

1. Regress: Modularity
2. Proxy Pattern

Lect 24
17-10-2023

Modularity

- Modularity is a fundamental attributes of any good design.
 - Defined in terms of cohesion and coupling.
 - Normally covered in First level Software Engineering subject
 - We are revisiting the concepts based on requests from many of you.

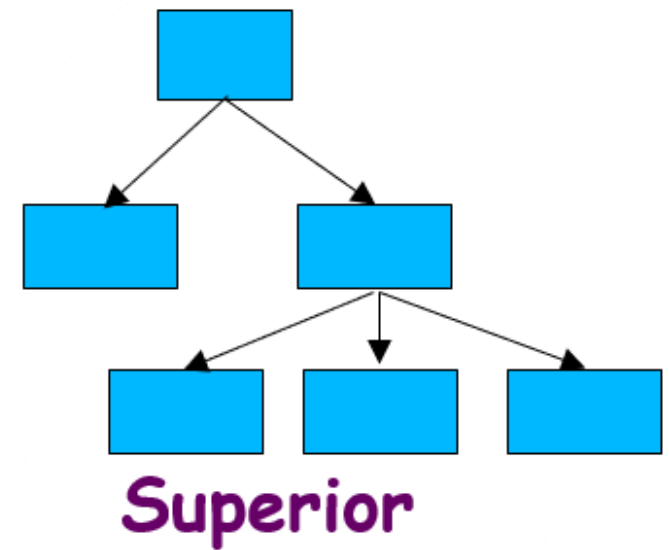
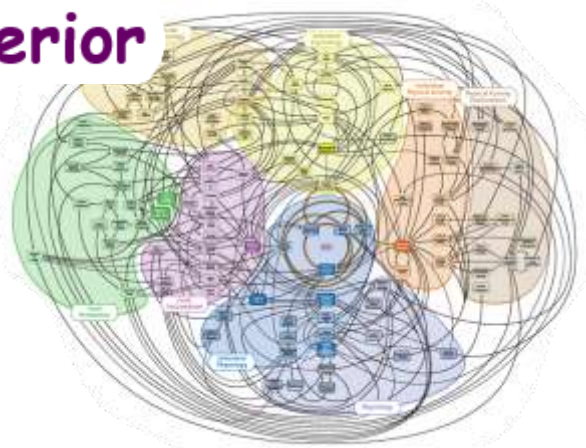
Modularity

- Modularity is a fundamental attributes of any good design.
 - Measures how effectively a design has been decomposed into a set of modules:
 - Modules should be almost independent of each other
 - Based on **divide and conquer principle**.

Modularity: Intuitive Explanation

- If modules are independent:
 - Each module can be understood separately,
 - reduces complexity greatly.

Inferior



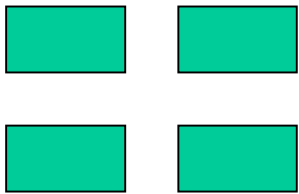
Modularity

- In technical terms, modules in a design have:
 - high cohesion
 - low coupling.
- We next discuss:
 - cohesion and coupling.

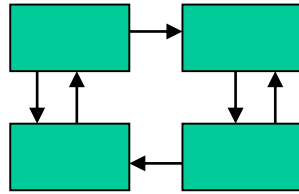
Modularity

- Arrangement of modules in a hierarchy ensures:
 - Low fan-out
 - Abstraction

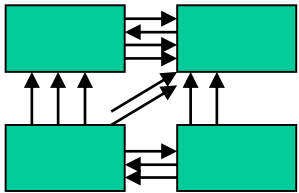
Coupling: Degree of dependence among components



No dependencies



Loosely coupled-some dependencies



Highly coupled-many dependencies

High coupling makes modifying parts of the system difficult, e.g., modifying a component affects all the components to which the component is connected.

Source:

Pfleeger, S., *Software Engineering Theory and Practice*. Prentice Hall, 2001.

Cohesion and Coupling

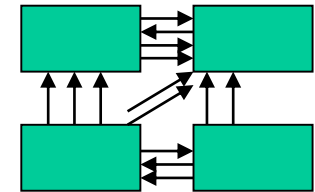
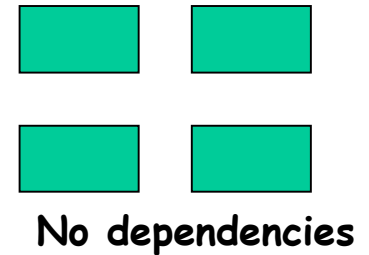
- Cohesion is a measure of:
 - functional strength of a module.
 - **A cohesive module performs a single task or function.**
- Coupling between two modules:
 - **A measure of the degree of interdependence or interaction between the two modules.**

Cohesion and Coupling

- A module having **high cohesion and low coupling**:
 - Called functionally independent of other modules:
- A functionally independent module:
 - Needs very little help from other modules during execution and therefore has minimal interaction with other modules.

Advantages of Functional Independence

- Better understandability
- Complexity of design is reduced,
- Different modules easily understood in isolation:
 - Modules are independent
- Also: Reuse of modules is possible.



Measuring Functional Independence

- Unfortunately, there are no ways available:
 - To quantitatively measure the degree of cohesion and coupling:
 - At present classification of different kinds of cohesion and coupling:
- If we can classify a module with respect to its cohesion and coupling:
 - It will give us some rough idea regarding the degree of cohesiveness of a module.

Classification of Cohesiveness

- Classification usually has scope for ambiguity and incorrectness:
 - yet gives us some rough idea about cohesiveness of a module.
- By examining the type of cohesion exhibited by a module:
 - we can roughly tell whether it displays high cohesion or low cohesion.

Classification of Cohesiveness

functional

sequential

communicational

procedural

temporal

logical

coincidental

Degree of
cohesion



Coincidental cohesion

- The module performs a set of tasks:
 - which relate to each other very loosely, if at all.
- That is, the module contains a random collection of functions.
- functions have been put in the module out of pure coincidence without any thought or design.

Coincidental Cohesion - example

Module AAA{

Print-inventory();

Register-Student();

Issue-Book();

};

Logical cohesion

- All elements of the module perform similar operations:
 - e.g. error handling, data input, data output, etc.
- An example of logical cohesion:
 - a set of print functions to generate an output report arranged into a single module.

Logical Cohesion

```
module print{  
    void print-grades(student-file){ ...}  
  
    void print-certificates(student-file){...}  
  
    void print-salary(teacher-file){...}  
}
```

Temporal cohesion

- The module contains functions so that:
 - all the functions must be executed in the same time span.
- Example:
 - The set of functions responsible for
 - initialization,
 - start-up, shut-down of some process, etc.

```
init() {  
    Check-memory();  
    Check-Hard-disk();  
    Initialize-Ports();  
    Display-Login-Screen();  
}
```

Temporal Cohesion - Example

Procedural cohesion

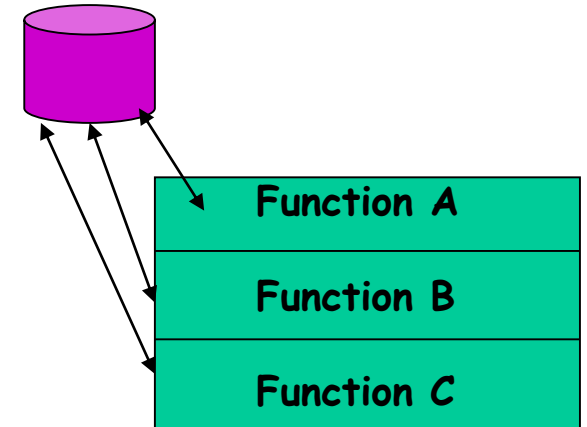
- The set of functions of the module:
 - all part of a procedure (algorithm)
 - certain sequence of steps have to be carried out in a certain order for achieving an objective,
 - e.g. the algorithm for decoding a message.

Communicational cohesion

- All functions of the module:
 - Reference or update the same data structure.
- **Example:**
 - The set of functions defined on an array or a stack.

Communicational Cohesion

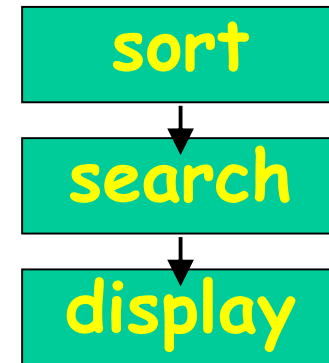
```
handle-Student- Data() {  
    Static Struct Student-data[10000];  
    Store-student-data();  
    Search-Student-data();  
    Print-all-students();  
};
```



Communicational
Access same data

Sequential cohesion

- Elements of a module form different parts of a sequence,
 - output from one element of the sequence is input to the next.
 - Example:



Functional cohesion

- Different elements of a module cooperate:
 - to achieve a single function,
 - e.g. managing an employee's pay-roll.
- When a module displays functional cohesion,
 - we can describe the function using a single sentence.

Determining Cohesiveness

- Write down a sentence to describe the function of the module
 - If the sentence is compound,
 - it has a sequential or communicational cohesion.
 - If it has words like "first", "next", "after", "then", etc.
 - it has sequential or temporal cohesion.
 - If it has words like initialize,
 - it probably has temporal cohesion.

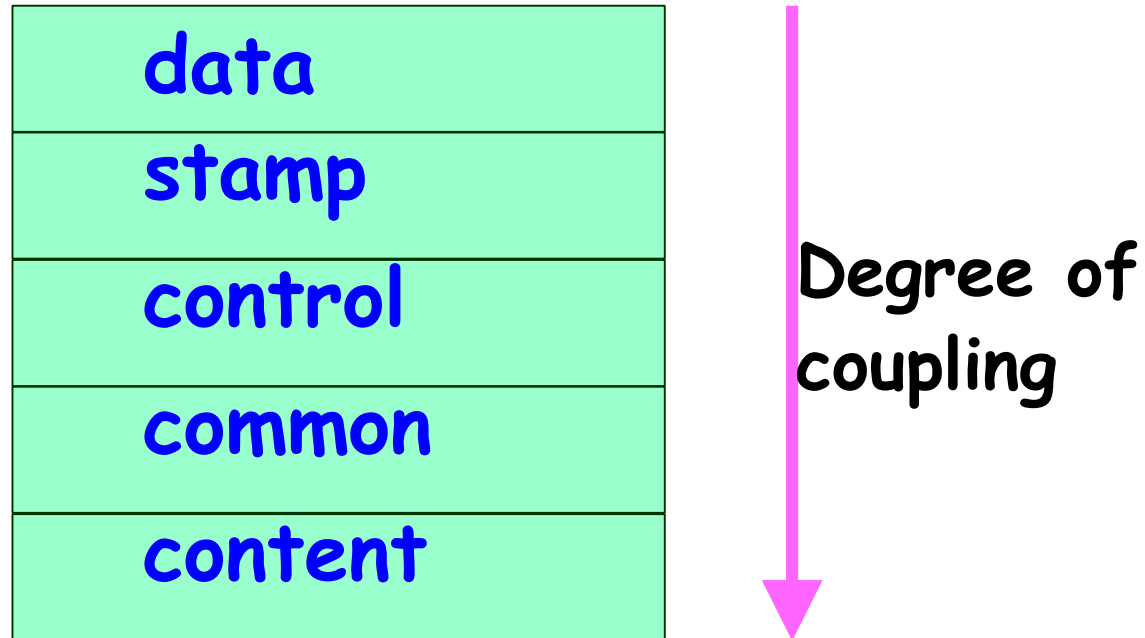
Coupling

- Coupling indicates:
 - How closely two modules interact or how interdependent they are.
 - The degree of coupling between two modules depends on their interface complexity.

Coupling

- There are no ways to precisely measure coupling between two modules:
 - classification of different types of coupling will help us to approximately estimate the degree of coupling between two modules.
- Five types of coupling can exist between any two modules.

Classes of coupling



Data coupling

- Two modules are data coupled,
 - if they communicate via a parameter:
 - that is an elementary data item,
 - e.g an integer, a float, a character variable.
 - The data item should be problem related:
 - not used for control purpose.

Stamp coupling

- Two modules are stamp coupled,
 - if they communicate via a composite data item
 - or an array or structure in C.

Control coupling

- Data from one module is used to direct
 - order of instruction execution in another.
- Example of control coupling:
 - a flag set in one module and tested in another module.

Common Coupling

- Two modules are common coupled,
 - if they share some global data.

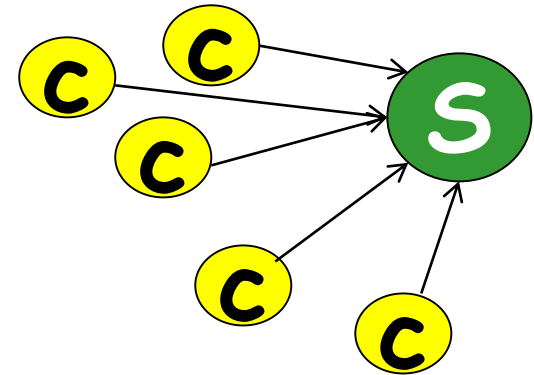
Content coupling

- Content coupling exists between two modules:
 - if they share code,
 - e.g, branching from one module into another module.
- The degree of coupling increases
 - from data coupling to content coupling.

Proxy Pattern

Proxy (Surrogate) Pattern

- **Problem:** How should a client invoke the services of a server when access to the server should be managed in some way?
- **Solution:** A proxy object should be created at the client side.
- **Explanation:** Manage service
 - Authenticate
 - Hide details of network operations.
 - Determine server address,
 - Communicate with the server, obtain server response and seamlessly pass that to the client, etc.



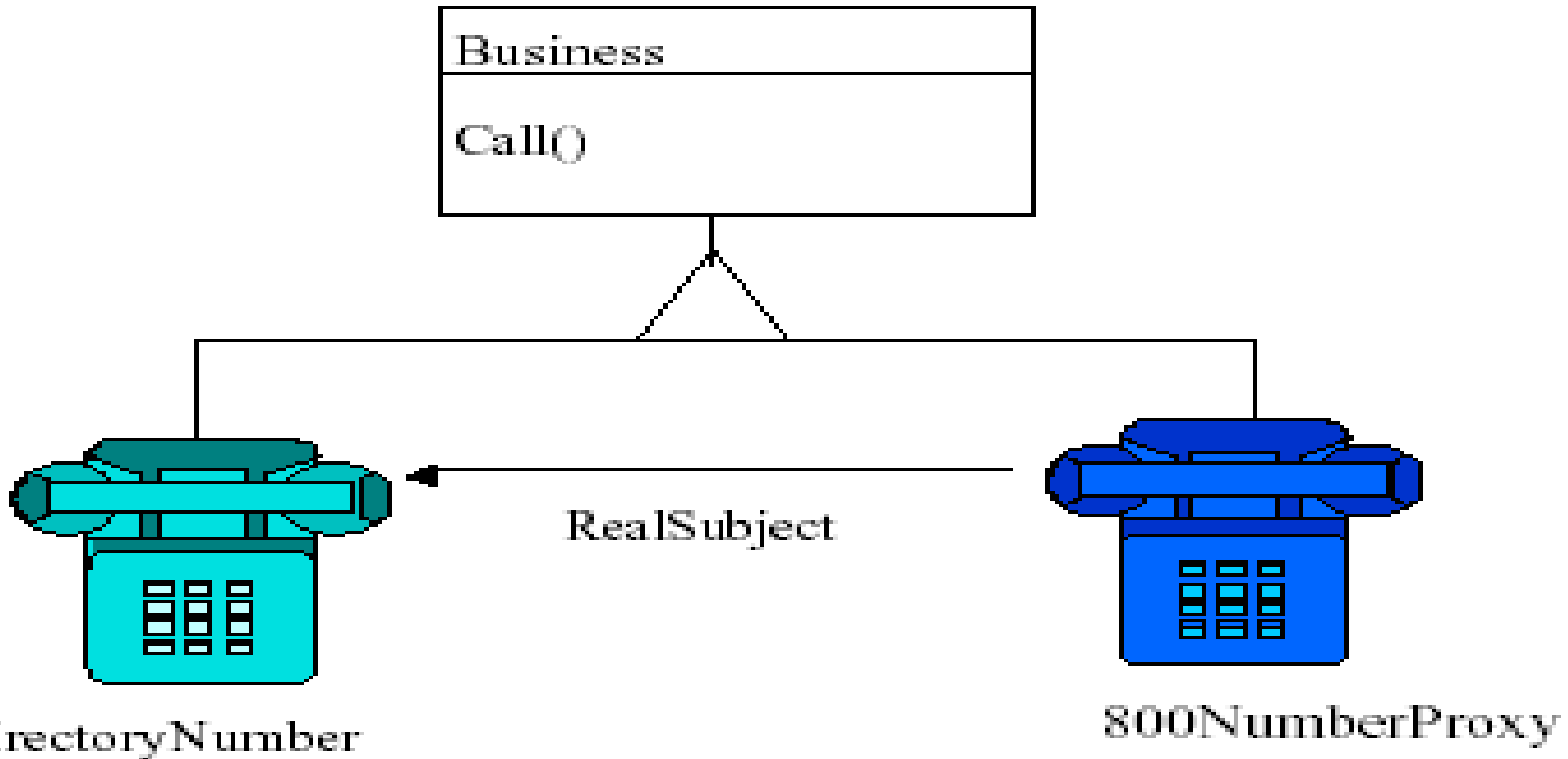
Proxy: Non-software example 1



- To control access to an object:
 - Provide a surrogate for the object

Proxy: Non-software example 2

Toll-free numbers



Client dials the "800" std code just like any other number
Proxy records billing information and connects to the
nearest call center.

Non-software Example 3: Payment By Cheque

- **Problem:**

A person has cash in his bank account but does not carry cash with him as it is cumbersome.
Wants to make purchase...

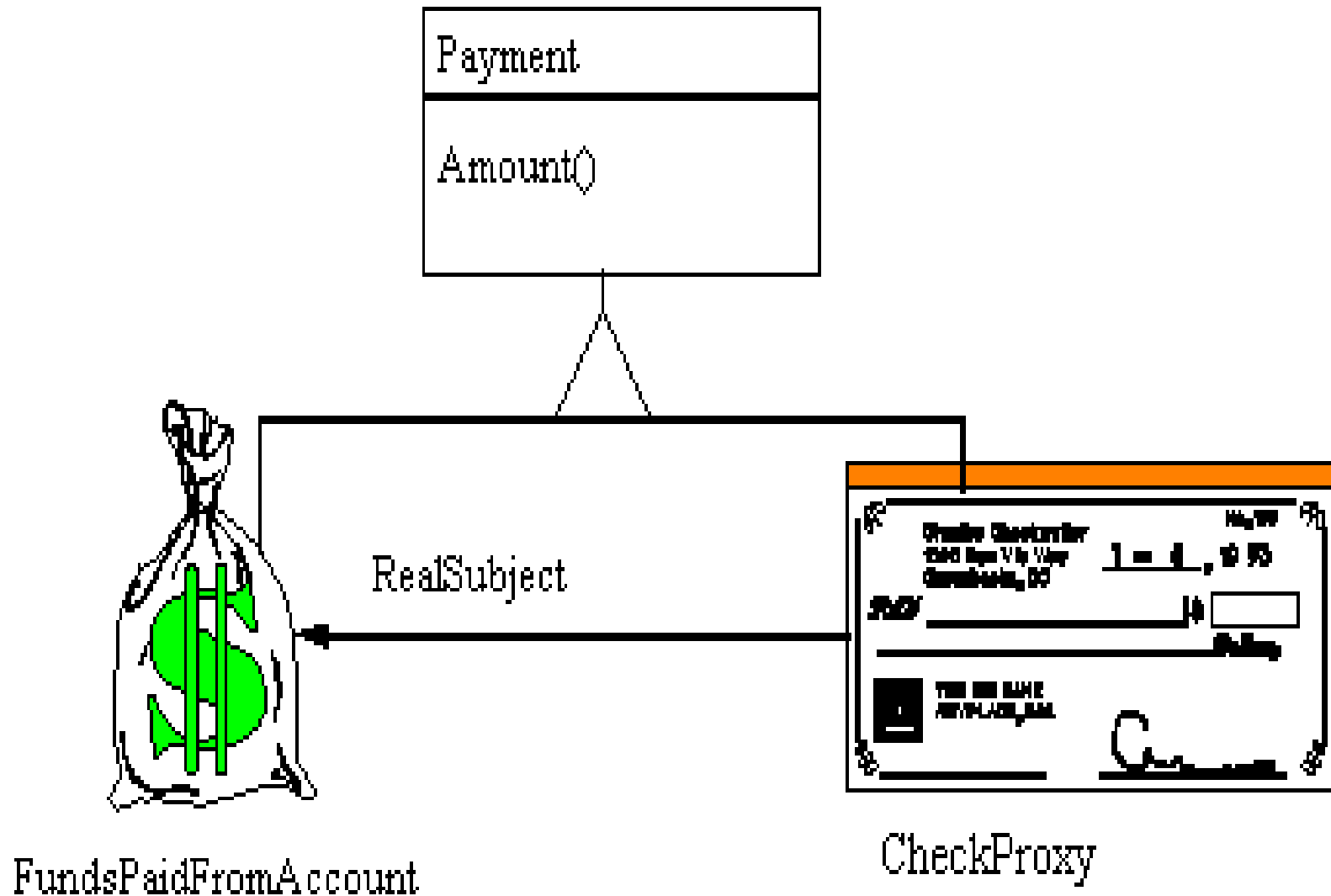
- **Solution:**

Proxy.

Payment by Cheque Example

- The cheque acts as a proxy for the fund in the account.
- The fund is the real subject.
 - The client asks for payment through a Cheque.
- The proxy (cheque) behaves:
 - As if it was actual fund.

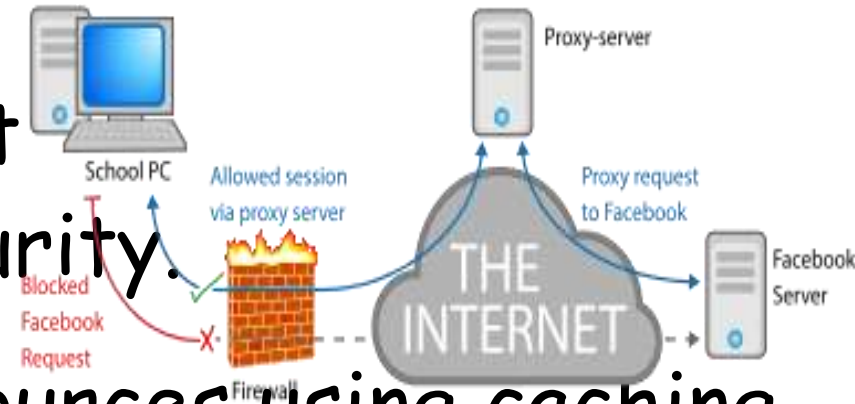
Proxy Design Pattern



Digression: Network Proxy Server

- What is the role of a network proxy server?

- To keep machines behind it anonymous, mainly for security.
- To speed up access to resources using caching.
- To prevent downloading the same content multiple times and save bandwidth.
- To log / audit usage, e.g. to provide company employee Internet usage reporting.
- Firewall and filtering: Scan for malware



- **Encryption / SSL acceleration:** Secure Sockets Layer (SSL) encryption is often not done by the web browser itself, but by proxy that is equipped with SSL acceleration hardware.
- **Load balancing:** can distribute the load to several web servers, each web server serving its own application area.
- **Compression:** proxy server can optimize and compress the content to speed up the load time.
- **Spoon feeding:** reduces resource usage caused by slow clients by caching the content the web server sent and slowly "spoon feeding" it to the client.⁴²

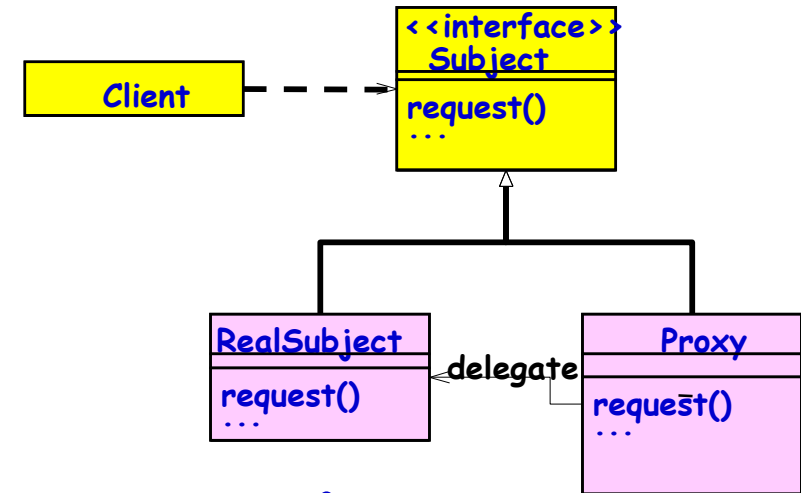
Proxy: Some Insights

- Use Proxy pattern whenever the services provided by a supplier need to be managed in some way without disturbing the supplier.
- **Example:** You require some additional conditions to be satisfied before the actual object is accessed:
 - Consider loading an image from disk only when it is actually needed.



Proxy Pattern

- The proxy object has the same interface as the target object:



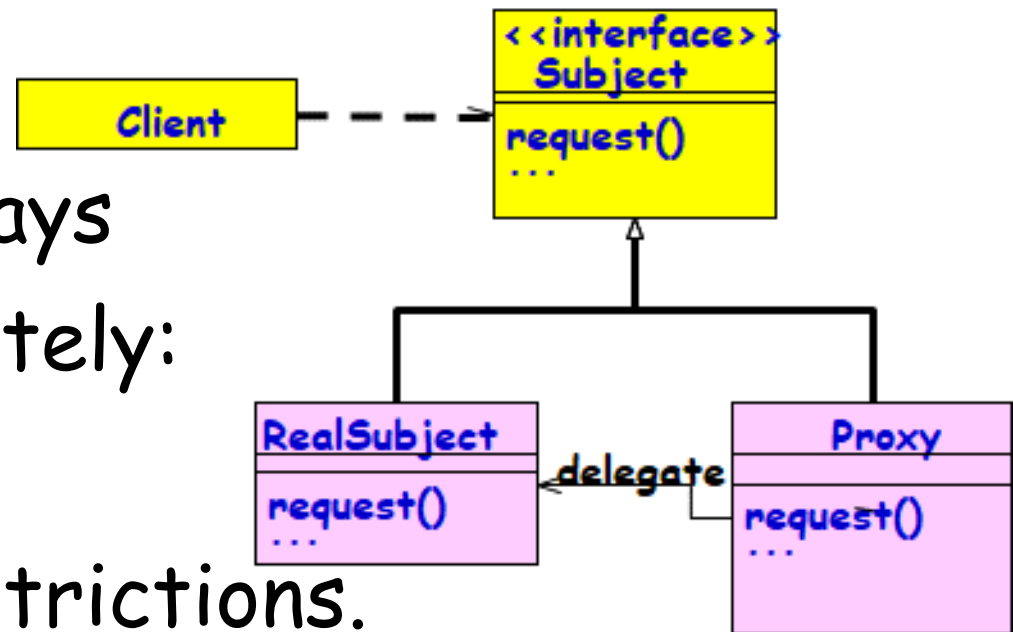
- Proxy stores a reference to the target object...
- Forwards (delegates) requests to it.
- Often more sophistication needed than just a simple method call to an object:
 - That is, we want to wrap code around the references to an object

Proxy: Another Wrapper Pattern

- **A Structural Pattern:**

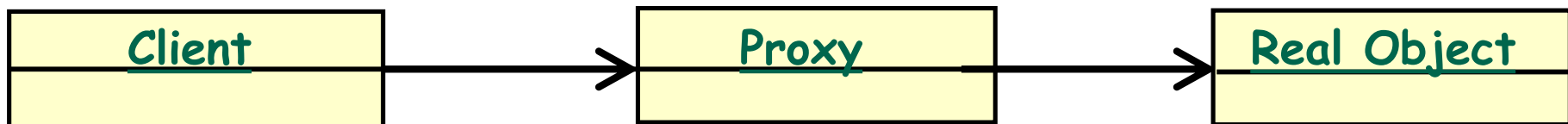
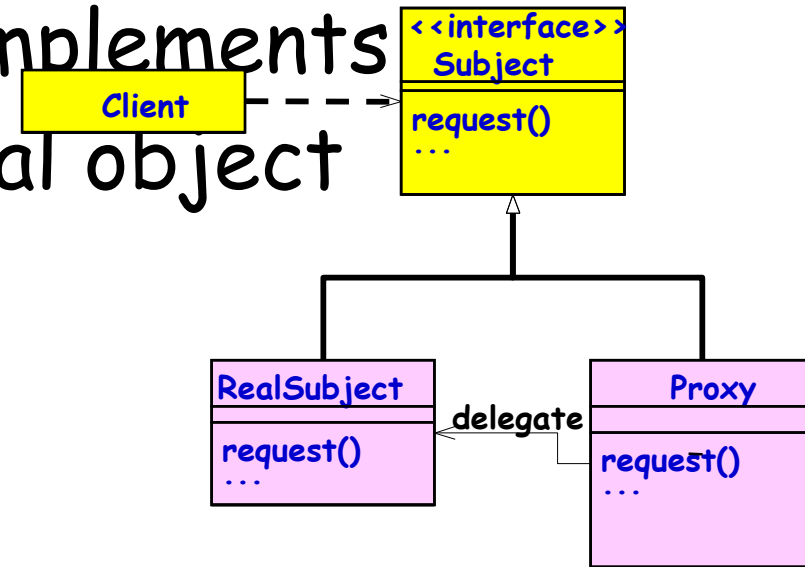
- Provides surrogate for some object,
- Controls access to real target object.

- Real target may not always be instantiated immediately:
 - Due to performance, location, or access restrictions.



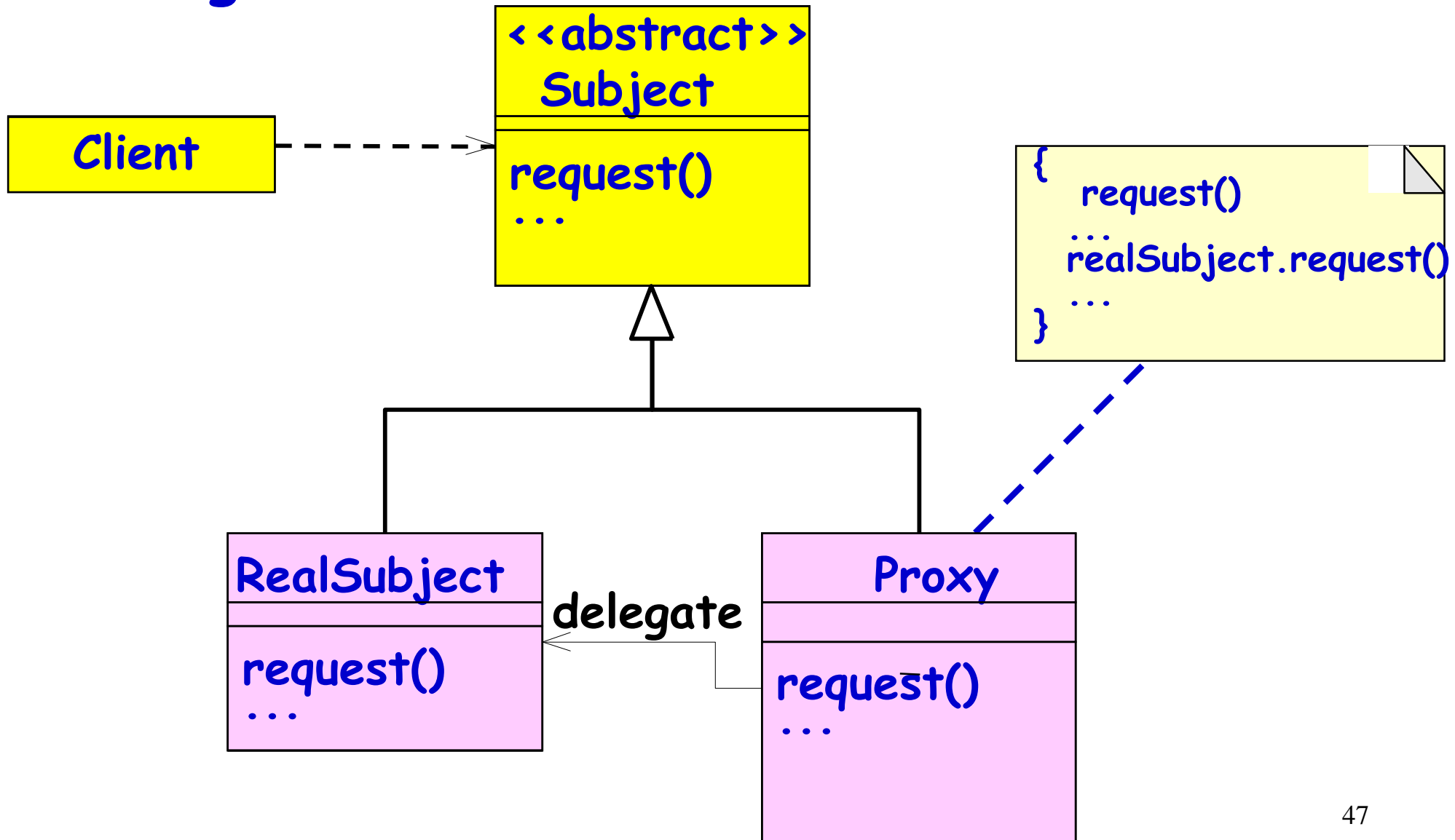
Proxy Solution

- Create a Proxy object that implements the same interface as the real object
- The Proxy object contains a reference to the real object
- Clients are given a reference to the Proxy:
 - Not the real object
- All clients invoke operations on the Proxy:
 - Allows the Proxy to perform additional processing



Proxy Structure

Class Diagram



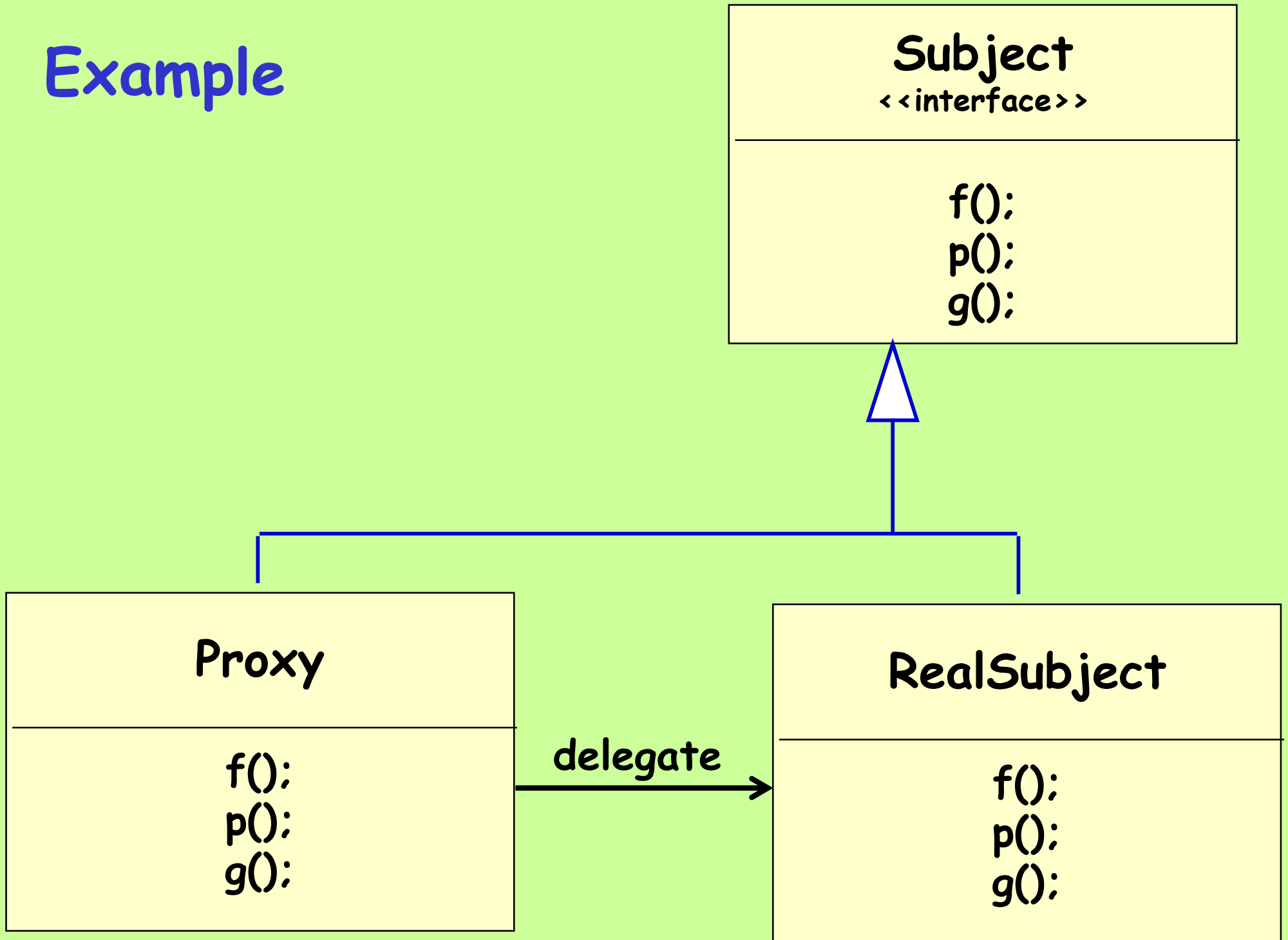
Proxy: Class Structure

- Three Classes: **Subject**, **RealSubject**, and **Proxy**.
- **Interface Subject:**
 - Implemented both by **RealSubject**, and **Proxy**.
- **Proxy:**
 - Delegates any calls to **RealSubject**.
- **Client:**
 - Always uses **Proxy**.

Proxy Usage 1: Helps Save Expensive Steps

- What is expensive?
 - Heavy weight object Creation
 - Object Initialization
- Helps defer object creation and initialization to the time we actually need the object.
- Proxy pattern:
 - Helps reduce the cost of accessing objects
 - The proxy object acts as a stand-in for the real object
 - The proxy creates the real object only if the user asks for it...

Example



Client Code:

```
Proxy p = new Proxy();  
public void test() { p.f();    p.g();    p.h(); }
```

Proxy Code

```
interface Subject { void f(); void g(); void h();}
```

```
class Proxy implements Subject {
```

```
    private Subject implementation;
```

```
    public Proxy() { implementation = new RealSubject(); }
```

```
    public void f() {implementation.f();}
```

```
    public void g() {implementation.g();}
```

```
    public void h() {implementation.h();}
```

```
}
```

```
class RealSubject implements Subject {
```

```
    public void f() {System.out.println("Implementation.f()");}
```

```
    public void g() {System.out.println("Implementation.g()");}
```

```
    public void h() {System.out.println("Implementation.h()");}
```

```
}
```

Proxy Design Pattern

Client

Instantiate with
Proxy object

**<<abstract>>
Subject
expensiveMethod()
anotherMethod()**

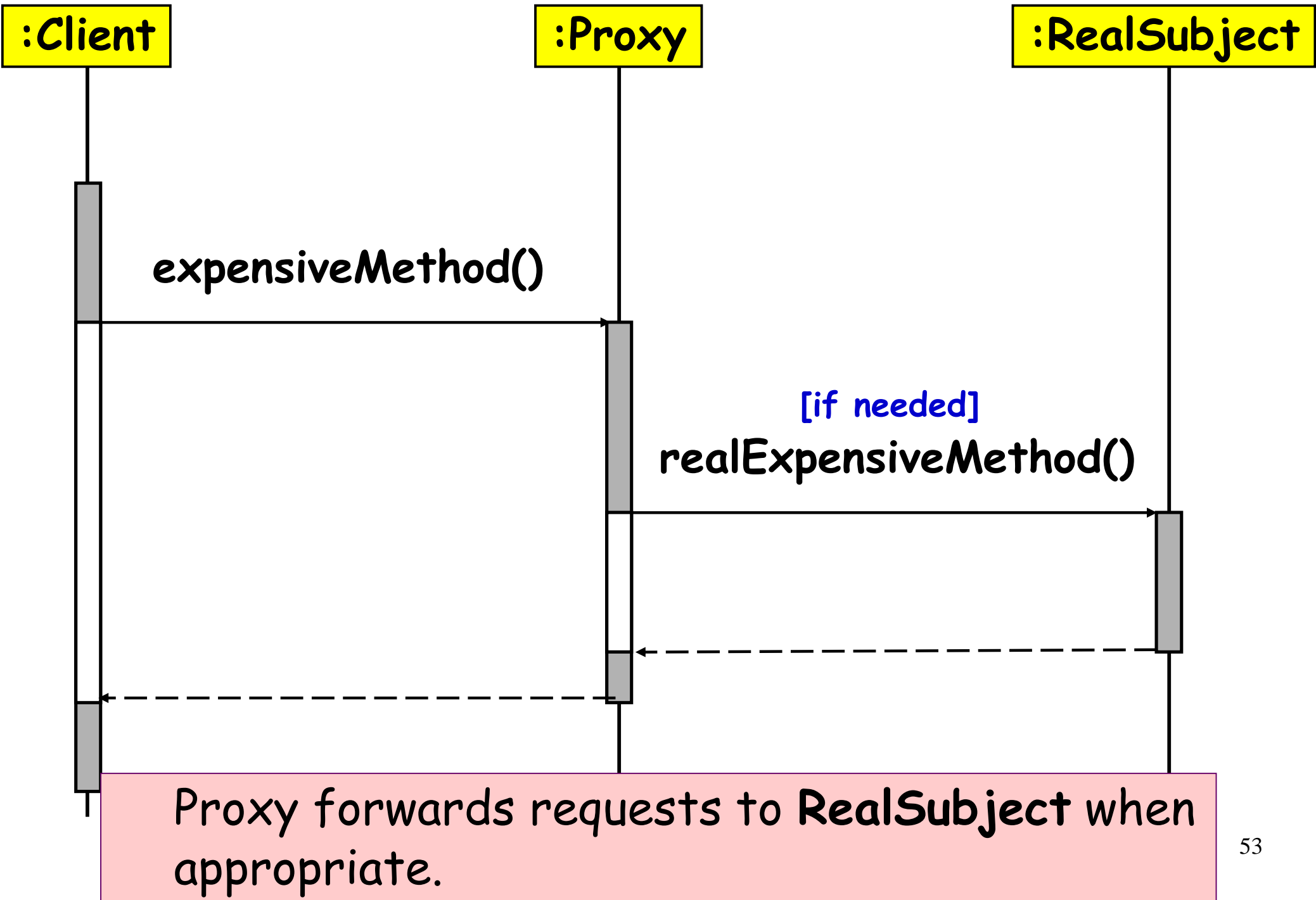
**RealSubject
expensiveMethod()**

realActiveObject

**Proxy
expensiveMethod()
anotherMethod()**

```
... // Check if it is really needed:  
if ( realActiveObject == null )           // never referenced  
{  
    realActiveObject = getRealActiveObject();  
    realActiveObject.expensiveMethod();  
}  
else // try to avoid calling the real expensiveMethod()
```

Sequence Diagram for Proxy



Kinds of proxies

- **Virtual proxy:**
 - Delays the creation or loading of large or computationally expensive objects (**lazy construction**)
 - Proxy is a standin --- **postpones accessing the real subject.**
- **Remote proxy:**
 - Use a local representative for a remote object (different address space)
 - **Hides the fact that an object is not local**
 - Encodes and sends the request to the real subject in a different address space.

Kinds of Proxies cont...

- **Synchronization Proxy**

- Controls access to a target object when multiple objects access it.

- **Cache Proxy**

- Hold results temporarily
- Saves data for clients to share so data is only fetched or calculated once
- **Caching of information: Helpful if information does not change too often.**

- **Copy-on-write:**

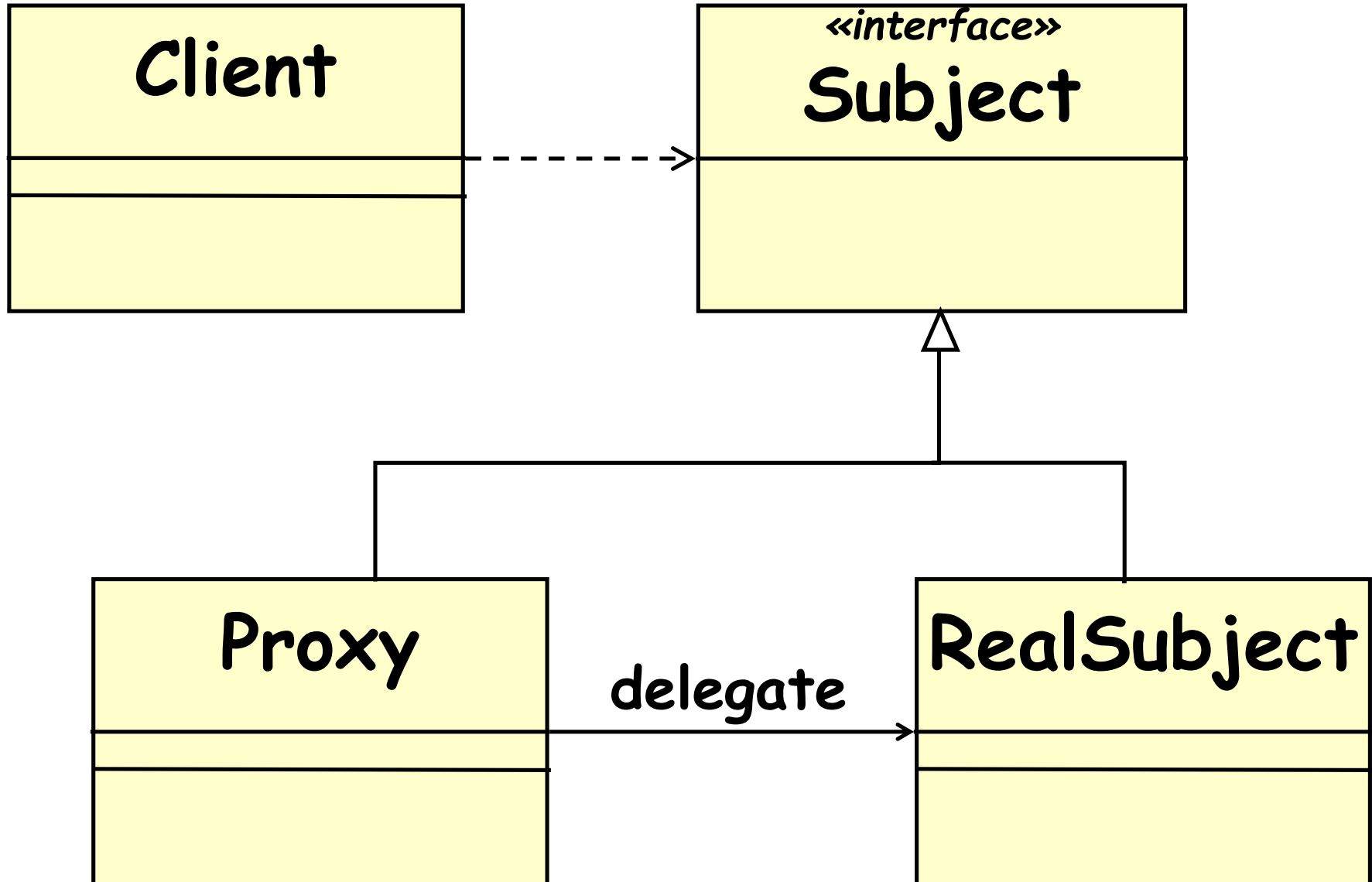
- Postpones creation of a copy of an object until it is necessary (if at all, changed from the original).

Kinds of Proxies: Few Details

- **Protection Proxy:**

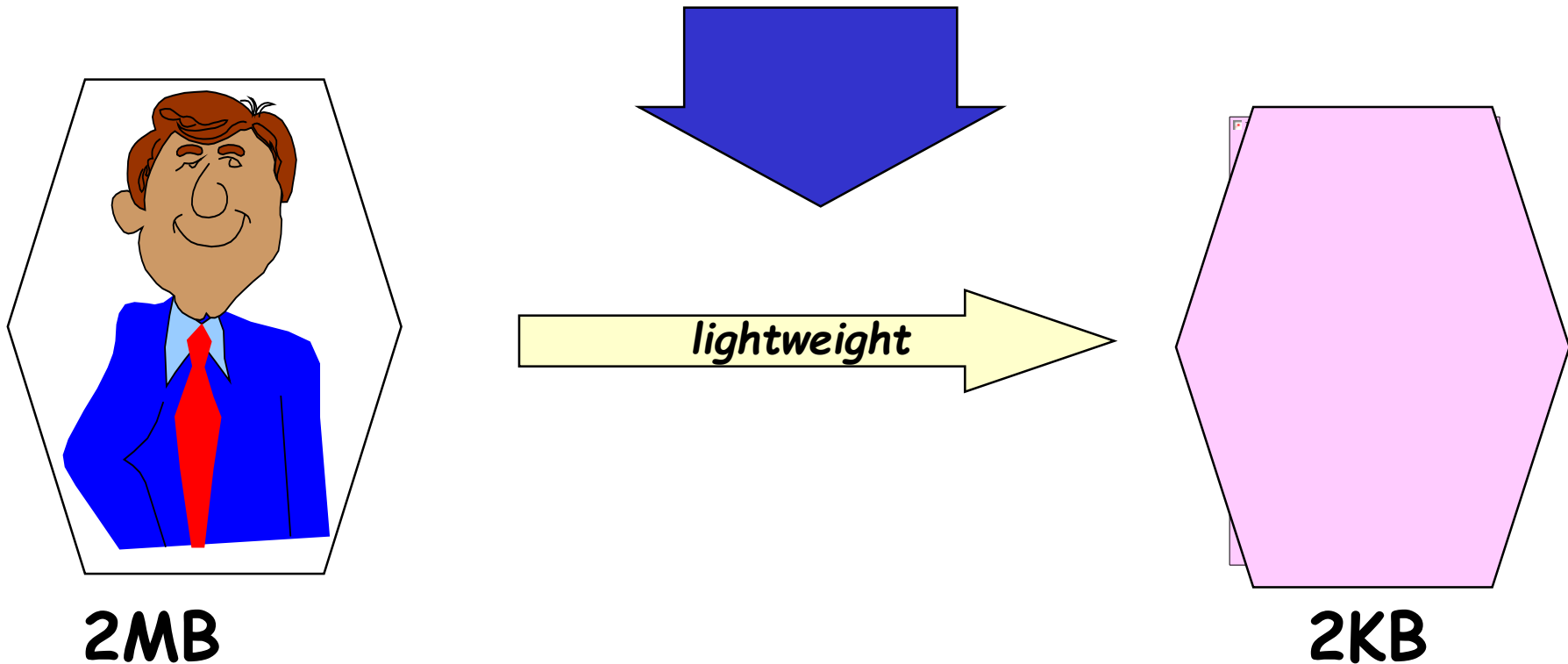
- Performs additional housekeeping chores when subject is accessed.
- Checks access permission to the real object
- Ensures that only authorized clients access a supplier in legitimate ways
- Useful when different objects should have different access and viewing rights for the same document.
- **Example:** Grade information shared by administrators, teachers and students.

Proxy Pattern

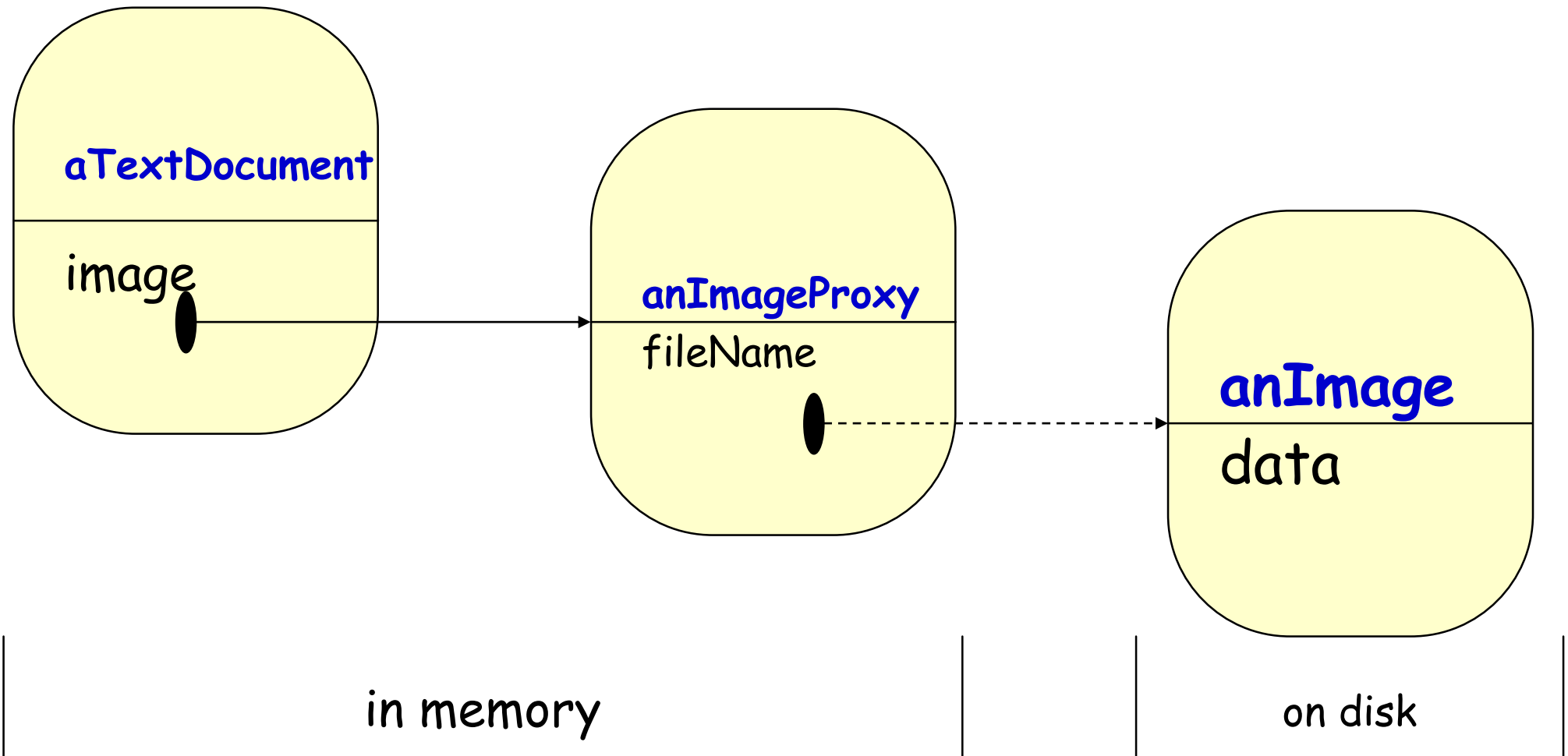


Virtual Proxy: Why Stand-in?

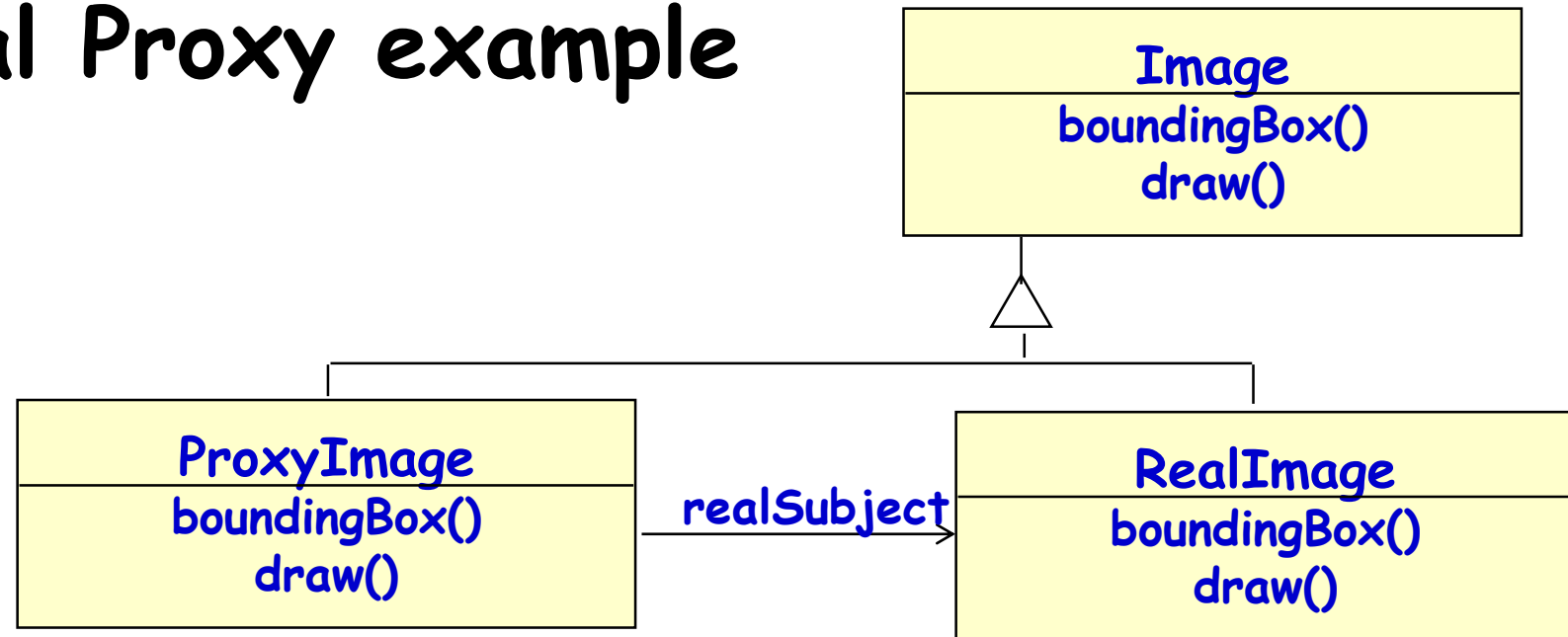
1. The image is expensive to load
2. The complete image is not always necessary



Virtual Proxy: Object Diagram



Virtual Proxy example



- Images are stored and loaded separately from text
- If a **RealImage** is not loaded a **ProxyImage** displays a grey rectangle in place of the image
- The client cannot know that it is dealing with a **ProxyImage** instead of a **RealImage**

Example Application: Picture Viewer

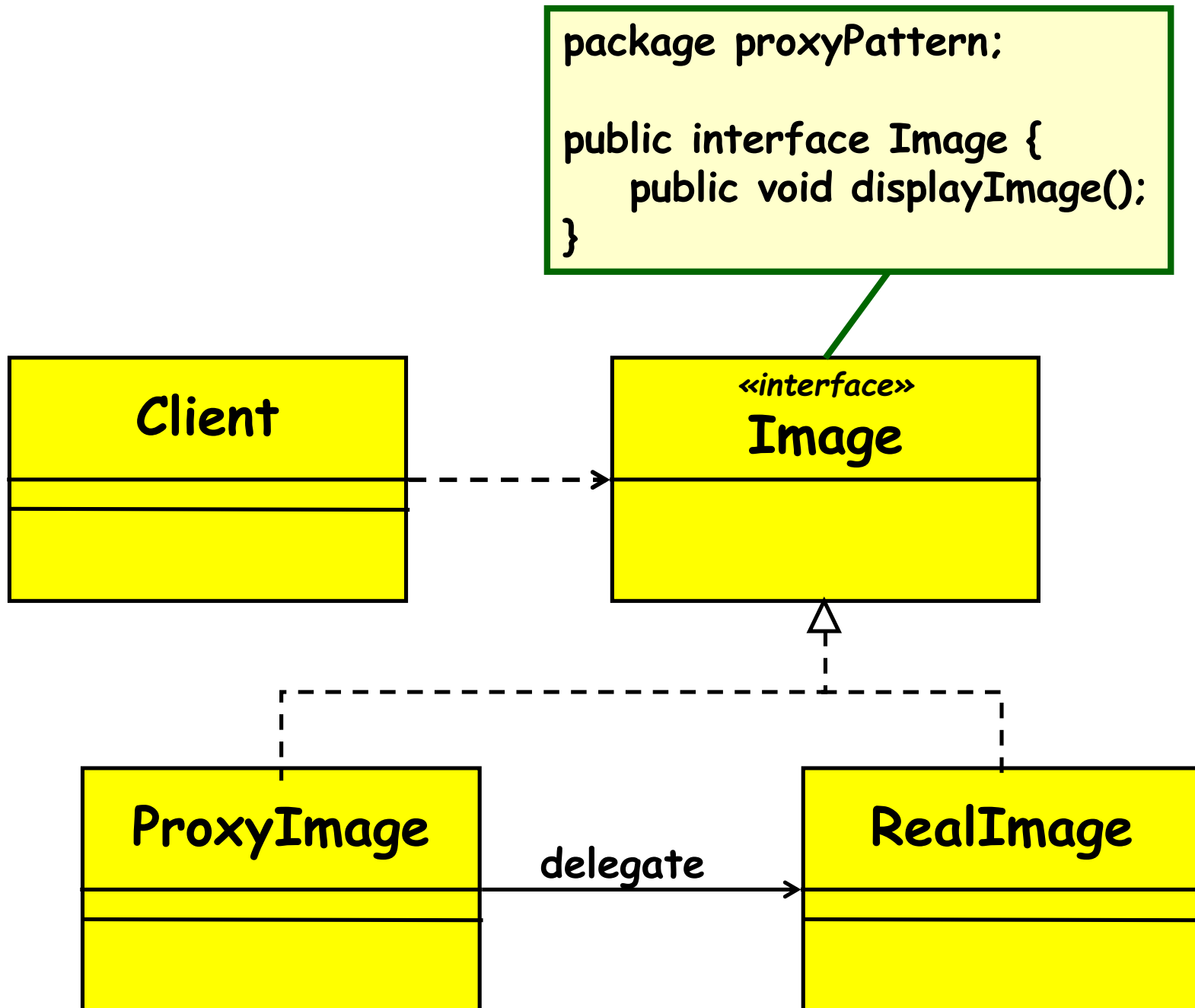
- A picture viewer has many high quality images.
- Opening the viewer should be fast:
 - Not so if all images must be loaded.
- No need to load all:
 - Not all will be viewable in the Window.
- **Load images on demand !**



190	191	192	193	194	195	196	197	198	199	
180	181	182	183	184	185	186	187	188	189	
170	171	172	173	174	175	176	177	178	179	
160	161	162	163	164	165	166	167	168	169	
150	151	152	153	154	155	156	157	158	159	
140	141	142	143	144	145	146	147	148	149	
130	131	132	133	134	135	136	137	138	139	
120	121	122	123	124	125	126	127	128	129	
110	111	112	113	114	115	116	117	118	119	
100	101	102	103	104	105	106	107	108	109	



Example 1: Image Proxy



Example 2: Lazy Loading in A Word Processor

- Suppose a document contains lots of multimedia objects:
 - Yet should load fast...
- Create proxies to represent large images, movies, etc.:
 - Only load these objects on demand as they become visible on the screen (Afterall, only a small part of the document is visible at a time)

Lazy Loading cont...

- Rather than instantiating an expensive object right away:
 - Create a proxy instead, and give the proxy to the client
- The proxy creates the object on demand when the client first uses it:
 - If the client never uses the object, the expense of creating it is never incurred...

Lazy Loading cont...

- A hybrid approach can be used:
 - The proxy implements some operations itself.
 - Create the real object only if the client invokes one of the operations it cannot perform...
- A proxy stores necessary information to create the object on-the-fly:
 - file name, network address, etc.

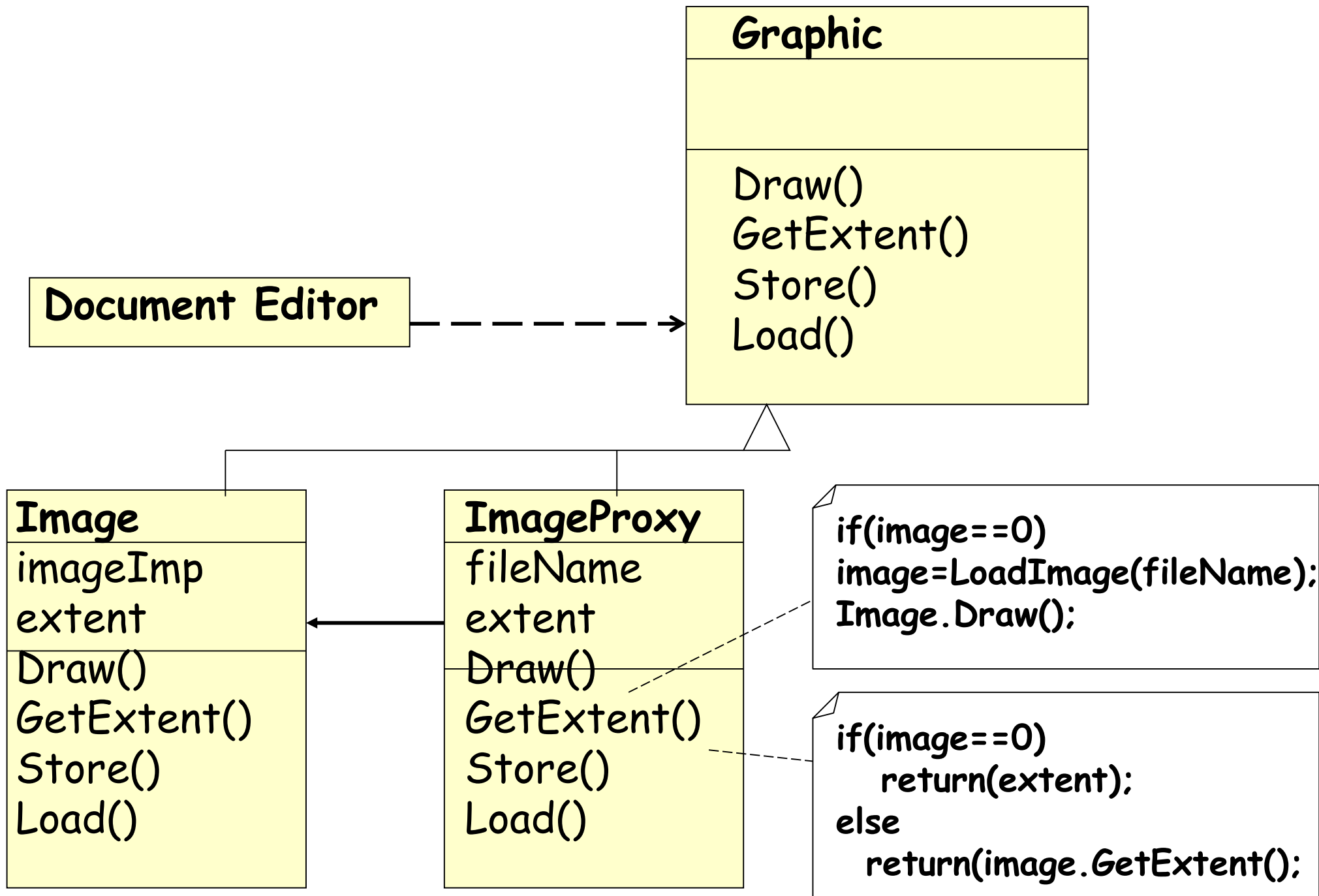


Image Proxy Example 1

```
interface Image {
```

```
    public void displayImage();
```

```
    public void loadImage();
```

```
}
```

```
class RealImage implements Image {
```

```
    private String filename;
```

```
    public RealImage(String filename) {
```

```
        this.filename = filename;
```

```
        System.out.println("Loading " + filename);
```

```
    }
```

```
    public void displayImage() {
```

```
        System.out.println("Displaying " + filename); }
```

```
}
```

Proxy Pattern Example 1

```
public class ProxyImage implements Image {
```

```
    private String filename;
```

```
    private Image image;
```

delegate request
to real subject

```
    public ProxyImage(String filename){  
        this.filename = filename;
```

```
    }
```

```
    public void displayImage() {
```

```
        if (image == null) {
```

```
            image = new RealImage(filename);
```

//load only on demand

```
        }
```

```
        image.displayImage();
```

```
    }
```

```
}
```

Proxy Example 1 - the Client


```
public class ProxyExample {  
    public static void main(String[] args) {  
  
        ArrayList<Image> images = new ArrayList<Image>();  
        images.add(new ProxyImage("HiRes_10MB_Photo1"));  
        images.add(new ProxyImage("HiRes_10MB_Photo2"));  
        images.add(new ProxyImage("HiRes_10MB_Photo3"));  
  
        images.get(0).displayImage();  
        images.get(1).displayImage();  
        images.get(0).loadImage();  
    }  
}
```

Before Proxy...

Netscape: Cyberian Outpost - Computers notebooks desktops hardware & software - Buy it Today - Use

Back Forward Reload Home Search Guide Images Print Security Stop

Netsite: <http://www.cyberianoutpost.com/>



CYBERIAN Outpost

The Cool Place to Shop For Computer Stuff!


We Ship Internationally!

Call: (800)856-9800 | (860)927-2050 | Fax: (860)-927-8375 | E-mail: sales@outpost.com

SPECIAL GIFTS FOR DADS & GRADS! CLICK!

OUTPOST TODAY
Jun. 10, 1998

- MAC
- PC
- DESKTOPS
- NOTEBOOKS
- PDA'S
- MEMORY
- SOFTWARE
- PERIPHERALS
- MODEMS
- NETWORKING
- GAMES
- ACCESSORIES
- DIGITAL CAMERAS
- DOWNLOADS
- WHAT'S NEW
- CUSTOMER SERVICE
- STOCK TRACKING
- ABOUT THE OUTPOST




56K X2 PC CARD MODEM

(LIMITED TIME ONLY)

PC

\$59.95

CLICK TO BUY IT!




SCREAMIN' G3

(LIMITED TIME ONLY)

Mac

\$1549.00

CLICK TO BUY IT!



ASTRA 300P FLATBED SCANNER

LIMITED TIME PRICE

NEW LOW PRICE!

PC


\$69.95

CLICK TO BUY IT!

GLOBAL VILLAGE

56K TelePort

PLUS MAC OS 8



metroworks

300Mhz/750w/1MB

Backside at 150Mhz

MAXPOWER G3

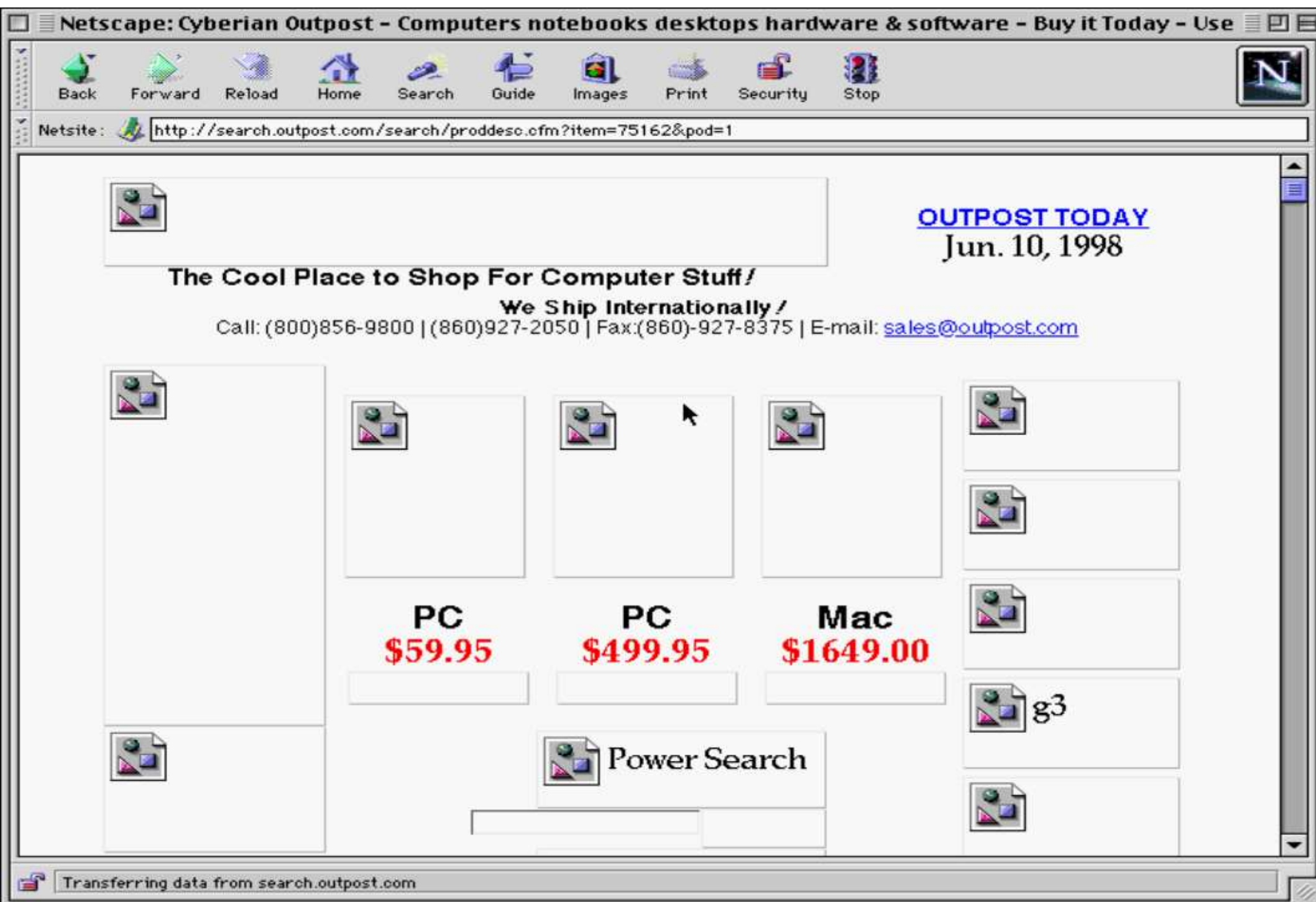
WE EAT INTEL PROCESSORS FOR LUNCH!

NEW! BUY IT TODAY - USE IT TODAY!

PRODUCT SEARCH

GO!

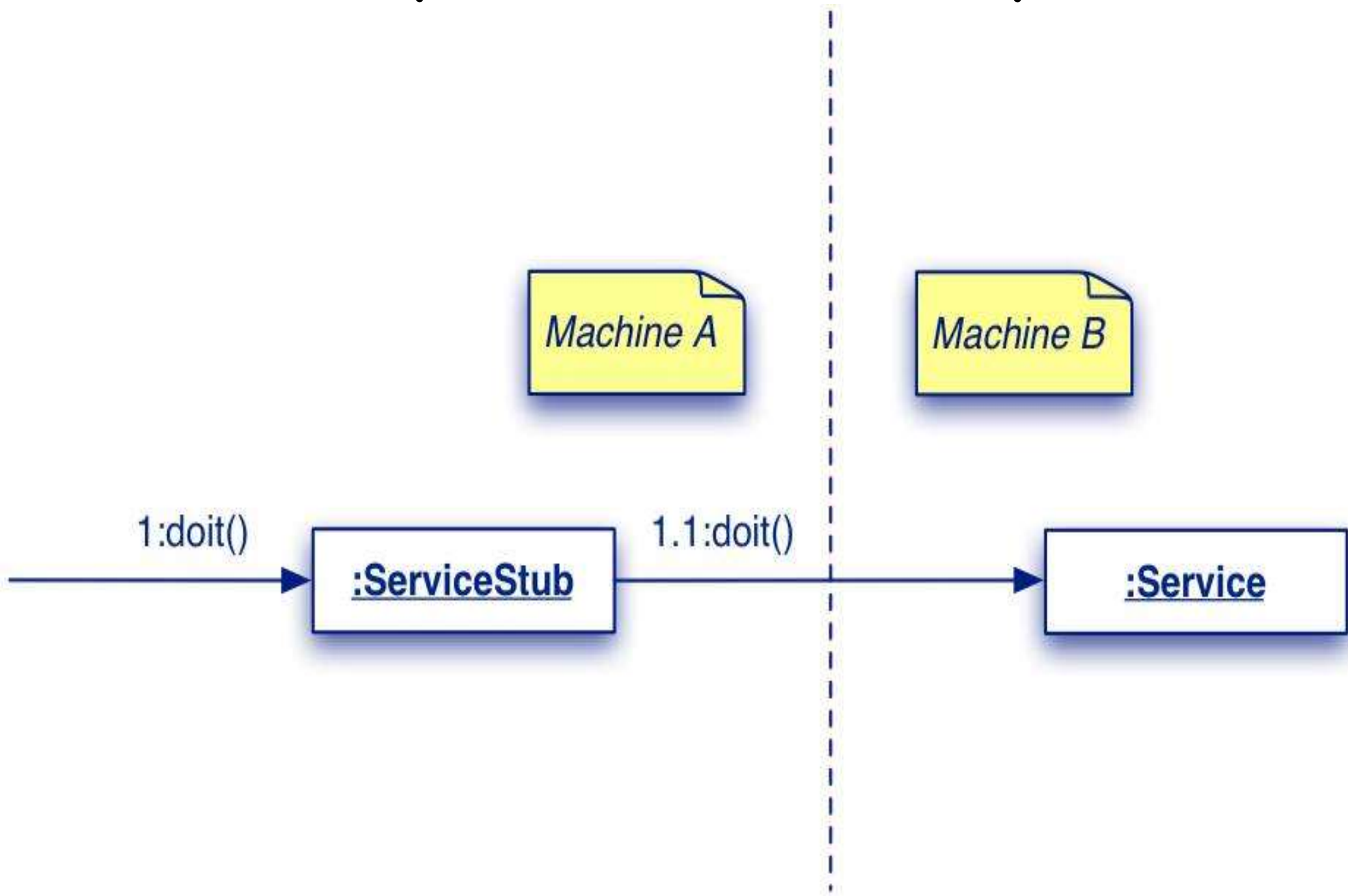
After Proxy...



Remote Proxy

- The proxy object:
 - Assembles data into a network message
 - Sends it to the remote object
- A remote receiver:
 - Receives data,
 - Turns it into a local message that is sent to the remote object

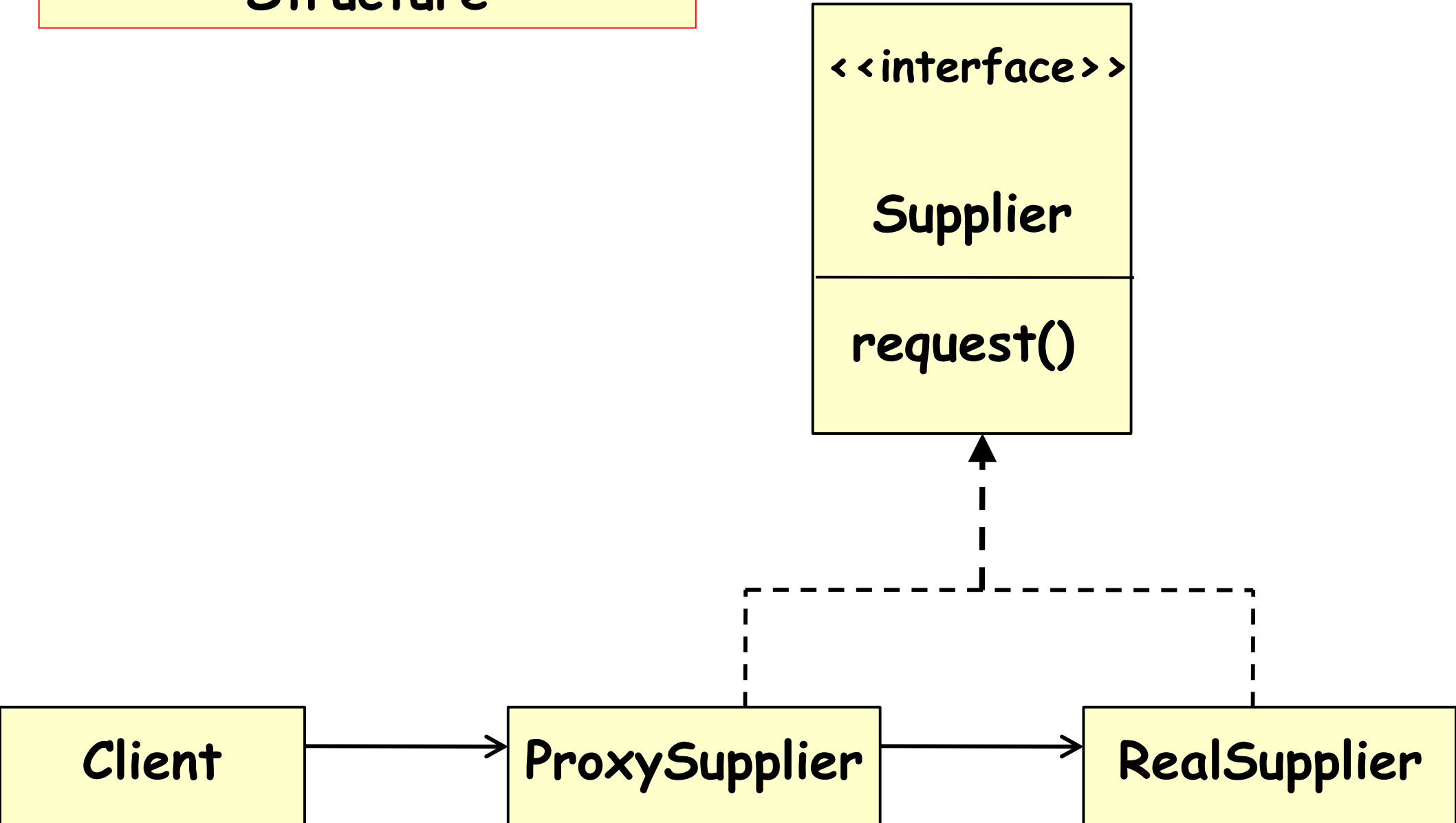
Proxy remote access example



Remote Proxy

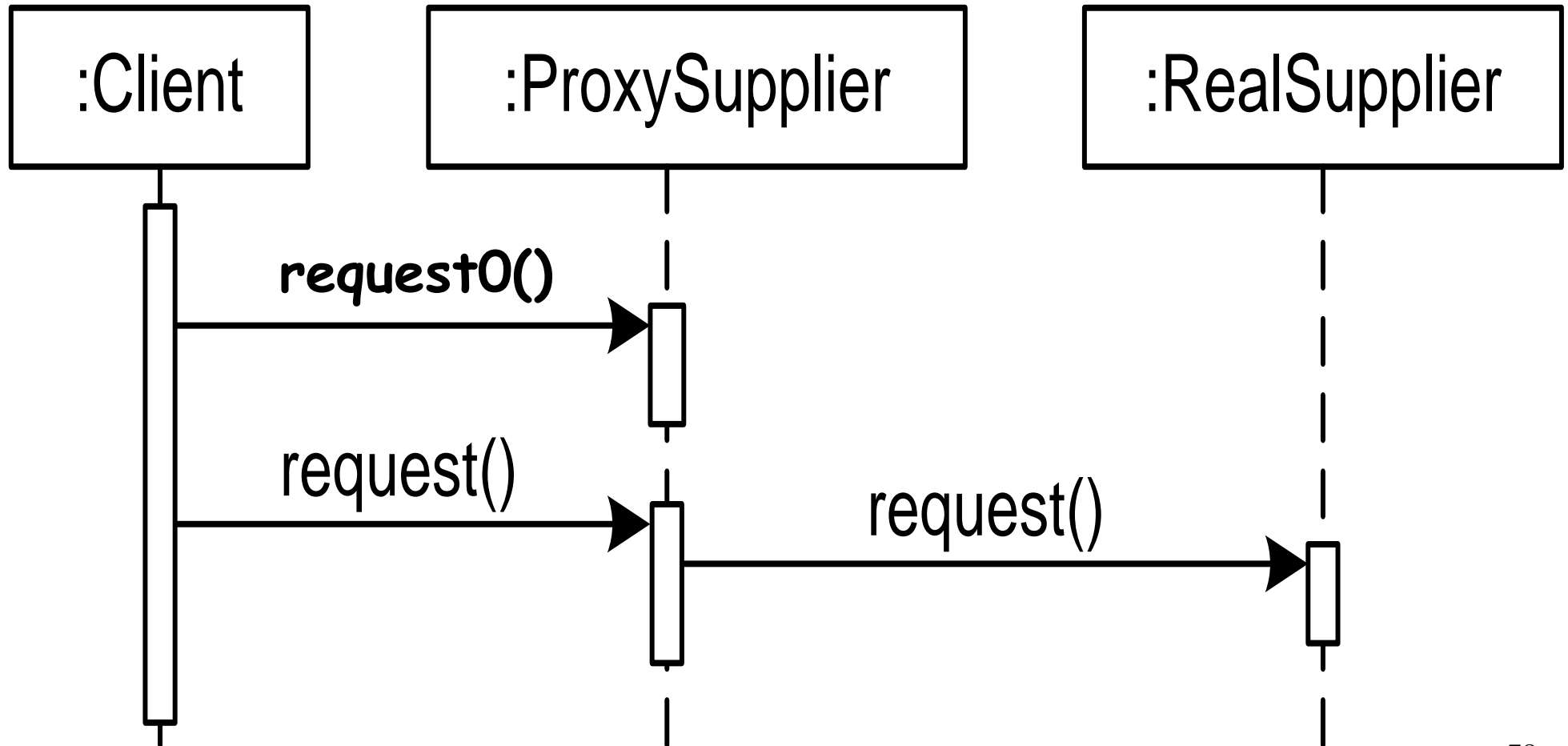
- **Stand-in for an object often needed because it is:**
 - Not locally available;
 - Instantiation and access are complex
 - Needs protected access for security.
- **The stand-in must:**
 - Have the same interface as the real object;
 - Handle as many messages as it can;
 - Delegate messages to the real object when necessary.

Remote Proxy Pattern Structure



Proxy Pattern Behavior

sd ProxyBehavior



Proxy: An Analysis

Consequences:

- A Proxy decouples clients from servers. A Proxy introduces a level of indirection.

Proxy some what similar to object adapter but differs from it in that it does not change the object's interface.

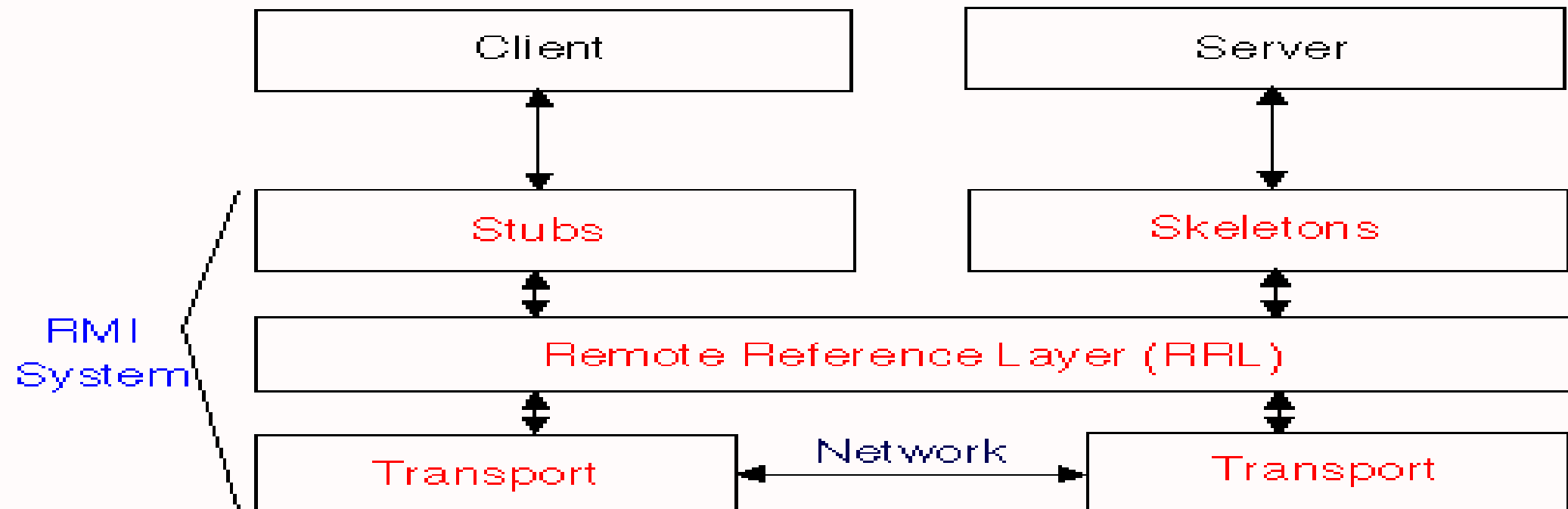
Proxy Usage: Another Example

- Enterprise JavaBeans and RMI provide examples of proxy usage:
 - A client for RMI or EJBs gets a proxy version of an object (called stub)
 - The proxy communicates (over the network) with the actual object
 - The proxy object makes remote communication appear (almost) identical to a local connection.

Java API Usage

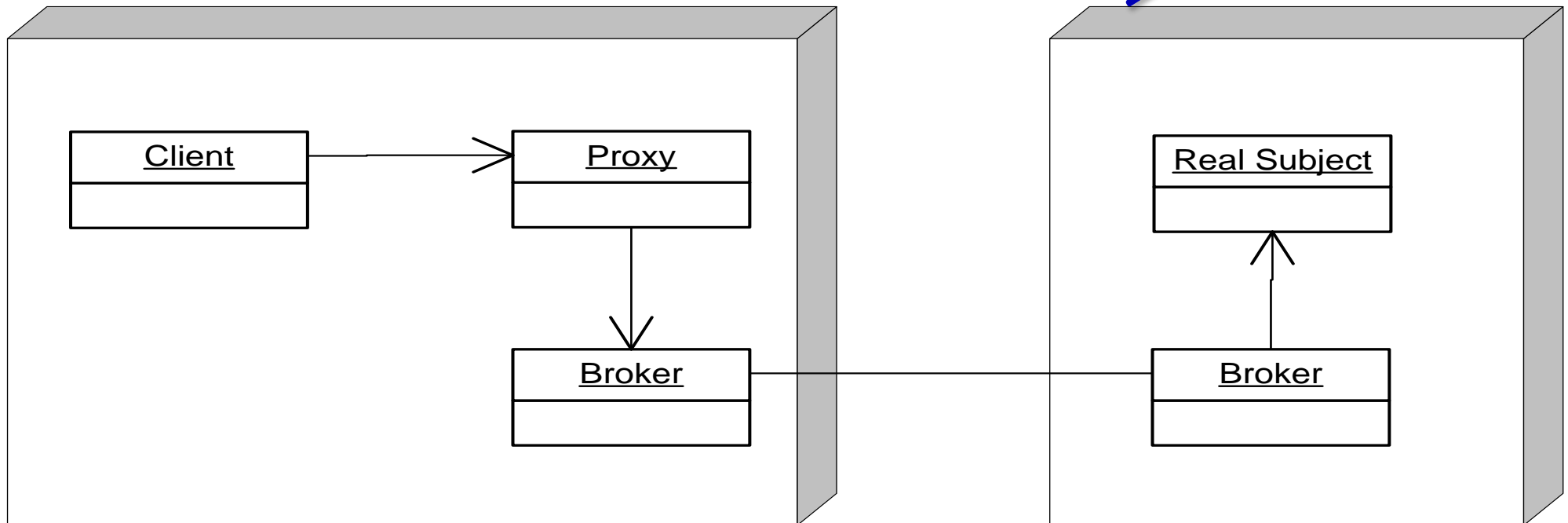
- java.rmi library --- "Remote Method Invocation"
- Allows objects in separate virtual machines to be used as if local
- **Uses "stub and skeleton" to handle communication**

Application



Known Uses: Broker Objects

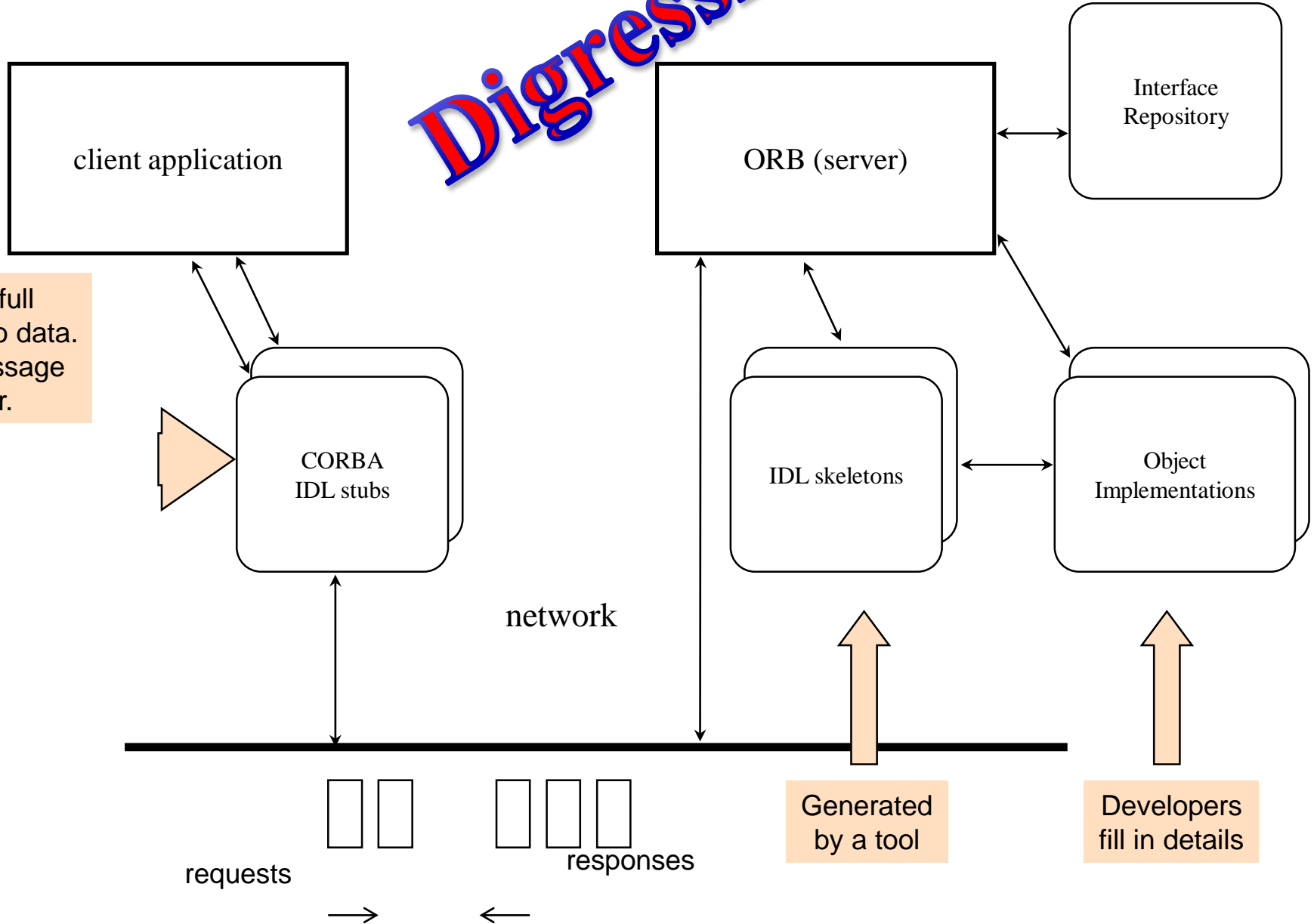
- The Client and Real Subject are in different processes or on different machines:
 - **So a direct method call will not work**
- The Proxy's job is to pass the method call across process or machine boundaries:
 - Return the result to the client (with Broker's help)



CORBA example

Digression

Stub objects have full public interface, but no data. They just push a message over to the server.



Uses #3 and #4: Java Collections

- **Read-only Collections:**

- Wrap collection object in a proxy that only allows read-only operations to be invoked on the collection
- All other operations throw exceptions
- **List unmodifiableList=Collections.unmodifiableList(List list);**
 - Returns read-only List proxy

- **Synchronized Collections**

- Wrap collection object in a proxy that ensures only one thread at a time is allowed to access the collection
- **Proxy acquires lock before calling a method, and releases lock after the method completes**
- **List Collections.synchronizedList(List list);**
 - Returns a synchronized List proxy

Proxy Uses #5: Secure Objects

- Different clients have different levels of access privileges to an object
- Clients access the object through a proxy
- The proxy either allows or rejects a method call:
 - Depending on what method is being called and who is calling it (i.e., the client's identity)

Proxy Use #6: Copy-on-Write

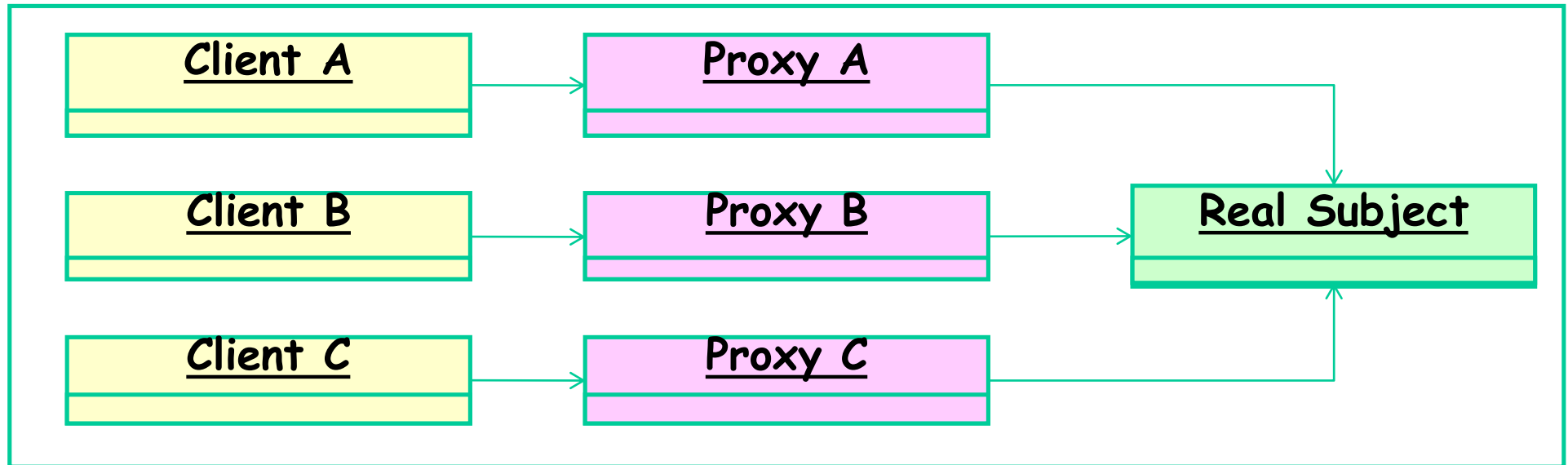
- Multiple clients share the same object:
 - as long as nobody tries to change it
- When a client attempts to change the object:
 - They get their own private copy of the object
- Read-only clients continue to share the original object:
 - while writers get their own copies
- **Maximize resource sharing, while making it look like everyone has own object.**

Use #6: Copy-on-Write

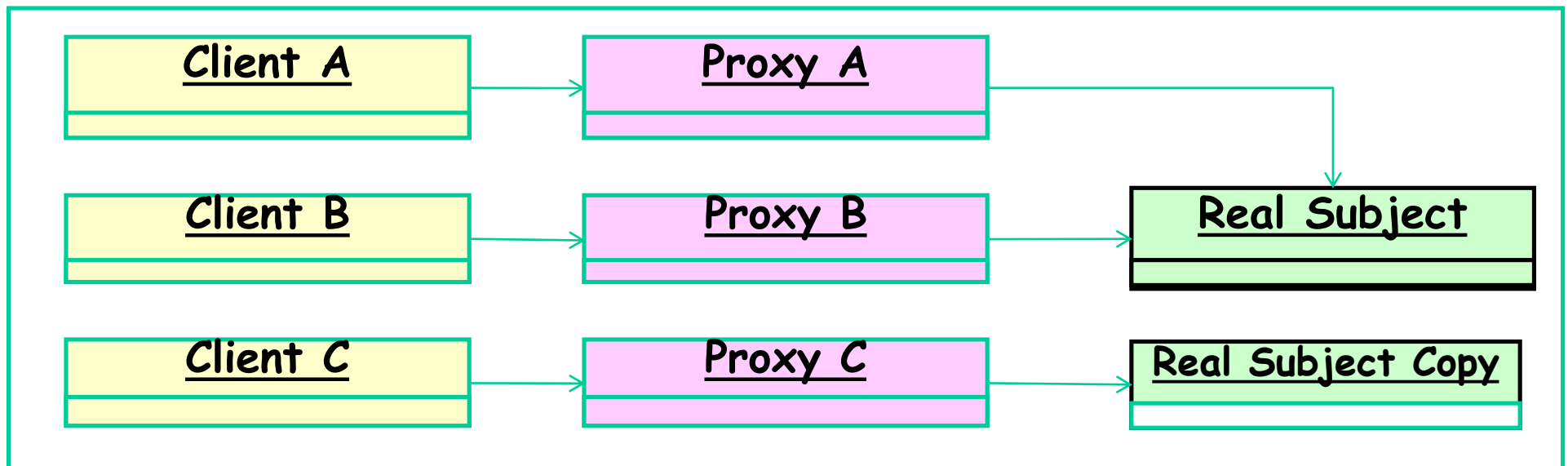
- Highly optimized String classes often use this approach.
- To make this work:
 - Clients are given proxies rather than direct references to the object.
- When a write operation occurs:
 - A proxy makes a private copy of the object on-the-fly to insulate other clients from the changes ...

Proxy Known Uses: Copy-on-Write

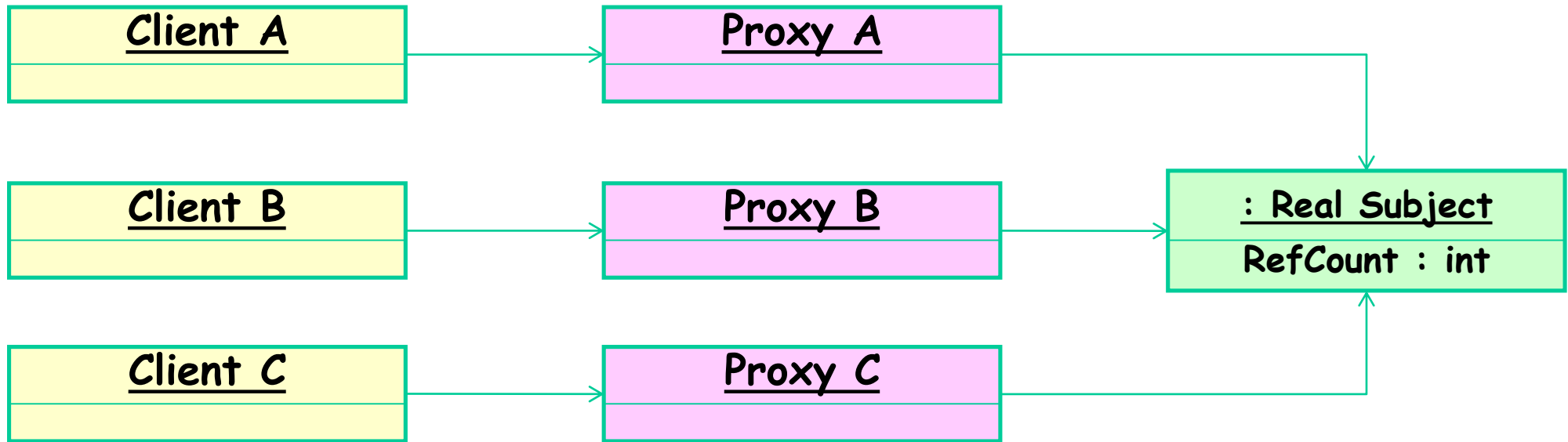
Before any Write



After Write by C



Uses #7: Reference Counting



- Proxies maintain the reference count inside the object
- The last proxy to go away is responsible for deleting the object:
 - That is, when the reference count goes to 0, delete the object.

Proxy: Related Patterns

- **Adapter:**

- Provides a different interface to an object.

- **Decorator:**

- Somewhat similar structure as proxy, but different purpose.
- **Decorator adds responsibilities whereas proxy controls access.**

Proxy vs. Decorator

- Both patterns create a “wrapper” around another object.
 - But an adapter has a different interface than the wrapped object.
 - While a Decorator adds additional responsibilities to an object, a Proxy “controls access” to an object.

- Not only that Proxy does not add any responsibilities, it might actually prevent access to some functionalities.
- E.g. Read-only collections, Secure objects