- 2 Removals + 2 Additions

```
// SPDX-License-Identifier: AGPL-3.0-or-later
pragma solidity 0.7.5;

interface IERC20 {
  /**
   * @dev Returns the amount of tokens in existence.
   */
  function totalSupply() external view returns (uint256);

  /**
   * @dev Returns the amount of tokens owned by `account`.
   */
  function balanceOf(address account) external view returns (uint256);

  /**
   * @dev Moves `amount` tokens from the caller's account to `recipient`.
   *
   * Returns a boolean value indicating whether the operation succeeded.
   *
   * Emits a {Transfer} event.
   */
  function transfer(address recipient, uint256 amount) external returns (bool);

  /**
   * @dev Returns the remaining number of tokens that `spender` will be
   * allowed to spend on behalf of `owner` through {transferFrom}. This is
   * zero by default.
   *
   * This value changes when {approve} or {transferFrom} are called.
   */
  function allowance(address owner, address spender) external view returns (uint256);

  /**
   * @dev Sets `amount` as the allowance of `spender` over the caller's tokens.
   *
   * Returns a boolean value indicating whether the operation succeeded.
   *
   * IMPORTANT: Beware that changing an allowance with this method brings the risk
   * that someone may use both the old and the new allowance by unfortunate
   * transaction ordering. One possible solution to mitigate this race
   * condition is to first reduce the spender's allowance to 0 and set the
   * desired value afterwards:
   * https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729
   *
   * Emits an {Approval} event.
```

```
// SPDX-License-Identifier: AGPL-3.0-or-later
pragma solidity 0.7.5;

interface IERC20 {
  /**
   * @dev Returns the amount of tokens in existence.
   */
  function totalSupply() external view returns (uint256);

  /**
   * @dev Returns the amount of tokens owned by `account`.
   */
  function balanceOf(address account) external view returns (uint256);

  /**
   * @dev Moves `amount` tokens from the caller's account to `recipient`.
   *
   * Returns a boolean value indicating whether the operation succeeded.
   *
   * Emits a {Transfer} event.
   */
  function transfer(address recipient, uint256 amount) external returns (bool);

  /**
   * @dev Returns the remaining number of tokens that `spender` will be
   * allowed to spend on behalf of `owner` through {transferFrom}. This is
   * zero by default.
   *
   * This value changes when {approve} or {transferFrom} are called.
   */
  function allowance(address owner, address spender) external view returns (uint256);

  /**
   * @dev Sets `amount` as the allowance of `spender` over the caller's tokens.
   *
   * Returns a boolean value indicating whether the operation succeeded.
   *
   * IMPORTANT: Beware that changing an allowance with this method brings the risk
   * that someone may use both the old and the new allowance by unfortunate
   * transaction ordering. One possible solution to mitigate this race
   * condition is to first reduce the spender's allowance to 0 and set the
   * desired value afterwards:
   * https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729
   *
   * Emits an {Approval} event.
   */
  function approve(address spender, uint256 amount) external returns (bool);

  /**
   * @dev Moves `amount` tokens from `sender` to `recipient` using the
   * allowance mechanism. `amount` is then deducted from the caller's
   * allowance.
   *
   * Returns a boolean value indicating whether the operation succeeded.
   *
   * Emits a {Transfer} event.
   */
  function transferFrom(address sender, address recipient, uint256 amount) external returns (bool);

  /**
   * @dev Emitted when `value` tokens are moved from one account (`from`) to
   * another (`to`).
   *
   * Note that `value` may be zero.
   */
  event Transfer(address indexed from, address indexed to, uint256 value);

  /**
   * @dev Emitted when the allowance of a `spender` for an `owner` is set by
   * a call to {approve}. `value` is the new allowance.
   */
  event Approval(address indexed owner, address indexed spender, uint256 value);
}

library LowGasSafeMath {
    /// @notice Returns x + y, reverts if sum overflows uint256
    /// @param x The augend
    /// @param y The addend
    /// @return z The sum of x and y
    function add(uint256 x, uint256 y) internal pure returns (uint256 z) {
        require((z = x + y) >= x);
    }

    /// @notice Returns x - y, reverts if underflows
    /// @param x The minuend
    /// @param y The subtrahend
    /// @return z The difference of x and y
    function sub(uint256 x, uint256 y) internal pure returns (uint256 z) {
        require((z = x - y) <= x);
    }

    /// @notice Returns x * y, reverts if overflows
    /// @param x The multiplicand
    /// @param y The multiplier
    /// @return z The product of x and y
    function mul(uint256 x, uint256 y) internal pure returns (uint256 z) {
        require(x == 0 || (z = x * y) / x == y);
    }

```

```
   */
  function approve(address spender, uint256 amount) external returns (bool);

  /**
   * @dev Moves `amount` tokens from `sender` to `recipient` using the
   * allowance mechanism. `amount` is then deducted from the caller's
   * allowance.
   *
   * Returns a boolean value indicating whether the operation succeeded.
   *
   * Emits a {Transfer} event.
   */
  function transferFrom(address sender, address recipient, uint256 amount) external returns (bool);

  /**
   * @dev Emitted when `value` tokens are moved from one account (`from`) to
   * another (`to`).
   *
   * Note that `value` may be zero.
   */
  event Transfer(address indexed from, address indexed to, uint256 value);

  /**
   * @dev Emitted when the allowance of a `spender` for an `owner` is set by
   * a call to {approve}. `value` is the new allowance.
   */
  event Approval(address indexed owner, address indexed spender, uint256 value);
}

library LowGasSafeMath {
    /// @notice Returns x + y, reverts if sum overflows uint256
    /// @param x The augend
    /// @param y The addend
    /// @return z The sum of x and y
    function add(uint256 x, uint256 y) internal pure returns (uint256 z) {
        require((z = x + y) >= x);
    }

    /// @notice Returns x - y, reverts if underflows
    /// @param x The minuend
    /// @param y The subtrahend
    /// @return z The difference of x and y
    function sub(uint256 x, uint256 y) internal pure returns (uint256 z) {
        require((z = x - y) <= x);
    }

    /// @notice Returns x * y, reverts if overflows
    /// @param x The multiplicand
    /// @param y The multiplier
    /// @return z The product of x and y
    function mul(uint256 x, uint256 y) internal pure returns (uint256 z) {
        require(x == 0 || (z = x * y) / x == y);
    }

    /// @notice Returns x + y, reverts if overflows or underflows
    /// @param x The augend
    /// @param y The addend
    /// @return z The sum of x and y
    function add(int256 x, int256 y) internal pure returns (int256 z) {
        require((z = x + y) >= x == (y >= 0));
    }

    /// @notice Returns x - y, reverts if overflows or underflows
    /// @param x The minuend
    /// @param y The subtrahend
    /// @return z The difference of x and y
    function sub(int256 x, int256 y) internal pure returns (int256 z) {
        require((z = x - y) <= x == (y >= 0));
    }
}

abstract contract ERC20 is IERC20 {

  using LowGasSafeMath for uint256;

  // Present in ERC777
  mapping (address => uint256) internal _balances;

  // Present in ERC777
  mapping (address => mapping (address => uint256)) internal _allowances;

  // Present in ERC777
  uint256 internal _totalSupply;

  // Present in ERC777
  string internal _name;

  // Present in ERC777
  string internal _symbol;

  // Present in ERC777
  uint8 internal _decimals;

  constructor (string memory name_, string memory symbol_, uint8 decimals_) {
    _name = name_;
    _symbol = symbol_;
    _decimals = decimals_;
  }

  function name() public view returns (string memory) {
    return _name;
  }

  function symbol() public view returns (string memory) {
    return _symbol;
  }

  function decimals() public view returns (uint8) {
    return _decimals;
  }

  function totalSupply() public view override returns (uint256) {
    return _totalSupply;
  }

```

```
    /// @notice Returns x + y, reverts if overflows or underflows
    /// @param x The augend
    /// @param y The addend
    /// @return z The sum of x and y
    function add(int256 x, int256 y) internal pure returns (int256 z) {
        require((z = x + y) >= x == (y >= 0));
    }

    /// @notice Returns x - y, reverts if overflows or underflows
    /// @param x The minuend
    /// @param y The subtrahend
    /// @return z The difference of x and y
    function sub(int256 x, int256 y) internal pure returns (int256 z) {
        require((z = x - y) <= x == (y >= 0));
    }
}

abstract contract ERC20 is IERC20 {

  using LowGasSafeMath for uint256;

  // Present in ERC777
  mapping (address => uint256) internal _balances;

  // Present in ERC777
  mapping (address => mapping (address => uint256)) internal _allowances;

  // Present in ERC777
  uint256 internal _totalSupply;

  // Present in ERC777
  string internal _name;

  // Present in ERC777
  string internal _symbol;

  // Present in ERC777
  uint8 internal _decimals;

  constructor (string memory name_, string memory symbol_, uint8 decimals_) {
    _name = name_;
    _symbol = symbol_;
    _decimals = decimals_;
  }

  function name() public view returns (string memory) {
    return _name;
  }

  function symbol() public view returns (string memory) {
    return _symbol;
  }

  function decimals() public view returns (uint8) {
    return _decimals;
  }

  function totalSupply() public view override returns (uint256) {
    return _totalSupply;
  }

  function balanceOf(address account) public view virtual override returns (uint256) {
    return _balances[account];
  }

  function transfer(address recipient, uint256 amount) public virtual override returns (bool) {
    _transfer(msg.sender, recipient, amount);
    return true;
  }

    function allowance(address owner, address spender) public view virtual override returns (uint256) {
        return _allowances[owner][spender];
    }

    function approve(address spender, uint256 amount) public virtual override returns (bool) {
        _approve(msg.sender, spender, amount);
        return true;
    }

    function transferFrom(address sender, address recipient, uint256 amount) public virtual override returns (bool) {
        _transfer(sender, recipient, amount);
        _approve(sender, msg.sender, _allowances[sender][msg.sender]
          .sub(amount));
        return true;
    }

    function increaseAllowance(address spender, uint256 addedValue) public virtual returns (bool) {
        _approve(msg.sender, spender, _allowances[msg.sender][spender].add(addedValue));
        return true;
    }

    function decreaseAllowance(address spender, uint256 subtractedValue) public virtual returns (bool) {
        _approve(msg.sender, spender, _allowances[msg.sender][spender]
          .sub(subtractedValue));
        return true;
    }

    function _transfer(address sender, address recipient, uint256 amount) internal virtual {
      require(sender != address(0), "ERC20: transfer from the zero address");
      require(recipient != address(0), "ERC20: transfer to the zero address");

      _beforeTokenTransfer(sender, recipient, amount);

      _balances[sender] = _balances[sender].sub(amount);
      _balances[recipient] = _balances[recipient].add(amount);
      emit Transfer(sender, recipient, amount);
    }

    function _mint(address account_, uint256 amount_) internal virtual {
        require(account_ != address(0), "ERC20: mint to the zero address");
```

```
  function balanceOf(address account) public view virtual override returns (uint256) {
    return _balances[account];
  }

  function transfer(address recipient, uint256 amount) public virtual override returns (bool) {
    _transfer(msg.sender, recipient, amount);
    return true;
  }

    function allowance(address owner, address spender) public view virtual override returns (uint256) {
        return _allowances[owner][spender];
    }

    function approve(address spender, uint256 amount) public virtual override returns (bool) {
        _approve(msg.sender, spender, amount);
        return true;
    }

    function transferFrom(address sender, address recipient, uint256 amount) public virtual override returns (bool) {
        _transfer(sender, recipient, amount);
        _approve(sender, msg.sender, _allowances[sender][msg.sender]
          .sub(amount));
        return true;
    }

    function increaseAllowance(address spender, uint256 addedValue) public virtual returns (bool) {
        _approve(msg.sender, spender, _allowances[msg.sender][spender].add(addedValue));
        return true;
    }

    function decreaseAllowance(address spender, uint256 subtractedValue) public virtual returns (bool) {
        _approve(msg.sender, spender, _allowances[msg.sender][spender]
          .sub(subtractedValue));
        return true;
    }

    function _transfer(address sender, address recipient, uint256 amount) internal virtual {
      require(sender != address(0), "ERC20: transfer from the zero address");
      require(recipient != address(0), "ERC20: transfer to the zero address");

      _beforeTokenTransfer(sender, recipient, amount);

      _balances[sender] = _balances[sender].sub(amount);
      _balances[recipient] = _balances[recipient].add(amount);
      emit Transfer(sender, recipient, amount);
    }

    function _mint(address account_, uint256 amount_) internal virtual {
        require(account_ != address(0), "ERC20: mint to the zero address");
        _beforeTokenTransfer(address( this ), account_, amount_);
        _totalSupply = _totalSupply.add(amount_);
        _balances[account_] = _balances[account_].add(amount_);
        emit Transfer(address(0), account_, amount_);
    }

    function _burn(address account, uint256 amount) internal virtual {
        require(account != address(0), "ERC20: burn from the zero address");

        _beforeTokenTransfer(account, address(0), amount);

        _balances[account] = _balances[account].sub(amount);
        _totalSupply = _totalSupply.sub(amount);
        emit Transfer(account, address(0), amount);
    }

    function _approve(address owner, address spender, uint256 amount) internal virtual {
        require(owner != address(0), "ERC20: approve from the zero address");
        require(spender != address(0), "ERC20: approve to the zero address");

        _allowances[owner][spender] = amount;
        emit Approval(owner, spender, amount);
    }

  function _beforeTokenTransfer( address from_, address to_, uint256 amount_ ) internal virtual { }
}

library Counters {
    using LowGasSafeMath for uint256;

    struct Counter {
        uint256 _value; // default: 0
    }

    function current(Counter storage counter) internal view returns (uint256) {
        return counter._value;
    }

    function increment(Counter storage counter) internal {
        counter._value += 1;
    }

    function decrement(Counter storage counter) internal {
        counter._value = counter._value.sub(1);
    }
}

interface IERC2612Permit {

    function permit(
        address owner,
        address spender,
        uint256 amount,
        uint256 deadline,
        uint8 v,
        bytes32 r,
```

```
        _beforeTokenTransfer(address( this ), account_, amount_);
        _totalSupply = _totalSupply.add(amount_);
        _balances[account_] = _balances[account_].add(amount_);
        emit Transfer(address(0), account_, amount_);
    }

    function _burn(address account, uint256 amount) internal virtual {
        require(account != address(0), "ERC20: burn from the zero address");

        _beforeTokenTransfer(account, address(0), amount);

        _balances[account] = _balances[account].sub(amount);
        _totalSupply = _totalSupply.sub(amount);
        emit Transfer(account, address(0), amount);
    }

    function _approve(address owner, address spender, uint256 amount) internal virtual {
        require(owner != address(0), "ERC20: approve from the zero address");
        require(spender != address(0), "ERC20: approve to the zero address");

        _allowances[owner][spender] = amount;
        emit Approval(owner, spender, amount);
    }

  function _beforeTokenTransfer( address from_, address to_, uint256 amount_ ) internal virtual { }
}

library Counters {
    using LowGasSafeMath for uint256;

    struct Counter {
        uint256 _value; // default: 0
    }

    function current(Counter storage counter) internal view returns (uint256) {
        return counter._value;
    }

    function increment(Counter storage counter) internal {
        counter._value += 1;
    }

    function decrement(Counter storage counter) internal {
        counter._value = counter._value.sub(1);
    }
}

interface IERC2612Permit {

    function permit(
        address owner,
        address spender,
        uint256 amount,
        uint256 deadline,
        uint8 v,
        bytes32 r,
        bytes32 s
    ) external;

    function nonces(address owner) external view returns (uint256);
}

abstract contract ERC20Permit is ERC20, IERC2612Permit {
    using Counters for Counters.Counter;

    mapping(address => Counters.Counter) private _nonces;

    // keccak256("Permit(address owner,address spender,uint256 value,uint256 nonce,uint256 deadline)");
    bytes32 public constant PERMIT_TYPEHASH = 0x6e71edae12b1b97f4d1f60370fef10105fa2faae0126114a169c64845d6126c9;

    bytes32 public DOMAIN_SEPARATOR;

    constructor() {
        uint256 chainID;
        assembly {
            chainID := chainid()
        }

        DOMAIN_SEPARATOR = keccak256(
            abi.encode(
                keccak256("EIP712Domain(string name,string version,uint256 chainId,address verifyingContract)"),
                keccak256(bytes(name())),
                keccak256(bytes("1")), // Version
                chainID,
                address(this)
            )
        );
    }

    function permit(
        address owner,
        address spender,
        uint256 amount,
        uint256 deadline,
        uint8 v,
        bytes32 r,
        bytes32 s
    ) public virtual override {
        require(block.timestamp <= deadline, "Permit: expired deadline");

        bytes32 hashStruct =
            keccak256(abi.encode(PERMIT_TYPEHASH, owner, spender, amount, _nonces[owner].current(), deadline));

        bytes32 _hash = keccak256(abi.encodePacked(uint16(0x1901), DOMAIN_SEPARATOR, hashStruct));

        address signer = ecrecover(_hash, v, r, s);
        require(signer != address(0) && signer == owner, "ERC20Permit: Invalid signature");

        _nonces[owner].increment();
        _approve(owner, spender, amount);
    }

```

```
        bytes32 s
    ) external;

    function nonces(address owner) external view returns (uint256);
}

abstract contract ERC20Permit is ERC20, IERC2612Permit {
    using Counters for Counters.Counter;

    mapping(address => Counters.Counter) private _nonces;

    // keccak256("Permit(address owner,address spender,uint256 value,uint256 nonce,uint256 deadline)");
    bytes32 public constant PERMIT_TYPEHASH = 0x6e71edae12b1b97f4d1f60370fef10105fa2faae0126114a169c64845d6126c9;

    bytes32 public DOMAIN_SEPARATOR;

    constructor() {
        uint256 chainID;
        assembly {
            chainID := chainid()
        }

        DOMAIN_SEPARATOR = keccak256(
            abi.encode(
                keccak256("EIP712Domain(string name,string version,uint256 chainId,address verifyingContract)"),
                keccak256(bytes(name())),
                keccak256(bytes("1")), // Version
                chainID,
                address(this)
            )
        );
    }

    function permit(
        address owner,
        address spender,
        uint256 amount,
        uint256 deadline,
        uint8 v,
        bytes32 r,
        bytes32 s
    ) public virtual override {
        require(block.timestamp <= deadline, "Permit: expired deadline");

        bytes32 hashStruct =
            keccak256(abi.encode(PERMIT_TYPEHASH, owner, spender, amount, _nonces[owner].current(), deadline));

        bytes32 _hash = keccak256(abi.encodePacked(uint16(0x1901), DOMAIN_SEPARATOR, hashStruct));

        address signer = ecrecover(_hash, v, r, s);
        require(signer != address(0) && signer == owner, "ERC20Permit: Invalid signature");

        _nonces[owner].increment();
        _approve(owner, spender, amount);
    }

    function nonces(address owner) public view override returns (uint256) {
        return _nonces[owner].current();
    }
}

interface IOwnable {
  function owner() external view returns (address);

  function renounceOwnership() external;

  function transferOwnership( address newOwner_ ) external;
}

contract Ownable is IOwnable {

  address internal _owner;

  event OwnershipTransferred(address indexed previousOwner, address indexed newOwner);

  constructor () {
    _owner = msg.sender;
    emit OwnershipTransferred( address(0), _owner );
  }

  function owner() public view override returns (address) {
    return _owner;
  }

  modifier onlyOwner() {
    require( _owner == msg.sender, "Ownable: caller is not the owner" );
    _;
  }

  function renounceOwnership() public virtual override onlyOwner() {
    emit OwnershipTransferred( _owner, address(0) );
    _owner = address(0);
  }

  function transferOwnership( address newOwner_ ) public virtual override onlyOwner() {
    require( newOwner_ != address(0), "Ownable: new owner is the zero address");
    emit OwnershipTransferred( _owner, newOwner_ );
    _owner = newOwner_;
  }
}

contract VaultOwned is Ownable {

  address internal _vault;

  event VaultTransferred(address indexed newVault);

  function setVault( address vault_ ) external onlyOwner() {
    require(vault_ != address(0), "IA0");
    _vault = vault_;
    emit VaultTransferred( _vault );
  }

  function vault() public view returns (address) {
    return _vault;
```

```
    function nonces(address owner) public view override returns (uint256) {
        return _nonces[owner].current();
    }
}

interface IOwnable {
  function owner() external view returns (address);

  function renounceOwnership() external;

  function transferOwnership( address newOwner_ ) external;
}

contract Ownable is IOwnable {

  address internal _owner;

  event OwnershipTransferred(address indexed previousOwner, address indexed newOwner);

  constructor () {
    _owner = msg.sender;
    emit OwnershipTransferred( address(0), _owner );
  }

  function owner() public view override returns (address) {
    return _owner;
  }

  modifier onlyOwner() {
    require( _owner == msg.sender, "Ownable: caller is not the owner" );
    _;
  }

  function renounceOwnership() public virtual override onlyOwner() {
    emit OwnershipTransferred( _owner, address(0) );
    _owner = address(0);
  }

  function transferOwnership( address newOwner_ ) public virtual override onlyOwner() {
    require( newOwner_ != address(0), "Ownable: new owner is the zero address");
    emit OwnershipTransferred( _owner, newOwner_ );
    _owner = newOwner_;
  }
}

contract VaultOwned is Ownable {

  address internal _vault;

  event VaultTransferred(address indexed newVault);

  function setVault( address vault_ ) external onlyOwner() {
    require(vault_ != address(0), "IA0");
    _vault = vault_;
    emit VaultTransferred( _vault );
  }

  function vault() public view returns (address) {
    return _vault;
  }

  modifier onlyVault() {
    require( _vault == msg.sender, "VaultOwned: caller is not the Vault" );
    _;
  }

}

contract TimeERC20Token is ERC20Permit, VaultOwned {

    using LowGasSafeMath for uint256;

    constructor() ERC20("Time", "TIME", 9) {
    }

    function mint(address account_, uint256 amount_) external onlyVault() {
        _mint(account_, amount_);
    }

    function burn(uint256 amount) external virtual {
        _burn(msg.sender, amount);
    }

    function burnFrom(address account_, uint256 amount_) external virtual {
        _burnFrom(account_, amount_);
    }

    function _burnFrom(address account_, uint256 amount_) internal virtual {
        uint256 decreasedAllowance_ =
            allowance(account_, msg.sender).sub(amount_);

        _approve(account_, msg.sender, decreasedAllowance_);
        _burn(account_, amount_);
    }
}
```

```
  }

  modifier onlyVault() {
    require( _vault == msg.sender, "VaultOwned: caller is not the Vault" );
    _;
  }

}

contract MaiaERC20Token is ERC20Permit, VaultOwned {

    using LowGasSafeMath for uint256;

    constructor() ERC20("Maia", "MAIA", 9) {
    }

    function mint(address account_, uint256 amount_) external onlyVault() {
        _mint(account_, amount_);
    }

    function burn(uint256 amount) external virtual {
        _burn(msg.sender, amount);
    }

    function burnFrom(address account_, uint256 amount_) external virtual {
        _burnFrom(account_, amount_);
    }

    function _burnFrom(address account_, uint256 amount_) internal virtual {
        uint256 decreasedAllowance_ =
            allowance(account_, msg.sender).sub(amount_);

        _approve(account_, msg.sender, decreasedAllowance_);
        _burn(account_, amount_);
    }
}
```