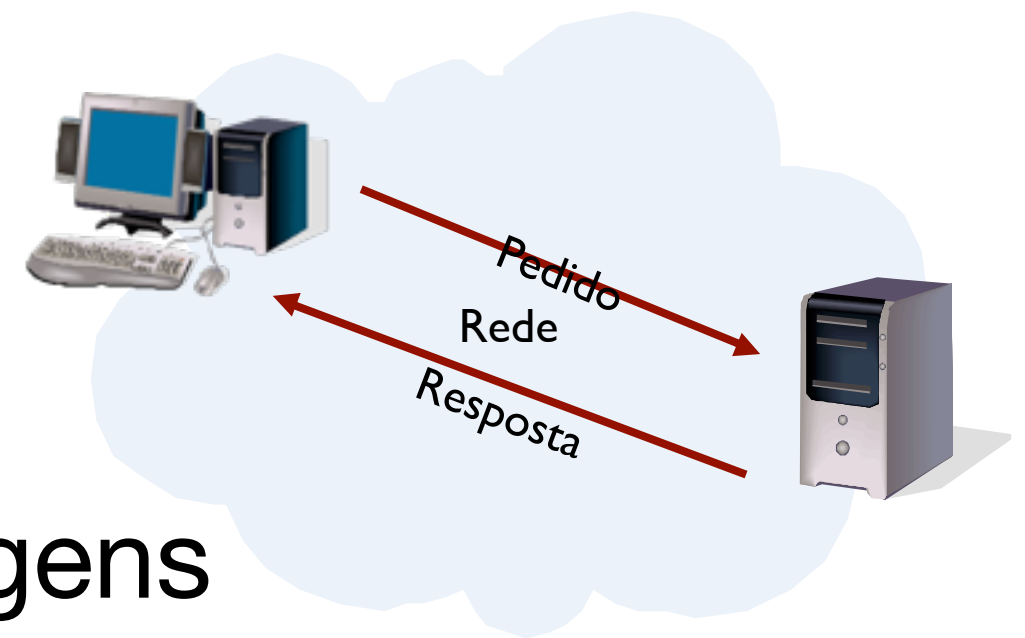


Remote Procedure Calls



Cliente/Servidor

- Mecanismos:
 - Comunicação
 - Formatação das mensagens
- Transparência para o programador:
 - Acesso
 - Localização
 - Falha
- Caso de estudo: Java RMI



Representação de Inteiros

1234

00000000 00000000 00000100 11010010

escrito em
big endian

escrito
little endian

n+0: 00000000

n+1: 00000000

n+2: 00000100

n+3: 11010010

n+0: 11010010

n+1: 00000100

n+2: 00000000

n+3: 00000000

lido em big endian

lido em little endian

lido em big endian

$0x256^3 + 0x256^2$

$+ 4x256 + 210$

$= 1234$

OK!

$210x256^3 + 4x256^2$

$+ 0x256 + 0$

$= 3523477504$

Erro!



Representação de Inteiros

1234

00000000 00000000 00000100 11010010

n+0: 00000000

n+1: 00000000

n+2: 00000100

n+3: 11010010

escrito em
big endian

Copiado na rede

n+0: 00000000

n+1: 00000000

n+2: 00000100

n+3: 11010010

lido em litte endian

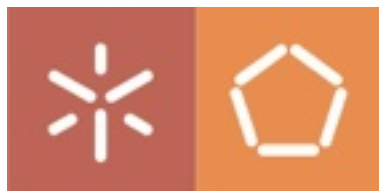
$$210 \times 256^3 + 4 \times 256^2 + 0 \times 256 + 0 = 3523477504$$

Erro!



Representação de Dados

- A solução é a conversão durante a transmissão
- Duas alternativas:
 - Conversão pelo emissor para um formato obrigatório
 - Conversão pelo receptor para o seu próprio formato
- Chama-se marshalling/unmarshalling
- Problemas semelhantes surgem com texto (e.g. caracteres acentuados) e outros tipos de dados

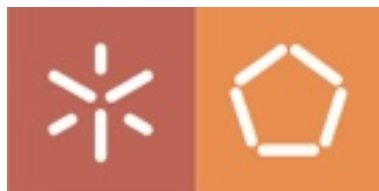


Objectos

- O marshalling de objectos implica:
 - Representar a classe do objecto
 - Representar os dados contidos no objecto
 - Basta saber qual a classe para saber qual a informação esperada:
 - Explícito ou implícito

ContactoTel:
Nome: Sra. A
Profissão: massagista
Número: 51723564

“ContactoTel” “Sra. A” “massagista”
517235564



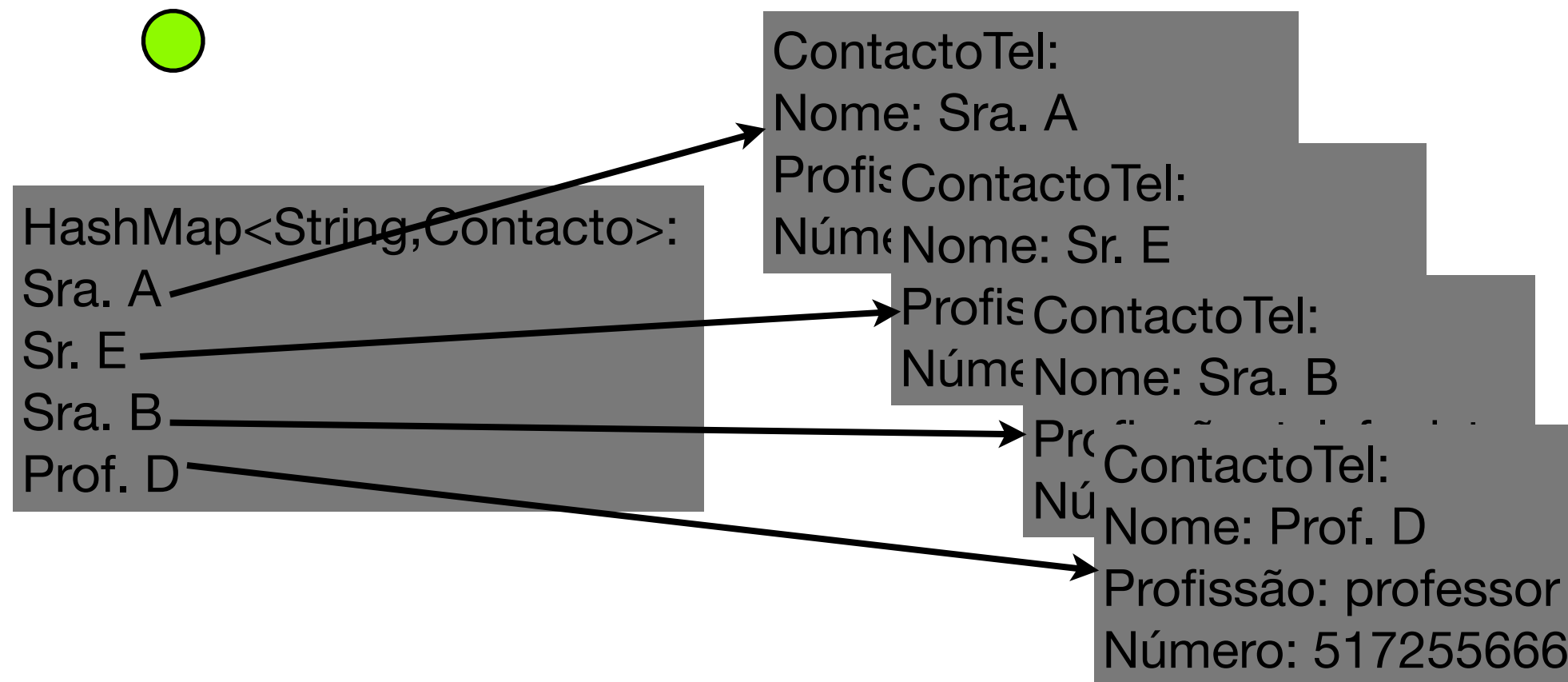
Objectos

- Cada um dos elementos é então convertido individualmente
- Algumas linguagens de programação conseguem enumerar o conteúdo de um objecto:
 - Java
- Noutras linguagens é necessário que o programador escreva filtros, que para cada tipo de dados enumeram os seus componentes



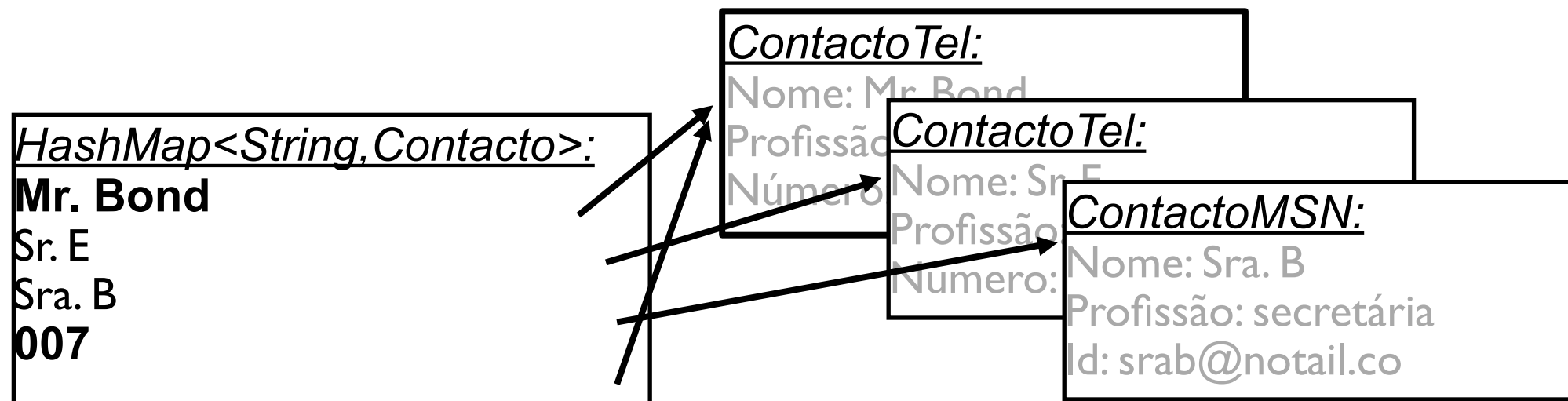
Referências para Objects

- Normalmente os objects fazem referência a outros objects
- É necessário fazer uma **travessia**:



Referências para Objectos

- Durante uma travessia pode ser encontrado o mesmo objecto várias vezes:

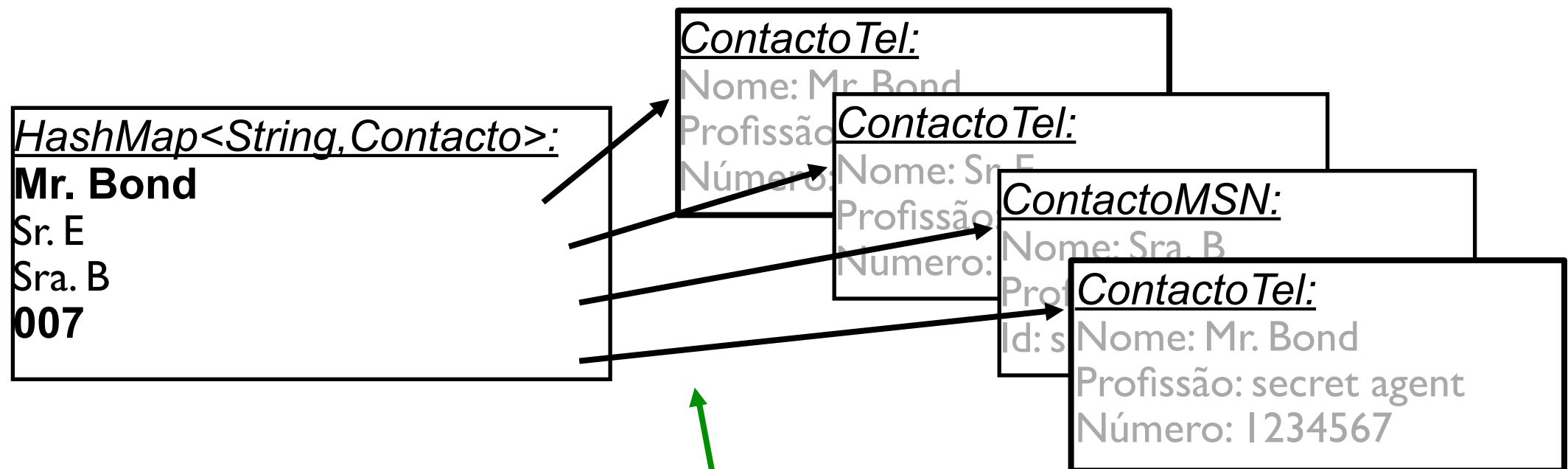


“HashMap<String, Contacto>” “Mr. Bond”
“ContactoTel” “Mr. Bond” “secret agent”
1234567 ... “007” “ContactoTel” “Mr. Bond”
“secret agent” 1234567



Referências para Objectos

- Depois de unmarshalled, temos duas cópias do mesmo objecto:

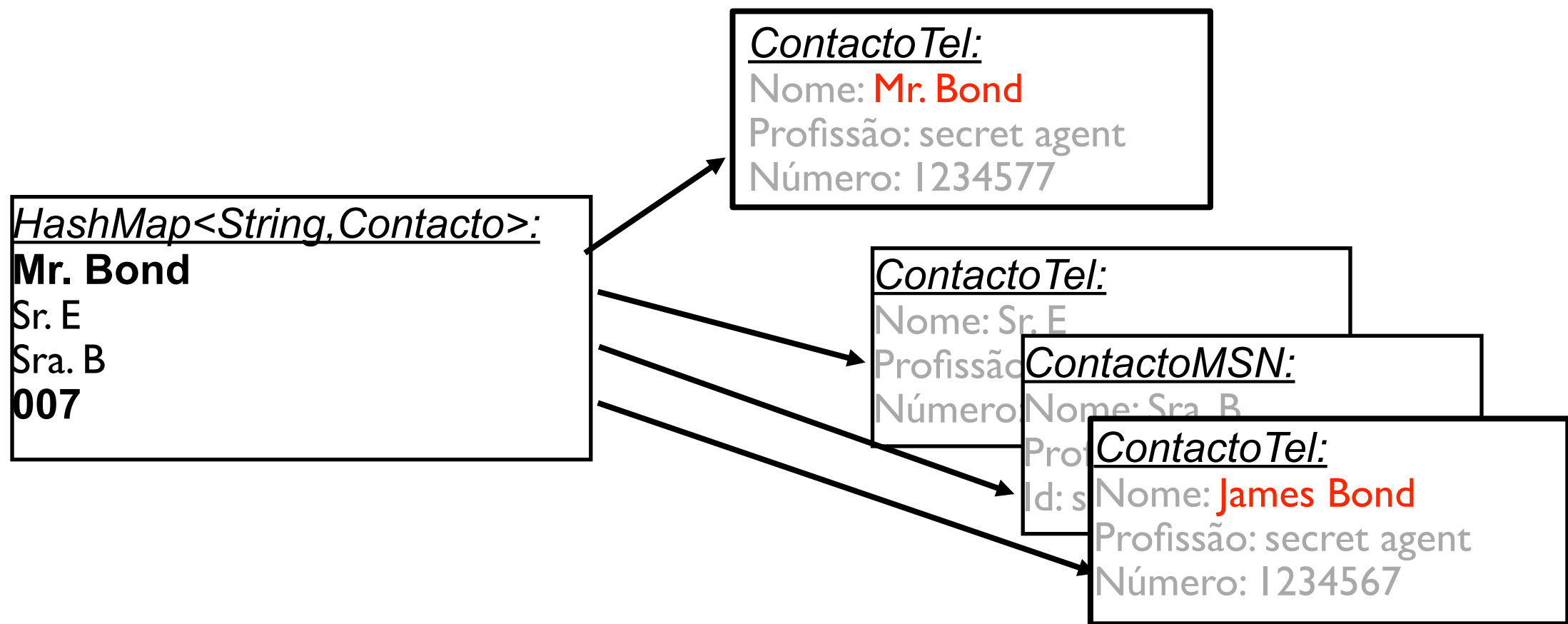


“HashMap<String, Contacto>” “Mr. Bond”
“ContactoTel” “Mr. Bond” “secret agent”
1234567 ... “007” “ContactoTel” “Mr. Bond”
“secret agent” 1234567



Referências para Objectos

- Se modificarmos o estado de uma das cópias do objecto a outra fica inalterada:

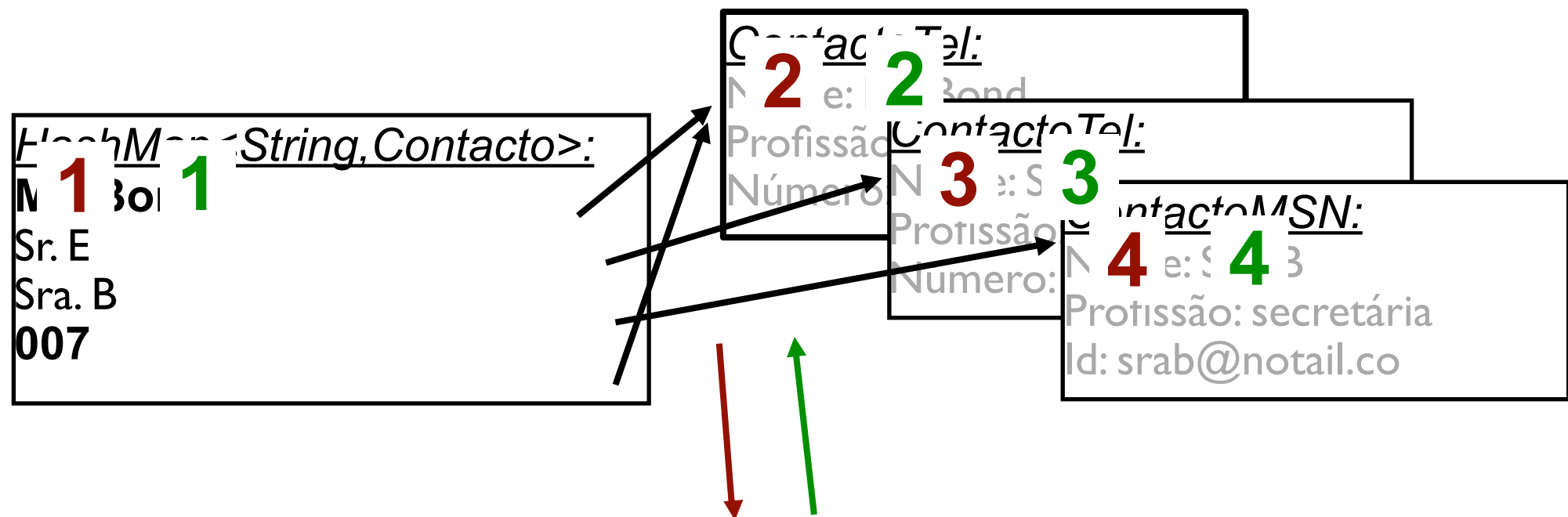


- Se a estrutura continha ciclos, as consequências eram ainda mais graves!



Referências para Objects

- A solução é:
 - Assinalar o início de cada objecto
 - Reconhecer objectos já visitados
 - Fazer referência a objectos anteriores:

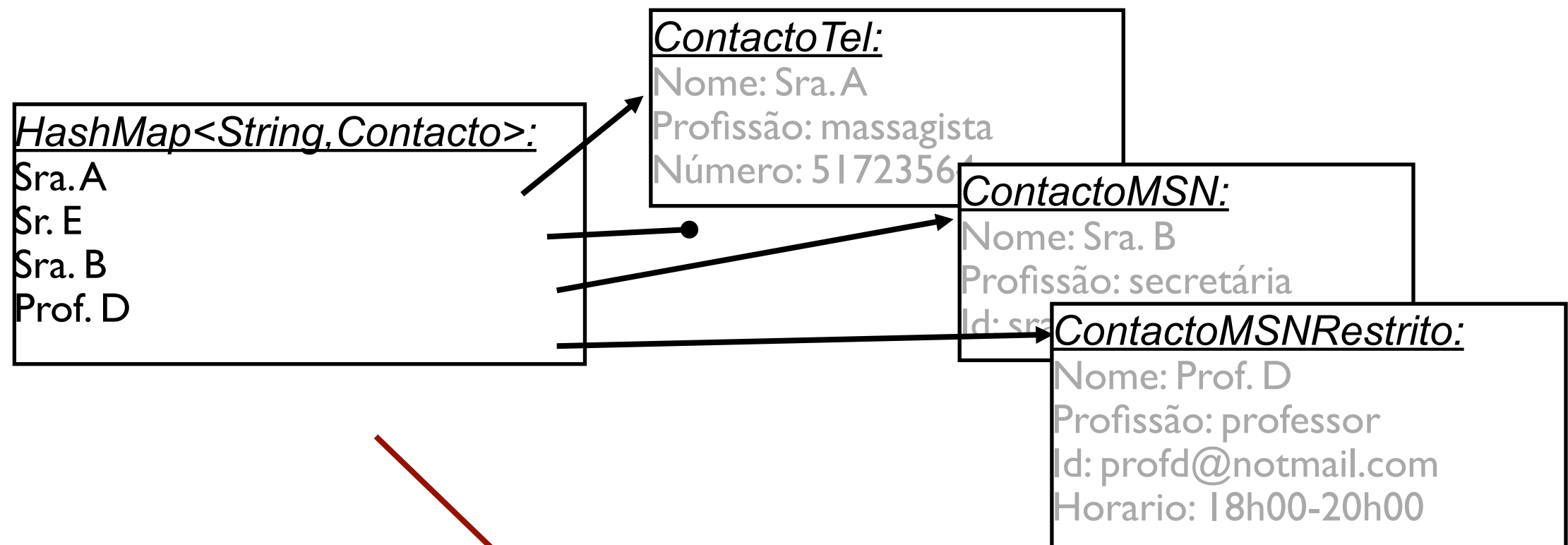


OBJ “HashMap<String, Contacto>” “Mr.
Bond” **OBJ** ... “Sr. E” **OBJ** ... “Sra. B” **OBJ** ...
“007” **REF(2)**



Referências para Objects

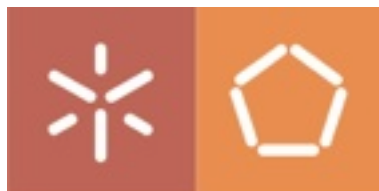
- Um mecanismo semelhante pode ser usado para referências nulas:



OBJ “HashMap<String, Contacto>” “Sra. A”

OBJ “ContactoTel” “Sra. A” “massagista”

517235564 “Sr. E” **NULL** “Prof. D” OBJ “Sra. B”....



Representação como Texto

- Todos os dados são convertidos para um formato inteligível

- Exemplo:

- XML

```
<map>
  <entrada>
    <chave>Sra.A</chave><valor> ... </valor>
  </entrada>
  ...
</map>
```

- Tolerar representações ligeiramente diferentes



Compromissos

- Velocidade de conversão:
 - Prejudicada pela detecção de alias
 - Prejudicada pelos formatos de texto
- Tamanho da representação:
 - Prejudicada pela anotação explícita de classes
 - Prejudicada pelos formatos de texto
- Robustez em sistemas heterogéneos:
 - Favorecida pela anotação explícita de classes
 - Favorecida pelos formatos de texto



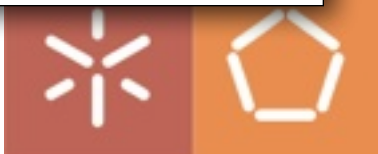
Exemplo de Aplicação

- Máquina de somar:

```
public interface Somador {  
    public int soma(int valor);  
    public void zero();  
}
```

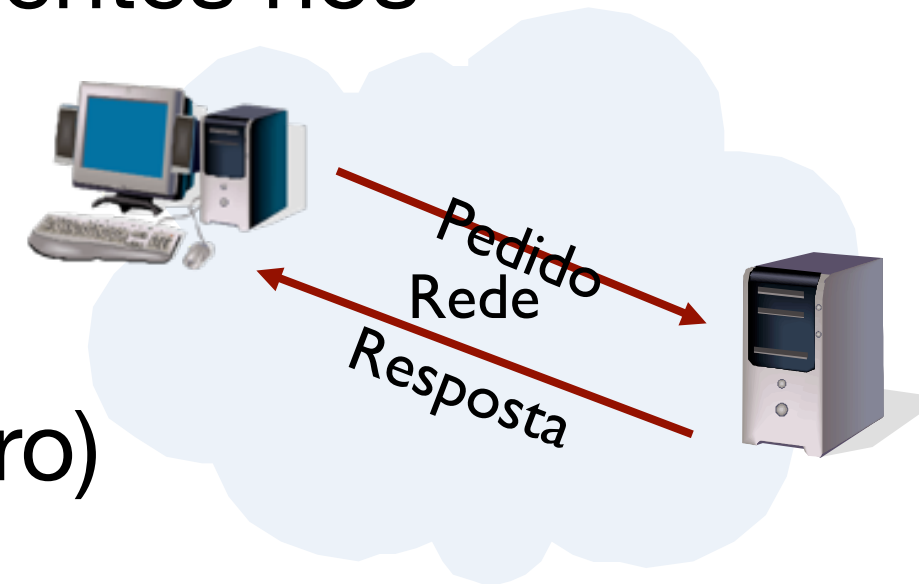
```
public class Local {  
    public static void main(String[] args) {  
        Somador s=new Calculadora();  
  
        int total=s.soma(123);  
  
        System.out.println(total);  
    }  
}
```

```
public class Calculadora implements Somador {  
    public class Registradora implements Somador {  
        private int total=0;  
        private List<Integer> registo=new LinkedList<Integer>();  
  
        public int soma(int valor) {  
            total=total+valor;  
            return total;  
        }  
  
        public void zero() {  
            registo.add(total);  
            total=0;  
        }  
    }  
}
```



Aplicação Distribuída

- Objectivos:
- Guardar o total num servidor central
- Utilizar o somador a partir de diferentes nós da rede
- Cada pedido contém:
 - nome da operação (soma ou zero)
 - dados para a operação (valor ou nada)
- Cada resposta contém:
 - total ou nada



Aplicação em Cliente/Servidor

```
try {
    Socket sock=new Socket("localhost", 12345);

    ObjectOutputStream oos=new
        ObjectOutputStream(sock.getOutputStream());
    ObjectInputStream ois=new
        ObjectInputStream(sock.getInputStream());

    oos.writeObject("soma");
    oos.writeObject(123);

    int total=(Integer) ois.readObject();

    System.out.println(total);

    oos.close();
    ois.close();
    sock.close();
} catch(Exception e) {
    e.printStackTrace();
}

try {
    int total=0;

    ServerSocket ssock=new ServerSocket(12345);
    while(true) {
        Socket sock=ssock.accept();
        ObjectOutputStream oos=new
            ObjectOutputStream(sock.getOutputStream());
        ObjectInputStream ois=new
            ObjectInputStream(sock.getInputStream());

        String op=(String) ois.readObject();
        if (op.equals("soma")) {
            int valor=(Integer) ois.readObject();
            total=total+valor;
            oos.writeObject(total);
        } else if (op.equals("zero")) {
            total=0;
        }

        oos.close();
        ois.close();
        sock.close();
    }
} catch(Exception e) {
    e.printStackTrace();
}
```



Transparência de Acesso?

- O código necessário para invocar a operação local e remota é diferente:

```
public class Local {  
    public static void main(String[] args) {  
        Somador s=new Calculadora();  
  
        int total=s.soma(123);  
  
        System.out.println(total);  
    }  
}
```

```
try {  
    Socket sock=new Socket("localhost", 12345);  
  
    ObjectOutputStream oos=new  
        ObjectOutputStream(sock.getOutputStream());  
    ObjectInputStream ois=new  
        ObjectInputStream(sock.getInputStream());  
  
    oos.writeObject("soma");  
    oos.writeInt(24);  
  
    int total=(Integer) ois.readObject();  
  
    oos.close();  
    ois.close();  
    sock.close();  
  
    System.out.println(total);  
} catch (Exception e) {  
    e.printStackTrace();  
}
```

Proxy
(implementa a mesma
interface Somador)



Solução: Classe Proxy

```
public class Cliente {  
    public static void main(String[] args) {  
        Somador s=new Proxy();  
  
        int total=s.soma(123);  
  
        System.out.println(total);  
    }  
}
```

```
public class Proxy implements Somador {  
  
    public int soma(int valor) {  
        int total=-1;  
        try {  
            Socket sock=new Socket("localhost", 12345);  
            ObjectOutputStream oos=new  
                ObjectOutputStream(sock.getOutputStream());  
            ObjectInputStream ois=new  
                ObjectInputStream(sock.getInputStream());  
  
            oos.writeObject("soma");  
            oos.writeObject(123);  
  
            total=(Integer) ois.readObject();  
  
            oos.close();  
            ois.close();  
            sock.close();  
        } catch(Exception e) {  
            e.printStackTrace();  
        }  
        return total;  
    }  
    ...  
}
```



Transparência de Acesso?

- O código para concretizar um serviço local e remoto é diferente:

```
public class Calculadora implements Somador {  
    private int total=0;  
  
    public int soma(int valor) {  
        total=total+valor;  
        return total;  
    }  
  
    public void zero() {  
        total=0;  
    }  
}
```

```
try {  
    int total=0;  
  
    ServerSocket ssock=new ServerSocket(12345);  
    while(true) {  
        Socket sock=ssock.accept();  
        ObjectOutputStream oos=new  
            ObjectOutputStream(sock.getOutputStream());  
        ObjectInputStream ois=new  
            ObjectInputStream(sock.getInputStream());  
  
        String op=(String)ois.readObject();  
        if (op.equals("soma")) {  
            int valor=(Integer)ois.readObject();  
            total=total+valor;  
            oos.writeObject(total);  
        } else if (op.equals("zero")) {  
            total=0;  
        }  
  
        oos.close();  
        ois.close();  
        sock.close();  
    }  
} catch(Exception e) {  
    e.printStackTrace();  
}
```

Skeleton
(usa qualquer
implementação de
Somador)



Solução: Classe Skeleton

```
try {
    Somador s=new Calculadora();

    ServerSocket ssock=new ServerSocket(12345);
    while(true) {
        Socket sock=ssock.accept();
        ObjectOutputStream oos=new
ObjectOutputStream(sock.getOutputStream());
        ObjectInputStream ois=new
        ObjectInputStream(sock.getInputStream());

        String op=(String) ois.readObject();
        if (op.equals("soma")) {
            int valor=(Integer) ois.readObject();
            int total=s.soma(valor);
            oos.writeObject(total);
        } else if (op.equals("zero")) {
            s.zero();
        }

        oos.close();
        ois.close();
        sock.close();
    }
} catch(Exception e) {
    e.printStackTrace();
}
```

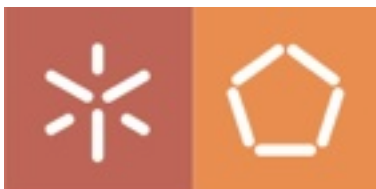
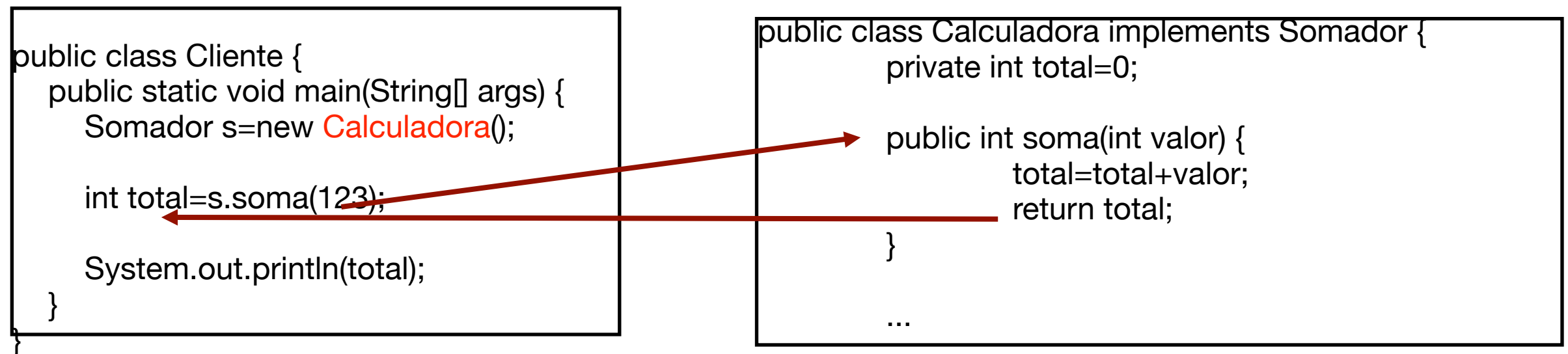
```
public class Calculadora implements Somador {
    private int total=0;

    public int soma(int valor) {
        total=total+valor;
        return total;
    }

    public void zero() {
        total=0;
    }
}
```



Invocação Local



Invocacão Remota

```
public class Cliente {  
    public static void main(String[] args) {  
        Somador s=new Proxy();  
  
        int total=s.soma(123);  
  
        System.out.println(total);  
    }  
}
```

```
public class Calculadora implements Somador {  
    private int total=0;  
  
    public int soma(int valor) {  
        total=total+valor;  
        return total;  
    }  
    ...  
}
```

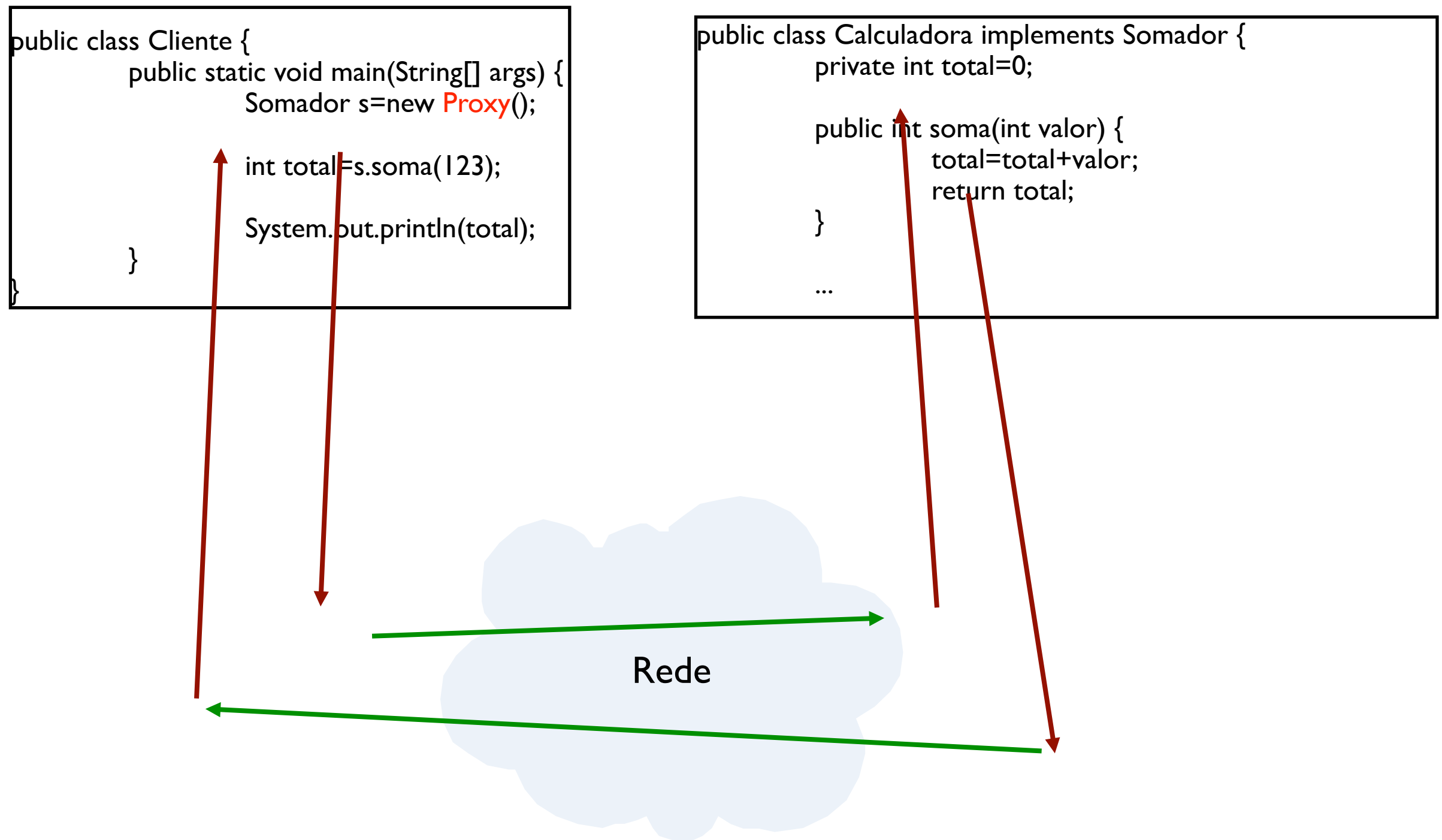
```
public class Proxy implements Somador {  
    public int soma(int valor) {  
        int total = -1;  
        try {  
            Socket sock=new Socket("localhost", 12345);  
  
            ObjectOutputStream oos=new  
            ObjectOutputStream(sock.getOutputStream());  
            ObjectInputStream ois=new  
            ObjectInputStream(sock.getInputStream());  
  
            oos.writeObject("soma");  
            oos.writeObject(123);  
  
            total=(Integer) ois.readObject();  
  
            oos.close();  
            ois.close();  
            sock.close();  
        } catch(Exception e) {  
            e.printStackTrace();  
        }  
        return total;  
    }  
    ...  
}
```

Rede

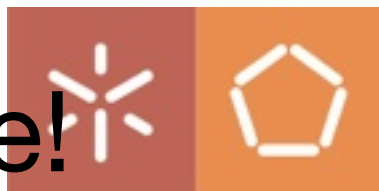
```
try {  
    Somador s=new Calculadora();  
  
    ServerSocket ssock=new ServerSocket(12345);  
    while(true) {  
        Socket sock=ssock.accept();  
        ObjectOutputStream oos=new  
        ObjectOutputStream(sock.getOutputStream());  
        ObjectInputStream ois=new  
        ObjectInputStream(sock.getInputStream());  
  
        String op=(String) ois.readObject();  
        if (op.equals("soma")) {  
            int valor=(Integer) ois.readObject();  
  
            int total=s.soma(valor);  
  
            oos.writeObject(total);  
        } else if (op.equals("zero")) {  
            s.zero();  
        }  
  
        oos.close();  
        ois.close();  
        sock.close();  
    }  
} catch(Exception e) {  
    e.printStackTrace();  
}
```



Middleware de Invocação Remota



Stub e Skeleton:
Gerados automaticamente a partir da interface!



Protocolo de Invocação Remota



liga (`new Socket(...)`)

nome da operação

parametro 1

parametro 2

parametro n

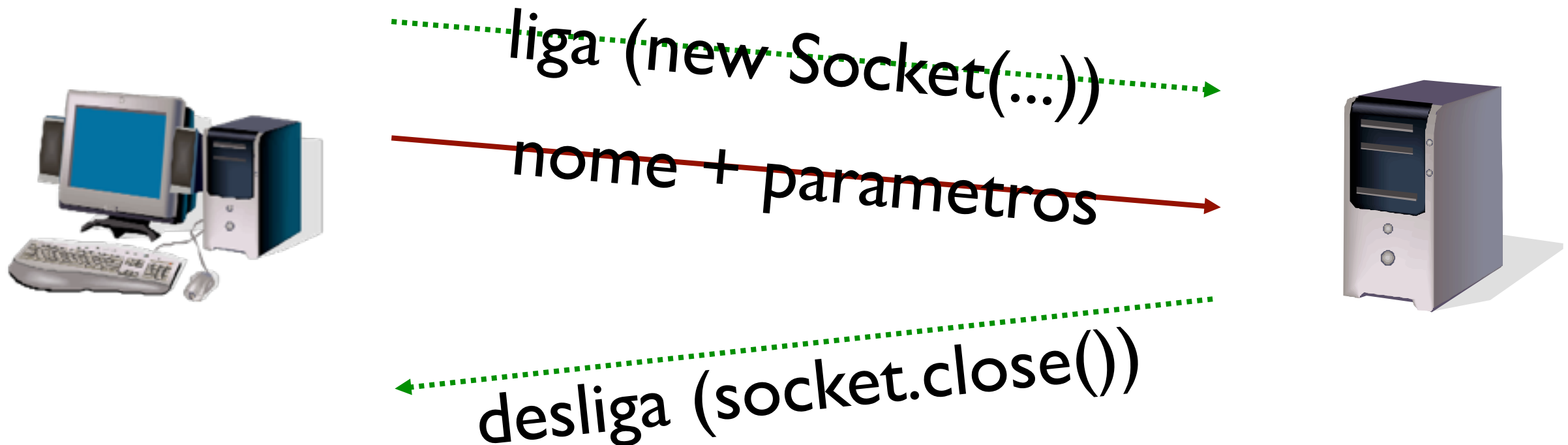
resultado

desliga (`socket.close()`)

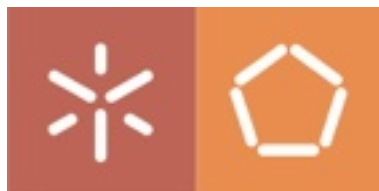


Variante “One-Way”

- Para operações sem resultados:

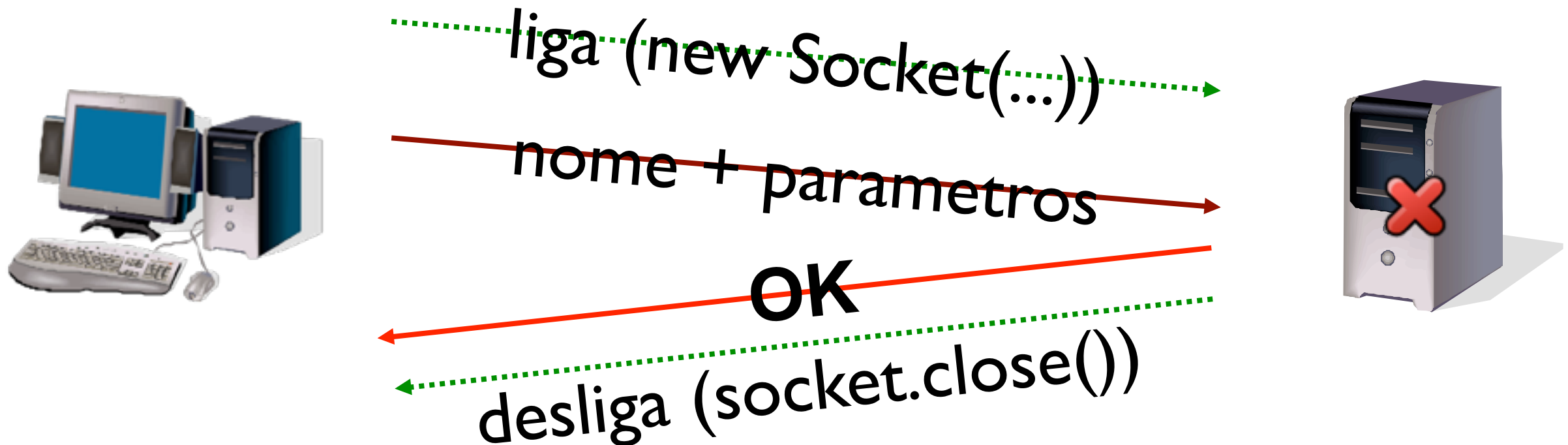


- O proxy não perde tempo à espera de resposta

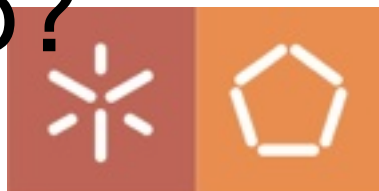


Pedido+Resposta

- Em caso de falha do servidor, como saber se a operação foi de facto executada?



- O que fazer quando o OK não é recebido?



Semântica de Falha

- Estratégia 1: Não fazer nada
 - A operação é executada no máximo uma vez
- Estratégia 2: Repetir a invocação
 - A operação é executada pelo menos uma vez
- O desejável seria uma estratégia que garantisse que a operação é executada **exatamente uma vez**



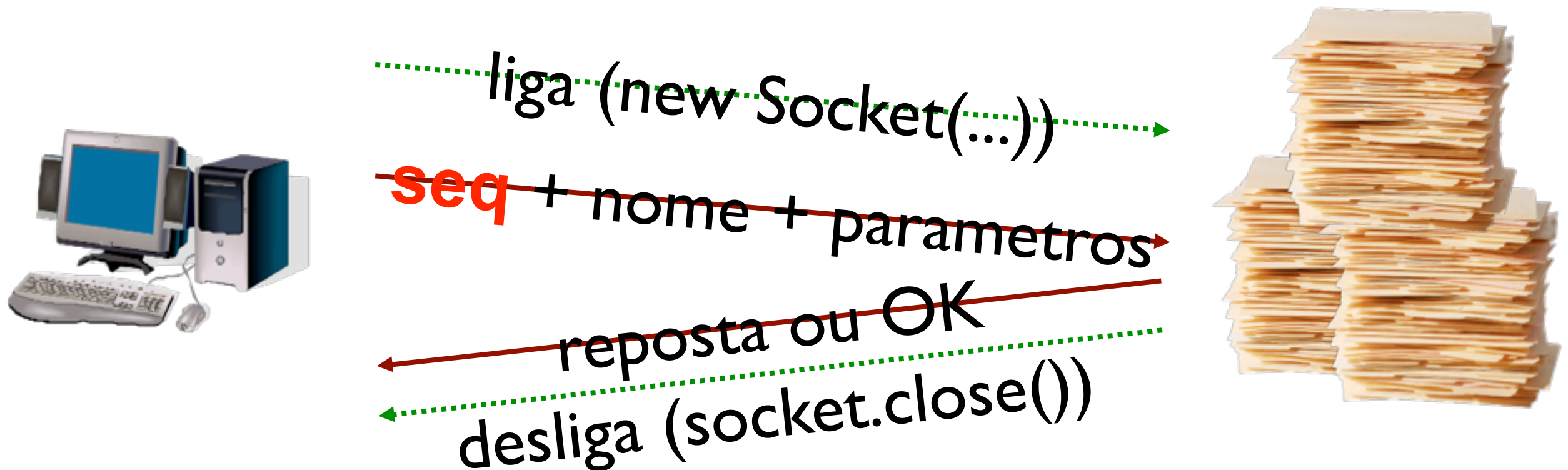
Idempotência

- Operação idempotente:
 - O efeito da operação efectuada várias vezes é o mesmo da operação efectuada apenas uma vez
- Exemplo:
 - escrita de um valor num ficheiro
- A estratégia 2 em operações idempotentes garante o desejado “exactamente uma vez”

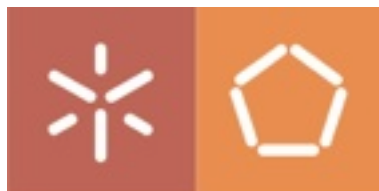


Retransmissão de Respostas

- Podemos numerar os pedidos para encontrar os duplicados e armazenar as respostas para retransmissão:

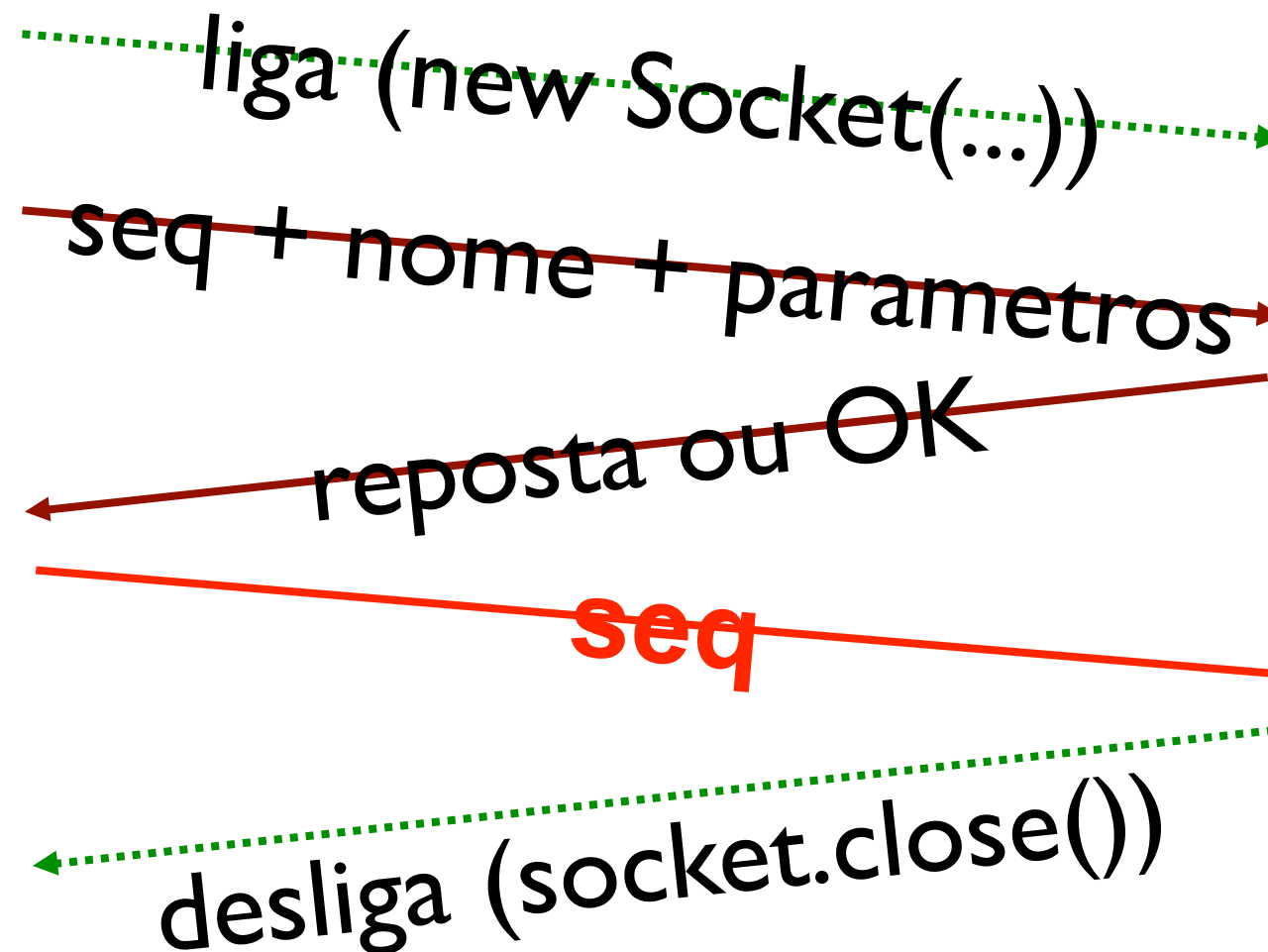


- Até quando temos que guardar as respostas?



Pedido+Resposta+Confirmação

- A confirmação pelo cliente permite eliminar respostas pendentes
 - O protocolo fica mais demorado



- O que acontece em caso de falha do cliente?



Transparência de Falha

- Não é possível proporcionar transparência de falha total
- O que fazer perante uma falha é mais facilmente resolvido:
 - Pela aplicação (e.g. pedidos idempotentes)
 - Pelo utilizador (e.g. verificando se o pedido foi efectuado)
- Transacções distribuídas ajudam, ao garantir que o pedido foi completamente executado ou completamente ignorado



Invocação Remota

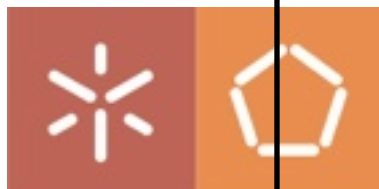
```
public class Cliente {  
    public static void main(String[] args) {  
        Somador s=  
            new Proxy("localhost", 12345);  
  
        int total=s.soma(123);  
  
        System.out.println(total);  
    }  
}
```

```
public class Proxy implements Somador {  
  
    private String host;  
    private int port;  
  
    public Proxy(String host, int port) {  
        this.host=host;  
        this.port=port;  
    }  
  
    public int soma(int valor) {  
        int total = -1;  
        try {  
            Socket sock=new Socket(host, port);  
  
            ...  
        }  
    }  
}
```

Rede

```
public class Calculadora implements Somador {  
    private int total=0;  
  
    public int soma(int valor) {  
        total=total+valor;  
        return total;  
    }  
  
    ...  
}
```

```
try {  
    Somador s=new Calculadora();  
  
    ServerSocket ssock=new ServerSocket(12345);  
    while(true) {  
        Socket sock=ssock.accept();  
        ObjectOutputStream oos=new  
            ObjectOutputStream(sock.getOutputStream());  
        ObjectInputStream ois=new  
            ObjectInputStream(sock.getInputStream());  
  
        String op=(String) ois.readObject();  
        if (op.equals("soma")) {  
            int valor=(Integer) ois.readObject();  
            int total=s.soma(valor);  
            oos.writeObject(total);  
        } else if (op.equals("zero")) {  
            s.zero();  
        }  
  
        oos.close();  
        ois.close();  
        sock.close();  
    }  
} catch (Exception e) {  
    e.printStackTrace();  
}
```



Referências a Objectos

```
public class Proxy implements Somador, Serializable {  
    ...  
}
```

- Quais as implicações de fazer marshalling de um proxy?
 - Podemos passar um proxy como parâmetro numa invocação remota
- Como resultado obtemos um sistema de **objectos distribuídos**:
 - Qualquer nó do sistema pode alojar objectos
 - Qualquer nó pode invocar objectos



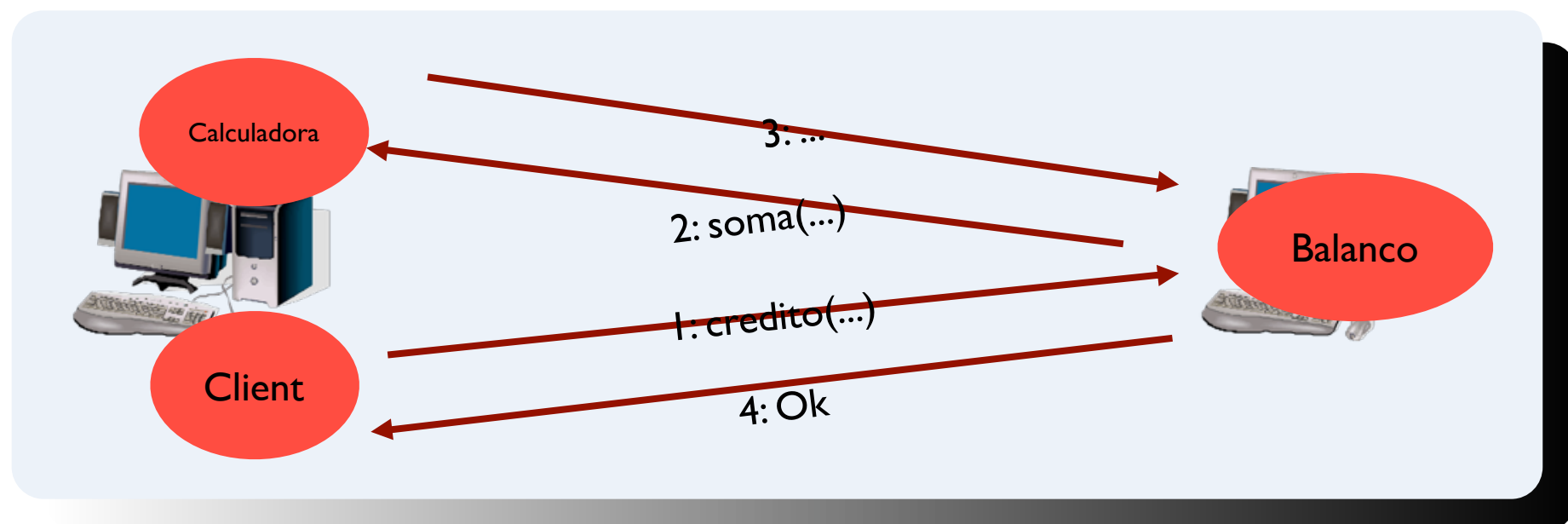
Referências a Objectos

- Exemplo de passagem de uma referência como parâmetro:

```
public interface Balanco {  
    public void debito(Somador soma);  
    public void credito(Somador soma);  
};
```

```
Somador s=new ProxySomador("localhost", 12345);  
Balanco b=new ProxyBalanco("outro", 12346);  
  
b.credito(s);
```

```
public class ImplBalanco implements Balanco {  
    ...  
    public void credito(Somador s) {  
        for(int i: creditos)  
            s.soma(i);  
    }  
};
```



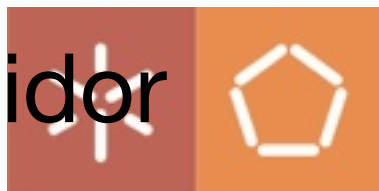
Referências a Objectos

- Se não há mais referências para o objecto, ele deve ser eliminado para não desperdiçar recursos
- Se o objecto é eliminado enquanto ainda há referências, os clientes receberão um erro
- Novo problema:
 - Quantas referências existem num determinado momento para um objecto?



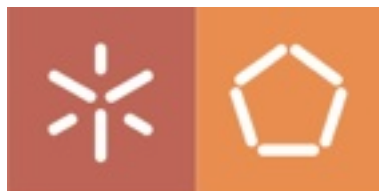
Garbage Collection

- Cada servidor mantém uma lista de proxies:
 - Cada vez que um proxy é criado, o servidor é contactado para o adicionar à lista
 - Quando um proxy é eliminado o servidor é contactado para o eliminar também
- Quando a lista fica vazia, o skeleton e o objecto são eliminados
- Opções de tolerância a faltas:
 - As operações de actualização da lista do servidor são idempotentes
 - Os clientes têm que avisar periodicamente o servidor que ainda têm um proxy



Serviços de Directoria

- Considere:
 - Um servidor de um mapa entre nomes e objectos (i.e. implementação, skeleton e dispatcher) colocado numa localização bem conhecida
 - Um proxy correspondente que saiba encontrar o servidor
- O resultado é uma **director**ia de objectos que pode ser usado para obter transparência de localização



Transparência de Localização

- Acesso a um objecto remoto:

```
Directoria dir=new ProxyDirectoria();  
  
Somador s=(Somador)dir.get("o meu somador");  
  
int v=s.soma(123);  
  
System.out.println(v);
```

```
Somador s=new Calculadora();  
Directoria dir=new ProxyDirectoria();  
dir.put("o meu somador",  
       new Proxy("host.uminho.pt", 12345));
```

...

Este servidor corre em
host.uminho.pt

- Note que o cliente não tem conhecimento do nome da máquina onde está o objecto



Transparência de Localização

- Acesso a um objecto local:

```
Directoria dir=new ProxyDirectoria();
```

```
Somador s=(Somador)dir.get("o meu somador");
```

```
int v=s.soma(123);
```

```
System.out.println(v);
```

```
Somador s=new Calculadora();
```

```
Directoria dir=new ProxyDirectoria();  
dir.put("o meu somador", s);
```

```
...
```

- O cliente continua inalterado



Transparência de Migração

- Acesso a um objecto que migrou:

```
Directoria dir=new ProxyDirectoria();  
  
Somador s=(Somador)dir.get("o meu somador");  
  
int v=s.soma(123);  
  
System.out.println(v);
```

```
Somador s=new Proxy("outro.uminho.pt", 12345);
```

...

Este servidor corre em
host.uminho.pt

```
Somador s=new Calculadora();  
Directoria dir=new ProxyDirectoria();  
dir.put("o meu somador", s);
```

...

Este servidor corre
em outro.uminho.pt

- Os novos clientes não passam pelo host.uminho.pt



Conclusões

- A invocação remota usa:
 - passagem de mensagens
 - marshalling
 - stubs e skeletons, gerados a partir da interface
- Obtém-se:
 - transparência de acesso (stubs)
 - transparência de localização (directoria)
 - transparência de migração (reencaminhamento)
- Não se obtém transparência de falha

