# Hardhat Upgrades

```
// contracts/Box.sol
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.8;

contract Box {
    uint256 internal value;

    // Emitted when the stored value changes
    event ValueChanged(uint256 newValue);

    // Stores a new value in the contract
    function store(uint256 newValue) public {
        value = newValue;
        emit ValueChanged(newValue);
    }

    // Reads the last stored value
    function retrieve() public view returns (uint256) {
        return value;
    }

    // Uncomment and redeploy to see the upgrade happen!
    // // Increments the stored value by 1
    // function increment() public {
    //     value = value + 1;
    //     emit ValueChanged(value);
    // }

    // Uncomment and redeploy to see the upgrade happen!
    // replace the "1" with a "2"
    function version() public pure returns (uint256) {
        return 1;
    }
}
```

```solidity
contract BoxV2 {
    uint256 internal value;

    // Emitted when the stored value changes
    event ValueChanged(uint256 newValue);

    // Stores a new value in the contract
    function store(uint256 newValue) public {
        value = newValue;
        emit ValueChanged(newValue);
    }

    // Reads the last stored value
    function retrieve() public view returns (uint256) {
        return value;
    }

    // Increments the stored value by 1
    function increment() public {
        value = value + 1;
        emit ValueChanged(value);
    }

    function version() public pure returns (uint256) {
        return 2;
    }
}
```

```solidity
import "@openzeppelin/contracts/proxy/transparent/ProxyAdmin.sol";

contract BoxProxyAdmin is ProxyAdmin {
    constructor(
        address /* owner */
    ) ProxyAdmin() {
        // We just need this for our hardhat tooling right now
    }
}
```

```
contract B {
    // NOTE: storage layout must be the same as contract A
    uint256 public num;
    address public sender;
    uint256 public value;

    function setVars(uint256 _num) public payable {
        num = _num;
        sender = msg.sender;
        value = msg.value;
    }
}

contract A {
    uint256 public num;
    address public sender;
    uint256 public value;

    function setVars(address _contract, uint256 _num) public payable {
        // A's storage is set, B is not modified.
        (bool success, bytes memory data) = _contract.delegatecall(
            abi.encodeWithSignature("setVars(uint256)", _num)
        );
    }
}
```

```
contract SmallProxy is Proxy {
    // This is the keccak-256 hash of "eip1967.proxy.implementation" subtracted by 1
    bytes32 private constant _IMPLEMENTATION_SLOT =
        0x360894a13ba1a3210667c828492db98dca3e2076cc3735a920a3ca505d382bbc;

    function setImplementation(address newImplementation) public {
        assembly {
            sstore(_IMPLEMENTATION_SLOT, newImplementation)
        }
    }

    function _implementation() internal view override returns (address implementationAddress) {
        assembly {
            implementationAddress := sload(_IMPLEMENTATION_SLOT)
        }
    }

    // helper function
    function getDataToTransact(uint256 numberToUpdate) public pure returns (bytes memory) {
        return abi.encodeWithSignature("setValue(uint256)", numberToUpdate);
    }

    function readStorage() public view returns (uint256 valueAtStorageSlotZero) {
        assembly {
            valueAtStorageSlotZero := sload(0)
        }
    }
}

contract ImplementationA {
    uint256 public value;

    function setValue(uint256 newValue) public {
        value = newValue;
    }
}
```

```
contract ImplementationB {
    uint256 public value;

    function setValue(uint256 newValue) public {
        value = newValue + 2;
    }
}
```