Proxy Pattern

The Proxy design pattern

- Provide a substitute for another object in order to control its access.

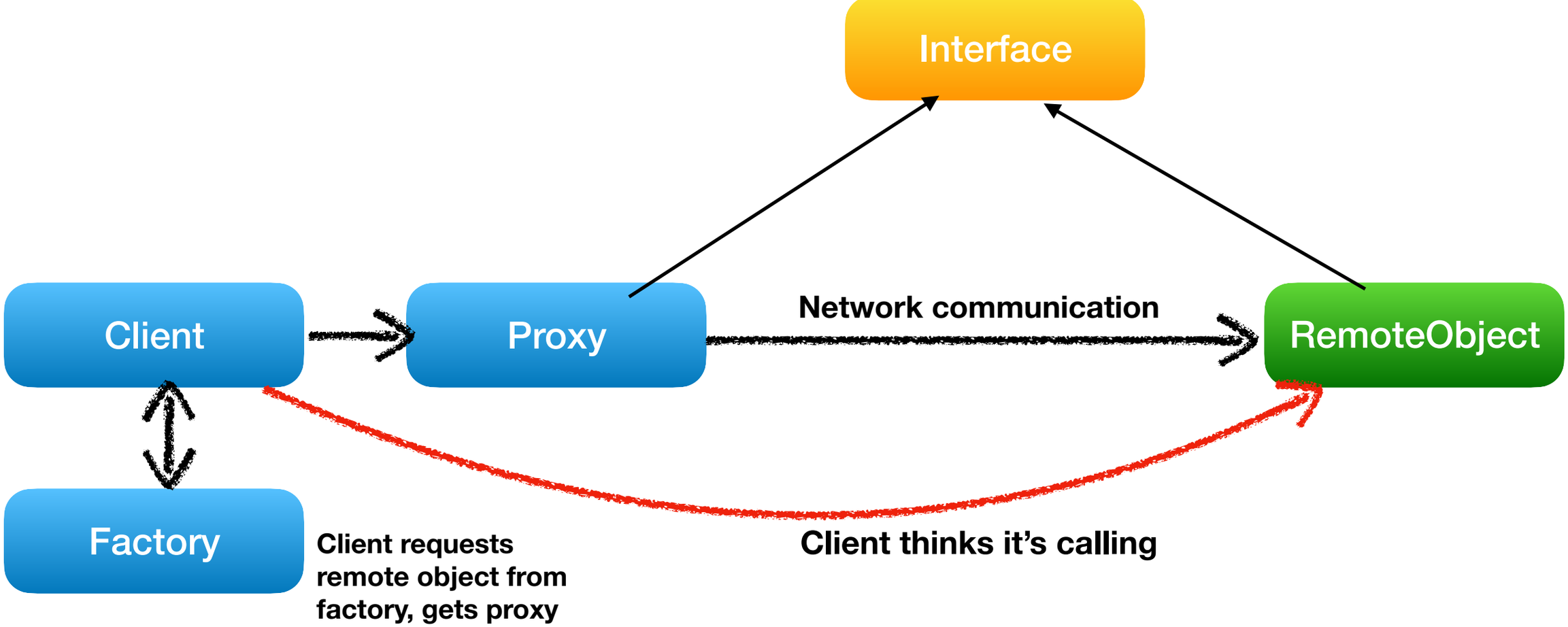
- The Proxy design template is your best friend when you have a remote object you want to work with and make it look like a local object. Or when you want to control access to that remote object in a certain way.

- Use the proxy design pattern to create a representative object that controls access to another object, which can be distant, costly to create, or requires security.

Remote Proxy Object

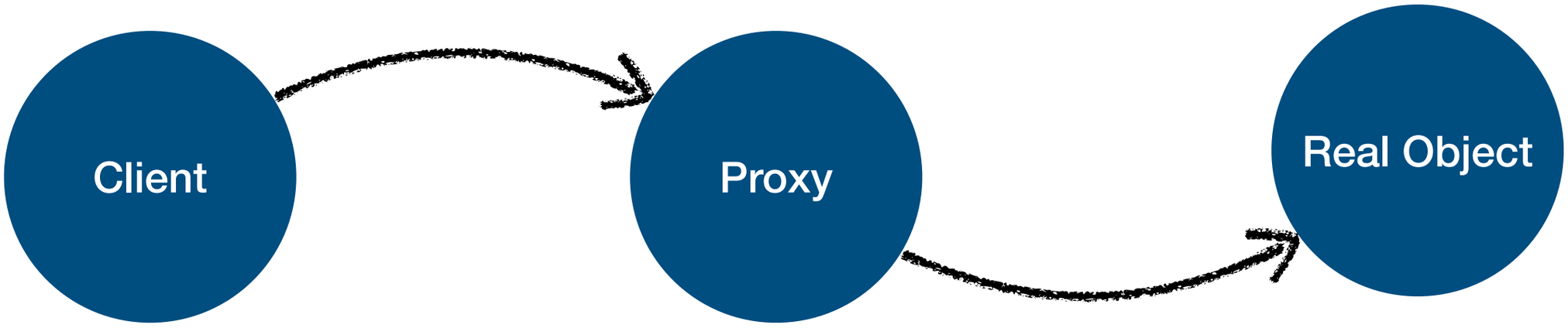
- A remote proxy object acts like a local representation of a remote object, which is another object that lives in the pile of a different computer. The local object you can use local methods and have it forward to the remote object.

- This means that you should write a code that takes a method invocation and transfers it to the network and invokes the same method on a remote object. Once the call is finished, the result is sent to the client through the network.



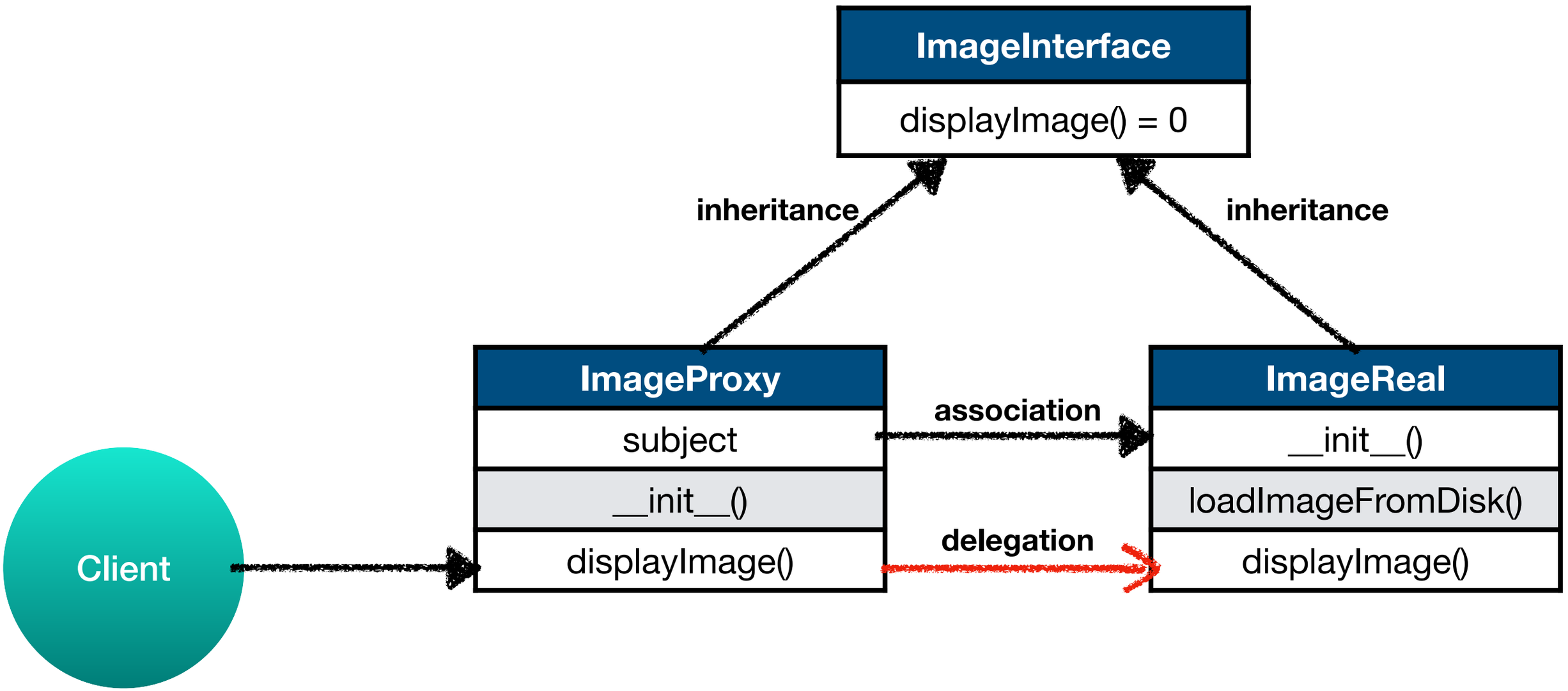
Virtual Proxy

- It acts as a representative for an object which can cause a lot of overheads to be created. The virtual proxy often postpones creating the object until it is required. It is also used as a substitute for the subject before and during its creation. Subsequently, the proxy delegates the requests directly to the real object.



Case Study - Virtual Proxy Demo

- The example simulates loading a large image file. Because loading a large image file takes a lot of time, it should only be loaded when the function is first called. After that, even the same function is called many times later, the image file does not need to be reloaded.

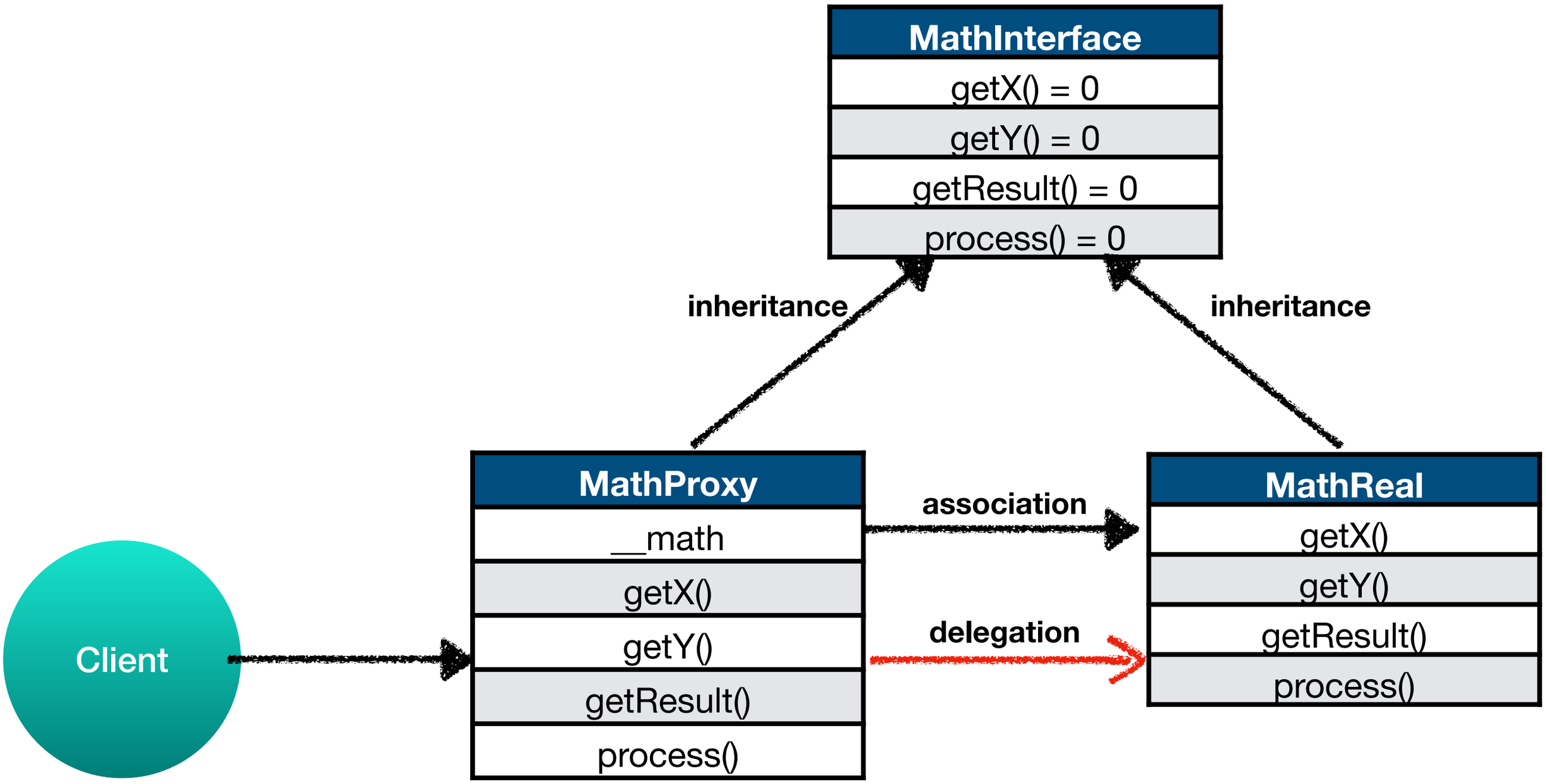


Case Study - Virtual Proxy Demo

```
from abc import ABC, abstractmethod

class ImageInterface(ABC):
    @abstractmethod
    def displayImage(self):
        pass

class ImageReal(ImageInterface):
    def __init__(self, filename):
        self._filename = filename

    def loadImageFromDisk(self):
        print("loading " + self._filename)

    def displayImage(self):
        print("display " + self._filename)

class ImageProxy(ImageInterface):
    def __init__( self, subject ):
        self._subject = subject
        self._proxystate = None

    def displayImage(self):
        if self._proxystate == None:
            self._subject.loadImageFromDisk()
            self._proxystate = 1
        print("display " + self._subject._filename)

def main():
    imag1 = ImageProxy(ImageReal('ABC.jpg'));
    imag2 = ImageProxy(ImageReal('DEF.jpg'));

    imag1.displayImage();
    imag2.displayImage();
    imag1.displayImage();
    imag2.displayImage();
    imag1.displayImage();
    imag2.displayImage();

main()
```

Case Study - Long Process

- The MathReal is a real object with a method referred to as "process". This method requires considerable time to calculate the result. Thus, instead of having the client wait until the method ends, we use the MathProxy object as a substitute before the real object ends the process.



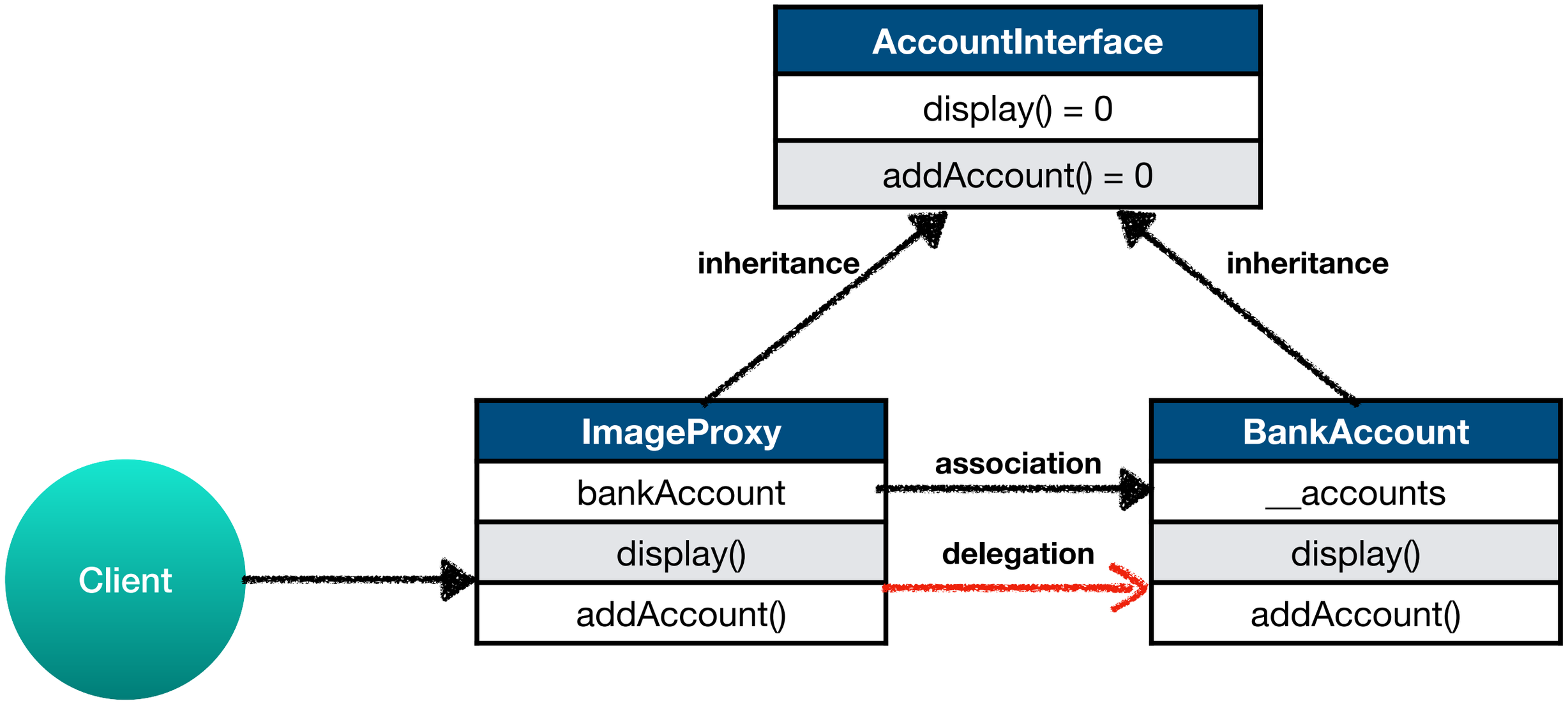
Case Study - Long Process

```
class MathInterface(ABC):
    @abstractmethod
    def getX(self):
        pass

    @abstractmethod
    def getY(self):
        pass

    @abstractmethod
    def getResult(self):
        pass

    @abstractmethod
    def process(self, x, y):
        pass

class MathReal(MathInterface):
    def __init__(self):
        self.__x = None
        self.__y = None
        self.__result = None

    def getX(self):
        return self.__x

    def getY(self):
        return self.__y

    def getResult(self):
        return self.__result

    def process(self, x, y):
        self.__x = x
        self.__y = y
        self.__result = x;

        while y > 0:
            self.__result *= x
            y -= 1
            time.sleep(1)
```

```
class MathProxy(MathInterface):
    def __init__(self):
        self.__inProgress = False
        self.__math = None

    def getX(self):
        if self.__main is None:
            return 0
        else:
            return self.__math.getX()

    def getY(self):
        if self.__main is None:
            return 0
        else:
            return self.__math.getY()

    def getResult(self):
        if self.__math is None:
            return 0
        else:
            return self.__math.getResult()

    def process(self, x, y):
        print("Processing, please wait...")
        self.__inProgress = True

        def execute(x, y):
            temp = MathReal()
            temp.process(x, y)
            self.__math = temp

        t = Thread(target=execute, args=(x,y,))
        t.start()
```

Protection Proxy

- The protection proxy design pattern controls access to a resource according to access privileges.
- To show the proxy model, let's implement a simple protection proxy to display and add accounts. It offers two options:
  - Displaying account list: This operation requires no special privileges.
  - Adding a new account: This operation requires a special secret message from the client.



The example - Protection Proxy

```
class AccountInterface:
    @abstractmethod
    def display(self):
        pass

    @abstractmethod
    def addAccount(self, account):
        pass

class BankAccount(AccountInterface):
    def __init__(self):
        self.__accounts = ['peter', 'tom', 'jim', 'lily']

    def display(self):
        nums = len(self.__accounts)
        print(f"There are {nums} accounts: {' '.join(self.__accounts)}")

    def addAccount(self, account):
        self.__accounts.append(account)
        print(f'Added account {account}')

class AccountProxy(AccountInterface):
    # protection proxy to BankAccount
    def __init__(self):
        self.bankAccount = BankAccount()
        self.secret = 'npuloveyou'

    def display(self):
        self.bankAccount.display()

    def addAccount(self, account):
        sec = input('What is the secret? ')
        self.bankAccount.addAccount(account) if sec == self.secret else print("That's incorrect!")
```

```
def main():
    accountProxy = AccountProxy()
    while True:
        print('1. Display list,  2. Add account,  3. quit')
        option = input('Select option: ')
        if option == '1':
            accountProxy.display()
        elif option == '2':
            name = input('Enter account name: ')
            accountProxy.addAccount(name)
        elif option == '3':
            exit()
        else:
            print(f'unknown option: {option}')
```