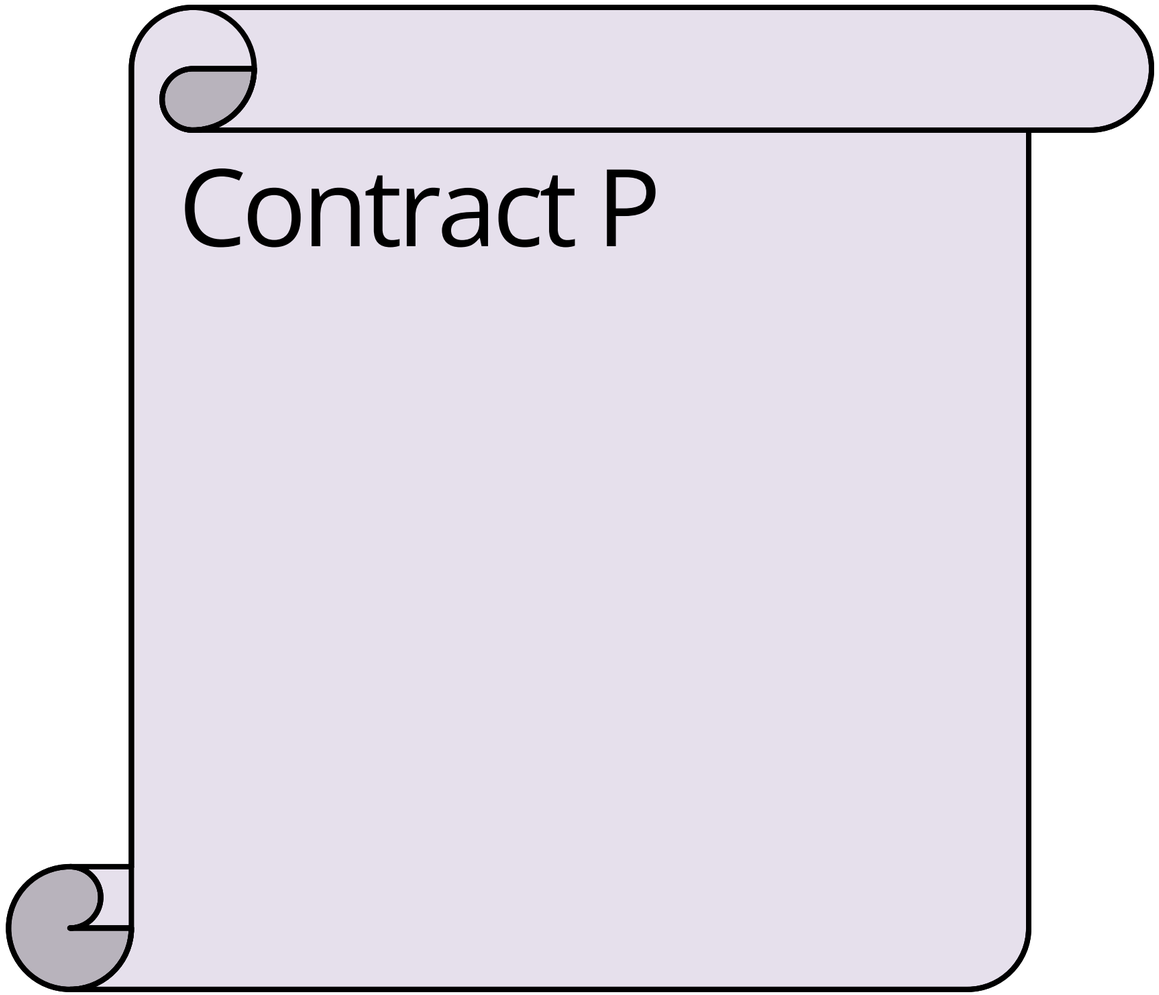
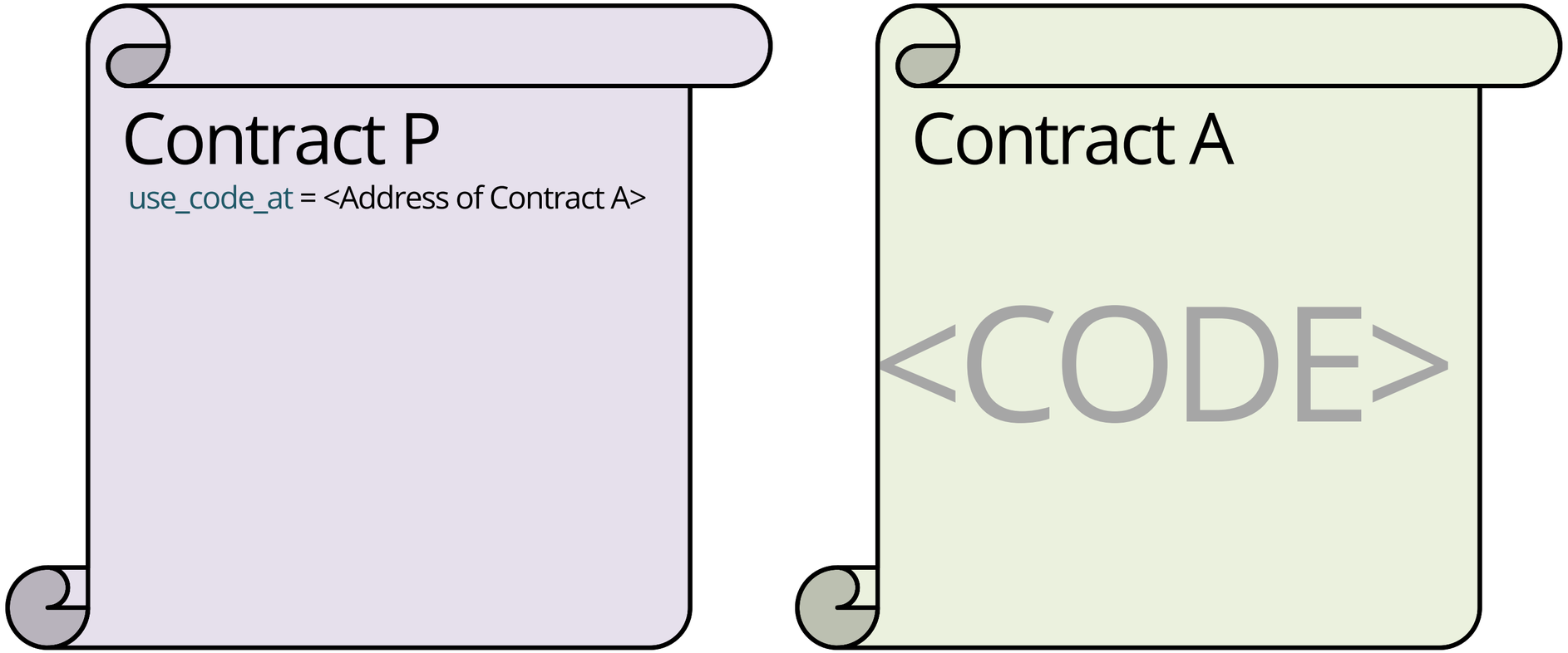
Sound Proxy Usage

Contents

What is a Proxy?


Contract P


Contract P
use_code_at = <Address of Contract A>

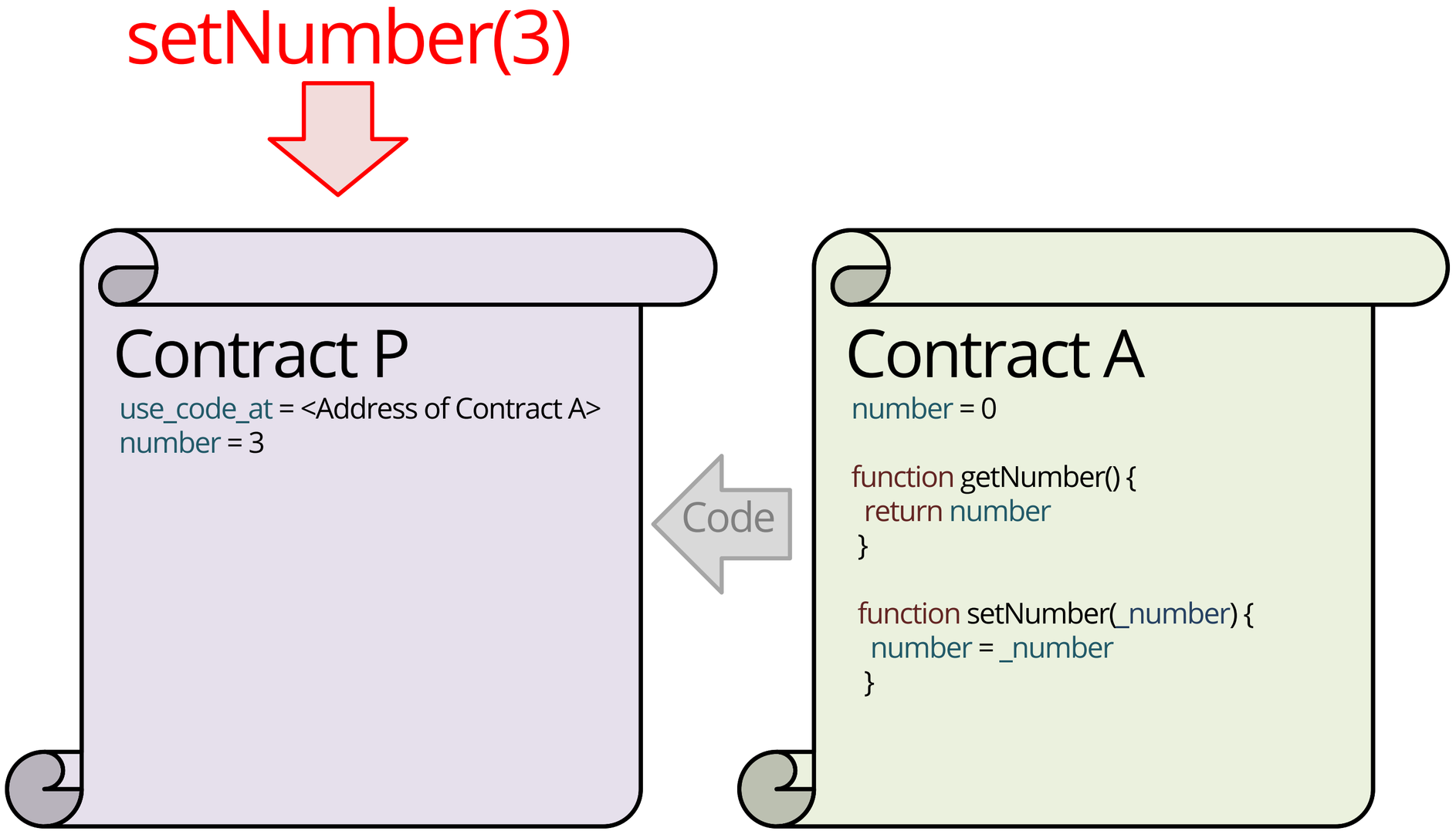

Contract P
use_code_at = <Address of Contract A>
Contract A
<CODE>

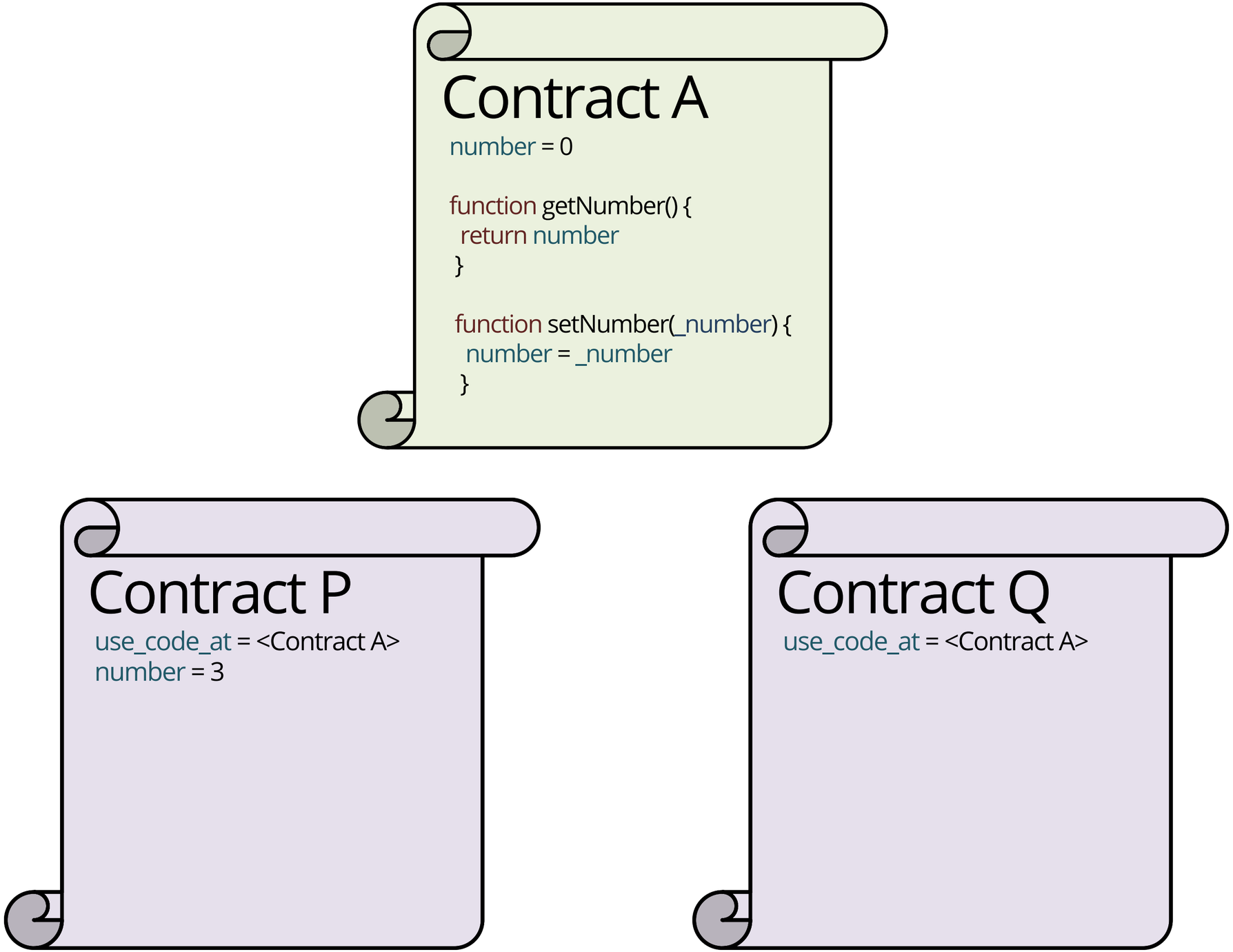
Contract P

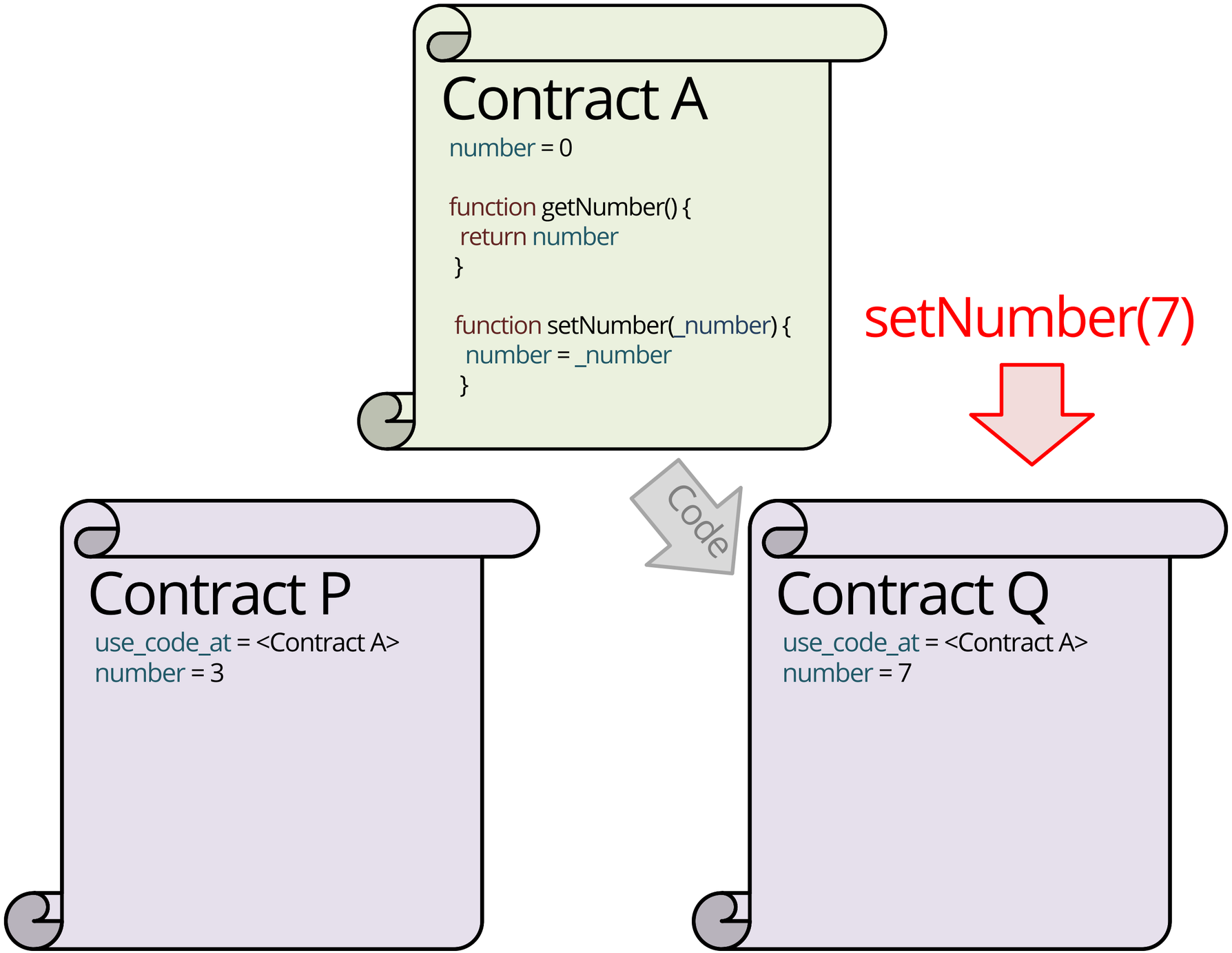
use_code_at = <Address of Contract A>

Contract A

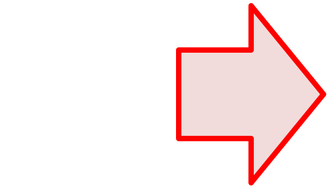
```
number = 0

function getNumber() {
  return number
}

 function setNumber(_number) {
   number = _number
 }
```


setNumber(3)
Contract P
use_code_at = <Address of Contract A>
number = 3
Code
Contract A
number = 0
function getNumber() {
return number
}
function setNumber(_number) {
number = _number
}


Contract A
number = 0
function getNumber() {
return number
}
function setNumber(_number) {
number = _number
}
Contract P
use_code_at = <Contract A>
number = 3
Contract Q
use_code_at = <Contract A>


Contract A
number = 0
function getNumber() {
return number
}
function setNumber(_number) {
number = _number
}
setNumber(7)
Code
Contract P
use_code_at = <Contract A>
number = 3
Contract Q
use_code_at = <Contract A>
number = 7

setNumber(1)



Contract A

number = 1

```
function getNumber() {
 return number
}

function setNumber(_number) {
 number = _number
}
```

Contract P

use_code_at = <Contract A>
number = 3

Contract Q

use_code_at = <Contract A>
number = 7

CALL

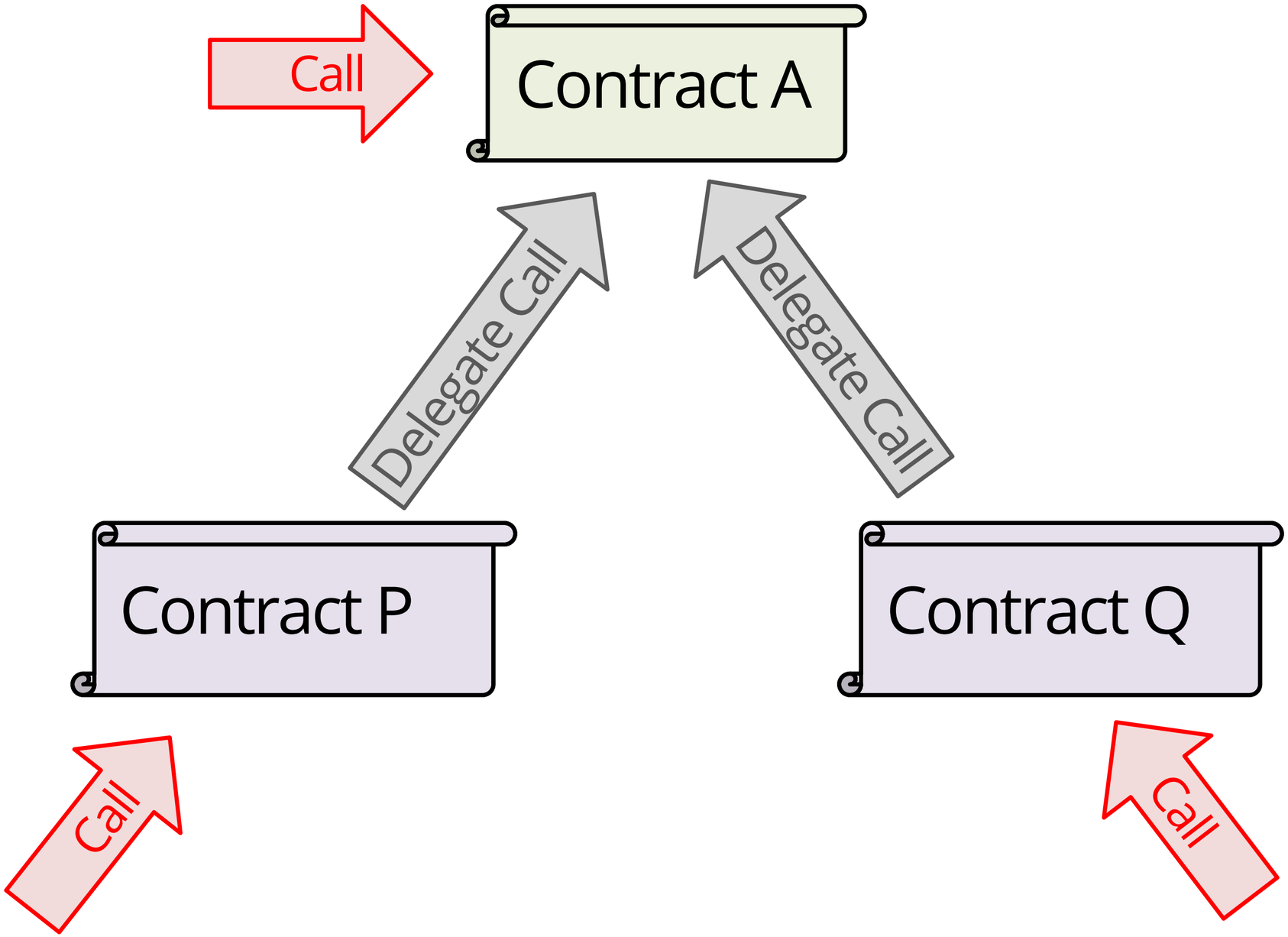
- Callee code
- Callee storage
- Callee address

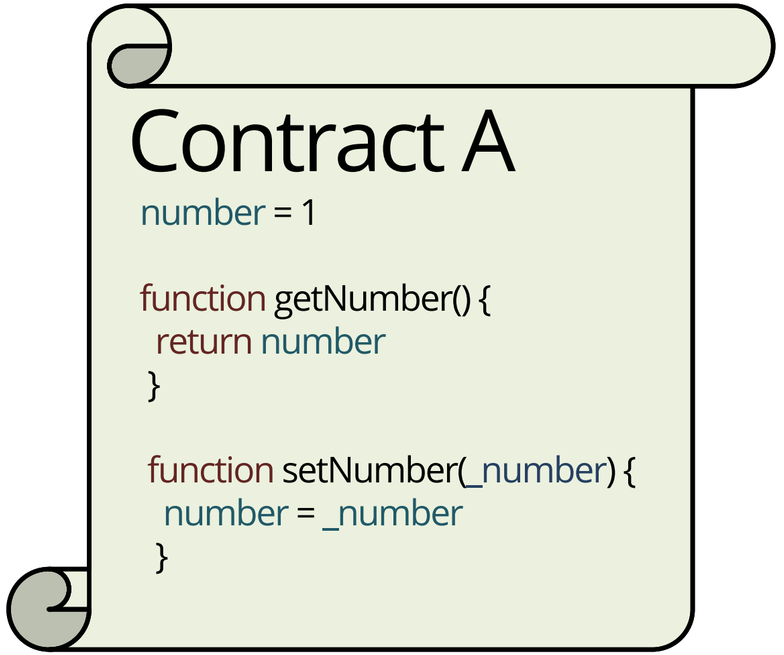
CALL

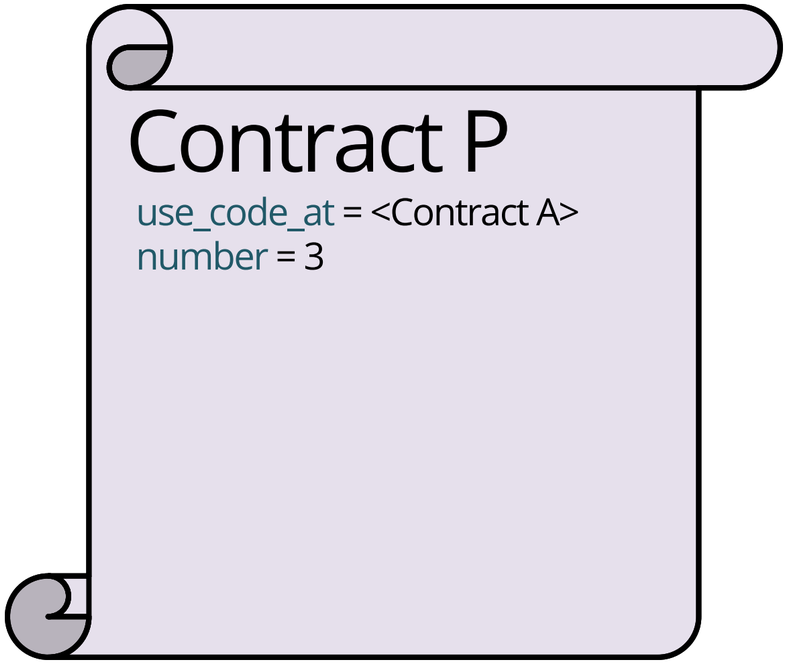
- Callee code
- Callee storage
- Callee address

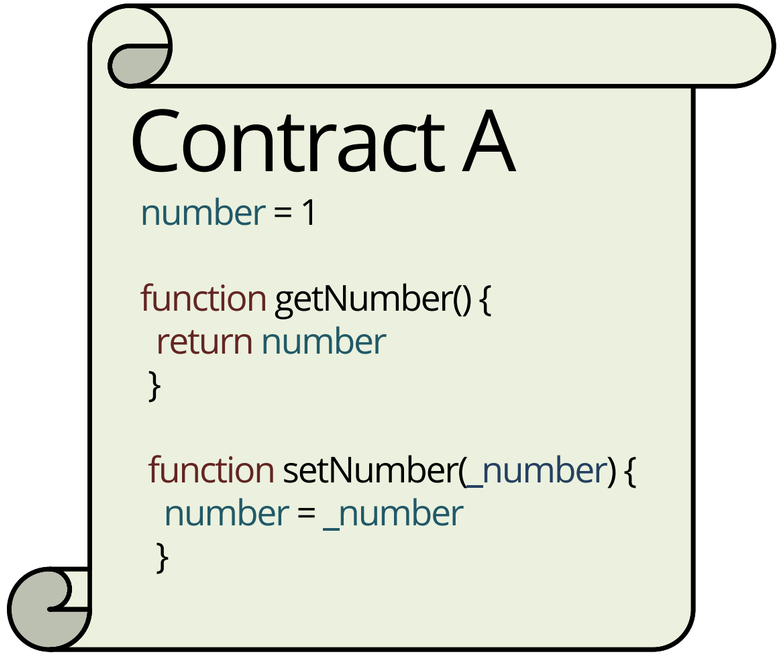
DELEGATE CALL

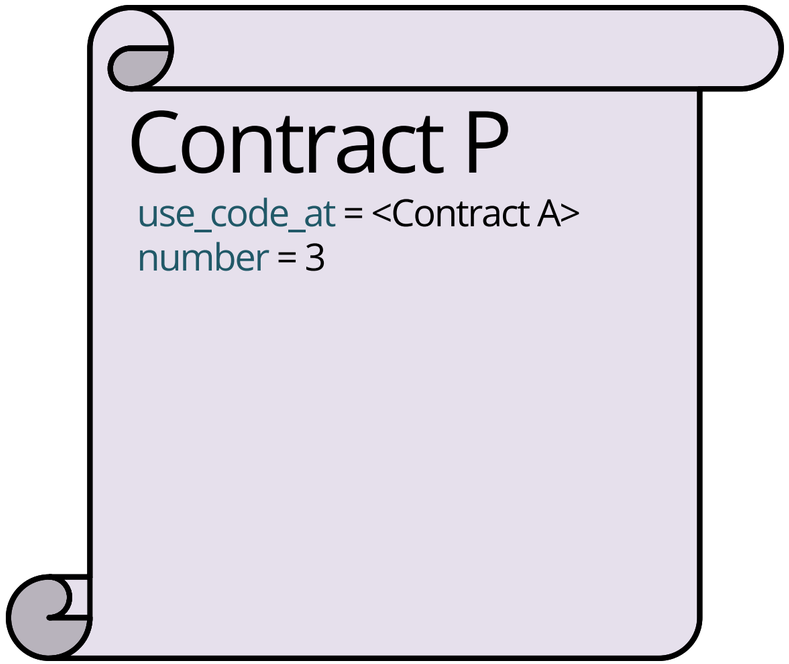
- Callee code
- Caller storage
- Caller address


Call
Contract A
Delegate Call
Delegate Call
Contract P
Contract Q
Call
Call


Contract A
number = 1
function getNumber() {
return number
}
function setNumber(_number) {
number = _number
}


Contract P
use_code_at = <Contract A>
number = 3


Contract A
number = 1
function getNumber() {
return number
}
function setNumber(_number) {
number = _number
}


Contract P
use_code_at = <Contract A>
number = 3

Contract A

```
number = 1

function getNumber() {
  return number
}

function setNumber(_number) {
  number = _number
}
```

Contract B

```
number = 0

function getNumber() {
  return number
}

function setNumber(_number) {
  number = _number
}

function getDoubleNumber() {
  return number * 2
}
```

Contract P

```
use_code_at = <Contract B>
number = 3
```

Contract A

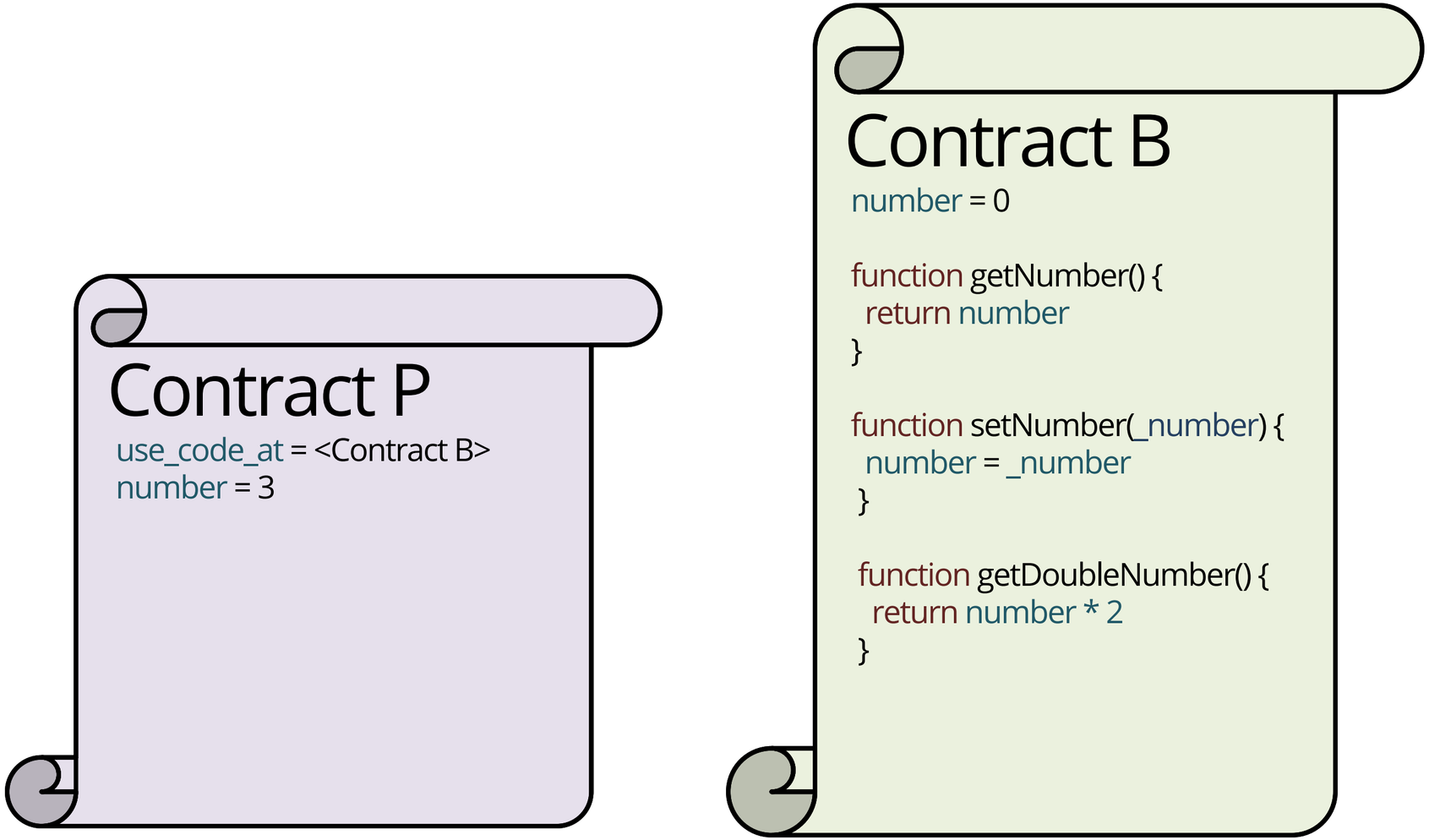
```
number = 1

function getNumber() {
  return number
}

function setNumber(_number) {
  number = _number
}
```

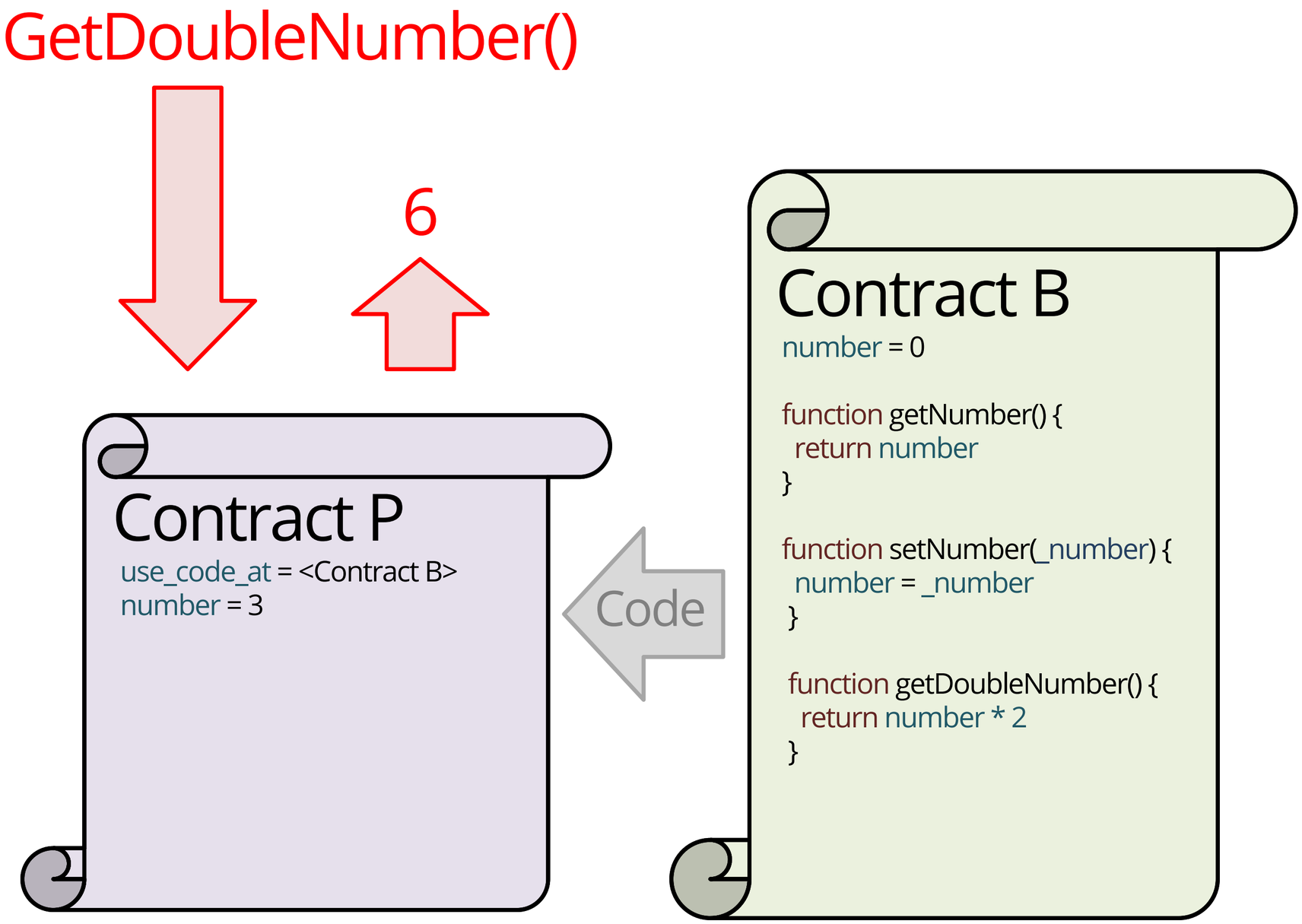
Contract B

```
number = 0

function getNumber() {
  return number
}

function setNumber(_number) {
  number = _number
}

function getDoubleNumber() {
  return number * 2
}
```

Contract P

```
use_code_at = <Contract B>
number = 3
```

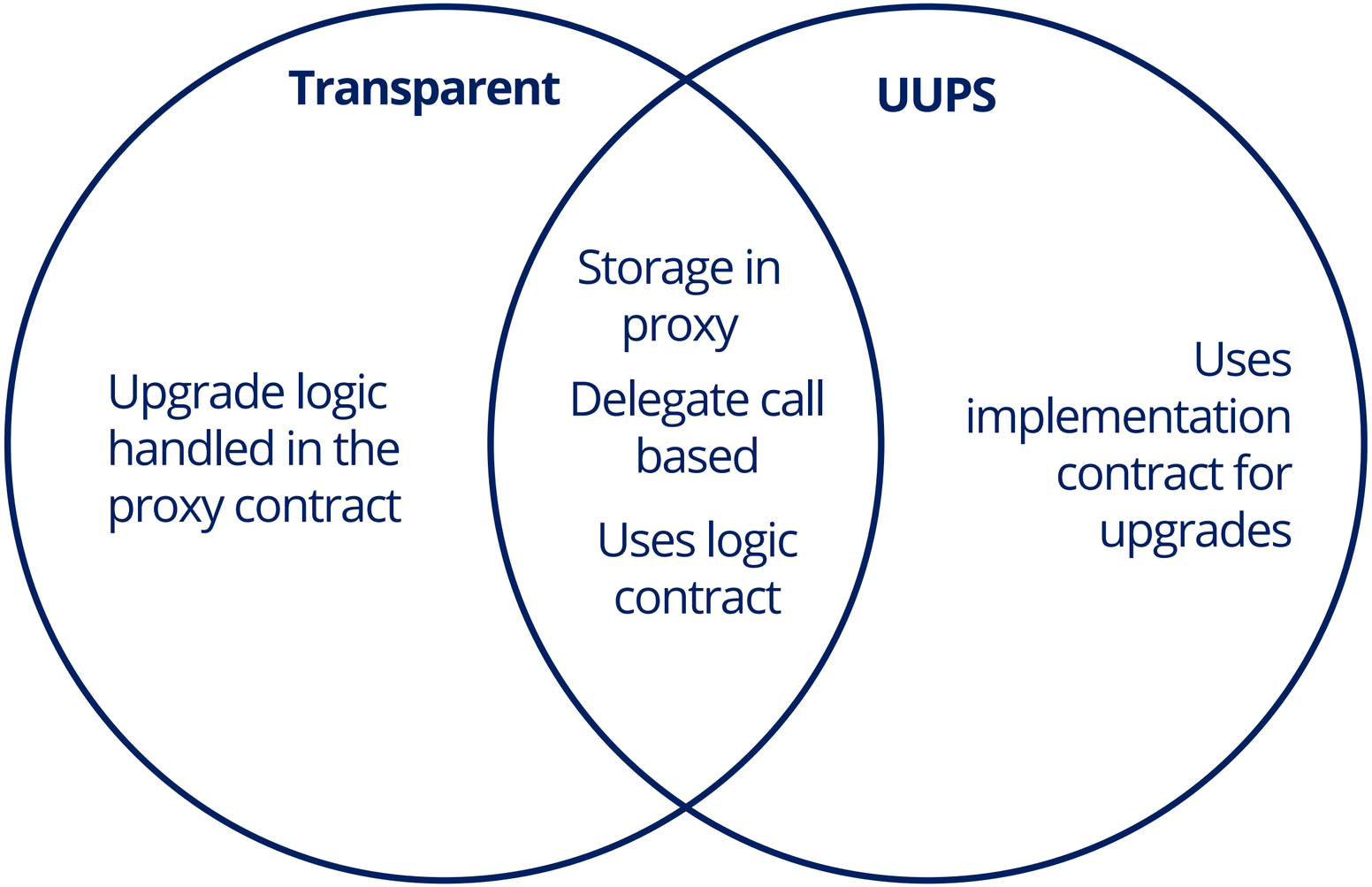

Contract P
use_code_at = <Contract B>
number = 3
Contract B
number = 0
function getNumber() {
return number
}
function setNumber(_number) {
number = _number
}
function getDoubleNumber() {
return number * 2
}


GetDoubleNumber()
6
Contract P
use_code_at = <Contract B>
number = 3
Code
Contract B
number = 0
function getNumber() {
return number
}
function setNumber(_number) {
number = _number
}
function getDoubleNumber() {
return number * 2
}

Transparent vs UUPS

TRANSPARENT vs UUPS

Both use DelegateCall, but differ in how they manage upgrading process.



TRANSPARENT vs UUPS

Transparent

Upgrade logic handled in the proxy contract

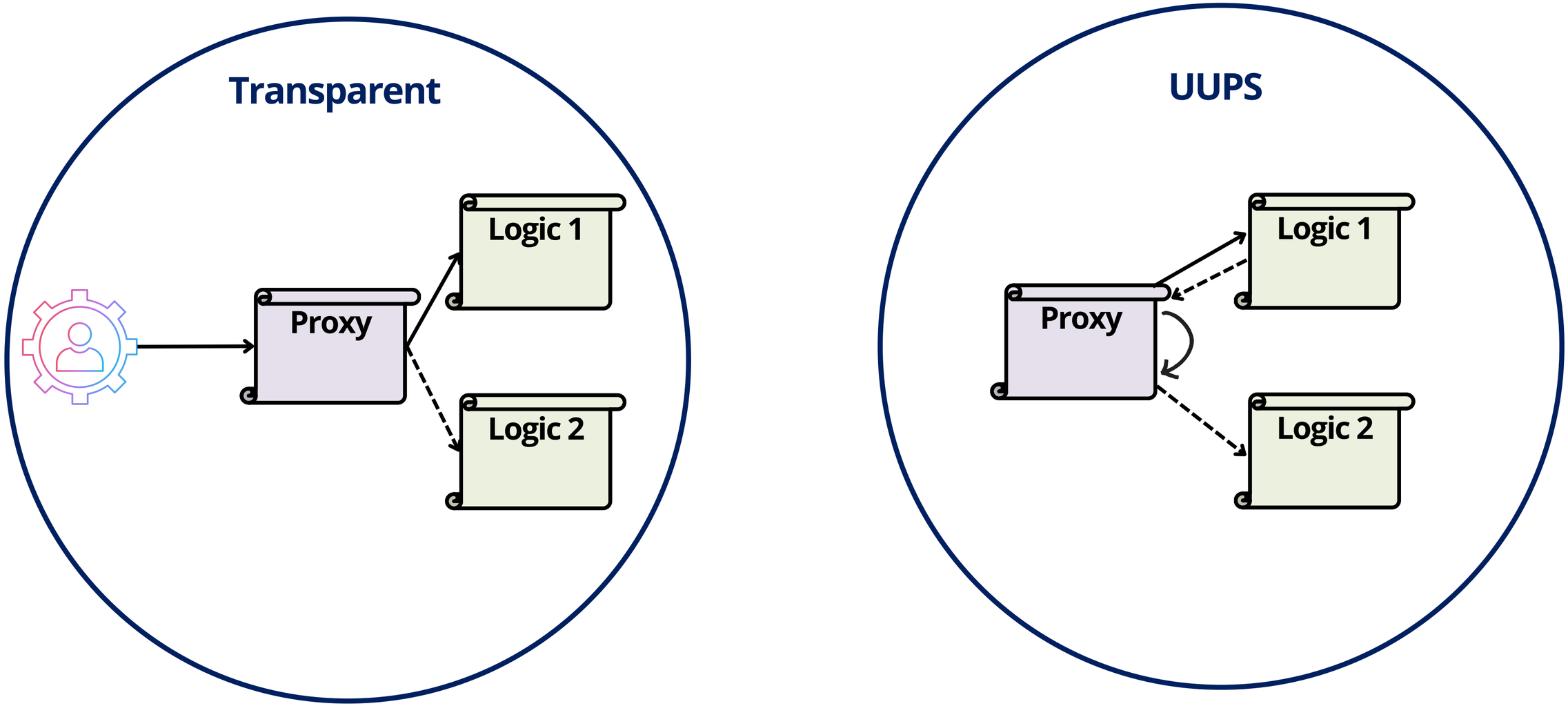
proxy functionality requires a designated administrator

UUPS

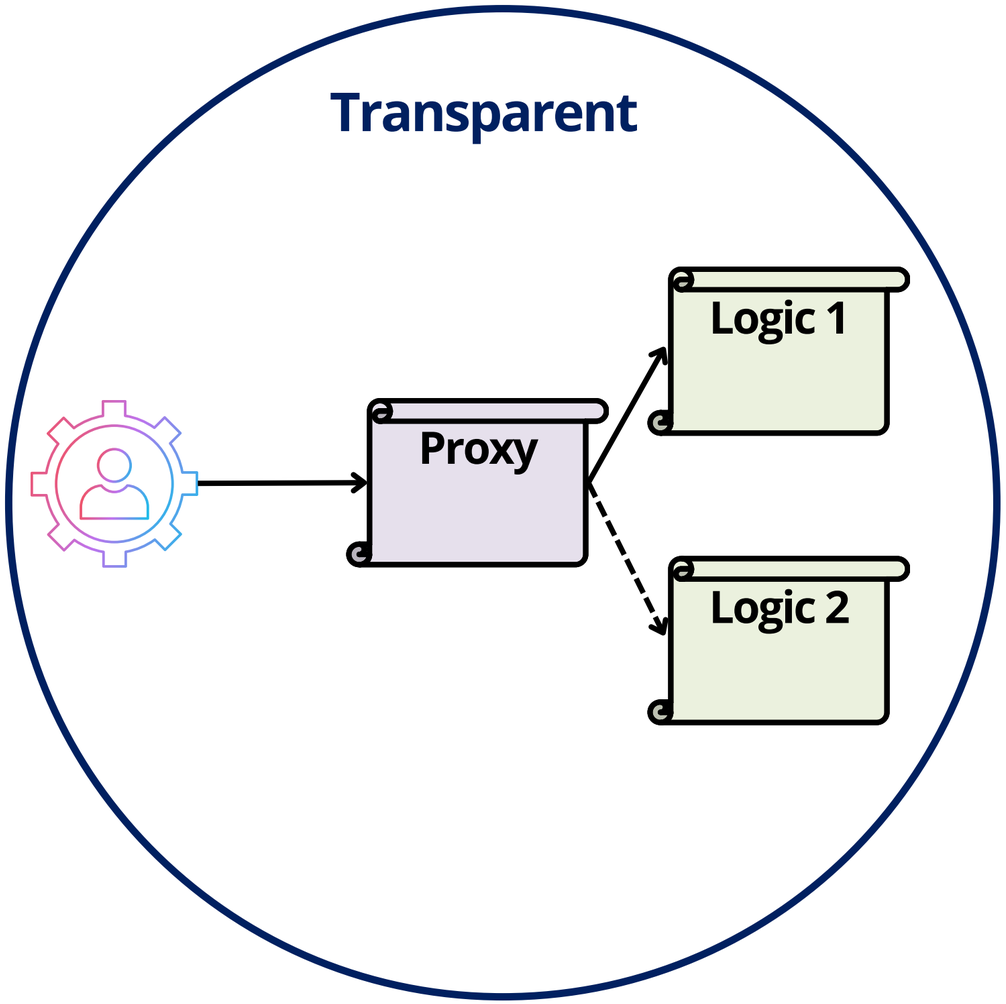
Uses implementation contract for upgrades

implementation contract decides who is able to upgrade base on it access control.

TRANSPARENT vs UUPS



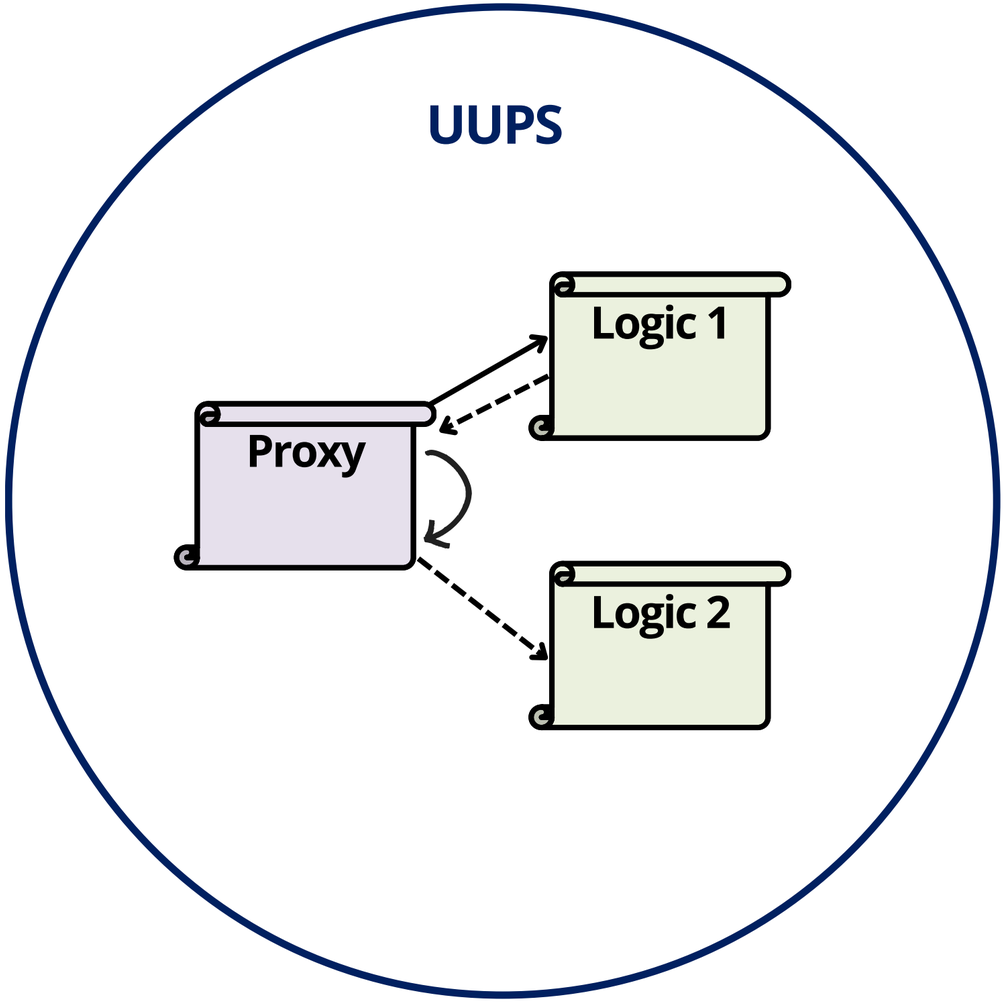
TRANSPARENT vs UUPS




```
/**
 * @dev If caller is the admin process the call internally, otherwise transparently fallbac
 */
function _fallback() internal virtual override {
    if (msg.sender == _admin) {
        bytes memory ret;
        bytes4 selector = msg.sig;
        if (selector == ITransparentUpgradeableProxy.upgradeTo.selector) {
            ret = _dispatchUpgradeTo();
        } else if (selector == ITransparentUpgradeableProxy.upgradeToAndCall.selector) {
            ret = _dispatchUpgradeToAndCall();
        } else {
            revert ProxyDeniedAdminAccess();
        }
        assembly {
            return(add(ret, 0x20), mload(ret))
        }
    } else {
        super._fallback();
    }
}

/**
```

TRANSPARENT vs UUPS

```
 *
 * @custom:oz-upgrades-unsafe-allow-reachable delegatecall
 */
function upgradeToAndCall(address newImplementation, bytes memory data) public payable virtual onlyProxy {
    _authorizeUpgrade(newImplementation);
    _upgradeToAndCallUUPS(newImplementation, data, true);
}
```



Best Practices

Best Practices

1. Don’t implement upgrade logic if you don’t need it.

Best Practices

1. Don’t implement upgrade logic if you don’t need it.
2. Use intializers, not constructors.

Best Practices

1. Don’t implement upgrade logic if you don’t need it.
2. Use intializers, not constructors.

```
constructor() {
    _disableInitializers();
}
```

Best Practices

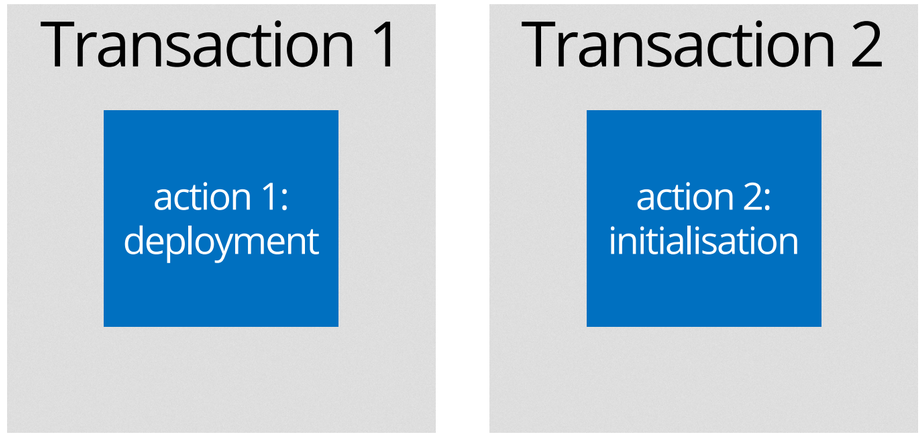
1. Don’t implement upgrade logic if you don’t need it.
2. Use intializers, not constructors.
3. Set up and initialize in one transaction.

Best Practices

1. Don’t implement upgrade logic if you don’t need it.
2. Use intializers, not constructors.
3. Set up and initialize in one transaction.

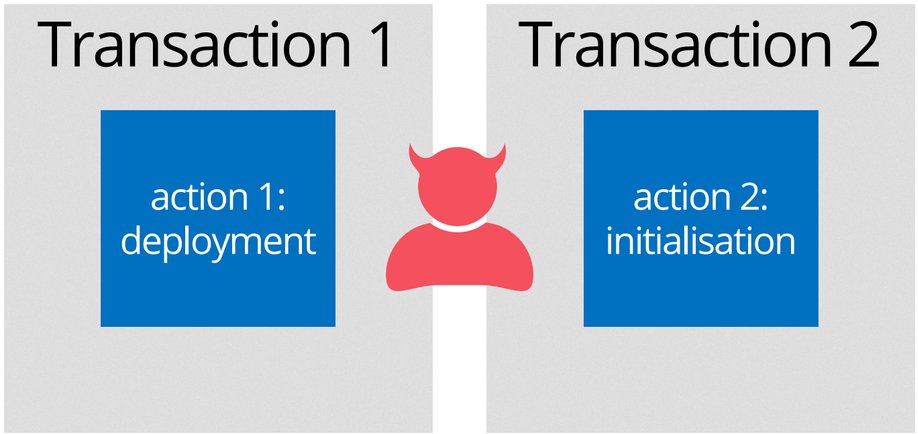
Best Practices

1. Don’t implement upgrade logic if you don’t need it.
2. Use intializers, not constructors.
3. Set up and initialize in one transaction.



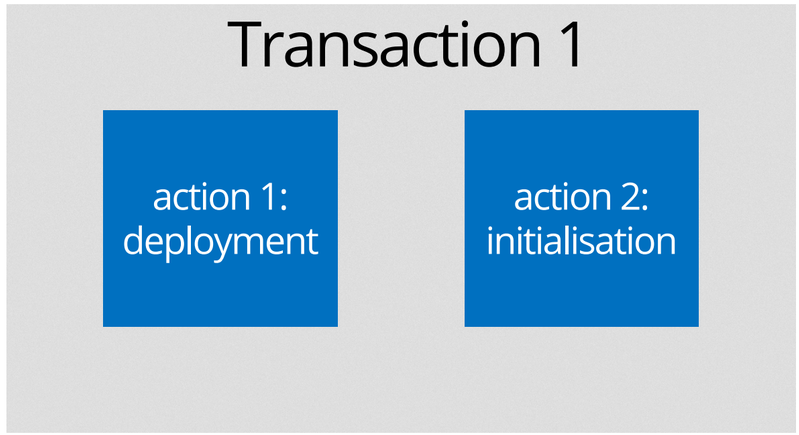
Best Practices

1. Don’t implement upgrade logic if you don’t need it.
2. Use intializers, not constructors.
3. Set up and initialize in one transaction.



Best Practices

1. Don’t implement upgrade logic if you don’t need it.
2. Use intializers, not constructors.
3. Set up and initialize in one transaction.



Best Practices

1. Don’t implement upgrade logic if you don’t need it.
2. Use intializers, not constructors.
3. Set up and initialize in one transaction.
4. Don’t use both Transparent and UUPS.

Best Practices

1. Don’t implement upgrade logic if you don’t need it.
2. Use intializers, not constructors.
3. Set up and initialize in one transaction.
4. Don’t use both Transparent and UUPS.
5. Only use trusted code for implementation contracts.