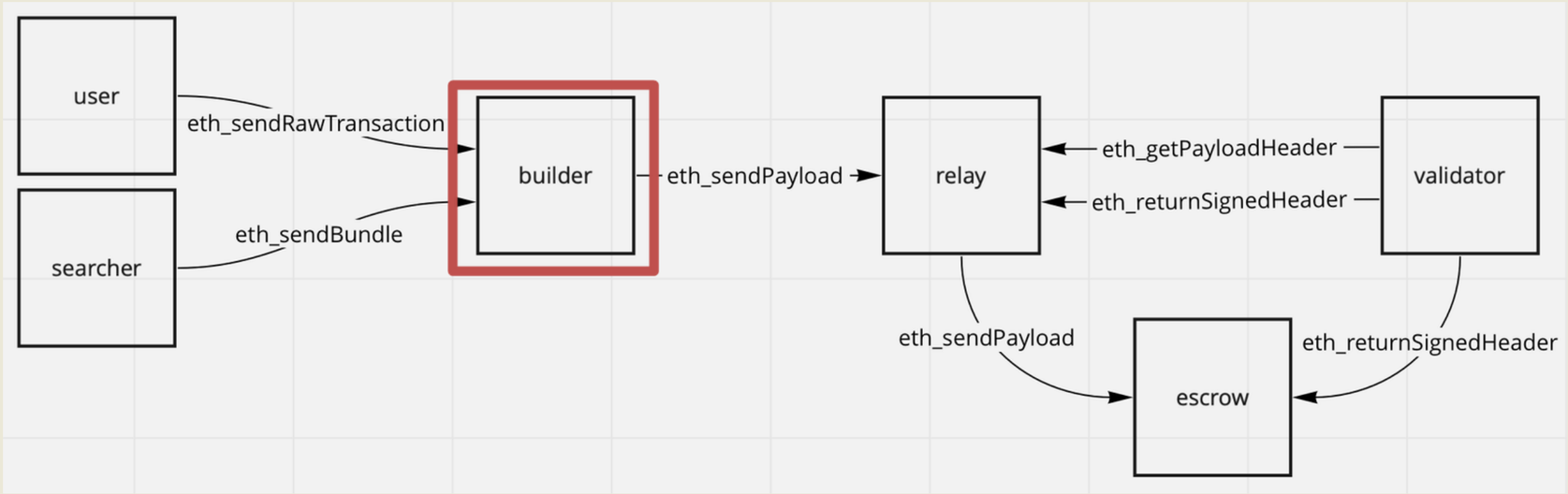
Idea

- Miner가 Searcher의 MEV를 훔칠 수 있는 문제
  → PoW에서는 Block proposer의 Whitelisting이 가능했으나, PoS에서는 불가능

- Builder라는 역할군의 추가 + Signing system 도입

MEV-BOOST

Architecture

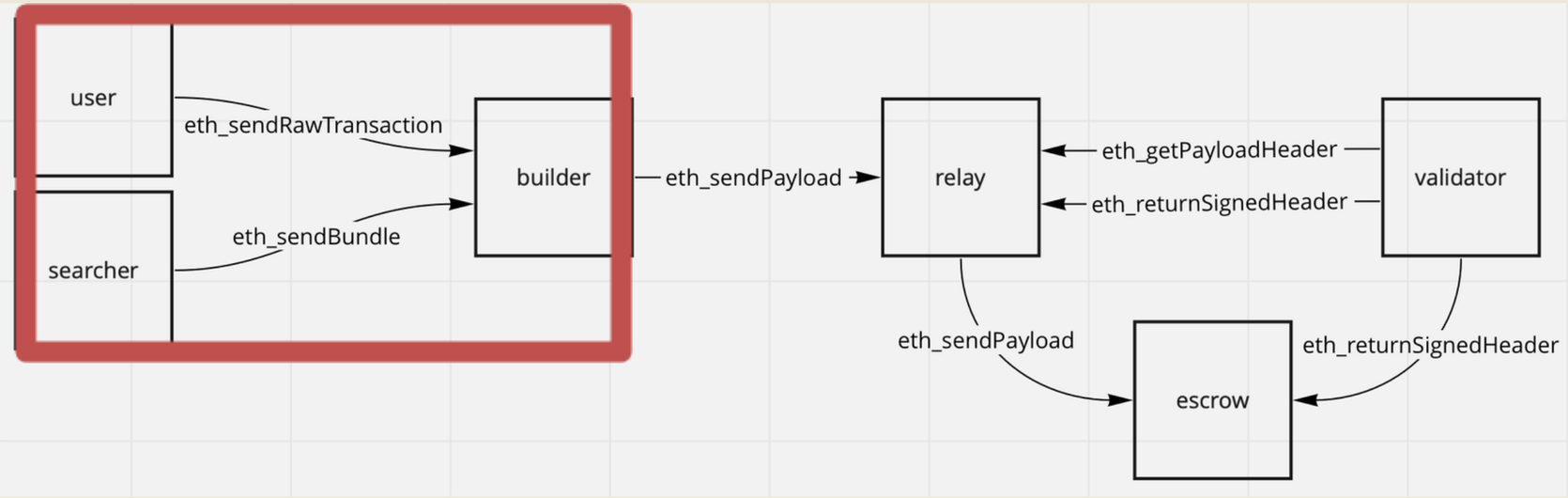


Builder

- Searcher에게 transaction bundle을 받음
- bundle + Exclusive oreder flow(EOF) → 완전한 블록 생성

MEV-BOOST

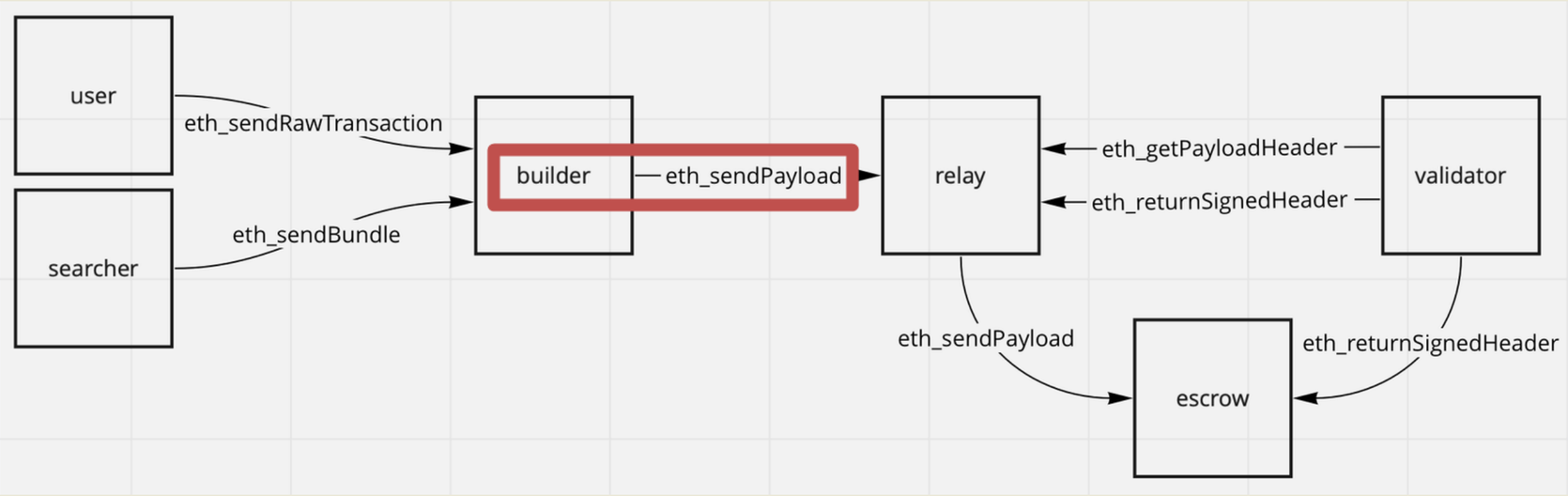
Transaction Flow



1. Builder : Searcher + EOF로부터 받은 tx로 완전한 블록을 생성

MEV-BOOST

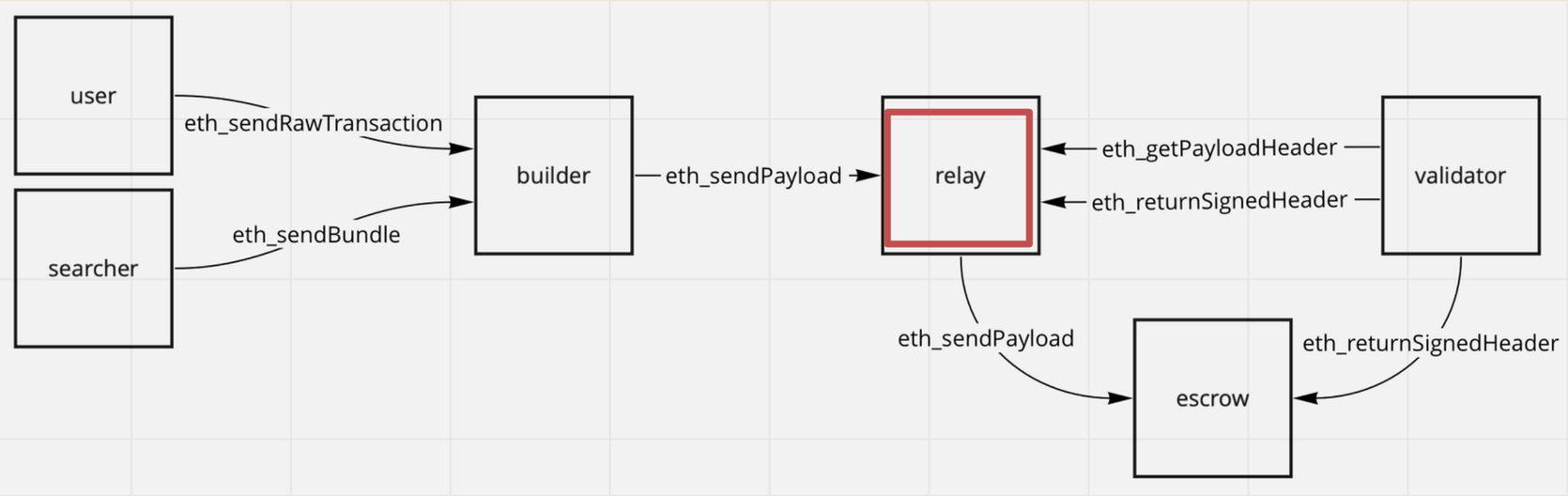
Transaction Flow



2. Builder → Relay : 블록에 bid를 붙여 전송

MEV-BOOST

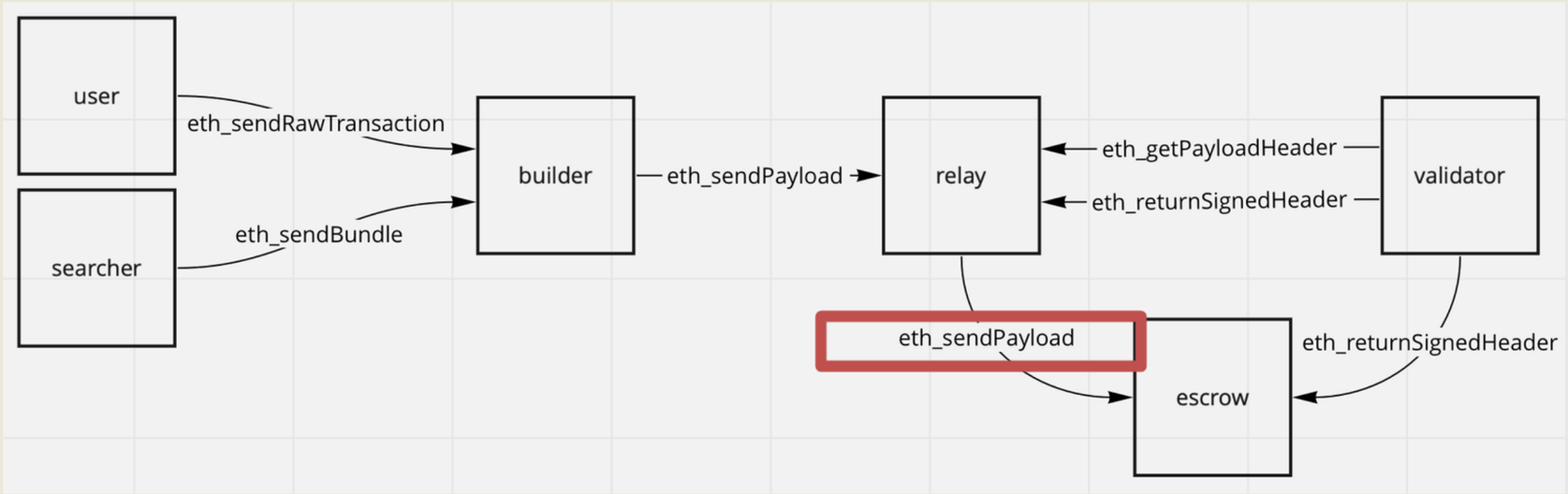
Transaction Flow



3. Relay : 블록의 타당성 검증, 블록 저장

MEV-BOOST

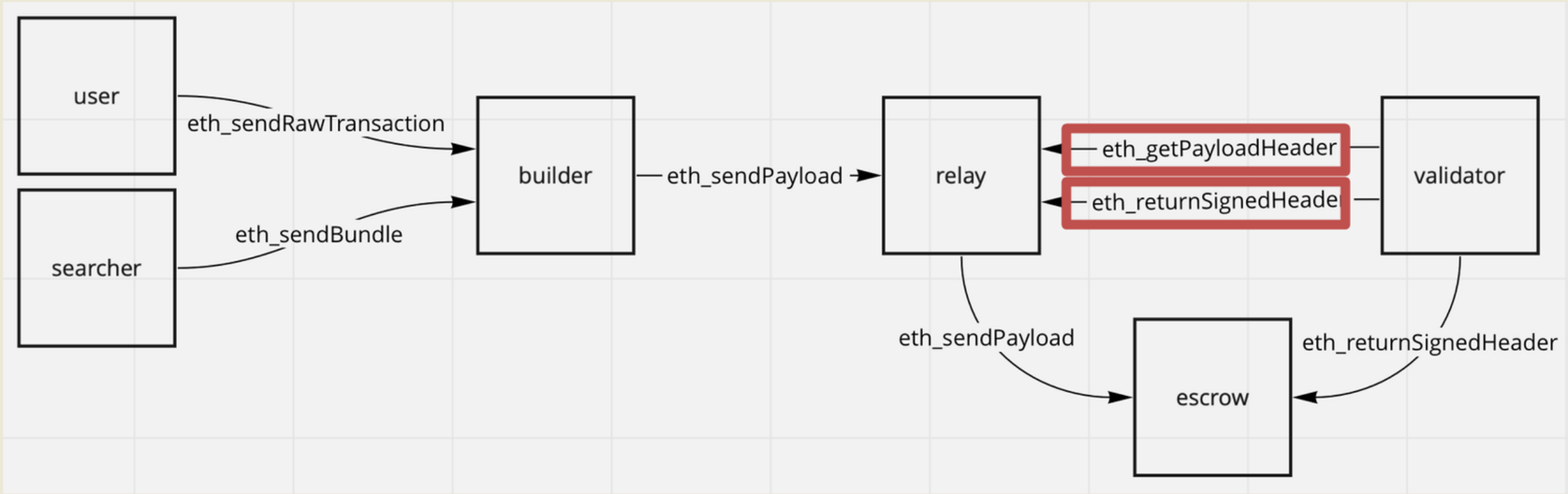
Transaction Flow



4. Relay → Proposer : 블록의 헤더와 bid만 전송

MEV-BOOST

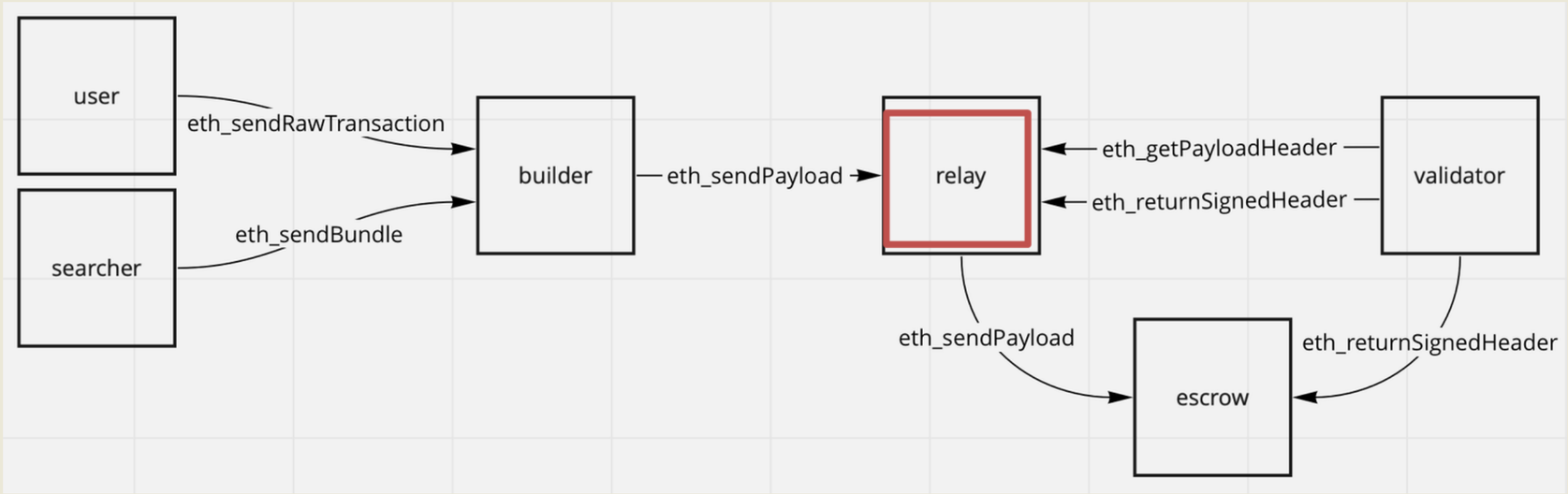
Transaction Flow



5. Proposer → Relay : bid가 높은 헤더에 서명하여 전송

MEV-BOOST

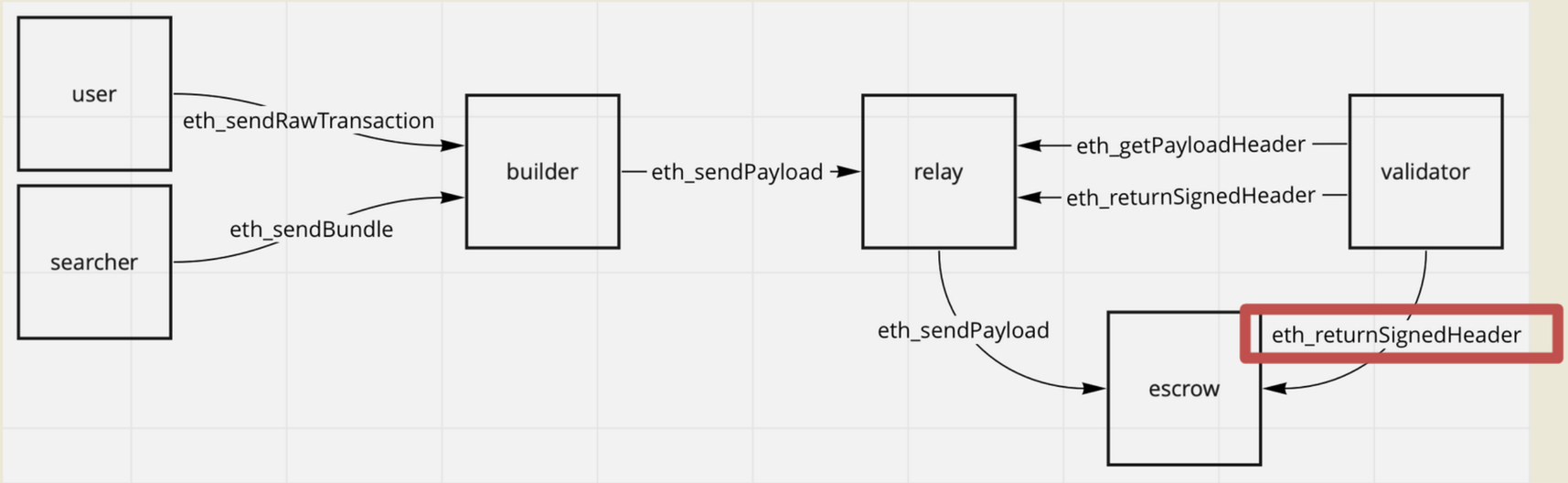
Transaction Flow



6. Relay → Proposer : 헤더에 해당하는 블록 내용 공개

MEV-BOOST

Transaction Flow



7. Proposer : 블록 제출, bid 수령

서명한 Block이 아닌 다른 Block을 제출할 경우, 더블사이닝으로 인한 슬래싱을 당하게 됨

MEV-BOOST

장점

- MEV 수익의 탈중앙화

단점

- Relay의 중앙화
- Relay의 안정성과 신뢰성이 Builder, proposer의 수익에 큰 영향을 미치기 때문
- Builder, Relay의 검열 문제
- Proposer가 블록 구성에 관여할 여지가 없기 때문

MEV-BOOST의 한계

Relay의 중앙화, 검열 문제

- 원인
  - 릴레이의 안정성 및 악의적 행동 가능성에 따라 빌더와 proposer가 손해를 입을 수 있고,
  - 빌더 혹은 릴레이가 의도적으로 tx를 검열해도 프로포저가 막을 방법이 없기 때문

- 솔루션
  - crList + PBS 도입으로 완화될 것으로 보임
    - PBS : Relay의 역할*을 프로토콜이 대체
      - 빌더와 프로포저간의 약속*을 이행하는 것
        - 프로포저가 bid를 수령하기 위해 블록을 제출하겠다는 약속
        - 빌더가 bid를 프로포저에게 지불하겠다는 약속
    - crList : 블록에 포함될 일부 transaction을 proposer가 지정

MEV-BOOST의 한계

Builder의 중앙화 문제

- 원인
  - Exclusive orderflow
    - 다양한 기능들을 제공하는 빠르고 강력한 빌더에게 사용자가 유입될 수 있음
    - 빌더가 orderflow를 돈주고 살 수 있음
  - cross-domain MEV 가능성
    - cross-domain MEV를 하는 것이 빌더에게 경제적으로 이득
      → 두 체인의 시퀀서가 공모하는 경우 cross-domain MEV가 더 쉬워짐
      → flashbot 맥락에서는 몇몇 빌더가 서로 공모하면서 중앙화될 가능성 존재

- 솔루션
  - SUAVE, Anoma Project 등
  - → 로드맵 달성까지 시간이 오래 걸릴 것으로 예상 (4~5년)

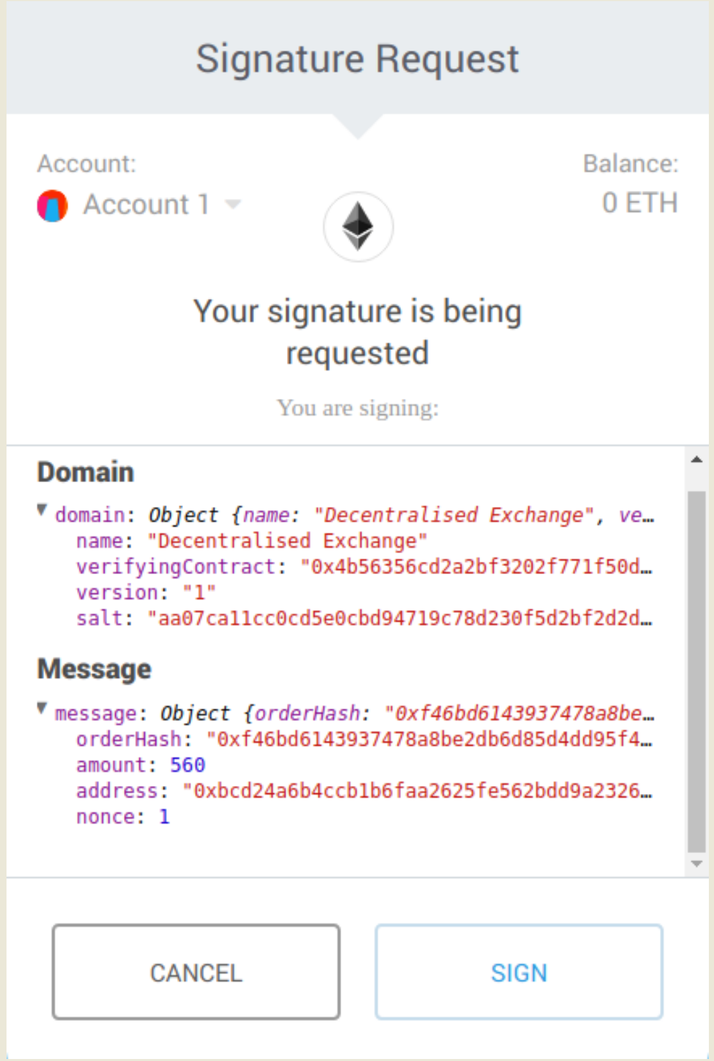
DeFi MEV Solution

- 1inch

1INCH LIMIT ORDER PROTOCOL

Idea

- data structure created off-chain and signed according to EIP-712
  - EIP-712: shows pre-hashed raw data
  - 기존 EIP-712 없이는 message 자체가 아래의 Object 해쉬이기에, 이것만 보고 서명했어야 했음



1INCH LIMIT ORDER PROTOCOL

Limit order : 고정된 가격으로 토큰 판매

1.유저는 1inch App을 통해 limit order를 주문할 수 잇음

- maker token의 컨트랙트 상에서 approve() 로 한도 지출을 승인하고, dAPP 인터페이스 혹은 TypeScript로 주문 생성 가능 (making 매도 / taking 매수)
- 아래 정보는 off-chain 상에 저장됨

```
const limitOrder = limitOrderBuilder.buildLimitOrder({
    makerAssetAddress: '0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2',
    takerAssetAddress: '0x111111111117dc0aa78b770fa6a738034120c302',
    makerAddress: '0xfb3c7ebccccAA12B5A884d612393969Adddddddd',
    makingAmount: '100',
    takingAmount: '200',
    //predicate = '0x',
    //permit = '0x',
    //receiver = ZERO_ADDRESS,
    //allowedSender = ZERO_ADDRESS,
    //getMakingAmount = ZERO_ADDRESS,
    //getTakingAmount = ZERO_ADDRESS,
    //preInteraction  = '0x',
    //postInteraction = '0x',
});

// 위와 같이 주문을 생성시에
// WETH 100개를 1INCH 200개에 매도
```

1INCH LIMIT ORDER PROTOCOL

Limit order : 고정된 가격으로 토큰 판매

2. API를 통해 Limit order 주문을 확인할 수 있고, on-chain 으로 거래 요청
    - 아래와 같이 Order 를 설정하여 컨트랙트 상의 fillOrder() 호출

```
struct Order {
        uint256 salt;
        Address maker;
        Address receiver;
        Address makerAsset;
        Address takerAsset;
        uint256 makingAmount;
        uint256 takingAmount;
        MakerTraits makerTraits;
    }

function fillOrder(
        Order calldata order,
        bytes32 r, // 서명 정보
        bytes32 vs, // 서명 정보
        uint256 amount,
        TakerTraits takerTraits
    ) external payable returns(uint256 makingAmount, uint256 takingAmount, bytes32 orderHash);
```

1INCH LIMIT ORDER PROTOCOL

Limit order : 고정된 가격으로 토큰 판매

3. 지정가 주문을 취소하려면 컨트랙트 상의 cancelOrder() 호출
    - 아래와 같이 주문 취소 → 지정가 주문 filled 100%로 가는 것으로
      사실상 자기가 자신의 자산을 사는 것으로 처리중 (따라서 취소시에는 가스비 본인 부담)

```
function cancelOrder(OrderLib.Order calldata order) external returns(uint256 orderRemaining, bytes32 orderHash) {
        if (order.maker != msg.sender) revert AccessDenied();

        orderHash = hashOrder(order);
        orderRemaining = _remaining[orderHash];
        if (orderRemaining == _ORDER_FILLED) revert AlreadyFilled();
        emit OrderCanceled(msg.sender, orderHash, orderRemaining);
        _remaining[orderHash] = _ORDER_FILLED;
    }
```

1INCH LIMIT ORDER PROTOCOL

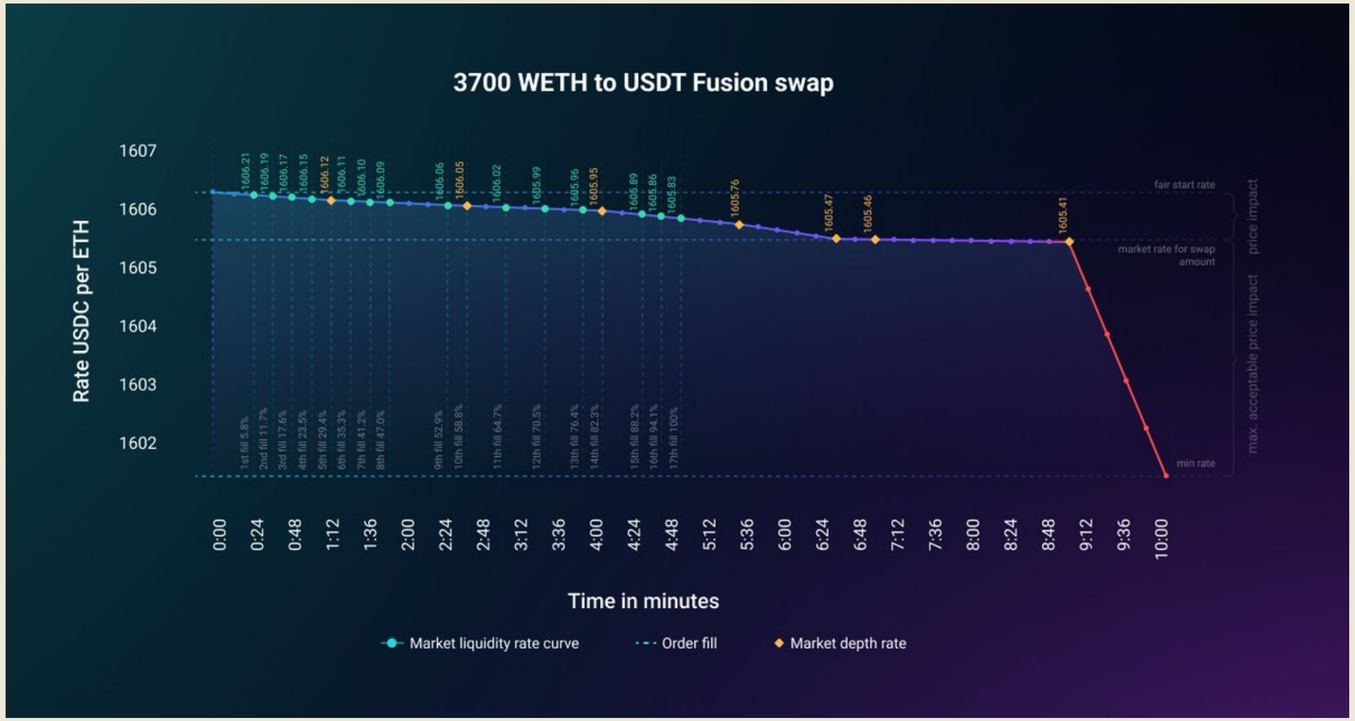
RFQ order

- 구매를 위해 마켓메이커로부터 quote을 요청
- 방식은 Limit order와 동일

1INCH FUSION

Idea

- Limit Order + Dutch Auction 도입
- 1INCH 보유 상위 인원을 Resolver로서 1inch 내 MM 역할



1INCH FUSION

Fusion Swap

- Limit Order와 동일 flow, 즉 유저가 가격 범위를 정해서 스왑 요청시 off-chain 상에 거래 정보 저장
- 이를 경매에 부치고, 낙찰받은 이가 토큰을 가져와 P2P 형식으로 스왑을 성사시키는 구조로 MEV 부담을 **전문적인 Resolver**들이 지게 함

1INCH FUSION

장점

- 인터페이스 단에서 따로 설정없이 MEV 부담 낮춤

단점

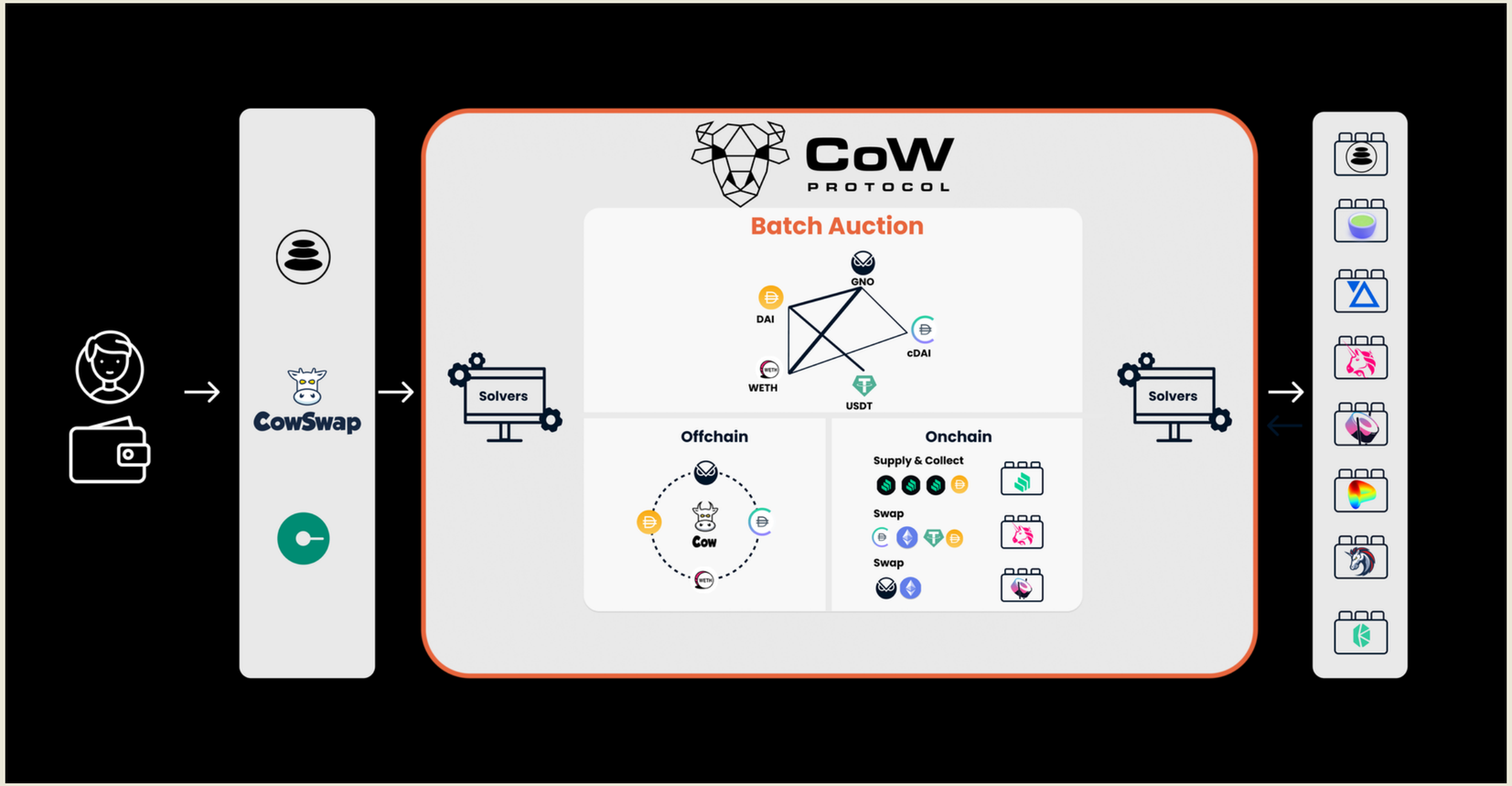
- 1inch 내에 있는 resolver들에게만 의존

DeFi MEV Solution

- CoW Swap

BATCH AUCTION

Architecture



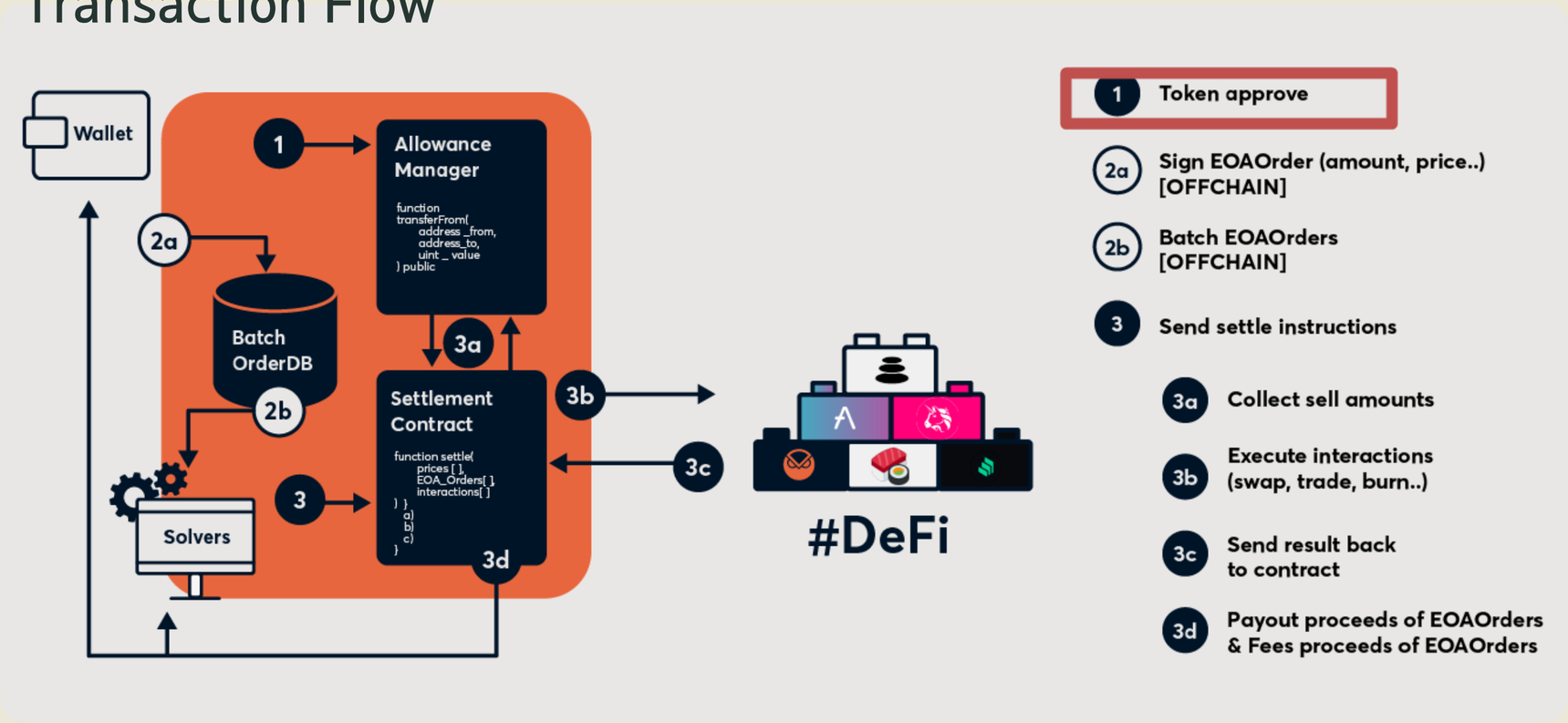
BATCH AUCTION

Idea

- combine off-chain interanctions with on-chain interactions, in the same transaction (CoWs + 다른 AMM)
- CoW Swap의 한 블록 내의 swap은 동일한 가격을 가짐 (batch transaction)
- 이를 묶는 주체로, Solver 도입
  - 가장 이상적으로 묶은 Solver에게 Reward 제공 → 경쟁 , batch ‘Auction’

BATCH AUCTION

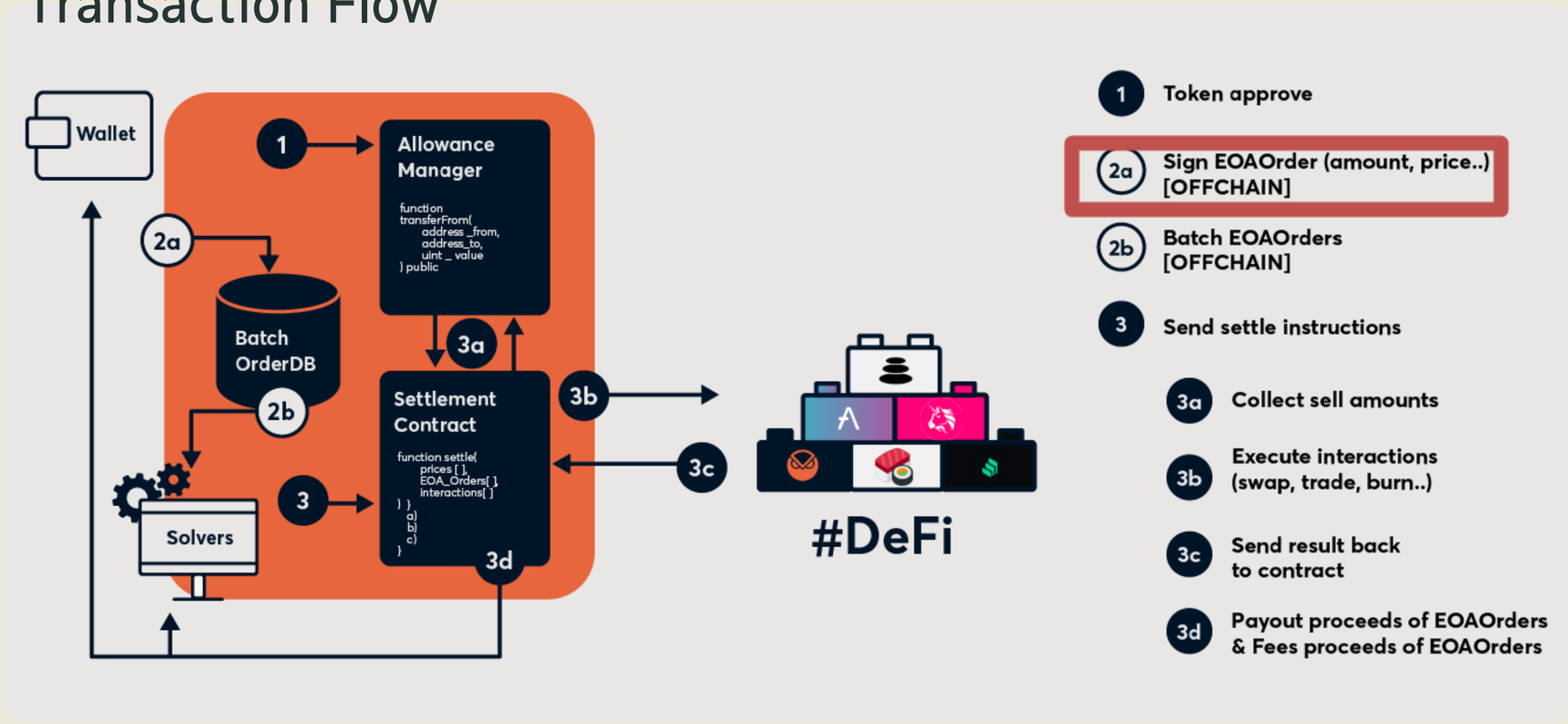
Transaction Flow



1. CoW Swap 컨트랙트(GPv2VaultRelayer)에 approve() 를 통해 지출한도 승인

BATCH AUCTION

Transaction Flow



2. 유저가 스왑을 요청 → signed order를 보냄 (off-chain)

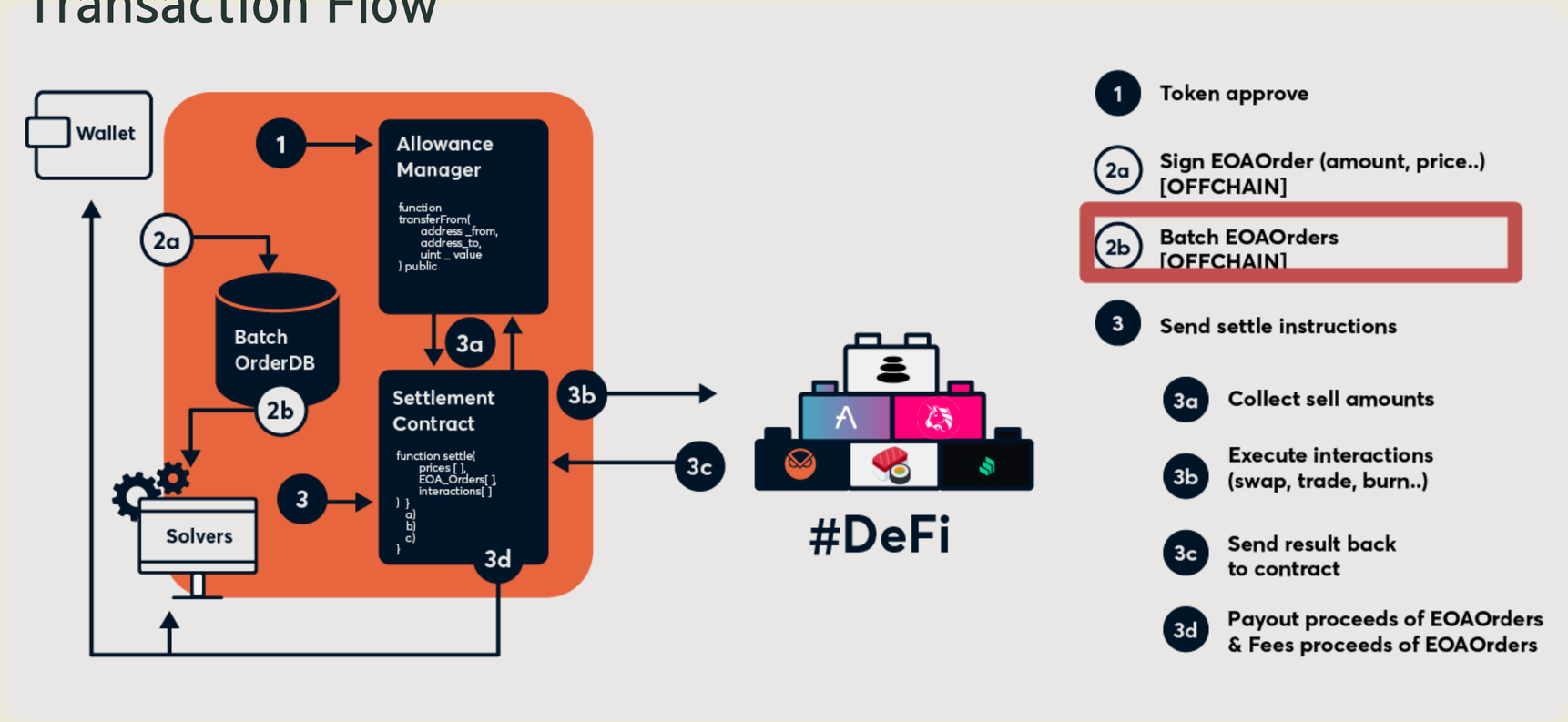
BATCH AUCTION

- signed order에 포함된 정보

```
struct Data {
        IERC20 sellToken;
        IERC20 buyToken;
        address receiver;
        uint256 sellAmount;
        uint256 buyAmount;
        uint32 validTo;
        bytes32 appData;
        uint256 feeAmount;
        bytes32 kind;
        bool partiallyFillable;
        bytes32 sellTokenBalance;
        bytes32 buyTokenBalance;
    }
```

BATCH AUCTION

Transaction Flow



3. 이후 작성된 batch transaction에 따라 (on-chain)

유저: 매도 토큰 → GPV2Settlement Contract → 1. CoWs 2. 타 AMM → 매수 토큰 : 유저

BATCH AUCTION

장점

- 동일 블록 동일 가격으로 프론트러닝 방어

단점

- Solver 신뢰 문제, 부족한 CoW Swap 유동성에서 더 괜찮은 가격이 나올 수 있을지 미지수

마무리

아쉬운 점을 위주로

결론

- 시중에 나와있는 MEV 방지책을 조사함

- 프로덕트를 구상까지는 발전시키지 못함

REFERENCE

FlashBots

- https://xangle.io/insight/research/63f5c63cb5a7786c6f6fc021

1inch Rabbithole

- https://blog.1inch.io/the-1inch-rabbithole-protection-from-sandwich-attacks/
- https://docs.1inch.io/docs/rabbithole/

REFERENCE

1inch Limit Order Protocol

- https://xangle.io/insight/research/63f5c63cb5a7786c6f6fc021
- https://xangle.io/insight/research/63f5c63cb5a7786c6f6fc021
- https://xangle.io/insight/research/63f5c63cb5a7786c6f6fc021
- https://xangle.io/insight/research/63f5c63cb5a7786c6f6fc021

1inch Fusion

- https://docs.1inch.io/docs/fusion-swap
- https://docs.1inch.io/docs/educational-resources/intermediate/fusion-swap-faq

REFERENCE

CoW Swap

- https://swap.cow.fi/#/faq
- https://docs.cow.fi/
- https://github.com/cowprotocol/contracts