1. **CCIPReader**.sol
2. Forward Resolution
   UR.sol
3. Reverse Resolution
   ReverseUR.sol
4. Wrapped Resolution
   HumanUR.sol

CCIPReader.sol

`https://github.com/unruggable-labs/CCIPReader.sol`

When a CCIP-Read function is called in the EVM, the call flow **bifurcates,** either:

(1) it returns immediately
or (2) continues after an `OffchainLookup` revert is processed.

CCIPReader.sol

https://github.com/unruggable-labs/CCIPReader.sol

```
function ccipRead(
    address target,
    bytes memory call,
    bytes4 mySelector,
    bytes memory myCarry
) internal view returns (bytes memory);
```

Similar to `staticcall()`, except:

```
contract Wrapper is CCIPReader {
```

```solidity
import {CCIPReader} from "../contracts/CCIPReader.sol";

contract Wrapper is CCIPReader {
    function wrap(
        address target,
        bytes memory data,
        bytes memory carry
    ) external view returns (bytes memory, bytes memory) {
        bytes memory v = ccipRead(
            target,
            data,
            this.wrapCallback.selector,
            carry
        );
        assembly {
            return(add(v, 32), mload(v))
        }
    }

    function wrapCallback(
        bytes memory ccip,
        bytes memory carry
    ) external pure returns (bytes memory, bytes memory) {
        return (ccip, carry);
    }
}
```

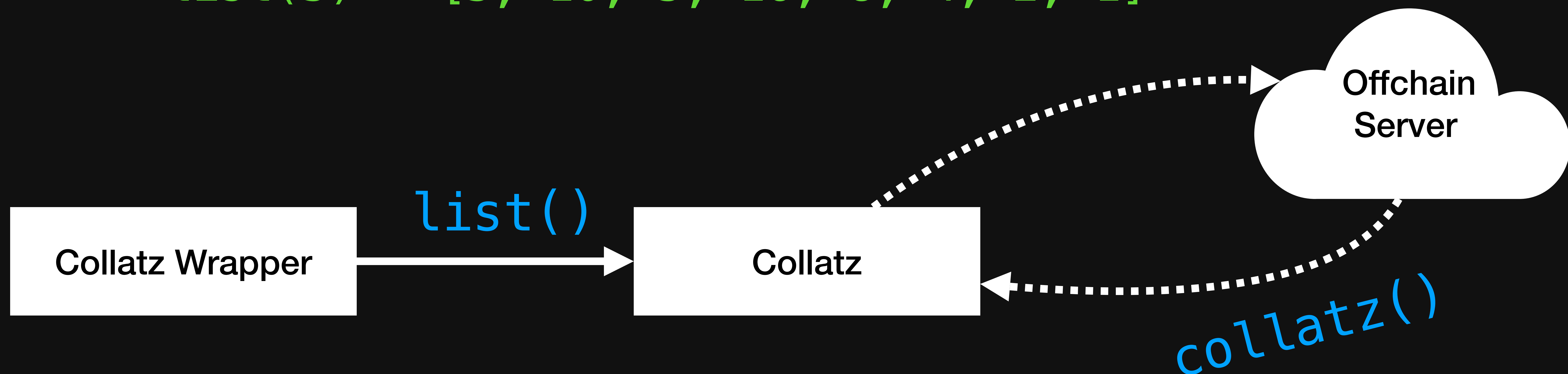
Demo: Collatz Sequence

https://github.com/unruggable-labs/CCIPReader.sol

$$f(n) = \begin{cases} n/2 & \text{if } n \equiv 0 \pmod{2}, \\ 3n+1 & \text{if } n \equiv 1 \pmod{2}. \end{cases}$$

```
function list(uint256 x) external view returns (uint256[] memory) {
```

```
list(3) = [3, 10, 5, 16, 8, 4, 2, 1]
```



UR.sol

`https://github.com/unruggable-labs/unruggable-resolve`

1. Rewrite `UniversalResolver` using **CCIPReader**.sol
2. Support `resolve(name, multicall(…))`
3. Standardize an Interface
4. Imagine as a "**lego**" that encapsulates **forward-resolution on-chain**

UR.sol

https://github.com/unruggable-labs/unruggable-resolve

```
struct Lookup {
    bytes dns; // dns-encoded name (safe to decode)
    uint256 offset; // byte offset into dns for basename
    bytes32 node; // namehash(dns)
    bytes32 basenode; // namehash(dns.slice(offset))
    address resolver; // resolver(basenode), null if invalid
    bool extended; // IExtendedResolver
}
```

```
struct Response {
    uint256 bits; // ResponseBits
    bytes call; // record calldata
    bytes data; // answer (or error)
}

library ResponseBits {
    uint256 constant ERROR = 1 << 0; // resolution failed
    uint256 constant OFFCHAIN = 1 << 1; // reverted OffchainLookup
    uint256 constant BATCHED = 1 << 2; // used Batched Gateway
    uint256 constant RESOLVED = 1 << 3; // resolution finished (internal flag)
}
```

```
interface IUR {
    function registry() external view returns (address);
    function lookupName(bytes memory dns) external view returns (Lookup memory lookup);
    function resolve(bytes memory dns, bytes[] memory calls, string[] memory batchedGateways)
        external
        view
        returns (Lookup memory lookup, Response[] memory res);
```

UR.sol

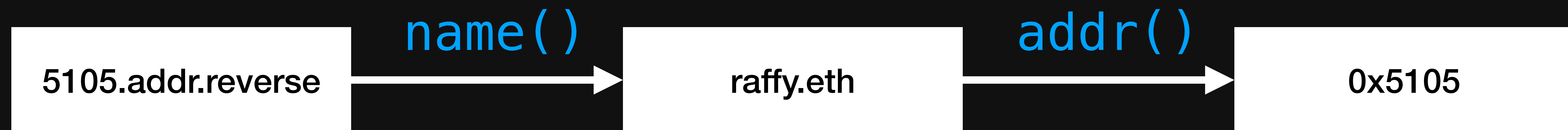
https://github.com/unruggable-labs/unruggable-resolve

1. Less than 200 lines of code
2. Super-easy to understand
3. Supports “intelligent” multicall
4. Designed to be wrapped

ReverseUR.sol

https://github.com/unruggable-labs/unruggable-resolve

1. Implement Reverse Resolution using UR



2. Standardize an Interface

3. Imagine as a "**lego**" that encapsulates **reverse-resolution on-chain**

ReverseUR.sol

https://github.com/unruggable-labs/unruggable-resolve

```
interface IReverseUR {
    function ur() external view returns (IUR);
    function reverseName(bytes memory addr, uint256 coinType) external pure returns (string memory);
    function reverse(bytes memory addr, uint256 coinType, string[] memory batchedGateways)
        external
        view
        returns (Lookup memory rev, Lookup memory fwd, bytes memory answer);
}
```

100 lines of code, very simple, uses multicall

HumanUR.sol (WIP)

https://github.com/unruggable-labs/unruggable-resolve

1. Normalization / DNS-Encoding
2. Address Coders
3. Easy to Call

```
function resolve(
    string memory name,
    string[] memory keys,
    uint256[] memory coins,
    bool useContenthash,
    string[] memory gateways
)
    external
    view
    returns (Lookup memory lookup, string[] memory texts, bytes[] memory addrs, bytes memory contenthash)
{
```

WrappedUR.sol

https://github.com/unruggable-labs/unruggable-resolve

```
function resolve(bytes memory dns, bytes[] memory calls, string[] memory gateways)
    external
    view
    returns (Lookup memory lookup, Response[] memory res)
{
    lookup = ur.lookupName(dns);
    if (lookup.resolver == address(0)) return (lookup, res);
    //
    // insert resolver based logic here
    //
    bytes memory v = ccipRead(
        address(ur), abi.encodeCall(IUR.resolve, (dns, calls, gateways)), this.resolveCallback.selector, ""
    );
    assembly {
        return(add(v, 32), mload(v))
    }
}
```

UniversalResolver.sol

`https://github.com/unruggable-labs/unruggable-resolve`

1. Reimplement ENS UniversalResolver
2. Reimplement ENS UniversalResolver (v3)
3. ABI-equivalent
4. ~150 lines of code