1. OxPool.sol

```
// SPDX-License-Identifier: MIT pragma solidity 0.8.11; import "@openzeppelin/contracts/to-
ken/ERC20/ERC20.sol"; import "./interfaces/ISolidlyLens.sol"; import "./libraries/Math.sol"; im-
port "./interfaces/IVoterProxy.sol"; import "./interfaces/IBribe.sol"; import "./interfaces/ITokensAl-
lowlist.sol"; import "./interfaces/IUserProxy.sol"; import "./interfaces/IOxPoolFactory.sol";

/**
 * @title OxPool
 * @author 0xDAO
 * @dev For every Solidly pool there is a corresponding oxPool
 * @dev oxPools represent a 1:1 ERC20 wrapper of a Solidly LP token
 * @dev For every oxPool there is a corresponding Synthetix MultiRewards contract
 * @dev oxPool LP tokens can be staked into the Synthetix MultiRewards contracts to allow LPs to
earn fees
 */ contract OxPool is ERC20 {
   /*******************************************************
    *                    Configuration
    *******************************************************/

   // Public addresses
   address public oxPoolFactoryAddress;
   address public solidPoolAddress;
   address public stakingAddress;
   address public gaugeAddress;
   address public oxPoolAddress;
   address public bribeAddress;
   address public tokensAllowlistAddress;

   // Token name and symbol
   string internal tokenName;
   string internal tokenSymbol;

   // Reward tokens allowlist sync mechanism variables
   uint256 public allowedTokensLength;
   mapping(uint256 => address) public rewardTokenByIndex;
   mapping(address => uint256) public indexByRewardToken;
   uint256 public bribeSyncIndex;
   uint256 public bribeNotifySyncIndex;
   uint256 public bribeOrFeesIndex;
   uint256 public nextClaimSolidTimestamp;
   uint256 public nextClaimFeeTimestamp;
   mapping(address => uint256) public nextClaimBribeTimestamp;

   // Internal helpers
   IOxPoolFactory internal _oxPoolFactory;
   ITokensAllowlist internal _tokensAllowlist;
   IBribe internal _bribe;
   IVoterProxy internal _voterProxy;
   ISolidlyLens internal _solidlyLens;
   ISolidlyLens.Pool internal _solidPoolInfo;

   /*******************************************************
    *                  oxPool Implementation
```

```
     ******************************************************/

    /**
     * @notice Return information about the Solid pool associated with this oxPool
     */
    function solidPoolInfo() external view returns (ISolidlyLens.Pool memory) {
        return _solidPoolInfo;
    }

    /**
     * @notice Initialize oxPool
     * @dev This is called by oxPoolFactory upon creation
     * @dev We need to initialize rather than create using constructor since oxPools are deployed using
EIP-1167
     */
    function initialize(
        address _oxPoolFactoryAddress,
        address _solidPoolAddress,
        address _stakingAddress,
        string memory _tokenName,
        string memory _tokenSymbol,
        address _bribeAddress,
        address _tokensAllowlistAddress
    ) external {
        require(oxPoolFactoryAddress == address(0), "Already initialized");
        bribeAddress = _bribeAddress;
        _bribe = IBribe(bribeAddress);
        oxPoolFactoryAddress = _oxPoolFactoryAddress;
        solidPoolAddress = _solidPoolAddress;
        stakingAddress = _stakingAddress;
        tokenName = _tokenName;
        tokenSymbol = _tokenSymbol;
        _oxPoolFactory = IOxPoolFactory(oxPoolFactoryAddress);
        address solidlyLensAddress = _oxPoolFactory.solidlyLensAddress();
        _solidlyLens = ISolidlyLens(solidlyLensAddress);
        _solidPoolInfo = _solidlyLens.poolInfo(solidPoolAddress);
        gaugeAddress = _solidPoolInfo.gaugeAddress;
        oxPoolAddress = address(this);
        tokensAllowlistAddress = _tokensAllowlistAddress;
        _tokensAllowlist = ITokensAllowlist(tokensAllowlistAddress);
        _voterProxy = IVoterProxy(_oxPoolFactory.voterProxyAddress());
    }

    /**
     * @notice Set up ERC20 token
     */
    constructor(string memory _tokenName, string memory _tokenSymbol)
        ERC20(_tokenName, _tokenSymbol)
    {}

    /**
     * @notice ERC20 token name
     */
    function name() public view override returns (string memory) {
```

```
    return tokenName;
}

/**
 * @notice ERC20 token symbol
 */
function symbol() public view override returns (string memory) {
    return tokenSymbol;
}

/*******************************************************
 * Core deposit/withdraw logic (taken from ERC20Wrapper)
 *******************************************************/

/**
 * @notice Deposit Solidly LP and mint oxPool receipt token to msg.sender
 * @param amount The amount of Solidly LP to deposit
 */
function depositLp(uint256 amount) public syncOrClaim {
    // Transfer Solidly LP from sender to oxPool
    IERC20(solidPoolAddress).transferFrom(
        msg.sender,
        address(this),
        amount
    );

    // Mint oxPool receipt token
    _mint(oxPoolAddress, amount);

    // Transfer oxPool receipt token to msg.sender
    IERC20(oxPoolAddress).transfer(msg.sender, amount);

    // Transfer LP to voter proxy
    IERC20(solidPoolAddress).transfer(address(_voterProxy), amount);

    // Stake Solidly LP into Solidly gauge via voter proxy
    _voterProxy.depositInGauge(solidPoolAddress, amount);
}

/**
 * @notice Withdraw Solidly LP and burn msg.sender’s oxPool receipt token
 */
function withdrawLp(uint256 amount) public syncOrClaim {
    // Withdraw Solidly LP from gauge
    _voterProxy.withdrawFromGauge(solidPoolAddress, amount);

    // Burn oxPool receipt token
    _burn(_msgSender(), amount);

    // Transfer Solidly LP back to msg.sender
    IERC20(solidPoolAddress).transfer(msg.sender, amount);
}

/*******************************************************
```

```
     *              Reward tokens sync mechanism
     ******************************************************/

    /**
     * @notice Fetch current number of rewards for associated bribe
     * @return Returns number of bribe tokens
     */
    function bribeTokensLength() public view returns (uint256) {
        return IBribe(bribeAddress).rewardsListLength();
    }

    /**
     * @notice Check a given token against the global allowlist and update state in oxPool allowlist if
state has changed
     * @param bribeTokenAddress The address to check
     */
    function updateTokenAllowedState(address bribeTokenAddress) public {
        // Detect state changes
        uint256 currentRewardTokenIndex = indexByRewardToken[bribeTokenAddress];
        bool tokenWasPreviouslyAllowed = currentRewardTokenIndex > 0;
        bool tokenIsNowAllowed = _tokensAllowlist.tokenIsAllowed(
            bribeTokenAddress
        );
        bool allowedStateDidntChange = tokenWasPreviouslyAllowed ==
            tokenIsNowAllowed;

        // Allowed state didn’t change, don’t do anything
        if (allowedStateDidntChange) {
            return;
        }

        // Detect whether a token was added or removed
        bool tokenWasAdded = tokenWasPreviouslyAllowed == false &&
            tokenIsNowAllowed == true;
        bool tokenWasRemoved = tokenWasPreviouslyAllowed == true &&
            tokenIsNowAllowed == false;

        if (tokenWasAdded) {
            // Add bribe token
            allowedTokensLength++;
            indexByRewardToken[bribeTokenAddress] = allowedTokensLength;
            rewardTokenByIndex[allowedTokensLength] = bribeTokenAddress;
        } else if (tokenWasRemoved) {
            // Remove bribe token
            address lastBribeAddress = rewardTokenByIndex[allowedTokensLength];
            uint256 currentIndex = indexByRewardToken[bribeTokenAddress];
            indexByRewardToken[bribeTokenAddress] = 0;
            rewardTokenByIndex[currentIndex] = lastBribeAddress;
            allowedTokensLength--;
        }
    }

    /**
     * @notice Return a list of whitelisted tokens for this oxPool
     * @dev This list updates automatically (upon user interactions with oxPools)
     * @dev The allowlist is based on a global allowlist
     */
    function bribeTokensAddresses() public view returns (address[] memory) {
        address[] memory _bribeTokensAddresses = new address[](
            allowedTokensLength
        );
        for (
            uint256 bribeTokenIndex;
            bribeTokenIndex < allowedTokensLength;
            bribeTokenIndex++
        ) {
            _bribeTokensAddresses[bribeTokenIndex] = rewardTokenByIndex[
                bribeTokenIndex + 1
            ];
        }
        return _bribeTokensAddresses;
    }

    /**
     * @notice Sync bribe token allowlist
     * @dev Syncs "bribeTokensSyncPageSize" (governance configurable) number of tokens at a time
     * @dev Once all tokens have been synced the index is reset and token syncing begins again from
the start index
     */
    function syncBribeTokens() public {
        uint256 virtualSyncIndex = bribeSyncIndex;
        uint256 _bribeTokensLength = bribeTokensLength();
        uint256 _pageSize = _tokensAllowlist.bribeTokensSyncPageSize();
        uint256 syncSize = Math.min(_pageSize, _bribeTokensLength);
        bool stopLoop;
        for (
            uint256 syncIndex;
            syncIndex < syncSize && !stopLoop;
            syncIndex++
        ) {
            if (virtualSyncIndex >= _bribeTokensLength) {
                virtualSyncIndex = 0;

                //break loop when we reach the end so pools with a small number of bribes don’t loop over
and over in one tx
                stopLoop = true;
            }
            address bribeTokenAddress = _bribe.rewards(virtualSyncIndex);
            updateTokenAllowedState(bribeTokenAddress);
            virtualSyncIndex++;
        }
        bribeSyncIndex = virtualSyncIndex;
    }

    /**
     * @notice Notify rewards on allowed bribe tokens
     * @dev Notify reward for "bribeTokensNotifyPageSize" (governance configurable) number of to-
kens at a time
```

```
     * @dev Once all tokens have been notified the index is reset and token notifying begins again from
the start index
     */
    function notifyBribeOrFees() public {
        uint256 virtualSyncIndex = bribeOrFeesIndex;
        (uint256 bribeFrequency, uint256 feeFrequency) = _tokensAllowlist
            .notifyFrequency();
        if (virtualSyncIndex >= bribeFrequency + feeFrequency) {
            virtualSyncIndex = 0;
        }
        if (virtualSyncIndex < feeFrequency) {
            notifyFeeTokens();
        } else {
            notifyBribeTokens();
        }
        virtualSyncIndex++;
        bribeOrFeesIndex = virtualSyncIndex;
    }

    /**
     * @notice Notify rewards on allowed bribe tokens
     * @dev Notify reward for "bribeTokensNotifyPageSize" (governance configurable) number of to-
kens at a time
     * @dev Once all tokens have been notified the index is reset and token notifying begins again from
the start index
     */
    function notifyBribeTokens() public {
        uint256 virtualSyncIndex = bribeNotifySyncIndex;
        uint256 _pageSize = _tokensAllowlist.bribeTokensNotifyPageSize();
        uint256 syncSize = Math.min(_pageSize, allowedTokensLength);
        address[] memory notifyBribeTokenAddresses = new address[](syncSize);
        bool stopLoop;
        for (
            uint256 syncIndex;
            syncIndex < syncSize && !stopLoop;
            syncIndex++
        ) {
            if (virtualSyncIndex >= allowedTokensLength) {
                virtualSyncIndex = 0;

                //break loop when we reach the end so pools with a small number of bribes don't loop over
and over in one tx
                stopLoop = true;
            }
            address bribeTokenAddress = rewardTokenByIndex[
                virtualSyncIndex + 1
            ];
            if (block.timestamp > nextClaimBribeTimestamp[bribeTokenAddress]) {
                notifyBribeTokenAddresses[syncIndex] = bribeTokenAddress;
            }
            virtualSyncIndex++;
        }

        (, bool[] memory claimed) = _voterProxy.getRewardFromBribe(
            oxPoolAddress,
            notifyBribeTokenAddresses
        );

        //update next timestamp for claimed tokens
        for (uint256 i; i < claimed.length; i++) {
            if (claimed[i]) {
                nextClaimBribeTimestamp[notifyBribeTokenAddresses[i]] =
                    block.timestamp +
                    _tokensAllowlist.periodBetweenClaimBribe();
            }
        }
        bribeNotifySyncIndex = virtualSyncIndex;
    }

    /**
     * @notice Notify rewards on fee tokens
     */
    function notifyFeeTokens() public {
        //if fee claiming is disabled for this pool or it's not time to claim yet, return
        if (
            _tokensAllowlist.feeClaimingDisabled(oxPoolAddress) ||
            block.timestamp < nextClaimFeeTimestamp
        ) {
            return;
        }

        // if claimed, update next claim timestamp
        bool claimed = _voterProxy.getFeeTokensFromBribe(oxPoolAddress);
        if (claimed) {
            nextClaimFeeTimestamp =
                block.timestamp +
                _tokensAllowlist.periodBetweenClaimFee();
        }
    }

    /**
     * @notice Sync a specific number of bribe tokens
     * @param startIndex The index to start at
     * @param endIndex The index to end at
     * @dev If endIndex is greater than total number of reward tokens, use reward token length as end
index
     */
    function syncBribeTokens(uint256 startIndex, uint256 endIndex) public {
        uint256 _bribeTokensLength = bribeTokensLength();
        if (endIndex > _bribeTokensLength) {
            endIndex = _bribeTokensLength;
        }
        for (
            uint256 syncIndex = startIndex;
            syncIndex < endIndex;
            syncIndex++
        ) {
            address bribeTokenAddress = _bribe.rewards(syncIndex);
        updateTokenAllowedState(bribeTokenAddress);
    }
}

/**
 * @notice Batch update token allowed states given a list of tokens
 * @param bribeTokensAddresses A list of addresses to update
 */
function updateTokensAllowedStates(address[] memory bribeTokensAddresses)
    public
{
    for (
        uint256 bribeTokenIndex;
        bribeTokenIndex < bribeTokensAddresses.length;
        bribeTokenIndex++
    ) {
        address bribeTokenAddress = bribeTokensAddresses[bribeTokenIndex];
        updateTokenAllowedState(bribeTokenAddress);
    }
}

/*******************************************************
 *                Modifiers
 *******************************************************/
modifier syncOrClaim() {
    syncBribeTokens();
    notifyBribeOrFees();

    // if it's time to claim more solid from the gauge, do so
    if (block.timestamp > nextClaimSolidTimestamp) {
        bool claimed = _voterProxy.claimSolid(oxPoolAddress);
        if (claimed) {
            nextClaimSolidTimestamp =
                block.timestamp +
                _tokensAllowlist.periodBetweenClaimSolid();
        }
    }
    _;
} }
```