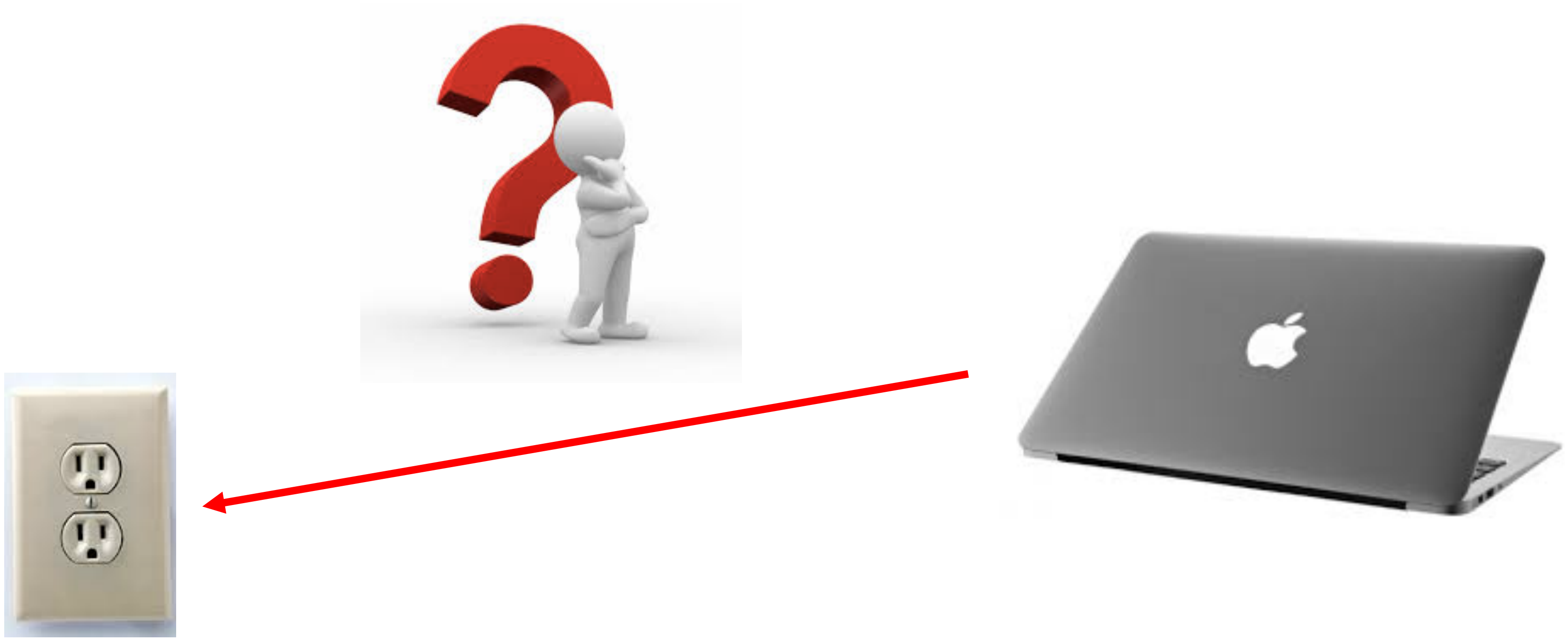
CMPSCI 220 Programming Methodology

09: Adapter Pattern

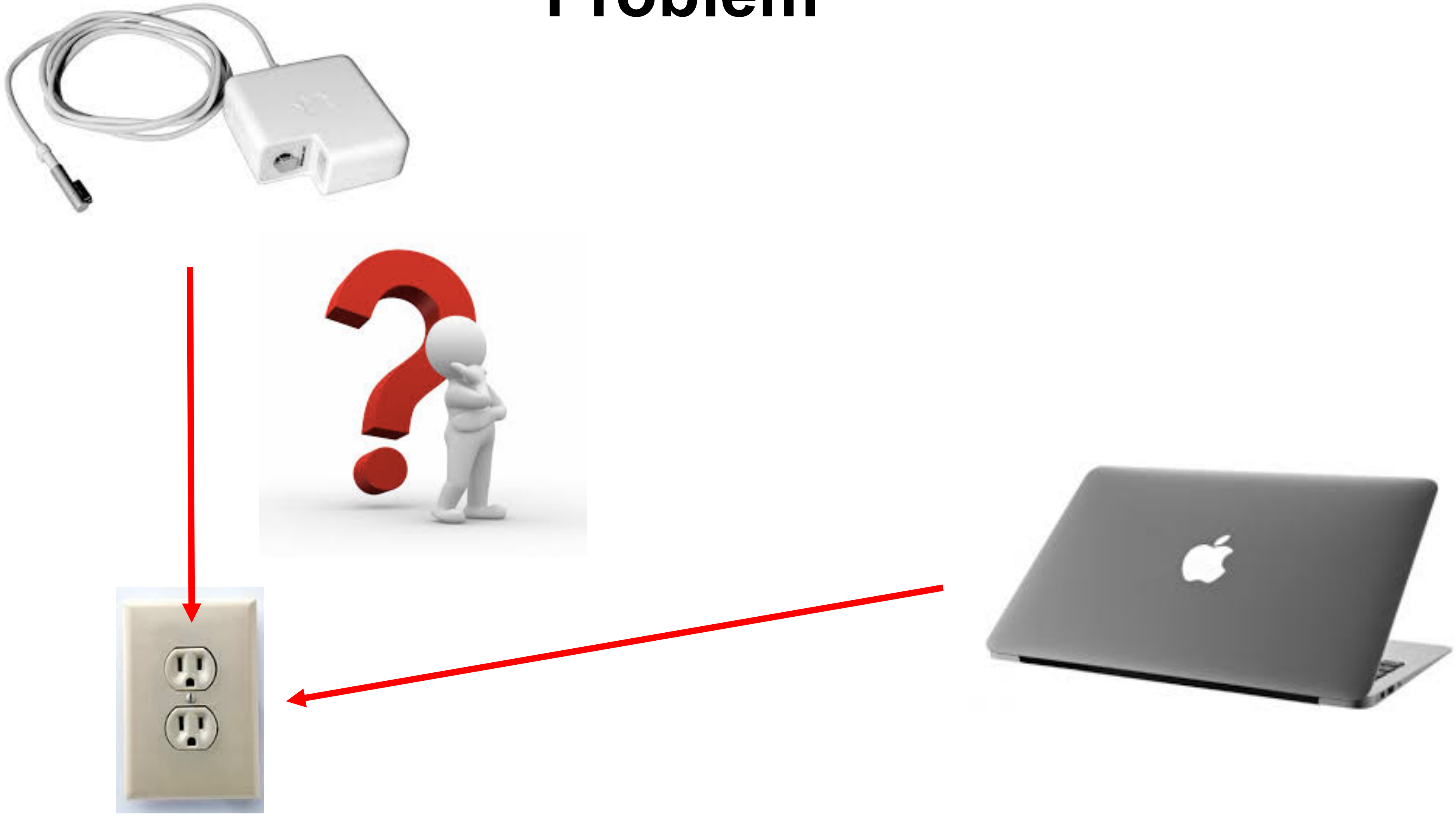
Based on Head First Design Patterns

Problem

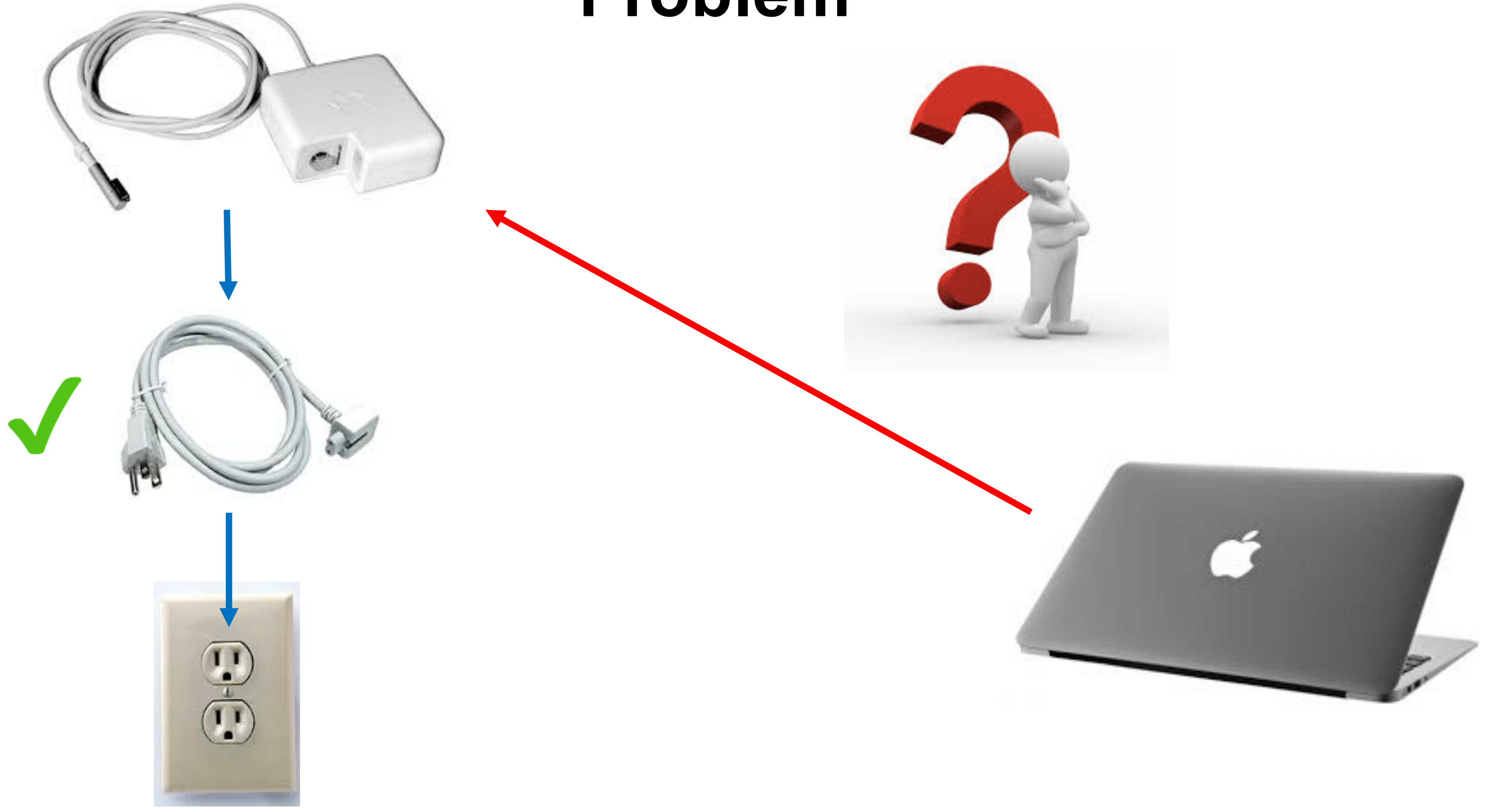
Problem



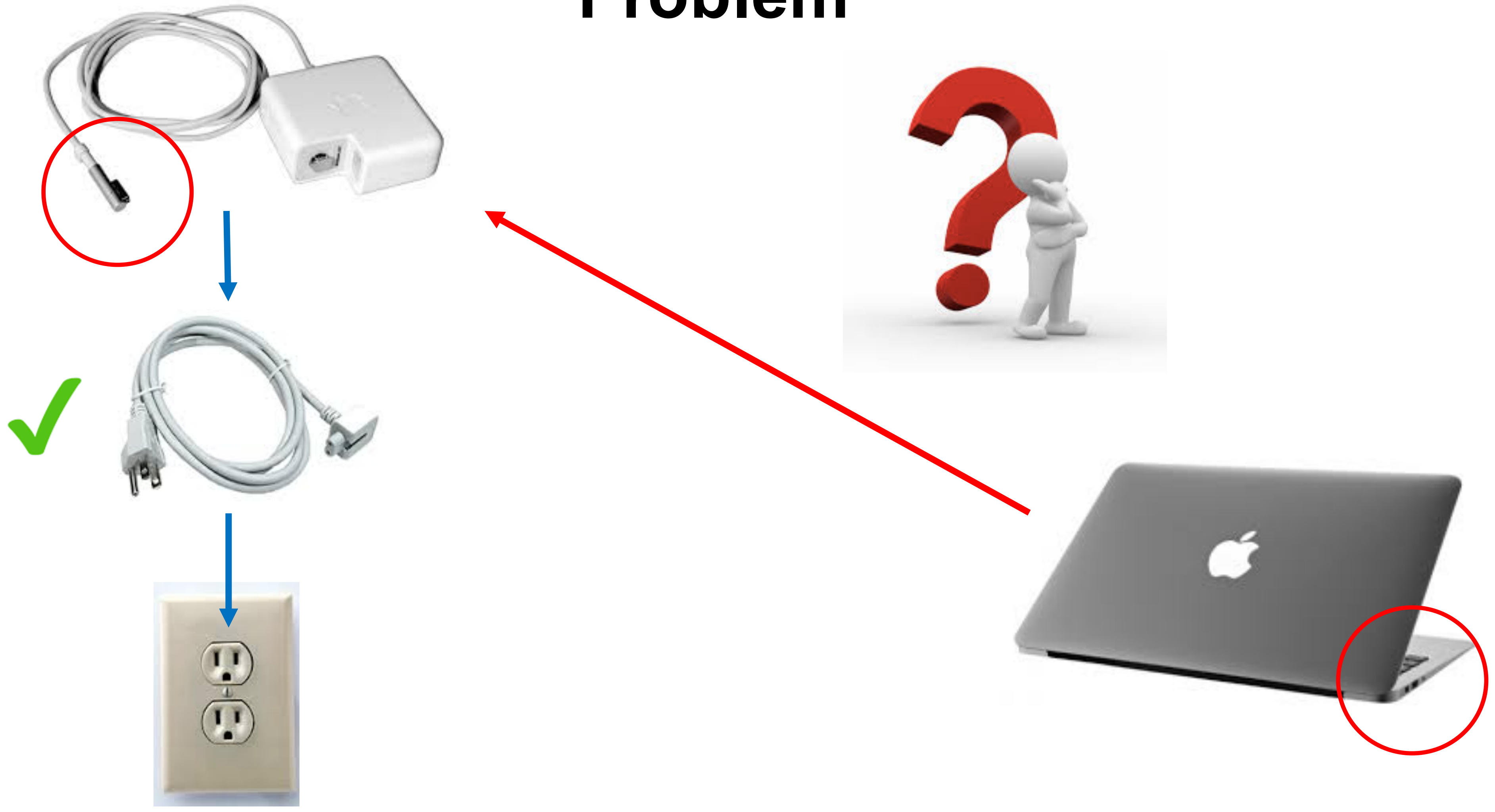
Problem



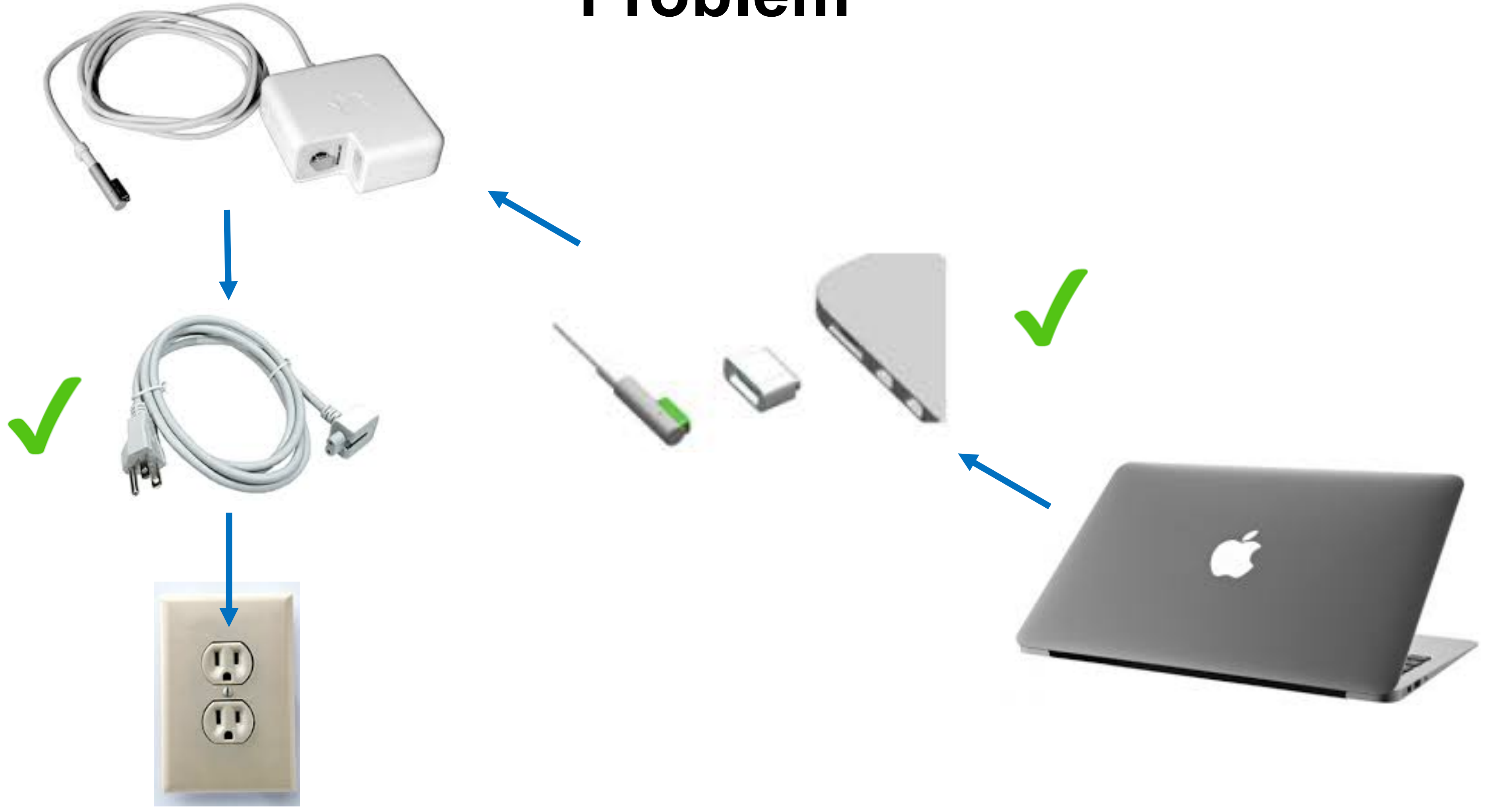
Problem



Problem



Problem

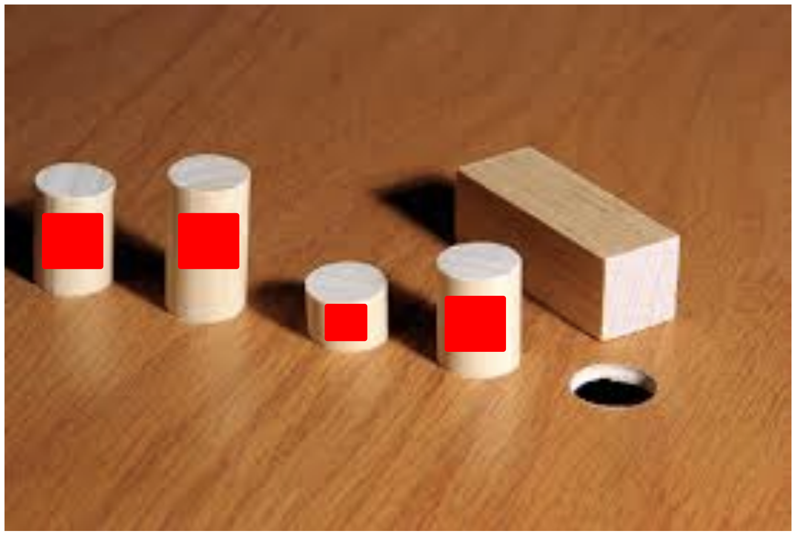


Adapter Pattern

In software design and implementation it is often the case where you need to fit a **square** peg into a **round** hole!

Fortunately, software is easier to manipulate than physical objects!

Adapter Pattern



The **adapter pattern** is a technique that wraps objects to given them *new responsibilities*.

We want to *adapt* a design expecting one interface to a class that implements a different interface!

Objectives

Adapter Pattern

- Learn how objects can adapt to different designs.
- Apply the adapter pattern in Scala.

i-clicker

What is the primary purpose of the observer pattern?

a) To tightly couple objects to other objects.
b) To abstract duplicate code into separate classes/objects.
c) To loosely couple objects to other objects.
d) To abstract algorithms into objects for composition.
e) None of these.

i-clicker

What is the primary purpose of the command pattern?

a) To allow for continual updates to related classes.
b) To eliminate code duplication.
c) To parameterize a request with different arguments.
d) To encapsulate a request into a separate object.
e) To dictate the implementation of subclasses.

i-clicker

What is the primary purpose of the factory pattern?

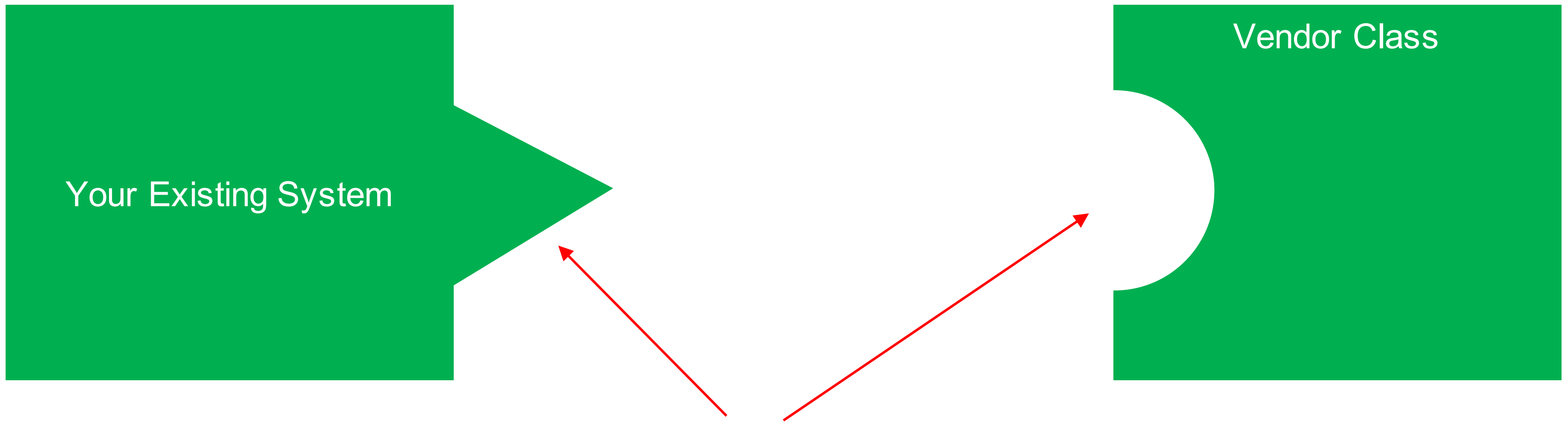
a) To allow for continual updates to related classes.
b) To decouple implementation from creation of objects.
c) To parameterize an object with different types.
d) To encapsulate an algorithm into a separate object.
e) To dictate the implementation of subclasses.

The Adapter Pattern

Converts the interface of a class into another interface the clients expect.

Adapter lets classes work together that couldn’t otherwise because of incompatible interfaces.

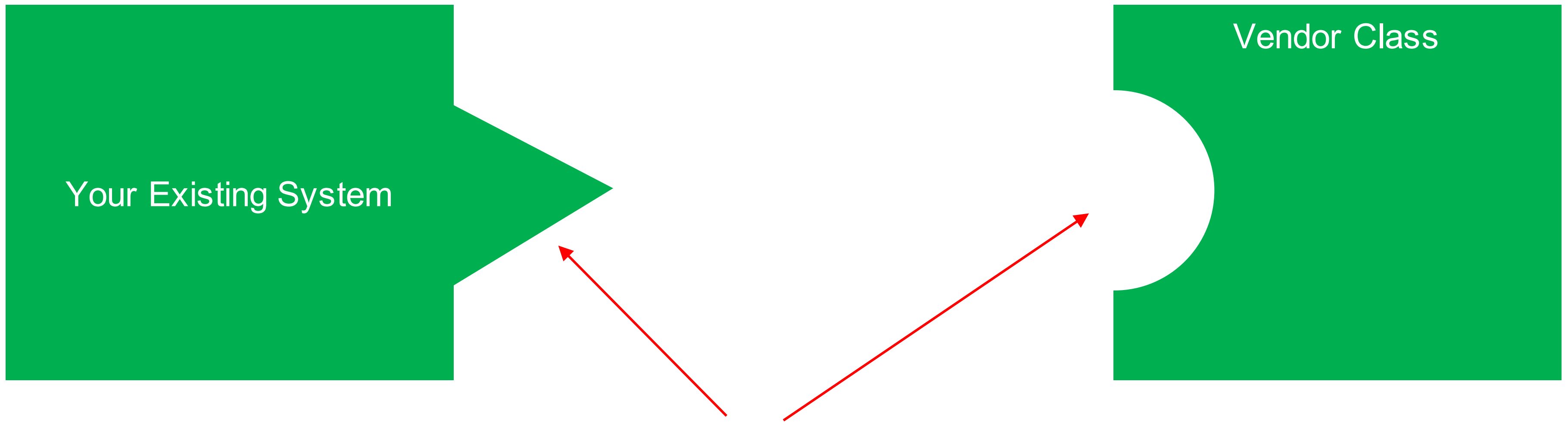
Object-Oriented Adapters



Their interface does not match the one you've written your code against!

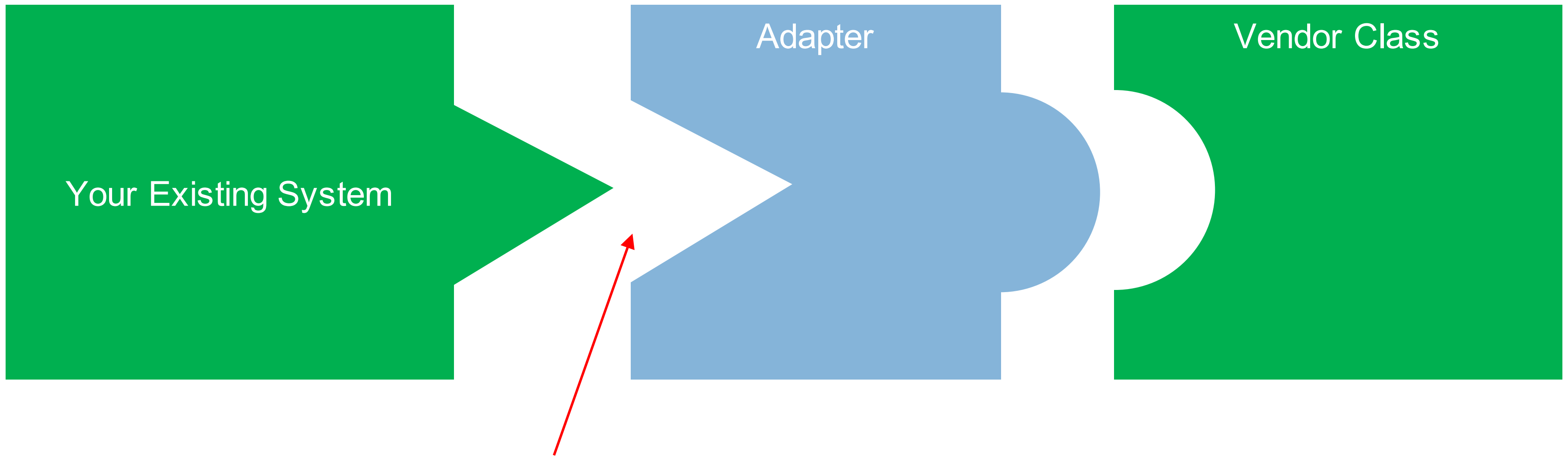
Object-Oriented Adapters

You certainly do not want to rewrite your system to conform to their interface. So, what do you do?



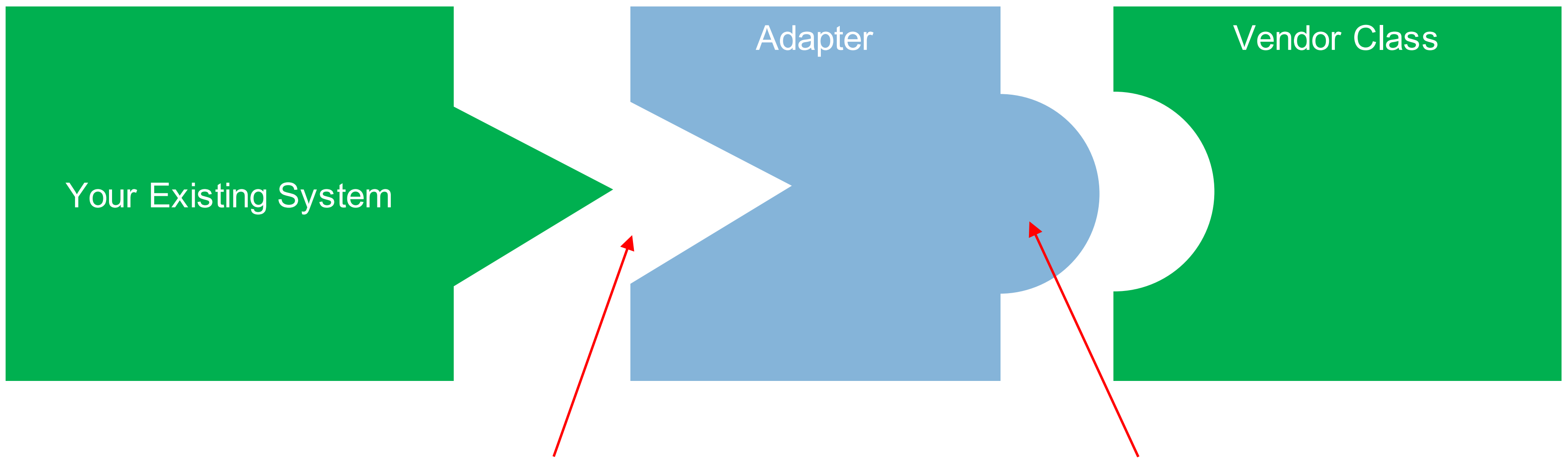
Their interface does not match the one you've written your code against!

Object-Oriented Adapters



The **adapter** implements the interface your classes expect.

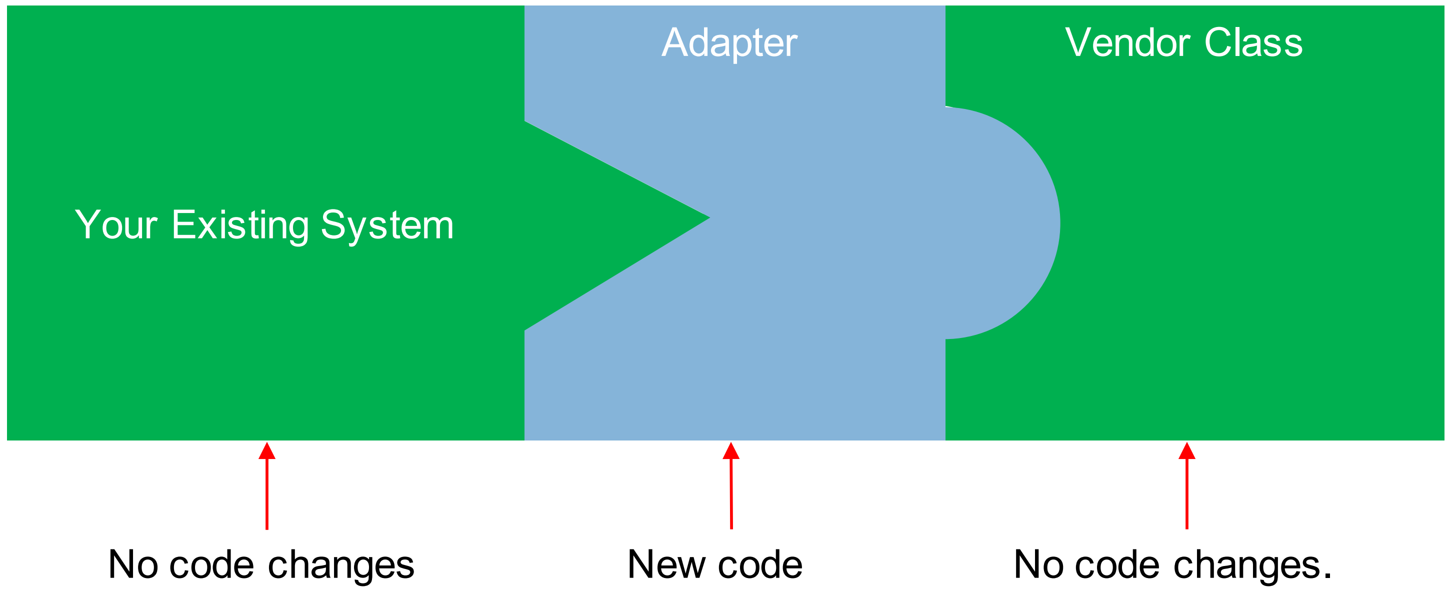
Object-Oriented Adapters



The **adapter** implements the interface your classes expect.

And, talks to the vendor interface to service your requests.

Object-Oriented Adapters



Ducks and Turkeys!

We have received a request from our technical lead that we need to extend the SimUDuck game with support for interfacing to 3rd party libraries!

Unfortunately, each library only works on either Ducks or Turkeys.

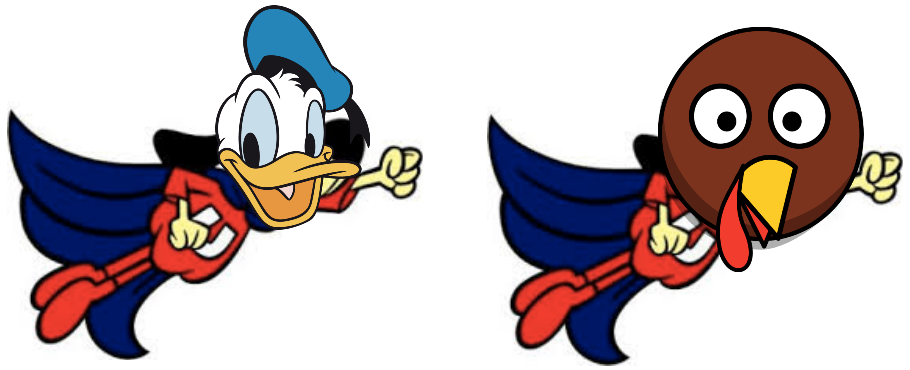
Ducks and Turkeys!

We have received a request from our technical lead that we need to extend the SimUDuck game with support for interfacing to 3rd party libraries!

Unfortunately, each library only works on either Ducks or Turkeys.

Adapters to the rescue!

Let us look at some code…



Question?

How much “adapting” does an adapter need to do? It seems like if I need to implement a large interface, I could have a LOT of work on my hands.

Question?

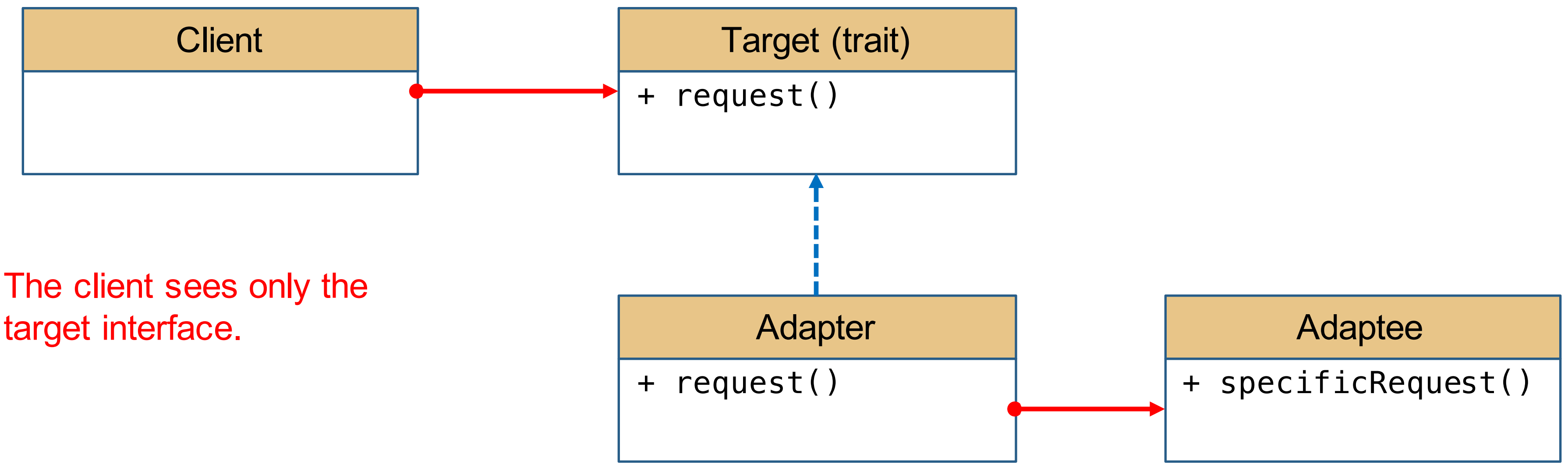
Does an adapter always wrap one and only one class?

Question?

What if I have old and new parts of my system, the old parts expect the old vendor interface, but we have already written the new parts to use the new vendor interface?

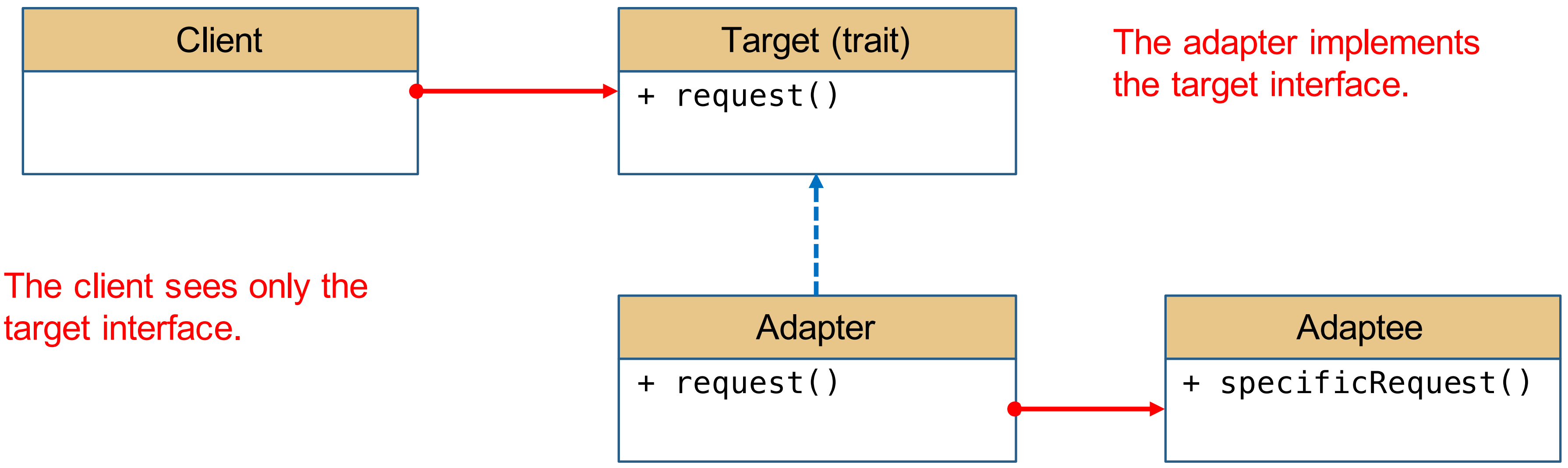
It is going to get confusing using an adapter here and the unwrapped interface there. Wouldn’t I be better off just rewriting my older code forgetting the adapter?

Adapter: Class Diagram



The client sees only the target interface.

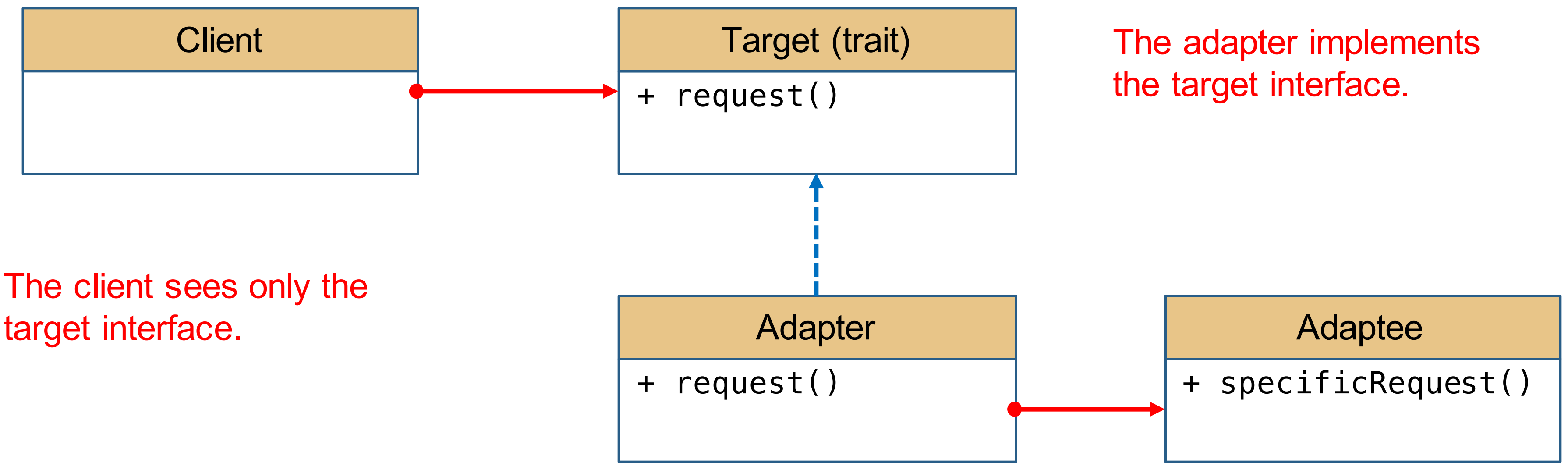
Adapter: Class Diagram



The adapter implements the target interface.

The client sees only the target interface.

Adapter: Class Diagram

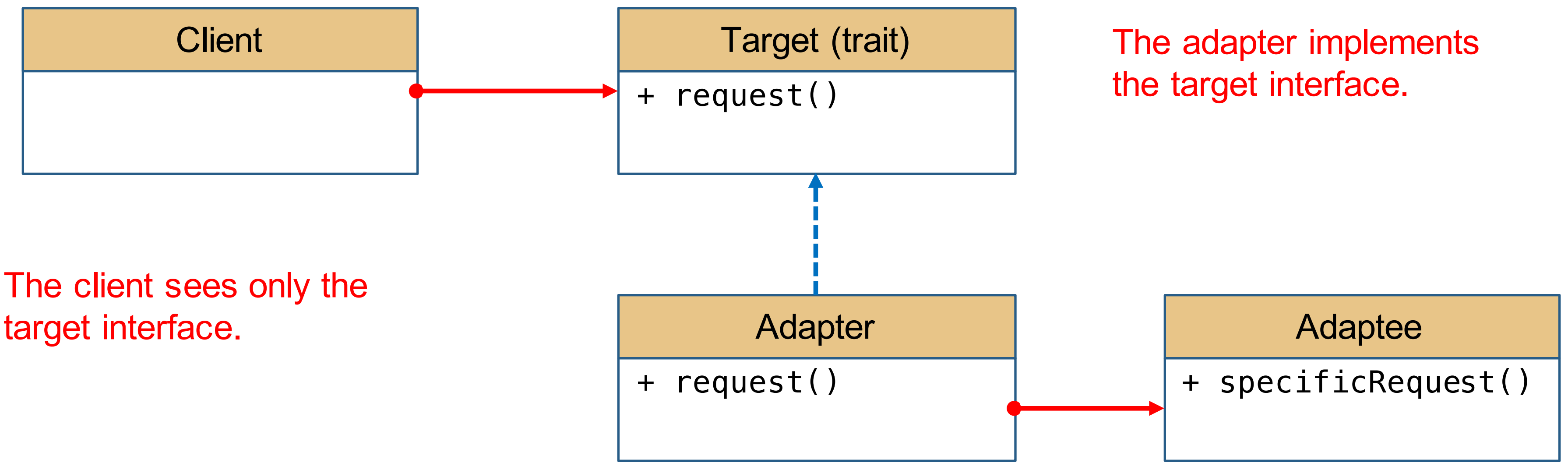


The adapter implements the target interface.

The client sees only the target interface.

The adapter is composed with the adaptee.

Adapter: Class Diagram



The adapter implements the target interface.

The client sees only the target interface.

The adapter is composed with the adaptee.

All requests get delegated to the adaptee.

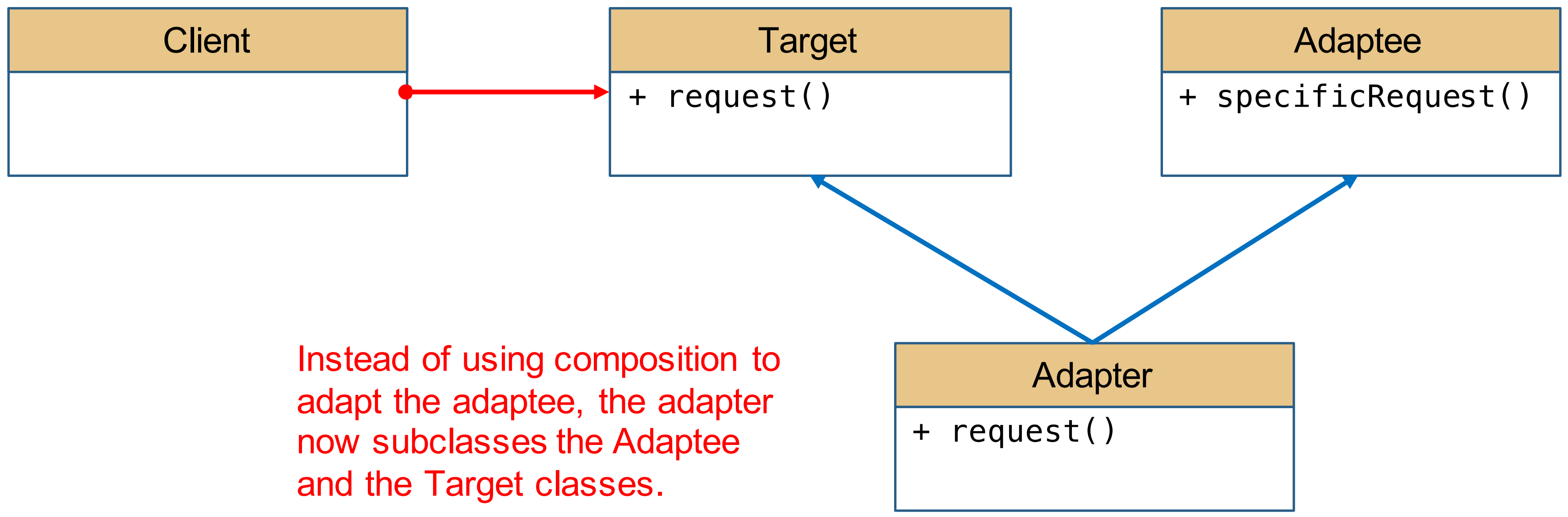
Object vs Class Adapters

It turns out there are two kinds of adapters…

- object adapters: use composition
- class adapters: use multiple inheritance

The first is supported by all OO languages.
The second, requires language support.

Object Adapters



Instead of using composition to adapt the adaptee, the adapter now subclasses the Adaptee and the Target classes.

Scala Traits

Scala, does not have multiple inheritance (The diamond problem). However, it does have traits that can provide implementation (Java 8 now has this as well).

Scala uses *linearization* to make this work… Let us look at some code…