Create2 Patterns

Frederico Lacs
Offchain Labs

How contract addresses are set

Characteristics we want to enforce:

- Unique
- Deterministic

How contract addresses are set

- EOA
  - Hash sender address and sender nonce
- Smart contract
  - Create
    - Same as EOA
  - Create2 (EIP-1014)
    - Salt instead of nonce, and init code hash
  - ~~Create3? (EIP-3171)~~
    - ~~Easier to handle immutable and constructor params~~
    - ~~Solidity implementation using self-destruct~~

How contract addresses are set

Contract calling `create`

```
pragma solidity ^0.8.0;

import "./Child.sol";

contract Factory {
    function deployChild() external returns (Child child) {
        child = new Child();
    }
}
```

How contract addresses are set

Contract calling `create`

- keccak256(rlp([sender, nonce]))[12:]

Each tx has unique nonce, so each contract has unique address

How contract addresses are set

Contract calling `create2`

```
pragma solidity ^0.8.0;

import "./Child.sol";

contract Factory {
    function deployChild(bytes32 userSalt) external returns (Child child) {
        bytes32 salt = keccak256(abi.encode(msg.sender, userSalt));
        child = new Child{ salt: salt }();
    }
}
```

How contract addresses are set

Contract calling `create2`

- keccak256(0xff ++ deployer ++ salt ++ keccak256(init_code))[12:]

If same salt is used again, the deployment fails

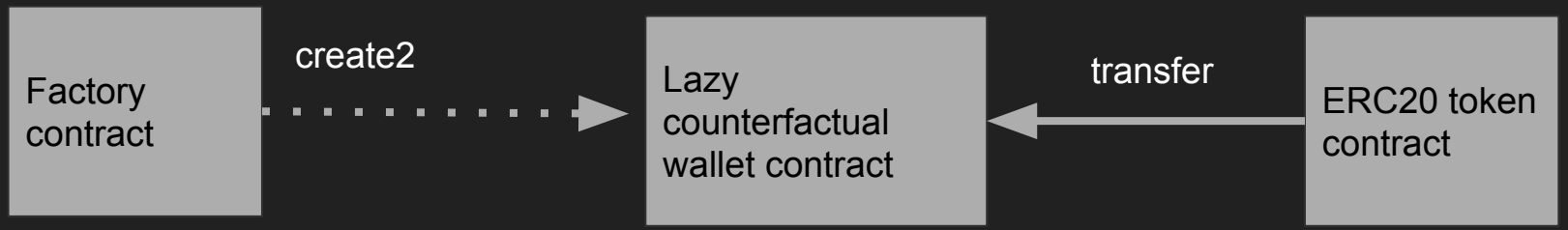
~~Can resurrect selfdestructed contracts~~

So what can I do with create2?

- You know which address a contract will be deployed to ahead of time
  - You also do with `create` but the smart contract has less control
- You can encode data into the address

Lazy Counterfactual Wallet

- Deploy contract to claim monies after monies is sent to address
- ~~Self-destruct to never have code on-chain~~



Address oracle

- Uniswap factory
  - Factory deploys pools with salt (tokenA, tokenB)
- Arbitrum token bridge
  - L2 token factory uses L1 token address as salt

Ownable proxy

Encode the owner to proxy address instead of saving in storage

Create2 encoding execution market

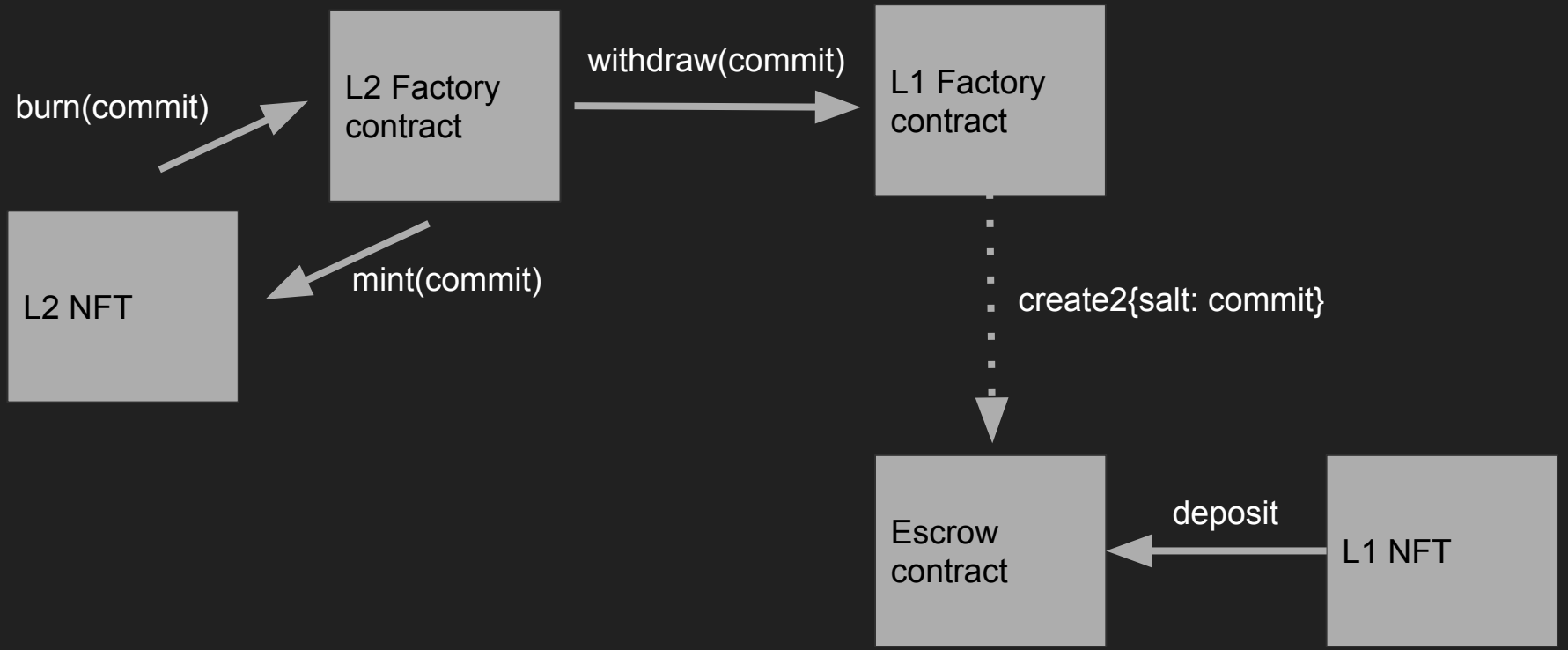
```
function transferAndCall(
    address to,
    uint256 value,
    bytes memory data
) external returns (bool success);
```

```
function withdraw(
  address asset,
  uint256 amount,
  address to
) external returns (uint256);
```

Create2 encoding execution market

- Hard to generalise discoverability
- Works for narrow use-cases

Create2 Counterfactual NFT bridge



```
library Lib {
    function getCommitHash(Commit721 calldata commit) internal pure returns (bytes32) {
        return keccak256(abi.encodePacked(
            commit.token, commit.tokenId, commit.minterUser, commit.fromChainId, commit.toChainId, commit.nonce
        ));
    }
}
```

Interesting links to read further

https://github.com/fredlacs/c2c-nft-bridge

https://github.com/frangio/cacheable-beacon-proxy

https://twitter.com/alpeh_v/status/1489317017924554752

https://github.com/pine-finance/contracts-v2

https://github.com/0xsequence/create3

This presentation
https://docs.google.com/presentation/d/1ClPlxSN_LbHMYvMi173dBMX5BhklDKp9difxDzD3QxE/edit?usp=sharing

We’re hiring

https://jobs.lever.co/offchainlabs

Questions?

DMs open

https://twitter.com/0x66726564