Computação Distribuída – Cap. IV

Licenciatura em Engenharia Informática

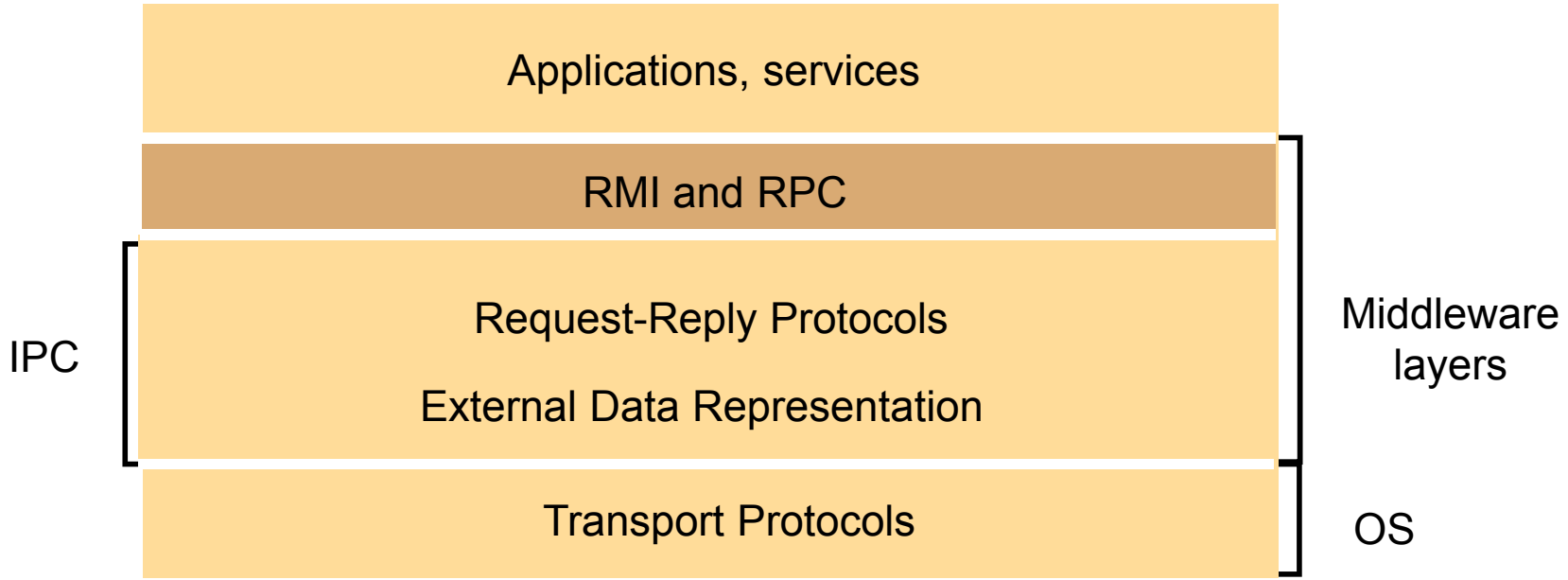
Universidade Lusófona

Prof. José Rogado
Prof. José Faísca

Invocação Remota e Objectos Distribuídos

- RPC: Modelo de execução
- Linguagem de definição de interfaces e passagem de parâmetros.
- Registo e descoberta de interfaces
- Plataforma de execução: Sun RPC
- Objectos Distribuídos: Modelo de Execução
- Plataformas de Execução (Java RMI, Corba)
- Nomeação e Serviços de Directório

Modelos de Programação



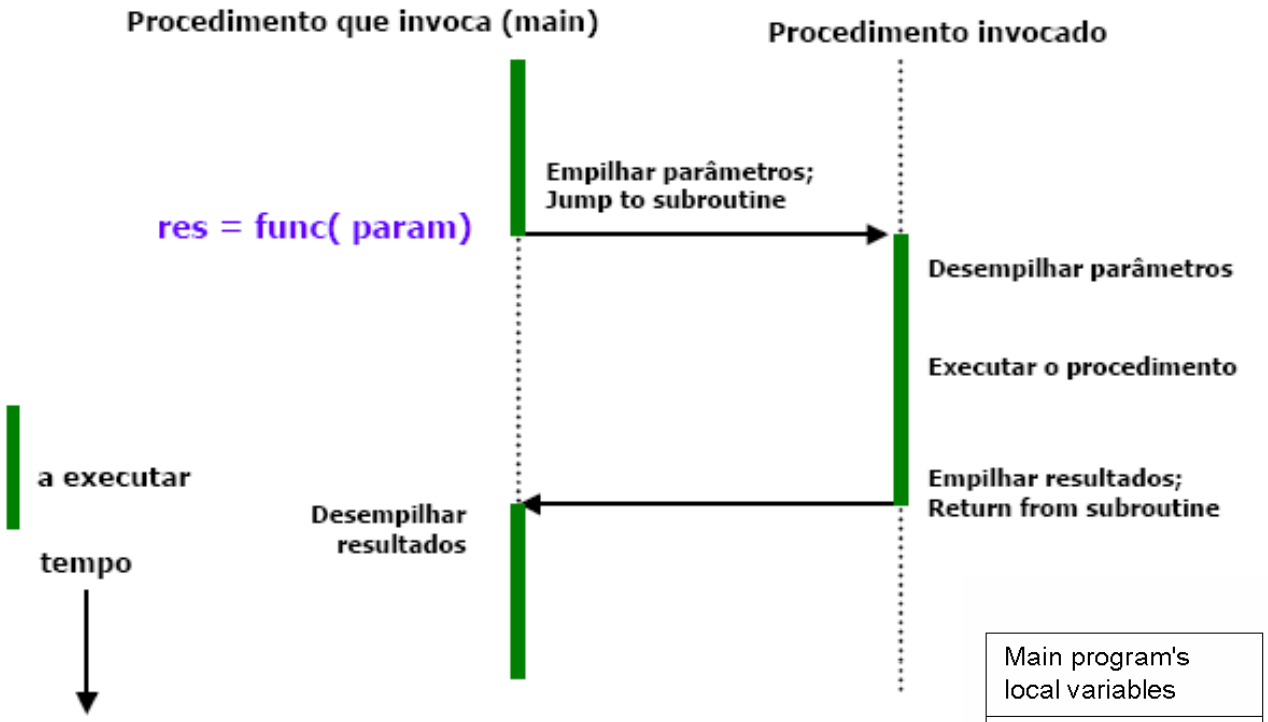
- O nível de invocação fornece um modelo de programação para as aplicações distribuídas
  - Extensão do modelo clássico de invocação de rotinas ou métodos
  - Implementação real da transparência de acesso e localização
- Modelos de programação
  - *Remote Procedure Call* - RPC
  - *Remote Method Invocation* - RMI

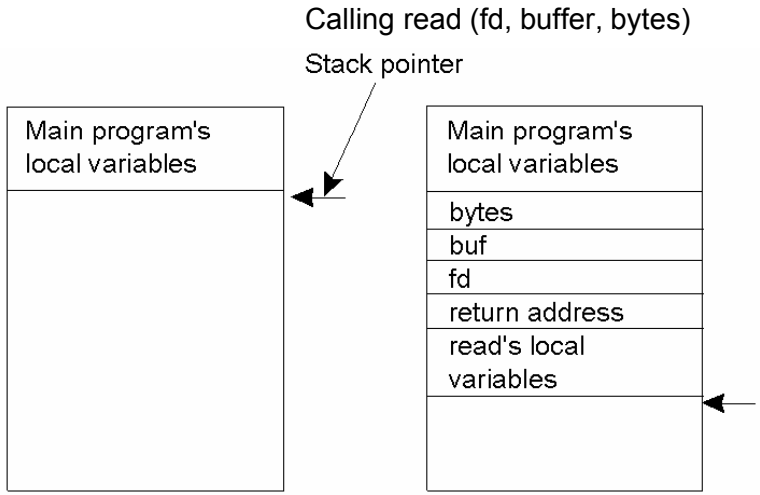
Invocação de Procedimentos Remotos

- O nível de RPC abstrai o nível de programação IPC
  - Esconde a complexidade da criação de ligações
  - Permite a estruturação das mensagens em função da aplicação
- No servidor, permite
  - Registar o serviço
  - A selecção da função a executar -> método
  - A extracção dos parâmetros relevantes das mensagens
  - O tratamento de erros e timeouts
- No cliente, permite
  - Procurar o ponto de acesso do serviço
  - Invocar a funcionalidade remota
  - O encapsulamento dos parâmetros de chamada nas mensagens
  - A recepção dos resultados da invocação
- Globalmente permite facilitar e automatizar as interacções entre cliente e servidor

Mecanismos de Invocação Local

Fonte: Sistemas Distribuídos, J. Legatheaux, N. Preguiça Martins, Universidade Nova, http://asc.di.fct.unl.pt/sd1

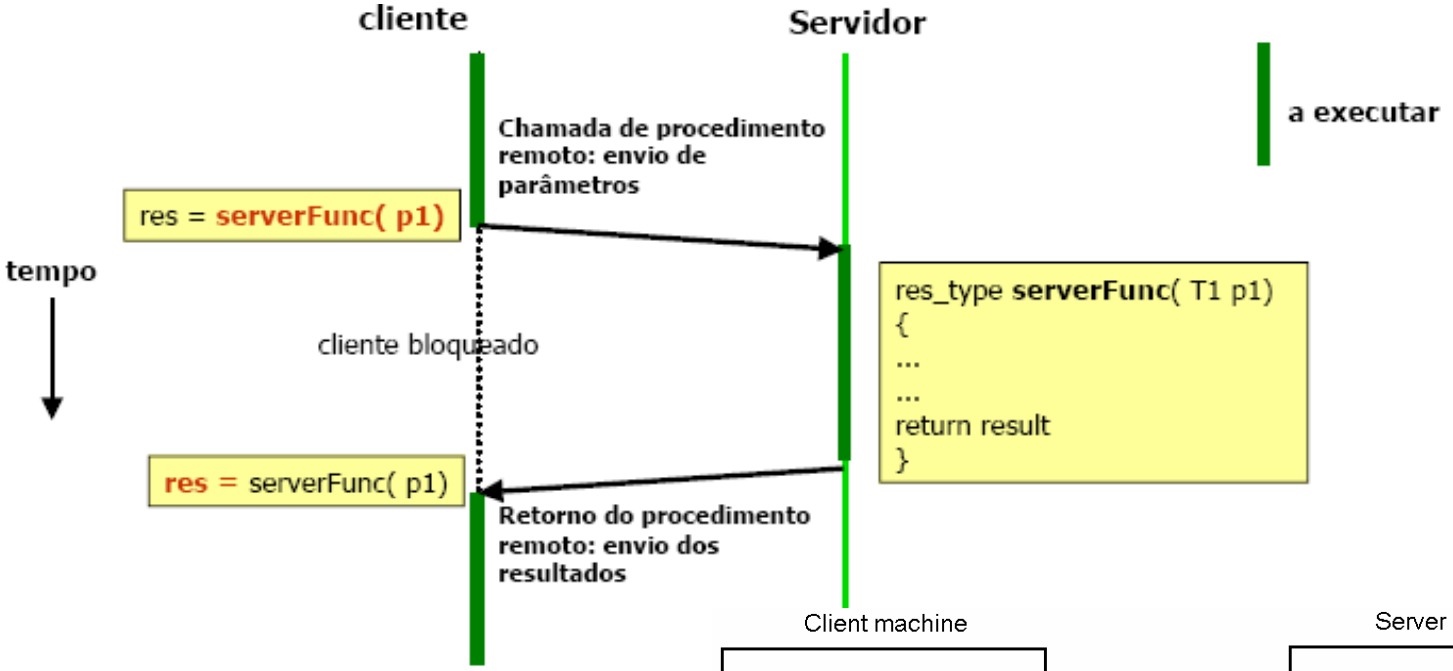




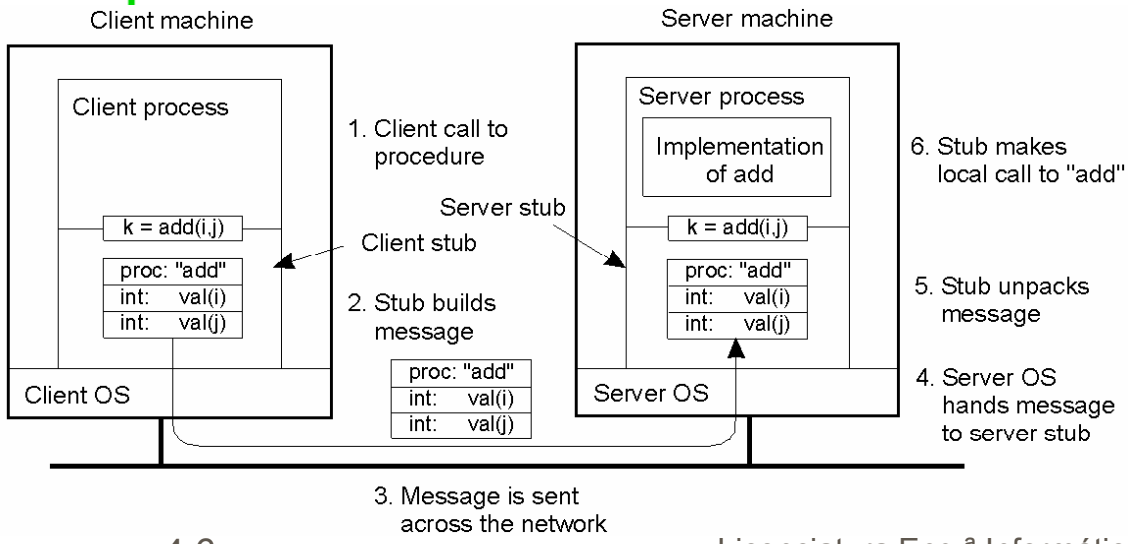
- O empillhamento e desempilhamento dos parâmetros é feito pelo compilador
- Faz parte da semântica de chamada da linguagem

Mecanismos de Invocação Remota

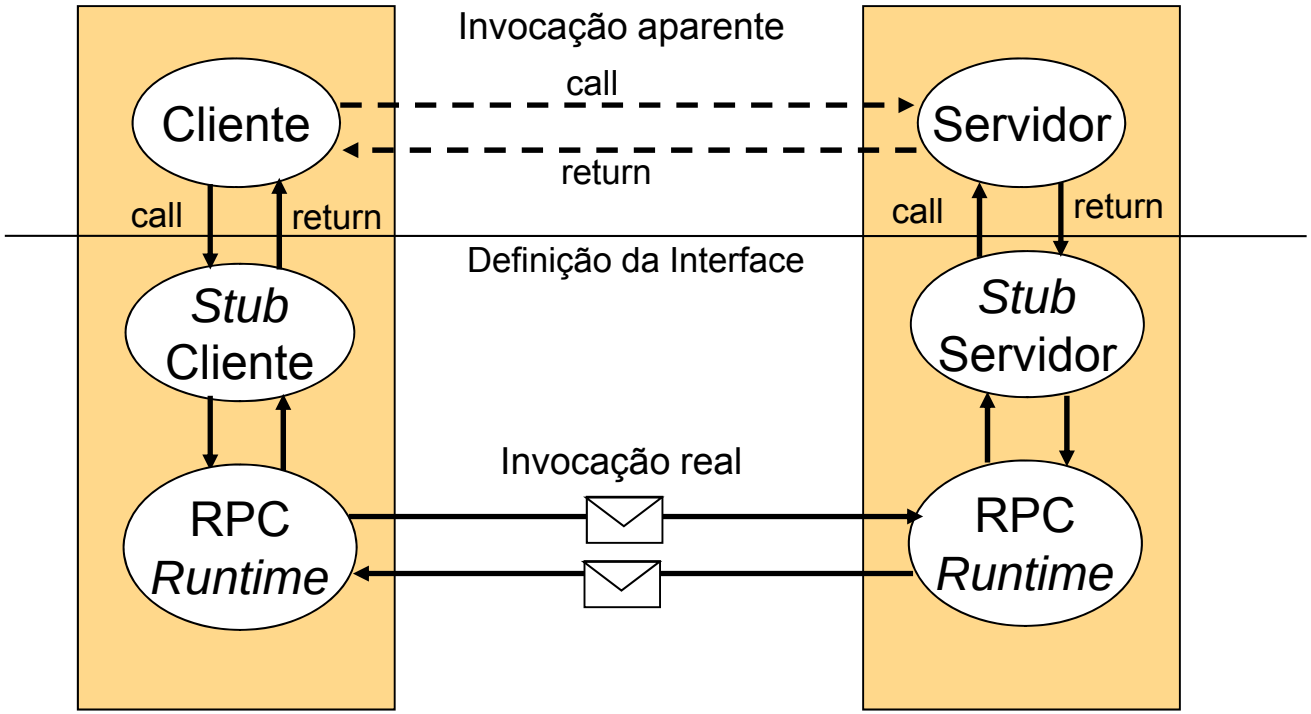
Fonte: Sistemas Distribuídos, J. Legatheaux, N. Preguiça Martins, Universidade Nova, http://asc.di.fct.unl.pt/sd1



- Envio dos parâmetros, invocação do procedimento remoto e recepção do resultado
- Realizados pelo *runtime* do RPC

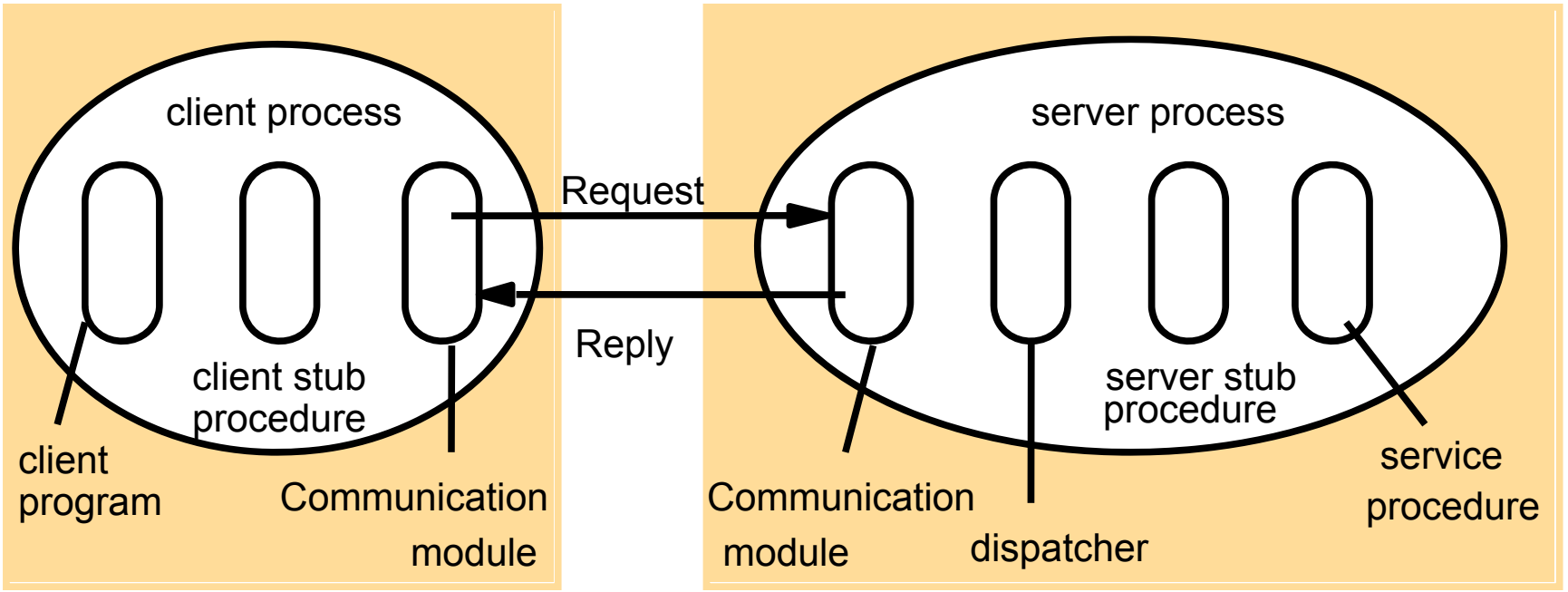


Sequência de Invocação de um RPC



- A sequência de invocação de um método remoto é implementada por vários módulos de software
  - *Client Stub*: funciona como o agente de representação (proxy) do método remoto
  - *Server Stub*: realiza a chamada no contexto do servidor ao método invocado
  - Os *runtimes* do RPC asseguram o encaminhamento das mensagens e o *dispatching* destas para o *stubs*

Módulos do *Runtime* RPC



- *Client Stub*:
  - Chamada: retira os parâmetros da pilha passa-os para um formato independente na mensagem
  - Resultado: passa o resultado da mensagem para a pilha de invocação
- *Dispatcher*
  - Recebe a mensagem no servidor e direcciona-a para o método invocado
- *Server Stub*:
  - Chamada: retira os parâmetros da mensagem e coloca-os na pilha para invocar o procedimento do serviço
  - Resultado: retira o resultados da pilha e passa-os para um formato independente na mensagem
- A coerência entre estas acções é assegurada pela definição formal da *Interface*

O Sun RPC

- O Sun RPC foi um dos primeiros exemplos de *middleware* para programação de aplicações distribuídas
- Desenvolvido para a implementação do sistema de ficheiros distribuído NFS - *Network File System*
  - Meados dos anos 80
  - Código posto no domínio público permitiu uma enorme expansão
  - Durante anos foi um standard *de facto* para aplicações distribuídas
- Plataforma simples e rudimentar
  - Inicialmente só suportava UDP, depois também TCP
  - Semântica de invocação *at-least-once*
- Fornece alguns serviços complementares
  - Autenticação
    - Unix style, Kerberos
  - Broadcast
    - Feita através do *Port Mapper*

Definição de Interfaces (i)

- A interface especifica o formato e o tipo de interacção fornecido por uma função ou objecto
  - Define as funcionalidades de um serviço
- A interface define os parâmetros que são fornecidos à rotina invocada
  - Passagem por valor: definição de parâmetros de entrada e de saída
  - Passagem por referência: impossível se não está no mesmo processo que o programa principal
    - Pode ser simulada através do contexto de chamada dos *stubs*
    - Ponteiros não têm significado
- A especificação de interfaces no contexto de RPC é feita utilizando uma linguagem de definição
  - IDL: *Interface Definition Language*
  - Integrada com a linguagem de invocação
    - RPC SUN: C, C++, C# e Java

Definição de Interfaces (ii)

- O mecanismo de RPC localiza o serviço através do identificador das interfaces
  - Identificador único
  - Mecanismo bastante rudimentar
- No Sun RPC o identificador é composto por 3 campos:
  - *Program number* -> identifica um grupo de procedimentos fornecidos por um serviço
  - *Version Number* -> identifica a versão do serviço
  - *Procedure Number* -> identifica o método ou procedimento
- O Identificador é atribuído pelo programador do serviço
  - Gama de identificadores reservados
  - 20000000 - 3FFFFFFF atribuídos ao utilizador
  - Ver detalhes em:
    - http://docs.sun.com/app/docs/doc/816-1435/6m7rrfn76?a=view

Exemplo de Definição de Interface

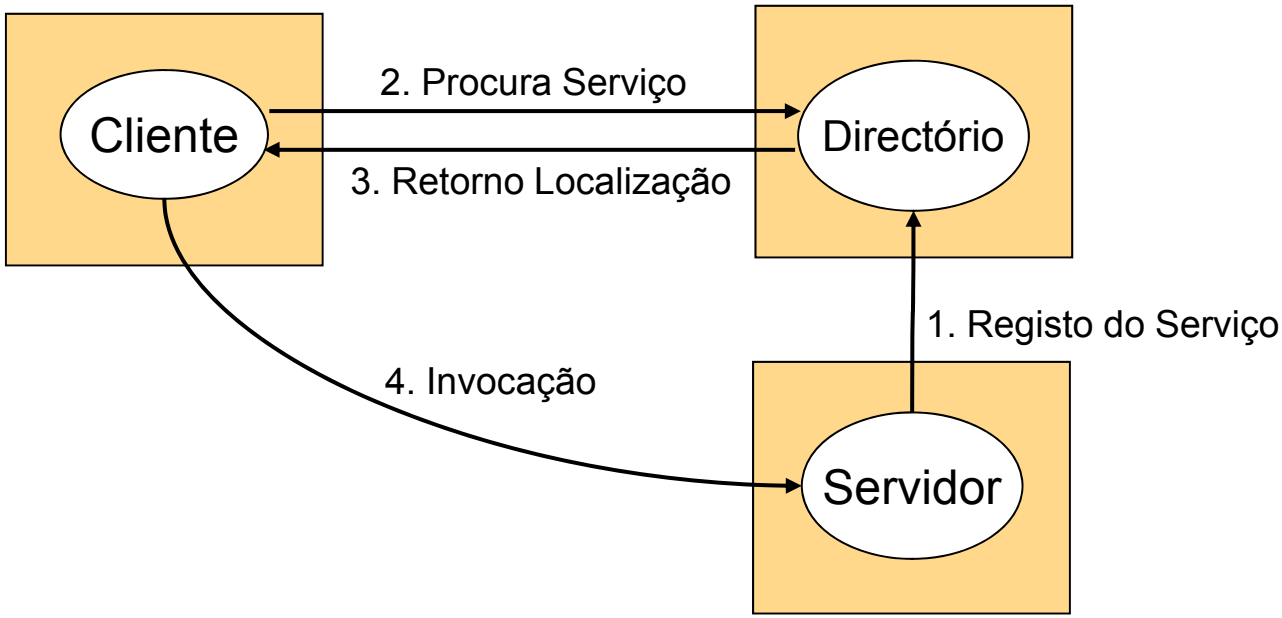
```
const MAX = 1000;

typedef int FileIdentifier;
typedef int FilePointer;
typedef int Length;

struct Data {
    int length;
    char buffer[MAX];
};
struct writeargs {
    FileIdentifier f;
    FilePointer position;
    Data data;
};
struct readargs {
    FileIdentifier f;
    FilePointer position;
    Length length;
};
program FILEREADWRITE {
   version VERSION {
        void WRITE(writeargs) = 1;
        Data READ(readargs) = 2;
   } = 2;
} = 0x20000100;
```

Identificador do Método

Identificador do Serviço

- Serviço de Acesso a Ficheiros Remotos:
  - Program FILEREADWRITE id = 0x20000100
  - Version 2
- Fornece dois métodos:
  - Write id = 1
  - Read id = 2
- Argumentos
  - Definidos por estruturas
  - Sintaxe linguagem C
- Ver mais exemplos em
  - http://netlab.ulusofona.pt/cd/praticas/rpc

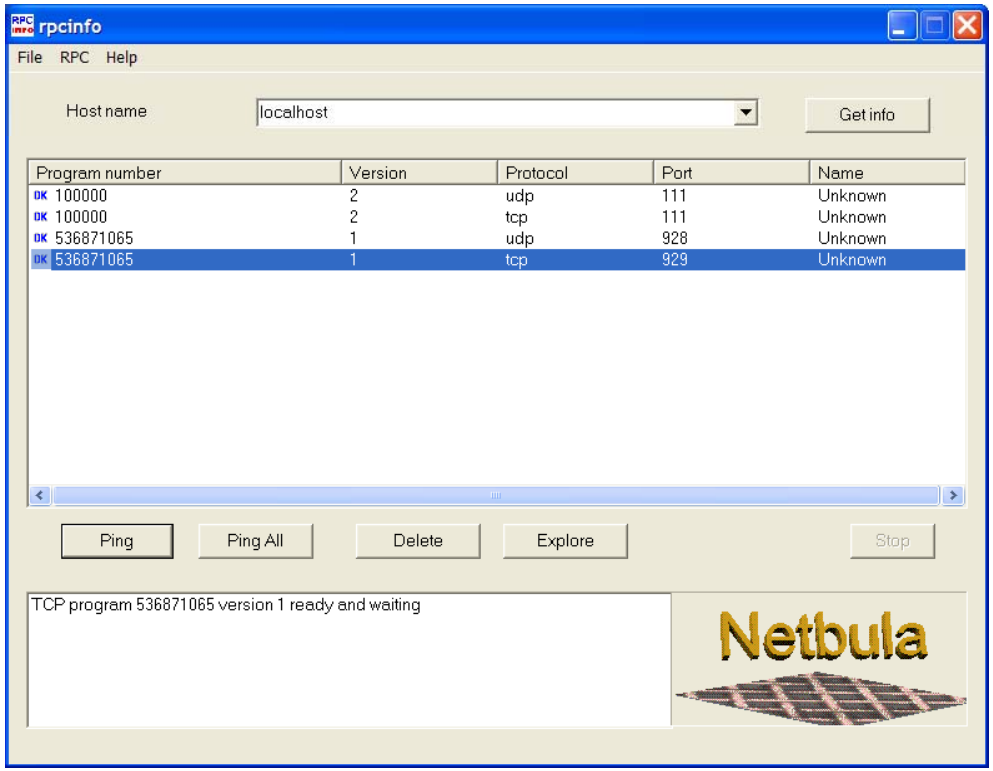
Registo e Descoberta (*binding*)



- A invocação de serviços implica a existência de um serviço de registo e descoberta
- O cliente tem de conhecer previamente a localização do Directório
- O processo é realizado em 4 passos
  1. O servidor regista-se no Directório de Serviços
  2. O cliente procura o serviço pretendido no Directório
  3. O Directório devolve a localização do serviço
  4. O cliente invoca o serviço pretendido

Exemplo do *binding* do Sun RPC

- O registo e descoberta são assegurados pelo *Port Mapper*
- Servidor que fica è escuta num porto predeterminado
  - *Well-known port*: 111 UDP e TCP
  - Um novo serviço regista-se junto do Port Mapper
  - Um cliente pede ao Port Mapper o porto de escuta do serviço



- Exemplo de Registo de Serviços
  - 100000: *Port Mapper*
  - 536871065: Serviço Registado

Trabalho Complementar

- Ler o documento da Sun
  - http://docs.sun.com/app/docs/doc/816-1435/6m7rrfn76?a=view
- Analisar outro exemplo de RPC
  - DCE - *Distributed Computing Environment*
    - http://www.opengroup.org/dce
  - Windows RPC
    - http://msdn.microsoft.com/en-us/library/aa378651(VS.85).aspx
- Realização de um serviço RPC
  - Implementação do cliente e do servidor demonstrando a utilização dos mecanismos apresentados
  - Utilizar por exemplo o interface para uma calculadora definido em;
    - http://netlab.ulusofona.pt/cd/praticas/rpc/calc

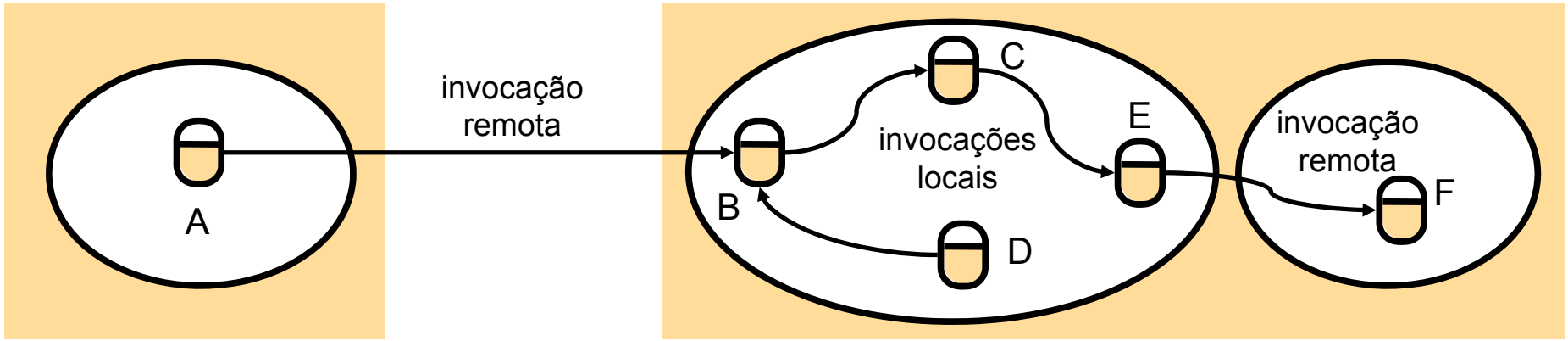
Modelo de Programação por Objectos

- Um **objecto** é uma entidade que encapsula dados e operações sobre os mesmos – ***métodos***
  - Corresponde à instanciação de uma ***classe***
- Os objectos são identificados e acedidos através de ***referências***
  - As referências podem ser afectadas a variáveis, passadas como parâmetros ou devolvidas pela execução de métodos
- Uma ***interface*** define a assinatura dos métodos de um objecto
  - Argumentos, tipos, valores de retorno e excepções
  - Uma classe pode implementar diversas interfaces
- A ***invocação*** de um método é uma ***acção*** que provoca uma mudança de estado no objecto invocado e o retorno de um valor
- A execução de um método pode provocar erros que geram ***excepções***
  - Forma elegante de gerir falhas sem ter de inserir testes em cada invocação
- A reciclagem de espaço memória libertada por objectos pode ser feita explícita ou implicitamente por ***Garbage Collection***

Objectos Distribuídos

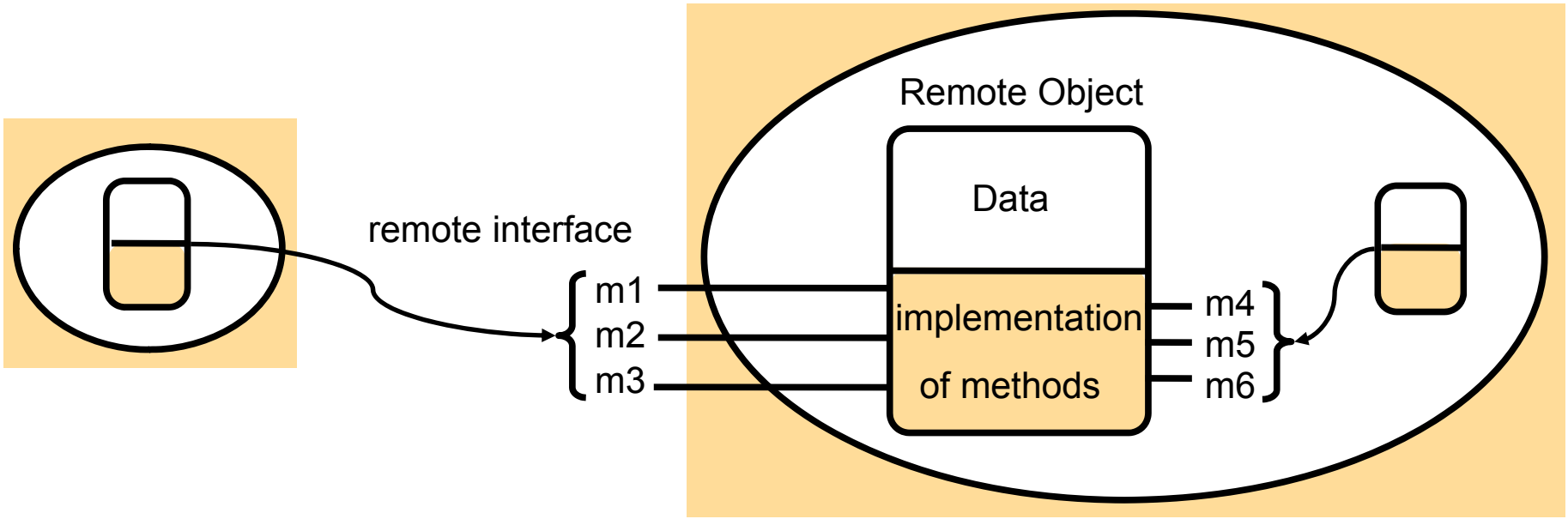
- Extensão do modelo a objectos não residentes no mesmo processo
  - Utilização de um Mecanismo de Invocação Remota de Métodos
    - A acção de invocação do método remoto é inserida numa mensagem
    - RMI: ***Remote Method Invocation*** - Modelo Cliente / Servidor
  - A invocação pode também utilizar outros modelos
    - Objectos replicados ou migrantes
- A encapsulação em objectos aumenta a modularidade das funcionalidades dos serviços
  - Funcionalidades acedidas só através dos métodos predefinidos
  - Favorece a concorrência e isolamento de falhas

Modelo de Objectos Distribuídos



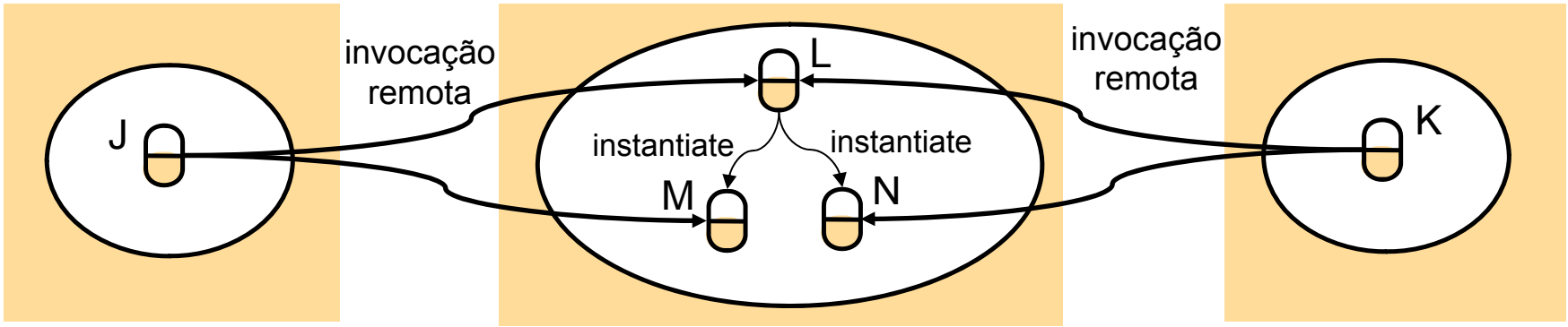
- O objecto A invoca um método do objecto remoto B
- Objectos remotos podem receber invocações locais ou remotas
  - Os processos podem residir ou não no mesmo servidor
- Para haver invocação remota é necessário
  - Possuir uma *referência* para o objecto remoto
    - Extensão da referência local de um objecto válida no contexto de um sistema distribuído
  - Conhecer a *definição* da interface do objecto remoto
    - Só podem ser invocados remotamente os métodos definidos na interface

Interfaces Remotas



- A classe que define um objecto remoto implementa os métodos da interface remota
- As interfaces são definidas através de uma linguagem de definição
  - ex.: Corba IDL
- Ou utilizando a própria linguagem de objectos
  - ex.: Java, por extensão da interface *remote*

Instanciação Remota

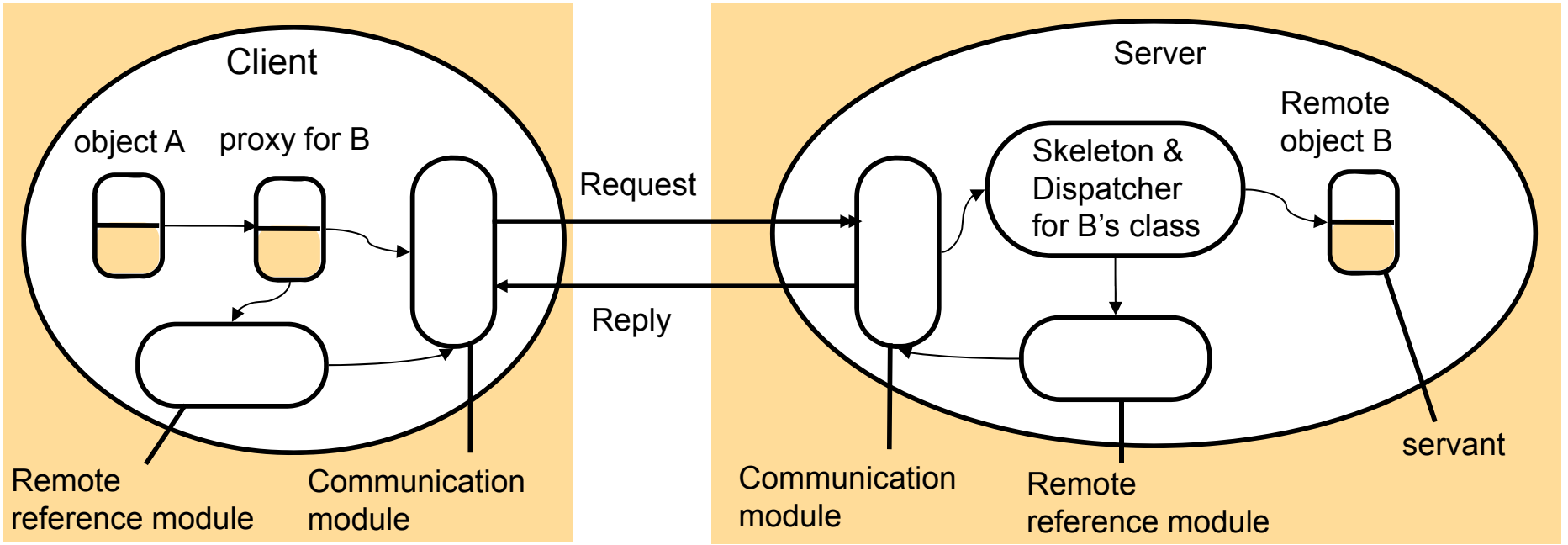


- Para que uma invocação seja efectuada, o objecto remoto tem de existir
  - Tem de ser instanciado no servidor e a sua referência exportada
- Uma invocação remota pode dar origem a uma instanciação de objecto
  - A classe pode ter um método construtor invocável remotamente
  - O objecto instanciado reside no servidor e pode ser invocável remotamente por outros objectos se a sua referência for passada ou exportada
- Os objectos remotos não referenciados devem ser libertados por *Garbage Collection* distribuído (se a linguagem o suportar)
- O mecanismo de RMI tem de garantir a propagação de excepções para o objecto que invoca remotamente para manter a mesma semântica da invocação local

Transparência

- O objectivo da invocação remota de métodos é esconder a natureza distribuída do sistema
  - Tornar transparente para o programador a utilização do protocolo de transporte e a linearização dos parâmetros de invocação
- Contudo não é desejável esconder totalmente a natureza remota da invocação
  - A distribuição introduz mais factores de falhas que a aplicação deve saber gerir
    - Excepções remotas
  - Existem grandes diferenças a nível temporal que podem condicionar o desempenho das aplicações
    - Latência do canal
    - Timeouts
- É portanto desejável que o facto de estar a invocar um método remoto não seja completamente escondido
  - Mesma sintaxe de invocação
  - Definição, encadeamento e mecanismo de invocação explícitos
  - Ex.: Java RMI
    - *Throws RemoteException*
    - *Implements Remote*

Implementação do RMI

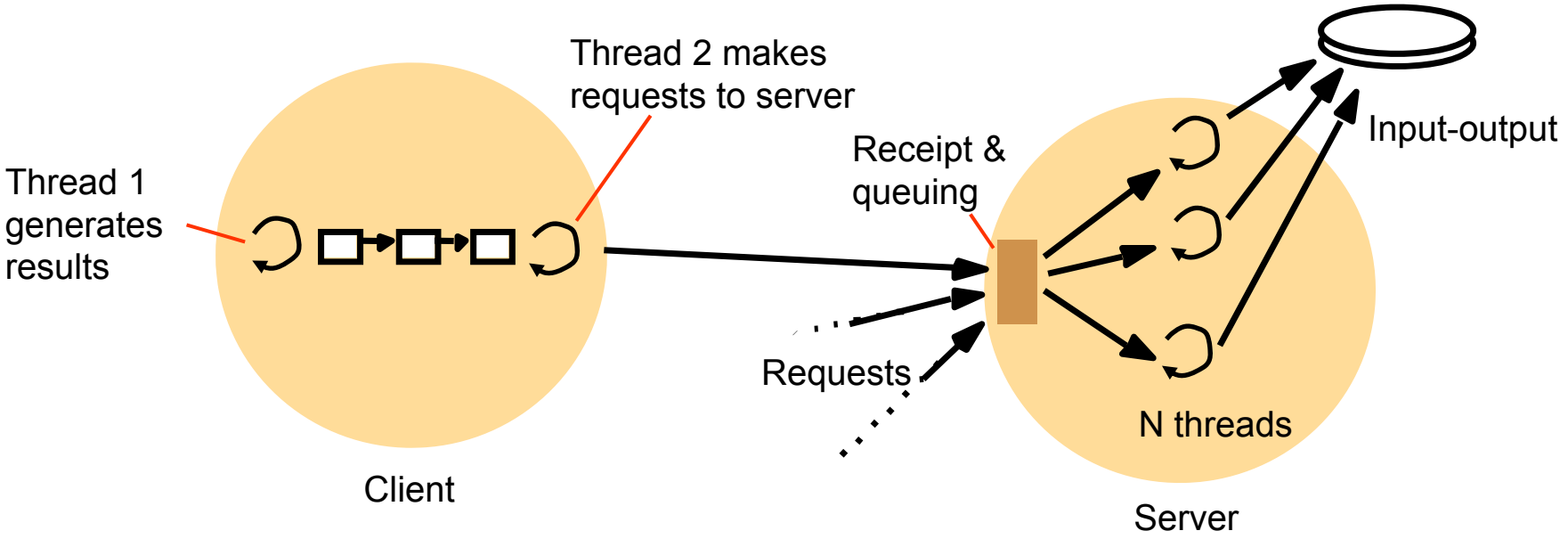


- *Proxy*
  - Representação local do objecto remoto (≡ *client stub rpc*)
- *Skeleton*
  - Classe de invocação do método remoto (≡ *server stub rpc*)
- *Dispatcher*
  - Cada classe tem um *dispatcher* que selecciona o método a invocar a partir do identificador
- *Remote Reference Module*
  - Realiza o mapeamento entre referências de objectos locais e remotos, contendo tabelas ligando os *proxies* aos *skeletons*
- Os *proxy, skeleton* e *class dispatcher* são gerados pelo compilador de interfaces

Funcionalidades Adicionais

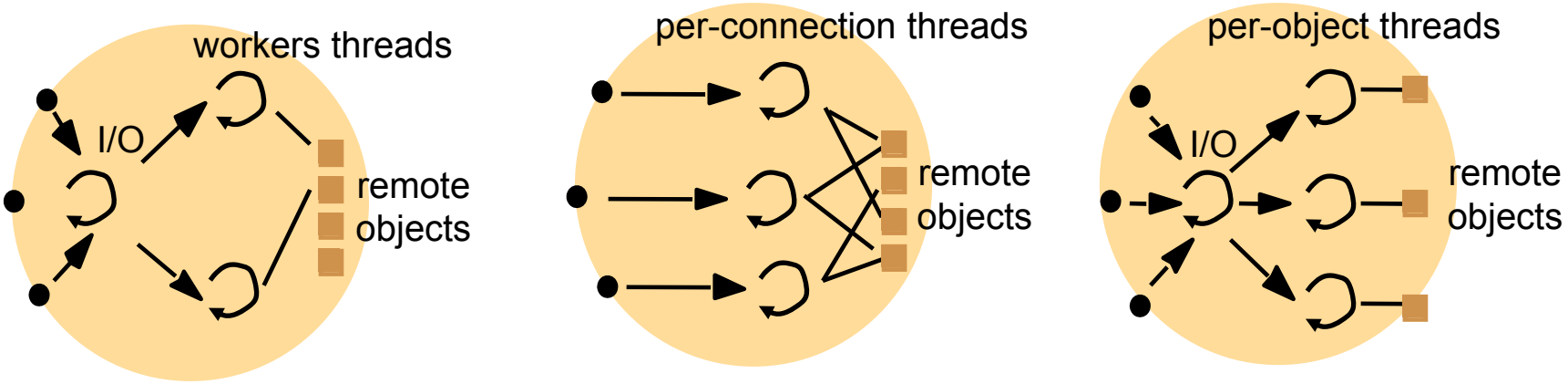
- Invocação Dinâmica de Interfaces (DII)
  - Permite invocar objectos cujas interfaces são desconhecidas *a priori*
  - Utilização directa da operação de invocação de operação remota (callRemote) e da construção dinâmica da mensagem a partir da definição da interface
    - A definição da interface pode ser obtida dinamicamente por *reflection* em Java RMI ou através do *Interface Repository* em Corba.
- Activação de Objectos Remotos
  - Possibilidade de manter objectos "adormecidos" e de os reactivar quando são invocados, no estado em que foram guardados (*snapshot*)
  - Os objectos adormecidos são designados por persistentes e armazenados serializados em repositórios de objectos
- *Garbage Collection* Distribuída
  - Garante que se não houver referências remotas para um objecto os seus recursos são reciclados
  - Envolve a contabilização das referências de objectos mantidas nos proxies de clientes remotos
  - Utiliza a noção de sessão, finda a qual a referência é reciclada

Arquitectura de Invocação



- Geralmente as aplicações distribuídas utilizam *threads* no cliente e no servidor
  - A programação com threads exige cuidados especiais para evitar acesso concorrente a recursos partilhados
  - Vários modelos de utilização de threads são possíveis
- No servidor são geralmente utilizadas uma thread de recepção de mensagens e um conjunto de threads (worker threads) para realizar as invocações

Várias Soluções no Servidor

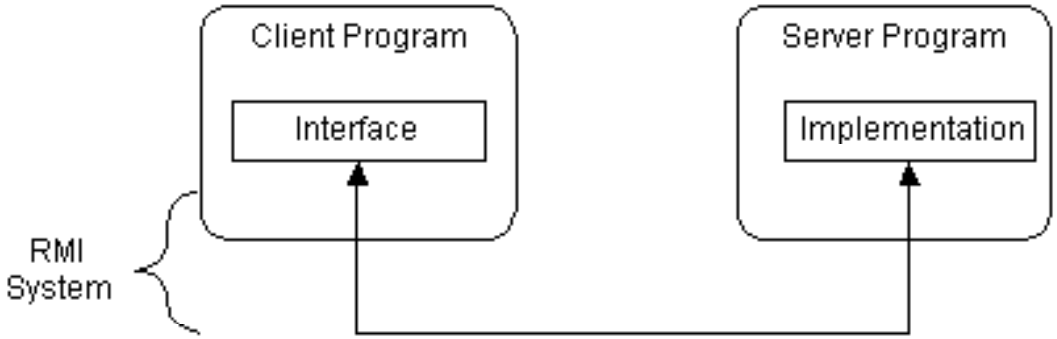


a. Thread-per-request b. Thread-per-connection c. Thread-per-object

- *Thread-per-request*: cada invocação é executada por uma *thread* é depois destruída
  - Concorrência no acesso aos objectos
  - Overhead de criação e destruição de threads
- Thread-per-connection: as invocações de uma conexão são executadas pela mesma *thread*
  - Concorrência de acesso aos objectos
- Thread-per-object: as invocações para um objecto são executadas pela mesma *thread*
  - Possibilidade de uma *thread* associada a um objecto não ter pedidos e outras terem muitos

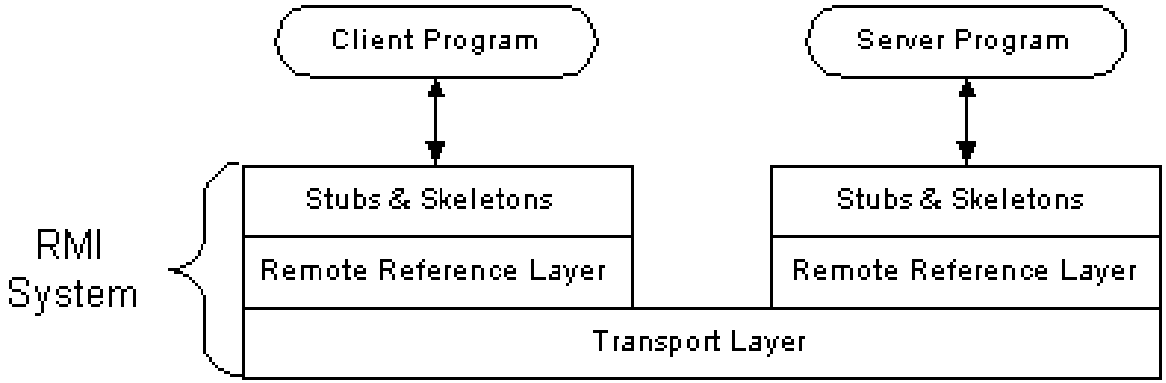
Java RMI

- Referência: Tutorial da Sun (Oracle) sobre RMI
  - http://java.sun.com/developer/onlineTraining/rmi/RMI.htm
- Objectivo
  - Disponibilizar um modelo de objectos distribuídos naturalmente integrado com a linguagem de programação
- Princípios Básicos
  - Comportamento: definido por interfaces
  - Implementação: definida por classes



RMI: Arquitectura

- A arquitectura do RMI é baseada em 3 níveis de abstracção
  - Nível *Stubs & Skeletons*: realiza a interface com os módulos servidor e cliente da aplicação
  - Nível *Remote Reference*: realiza a correspondência e ligação entre referências de objectos remotos e locais
    - Inicialmente uma ligação ponto a ponto
    - Na versão Java 2 suporta activações de objectos adormecidos (*dormant*)
  - Nível Transporte
    - Baseado em ligações TCP persistentes



Stubs & Skeletons

- O Proxy representa o objecto invocado do lado do cliente
  - Fornece uma interface idêntica à que é implementada no servidor
  - Realiza a serialização dos argumentos e dos resultados
- O *Skeleton* é uma classe auxiliar do lado do servidor
  - Até à versão JDK 1.1 o *skeleton* é utilizado para receber as invocações, recupera os argumentos e invoca o método da implementação através do *dispatcher*
- A partir da versão JDK 1.2, a utilização de *skeletons* deixa de ser necessária
  - A invocação e serialização são feitas dinamicamente usando as funcionalidades de ***reflection***
  - O dispatcher é igualmente genérico para todas as classes remotas
- A partir do JDK 1.5, o proxy deixa igualmente de ser necessário utilizando o conceito de *dynamic proxy*

Nomeação de Objectos Remotos

- O registo e descoberta de serviços RMI é feita através de um serviço de directório chamado *RMI Registry*
  - Pode utilizar outros serviços de directório (JNDI -> LDAP)
- O *RMI Registry* (semelhante ao *Port Mapper*) regista e fornece informação sobre os serviços de um servidor
  - Acessível por TCP no porto 1099 (por defeito)
- *RMI Registry* acessível através da classe *Naming*
  - Formato do URL:

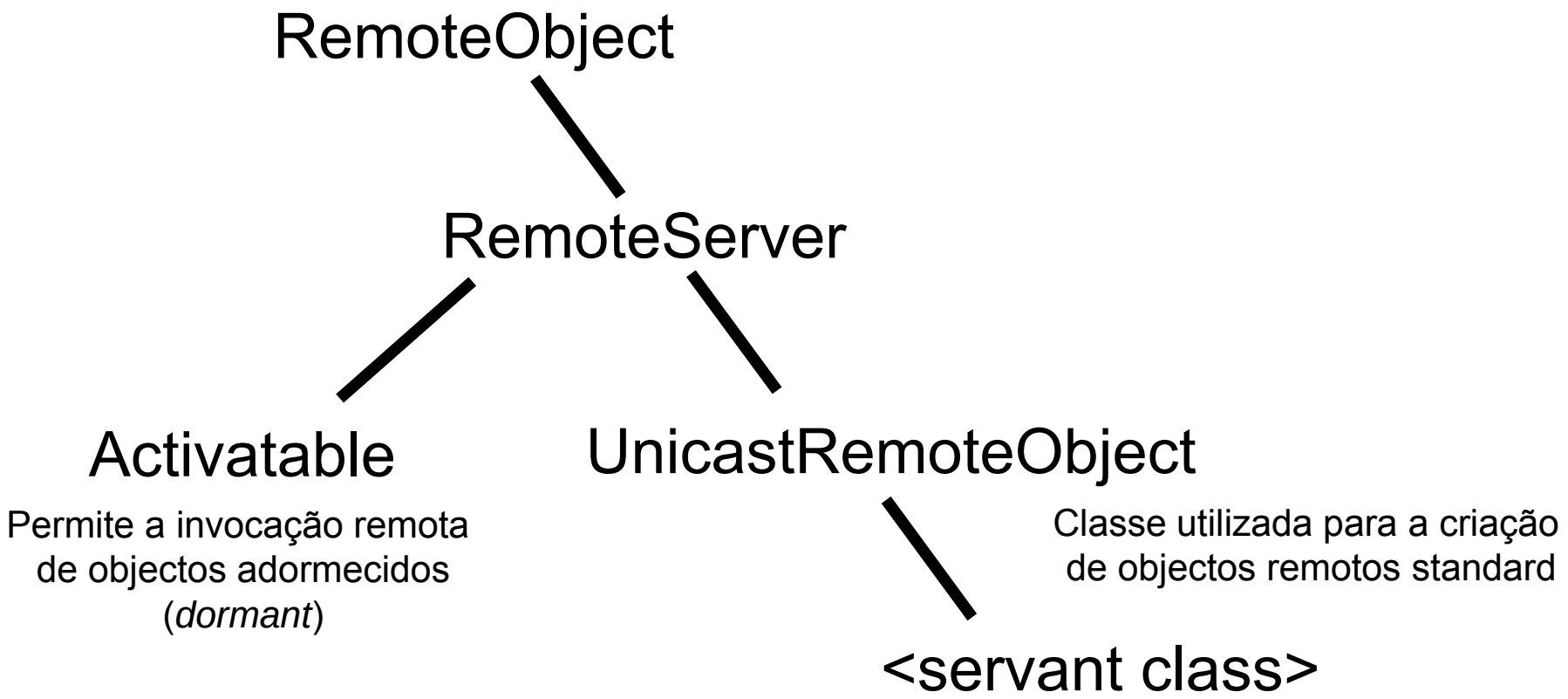
    ```
    rmi://<host_name> [:<name_service_port>] /<service_name>
    ```

  - Cliente: usa o método *lookup()* para procurar o serviço através do seu nome
  - Servidor: usa o método *rebind()* para registar o objecto remoto que implementa os métodos da interface

Interface, Implementação e Geração de *Stubs*

- A interface deve ser definida como uma extensão da classe *Remote*
  - `public interface <Name> extends java.rmi.Remote`
  - Deve conter a assinatura de todos os métodos implementados
- Deve ser fornecida uma implementação para cada método
  - A classe de implementação é uma extensão de uma classe que permite instanciar um objecto remoto de tipo *Unicast*
  - `public interface <Name> extends UnicastRemoteObject`
- A implementação deve prever excepções de tipo remoto
  - `throws java.rmi.RemoteException`
- A geração dos stubs (antes de JDK 1.2) é feita a partir da classe de implementação que realiza a definição da interface
  - `rmic <InterfaceImpl>`
- São gerados os ficheiros
  - `interfaceImpl_stub.class`
  - `interfaceImpl skeleton.class`

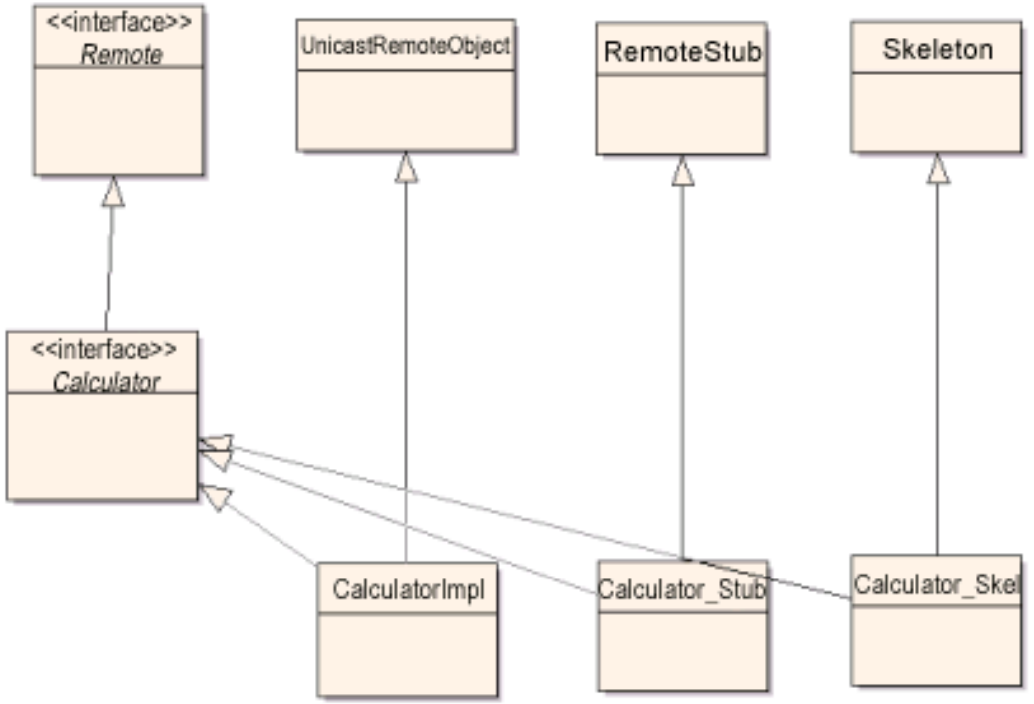
Hierarquia de Classes Remotas



Exemplo de Hierarquia de Classes

```java
import java.rmi.*;
import java.rmi.server.*;

public class CalculatorImpl extends UnicastRemoteObject implements Calculator {

    // Implementations must have an
    // explicit constructor
    // in order to declare the
    // RemoteException exception
    public CalculatorImpl() throws java.rmi.RemoteException {
        super();
    }

    public long add(long a, long b)
        throws java.rmi.RemoteException {
        return a + b;
    }

    public long sub(long a, long b)
        throws java.rmi.RemoteException {
        return a - b;
    }

    public long mul(long a, long b)
        throws java.rmi.RemoteException {
        return a * b;
    }

    public long div(long a, long b)
        throws java.rmi.RemoteException {
        return a / b;
    }
}
```

```java
public interface Calculator
          extends java.rmi.Remote {
    public long add(long a, long b)
        throws java.rmi.RemoteException;

    public long sub(long a, long b)
        throws java.rmi.RemoteException;

    public long mul(long a, long b)
        throws java.rmi.RemoteException;

    public long div(long a, long b)
        throws java.rmi.RemoteException;
}
```



Implementação do Cliente

- O cliente tem de incluir as classes de resolução de nomes
  - `import java.rmi.Naming`
- Deve inquirir o serviço para obter uma referência para o objecto remoto
  - `Ref = Naming.lookup(rmi://<host_name>[:<name_service_port>]/<service_name>`
- A partir da referência obtida, o cliente pode invocar os métodos do objecto remoto definidos na interface como se fossem locais
  - `Ex: remotecalc.add(3,4)`

Implementação do Servidor e Execução da Aplicação

- O servidor instancia a classe de implementação e regista o objecto criado no *RMI Register*
  - `Naming.rebind("rmi://<host>/<Service>", impl_obj)`
- Todas as classes devem ser compiladas
  - `javac File.java`
- O RMI Registry deve ser lançado
  - É necessário indicar onde estão armazenadas as classes compiladas
    - Variável de ambiente **`CLASSPATH`**
  - `rmiregistry`
- O servidor pode então ser lançado
  - Regista-se junto do RMIRegistry
  - O cliente pode invocar os seus serviços

Trabalho Complementar

- Ler um Tutorial mais avançado sobre RMI
  - http://download.oracle.com/javase/tutorial/rmi/index.html
- Ler o artigo "RMI, Dynamic Proxies and the Evolution of Deployment"
  - http://today.java.net/pub/a/today/2004/06/01/RMI.html?page=1
- Modificar o exemplo fornecido
  - Tornar o cliente capaz de ler as operações e os valores da linha de comandos
  - Adicionar novo método para implementar a operação "módulo"
- Exemplos no Netlab
  - http://netlab.ulusofona.pt/cd/praticas/rmi

Sistemas de Nomeação

- A identificação de recursos num ambiente distribuído é essencial para permitir a sua localização
  - Ficheiros
  - Serviços
  - Objectos
- É necessário um modo de designação
  - Nome: independente do sistema (geralmente em formato textual)
  - Identificador: referência interna do sistema (ex: formato binário)
- A relação entre um nome e um recurso é designada por associação - *name binding*
- A tradução do seu nome para um identificador é designada por resolução - *name resolution*
- Um nome pode ser constituído por uma série de campos com domínios de resolução distintos

Tipos de Nomes

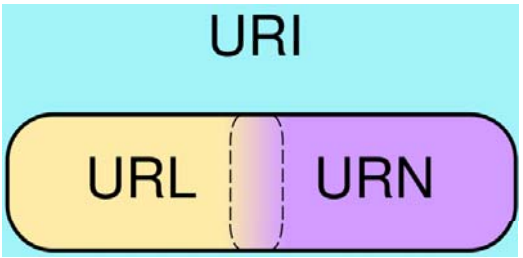
- Um nome está associado a um espaço de designação
- Nomes textuais
  - São nomes simbólicos orientados para o utilizador, facilmente memorizáveis
    - Exemplo: http://netlab.ulusofona.pt/cd/teoricas/index.php
- Identificadores únicos
  - UIDs: Unique Identifiers usados internamente pelos sistemas
    - FF.33.2B.45.8A.5F.2C.35.28.5E.29 designa um serviço no contexto de um servidor Java a correr numa JVM
    - Não faz sentido no contexto de um outro sistema
- Endereço
  - Forma particular de nomeação permitindo acesso directo ao recurso
    - 192.168.15.10 no contexto da rede interna do Laboratório
    - 21 757 70 06 designa um cliente da rede PSTN de Lisboa no contexto de numeração nacional
    - 0x0a005c8e endereço memória no contexto de um processo

Espaços de Nomeação

- Um *espaço de nomeação - name space* é o conjunto de todos os nomes válidos, reconhecidos e resolúveis por um determinado serviço
  - Num espaço de nomeação existe uma *única* autoridade administrativa (possivelmente replicada) que atribui e resolve nomes
    - 6.8.1.6.6.9.8.1.2.1.5.3.e164.arpa corresponde ao nº telefone +351 21 8916186 no ENUM - Electronic Number Mapping System
    - /home/alunos/a1234567/trabalho4 é um nome no espaço de nomeação Unix
  - Um espaço de nomeação tem uma sintaxe específica
- Um espaço de nomeação pode ser
  - Organizado hierarquicamente (DNS, UFS, XMLNS, X-500)
  - Plano ou linear (ex: sistema de nomeação humano)
- Um serviço de nomeação implementa a funcionalidade de resolução de nomes dentro de um determinado espaço de nomeação

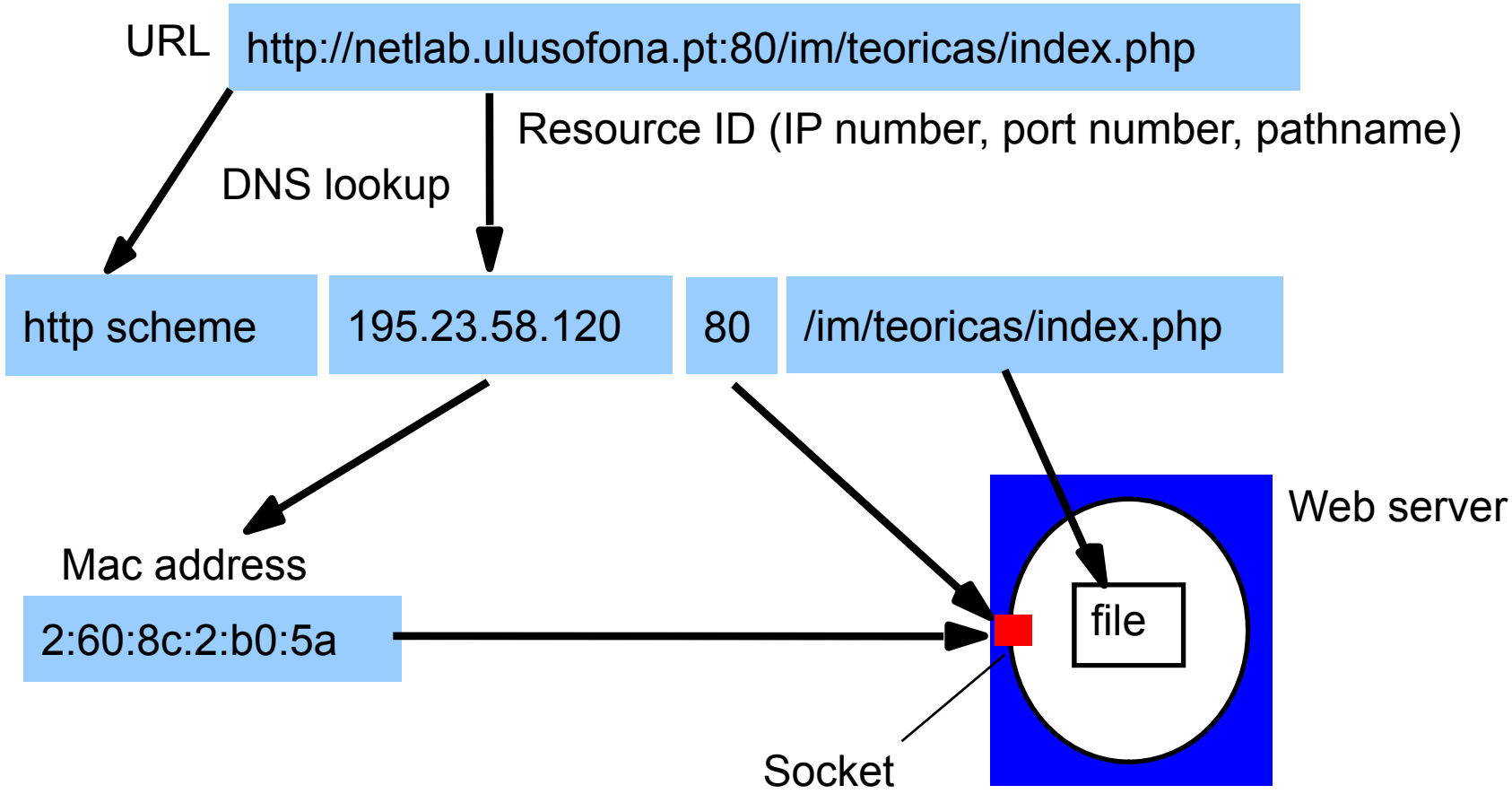
URIs: *Uniform Resource Identifiers*

- Sistema de designação que permite identificar recursos na Internet
  - Uniforme porque incorpora uma série de sistemas de designação
    - URI *scheme* - scheme:scheme-specific-name
  - Para cada *scheme* existem procedimentos específicos para a resolução do seu espaço de nomes
    - http://www.service.cc
    - ftp://server.school.edu
    - rmi://somehost/rmi-service



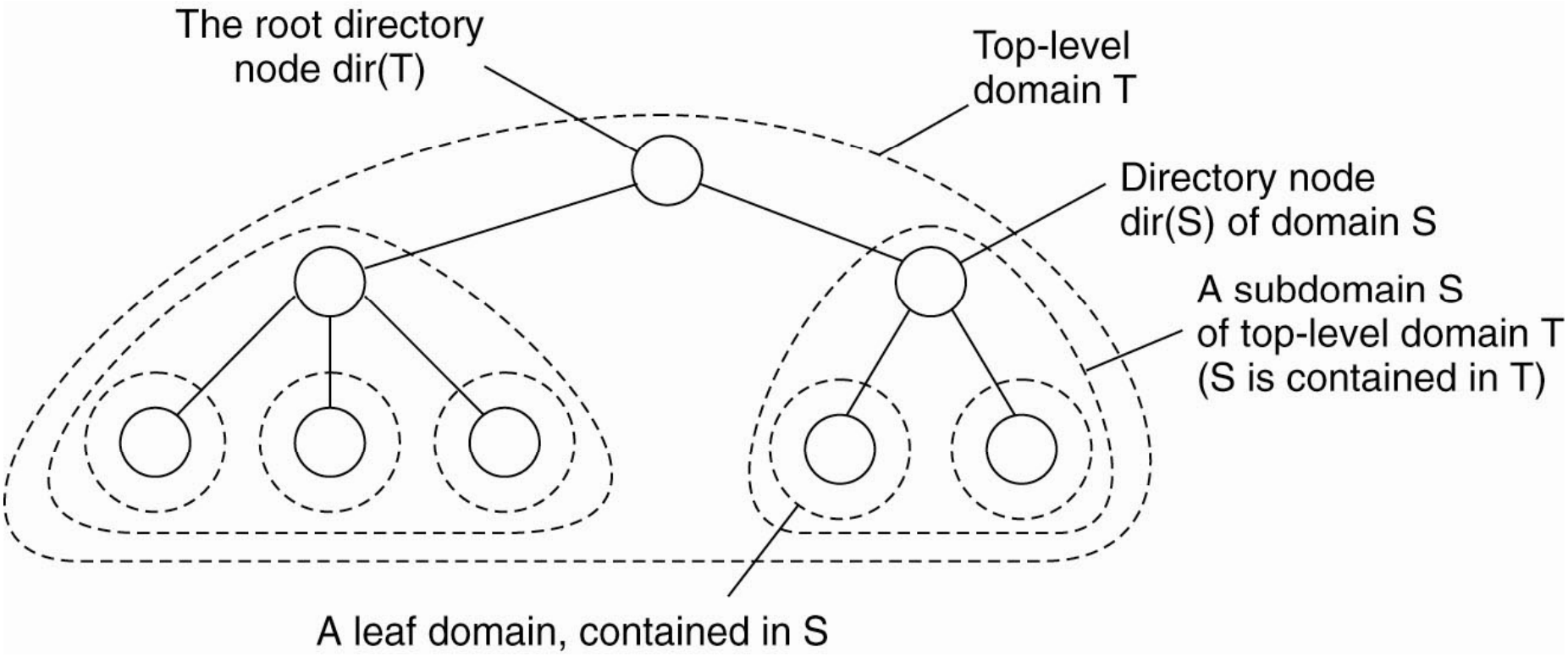
- URL: *Uniform Resource Locator*
  - É um URI específico para localização de recursos
    - scheme:scheme-specific-location
    - Ex: http://dns-name/filesystem-name
  - Está associado à localização do recurso
    - Se este se move ou é removido o URL perde o significado
- URN: *Uniform Resource Name*
  - É um URI de nomeação "puro" no qual o nome não tem associada qualquer localização
    - urn:nameSpace:nameSpace-specificName
    - Ex: urn:ISBN:0-201-64233-8 sistema de nomeação de livros
  - Para aceder a um URN é necessário convertê-lo num URL

Exemplo de Resolução de URL



- Um URL agrega vários domínios de resolução distintos
- Para resolver um URL, é necessário invocar vários sistemas de resolução de nomes

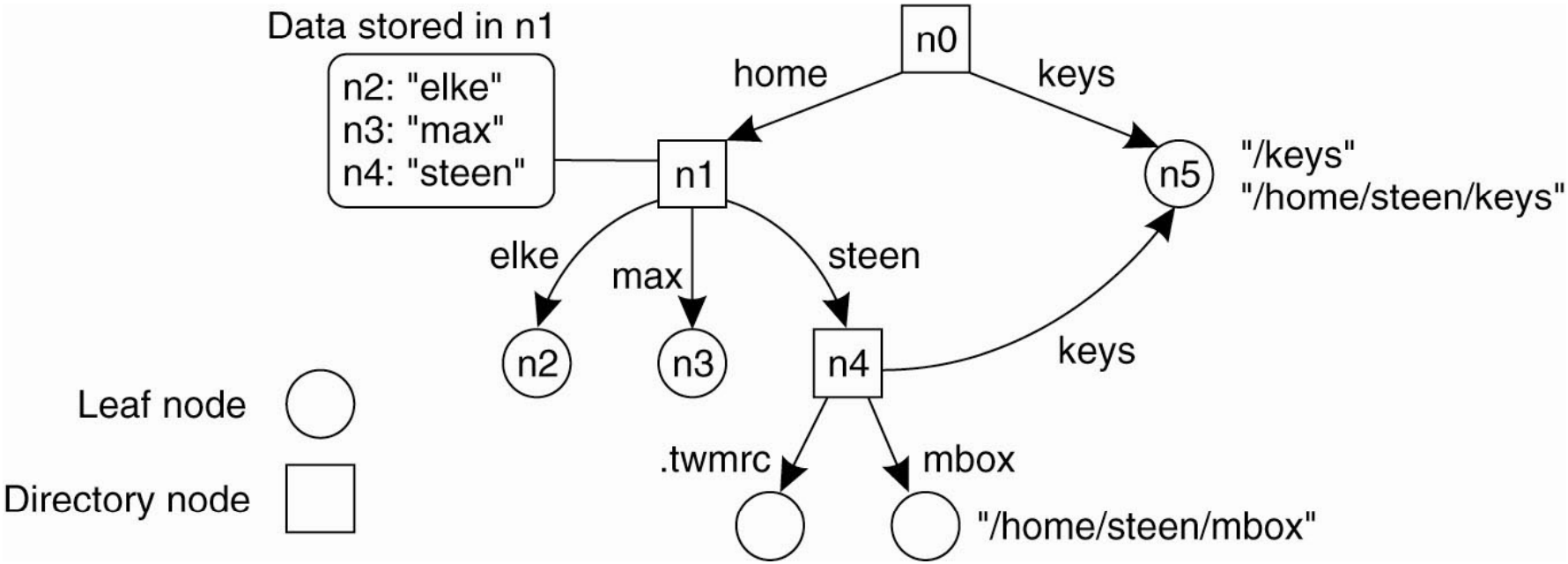
Espaço de Nomeação Hierárquico



Combinação de Espaços de Nomeação

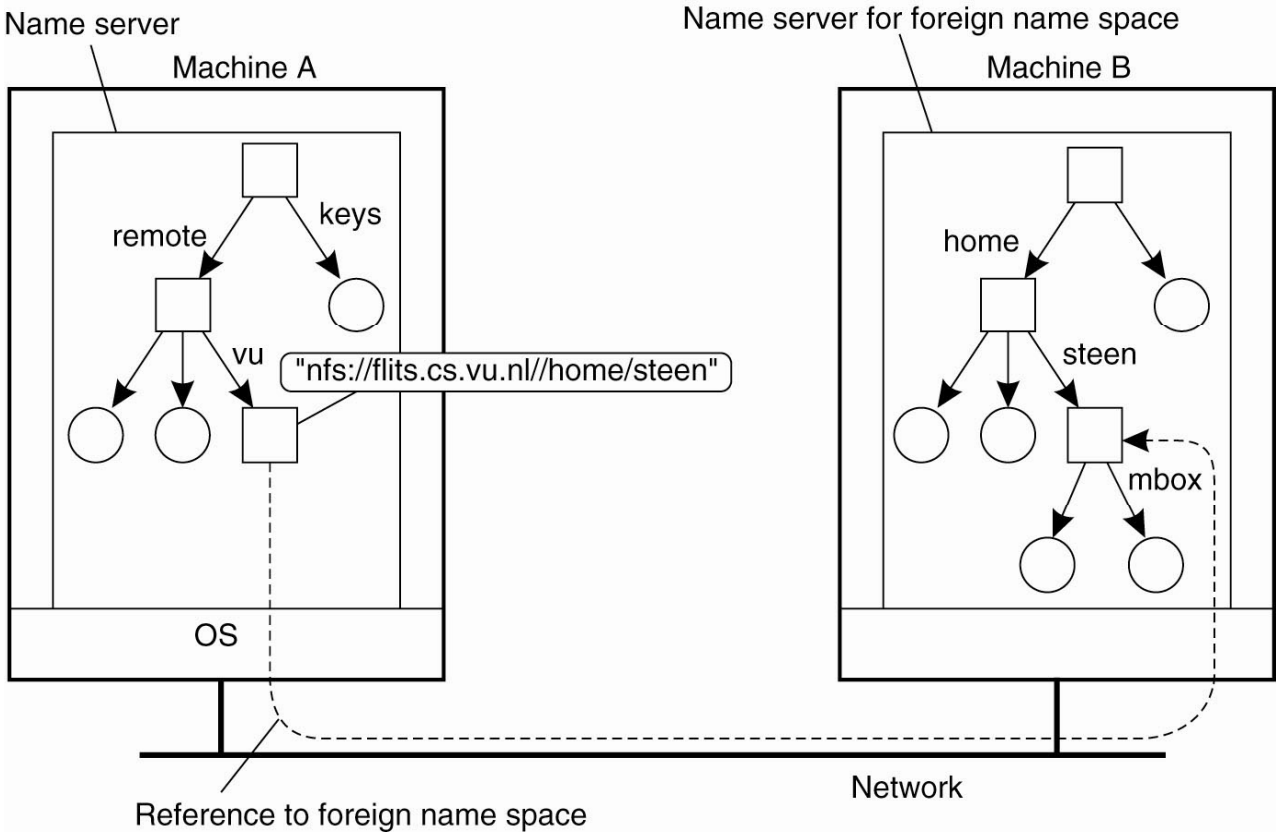
- Para a resolução de um nome global torna-se necessário combinar vários espaços de nomeação
  - Cada serviço de nomeação resolve a respectiva parte do nome
- Concatenação
  - Permite concatenar espaços de nomeação locais e remotos
  - Ex: *montagem* de sistemas de ficheiros NFS em Unix
    - mount srv1:/users /home/users
    - /home/users/aluno/documentos/aulas
  - A resolução do nome *atravessa* o ponto de montagem
- Customização
  - Permite adaptar o espaço de nomeação às exigências das aplicações ou utilizadores
  - Associações entre elementos de espaços de nomeação distintos
    - *aliases* de nomes DNS
    - *Symbolic links* Unix

Aliasing



- O nó n5 pode ser designado por:
  - /keys
  - /home/steen/keys

Concatenação NFS



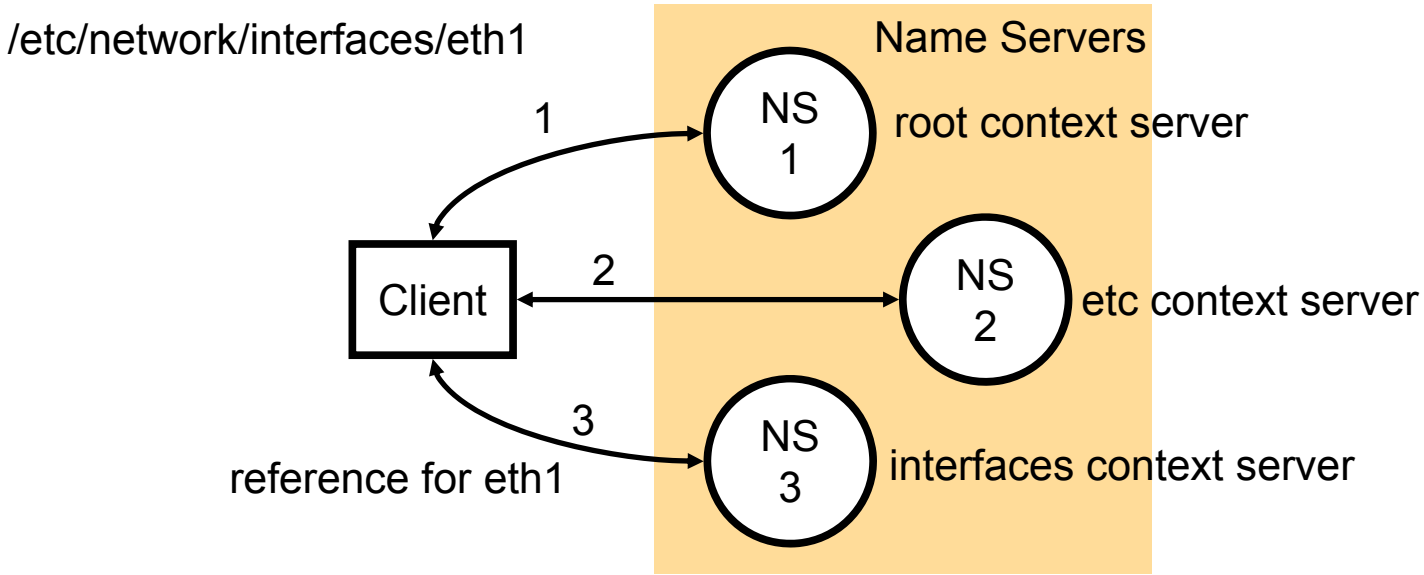
- Informação necessária para *montar* um espaço de nomeação remoto
  - O nome de um protocolo de acesso
  - O nome do servidor
  - O ponto de concatenação: nome do recurso local e do recurso de substituição

Serviço de Nomeação

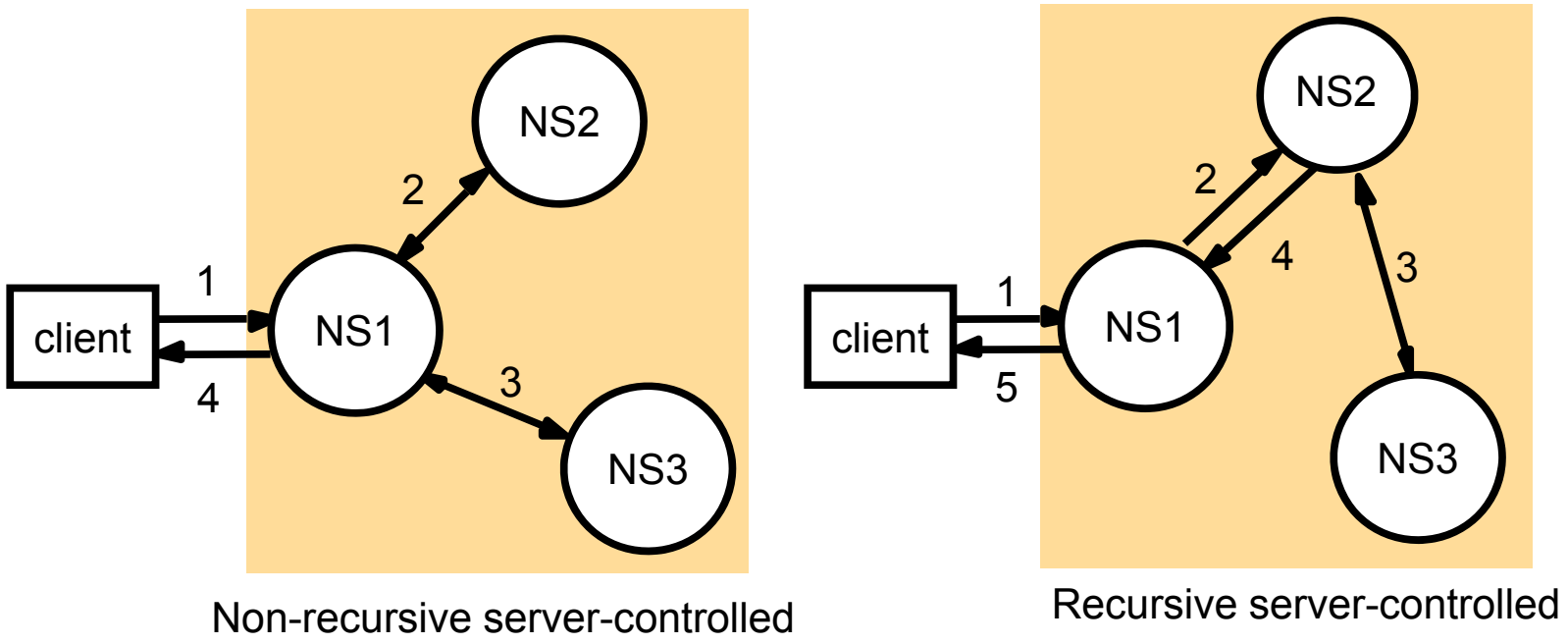
- Um Serviço de Nomeação permite realizar a resolução de um nome dentro do espaço de nomeação
- A resolução é sempre realizada num contexto ou directório associado
- O resultado de uma resolução é geralmente uma referência interna para um recurso, que permite aceder aos seus atributos ou funcionalidades
  - Características de um ficheiro (tamanho, tipo, permissões)
  - Métodos de um objecto
- As operações associadas à nomeação:
  - **Bind**: cria uma associação entre um nome e uma referência para os atributos do recurso
  - **Lookup**: permite realizar a resolução do nome

Resolução de Nomes

- A resolução é geralmente um processo iterativo em que cada componente do nome é apresentado ao servidor do contexto de topo (TLD)
  - Se o servidor conhece o nome devolve uma referência para os atributos
    - Se o nome é uma folha ou nó terminal o processo para
  - Se o nome é um nó, o processo continua no contexto do componente seguinte



Resolução Recursiva



Non-recursive server-controlled

Recursive server-controlled

- Se o servidor conhece o nome devolve uma referência para os atributos
  - Se o nome é uma folha ou nó terminal o processo para
- Se o nome é um nó, o processo continua o servidor de topo interage com os outros servidores para resolver o resto do nome
  - Pode interagir de forma iterativa ou recursiva

Caching e Réplica de Serviços de Nomes

- O serviços de nomeação são essenciais para o acesso a serviços distribuídos
- Por isso torna-se necessário mascarar as falhas dos servidores de nomes utilizando
  - *Caching* da resolução de nomes nos clientes
    - O *cache* pode ser organizado por contextos ou espaço de nomeação
  - Replicação dos vários servidores de contextos
    - Os servidores podem também ter *caches* de outros contextos
- O *caching* e a replicação permitem também melhorar as performances da resolução de nomes
  - Diminui latência e tráfego
- Técnicas de *caching* e replicação implicam a existência de manter a coerência entre as várias cópias da informação
  - Réplicas de tipo primário/secundário com propagação de actualizações
  - Gestão de *cache* baseado em *aging* ou com *call-backs* do servidor para o cliente (ex. AFS)

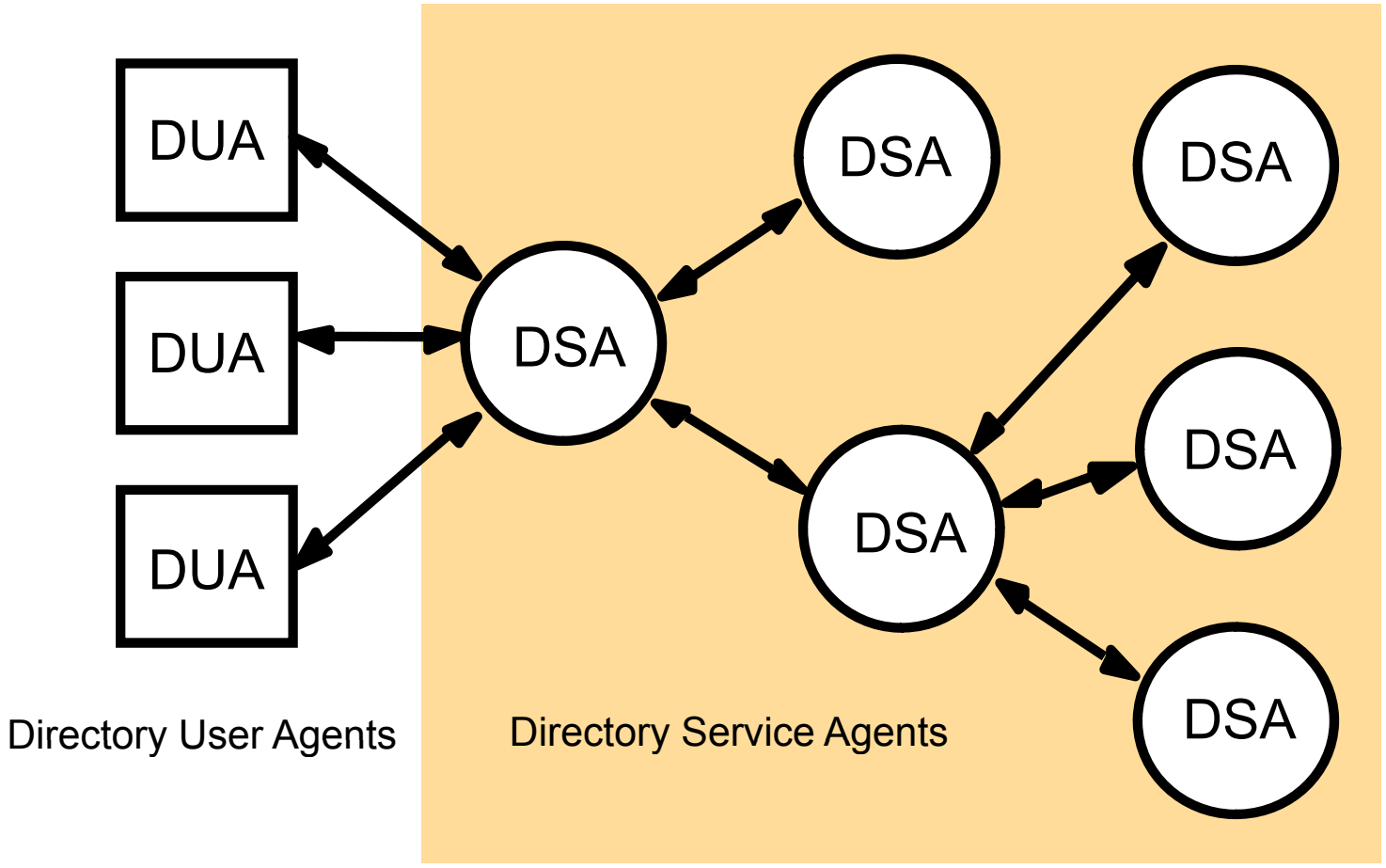
Serviços de Directório e de Descoberta

- Um **Serviço de Directório** permite obter uma referência para objectos através da indicação de um subconjunto dos seus atributos
  - O nome pode ser um dos atributos mas pode não ser necessário conhecê-lo
    - Ex. qual é o serviço de objectos que implementa cálculo matricial ?
  - O resultado de uma resolução pode ser um conjunto (vasto) de referências
  - Ex.: X-500, LDAP, UDDI são serviços de directório
- Um **Serviço de Descoberta** é um serviço de directório que permite o registo e descoberta dinâmica de funcionalidades de dispositivos voláteis num ambiente distribuído
  - A associação a dispositivos é feita baseada na funcionalidade e não no nome
  - A descoberta pode ser feita por *multicast* iniciado pelos clientes ou pelos servidores
  - Ex.: Jini, Bluetooth
- Serviço de Directório = *yellow pages*
  - Serviço de Nomeação = *white pages*

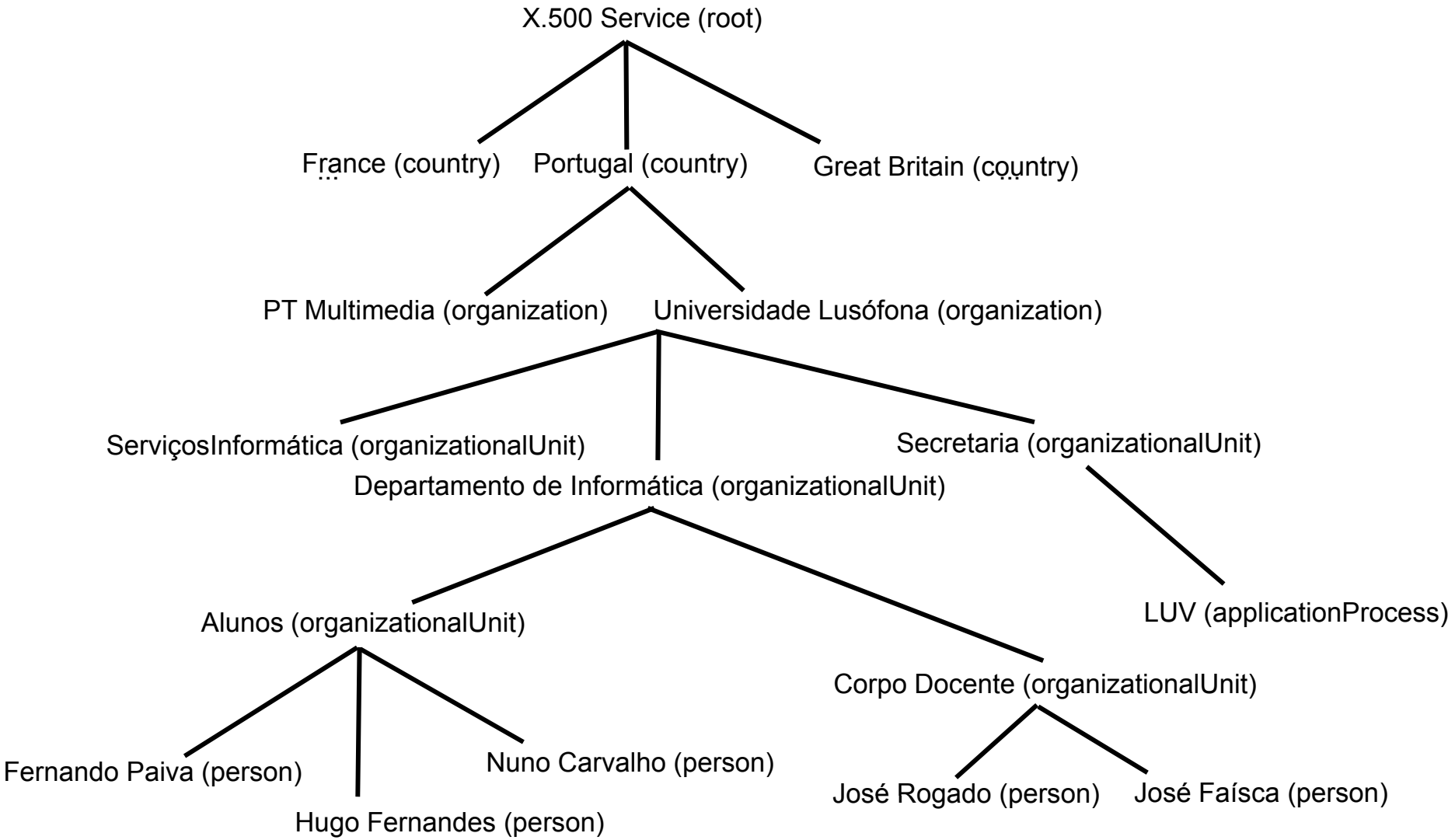
Serviço de Directório X.500

- Norma conjunta ISO/CCITT com objectivos ambiciosos:
  - Serviço de Directório distribuído de grande dimensão com servidores espalhados por todo o mundo
  - Permitir o armazenamento de grande variedade de informação
    - Desde informação respeitante a indivíduos até catálogos de componentes
  - Poder ser utilizado como um serviço de nomeação clássico
- Estrutura
  - Estrutura Hierárquica organizada em árvore
    - **DIT** - Directory Information Tree
  - A informação em cada nó é composta por um nome e um conjunto de atributos
    - **DIB** – Directory Information Base
  - O objectivo da norma era permitir a existência de um DIB a nível mundial com sub-directórios localizados em diferentes países
- Terminologia
  - Servidor: **DSA** - *Directory Service Agent*
  - Cliente: **DUA** - *Directory User Agent*
  - Nome local: **RDN** – *Relative Distinguished Name*
  - Nome Global: **DN** - *Distinguished Name*
    - Concatenação de todos os nomes locais desde a raiz

Arquitectura de Serviços X.500



Exemplo de Árvore X.500



Estrutura do DIB

- A estrutura de um elemento do DIB é flexível e consiste num conjunto variável de atributos
- Um atributo é constituído por um tipo e um ou mais valores
  - Existem tipos predefinidos e podem ser definidos novos tipos
    - *countryName, organizationName, commonName, telephoneNumber, mailbox, …*
  - Cada elemento do DIB é definido por uma ***objectClass*** que determina as suas características (tipo, sintaxe, etc…)
    - *Organization, organizationalUnit, person, …*
    - Novos objectClasses podem ser definidos
    - As características das *objectClasses* são herdadas
- Um elemento do DIB é determinado pelo seu *distinguished attribute*
  - É designado por *Relative Distinguished Name* e identifica um elemento de forma única no nível da DIT a que pertence
    - Ex: Portugal, Corpo Docente, Alunos, José Rogado, etc..
  - A concatenação de todos os RDNs desde a raíz forma o *Distinguished Name*
    - *Ex.: cn=José Rogado, ou=Corpo Docente, ou=Dpto de Informática, o=ULHT, c=Portugal*

Operações no Directório

- Procura
  - **A realização de procuras eficazes é uma das propriedades essenciais de um directório, pelo que a sua implementação deve fazer apelo a réplicas e *caching*.**
  - Tem como argumento o nome de um nó para o início da procura e um conjunto de valores de atributos para os quais se pretende realizar a filtragem
  - São devolvidos todos os elementos cujos atributos satisfaçam o critério de procura
  - A procura pode ser uma operação extremamente pesada se a DIT for vasta e os critérios de procura pouco selectivos
- Leitura
  - Tem como argumento um DN ou RDN e um conjunto de atributos dos quais se pretendem os valores
  - O nome é resolvido através da procura na DIT possivelmente percorrendo vários níveis e depois de localizado o DSA que gere o nome, este devolve a informação ao DUA

Operações no Directório

- Administração e actualização
  - A interface de acesso do DSA inclui primitivas para adicionar, apagar, mover e modificar elementos do directório
  - A operações de modificação não têm um desempenho elevado, pois o directório é sobretudo um repositório de consulta
  - Os acessos ao directório podem ser protegidos através de vários tipos de controlo
    - Simples password ou Public Key Encription baseado em certificados X.509
- Implementação
  - A norma X.500 não define a forma como o directório é implementado
  - A comunicação entre DUA e DSA é realizado através do Directory Access Protocol (DAP) baseado nos protocolos ISO, pouco utilizado hoje em dia
- A implementação e o protocolo de acesso constituem as principais razões pelas quais o X.500 não teve o sucesso previsto

O LDAP

- O X.500 não teve a propagação prevista a nível mundial
  - Sistema complexo e com sintaxe de nomeação pouco intuitiva
  - O DNS foi adoptado como standard de facto na Internet
  - É contudo muito utilizado em organizações como directório de serviços, utilizadores ou credenciais de autenticação
- *Lightweight Directory Access Protocol* - LDAP
  - Um protocolo mais leve baseado em TCP/IP foi desenvolvido na Univ. de Michigan e utilizado para aceder a directórios que implementam o standard X.500
  - Existe uma versão freeware: OpenLDAP (www.openldap.org)
- Inúmeras versões de LDAP de vários fabricantes são utilizados como repositórios de informação em plataformas IMS (*Identity Management Systems*)
  - Sun Directory Server
  - Microsoft Active Directory
  - Novell eDirectory
  - Isode M-Vault

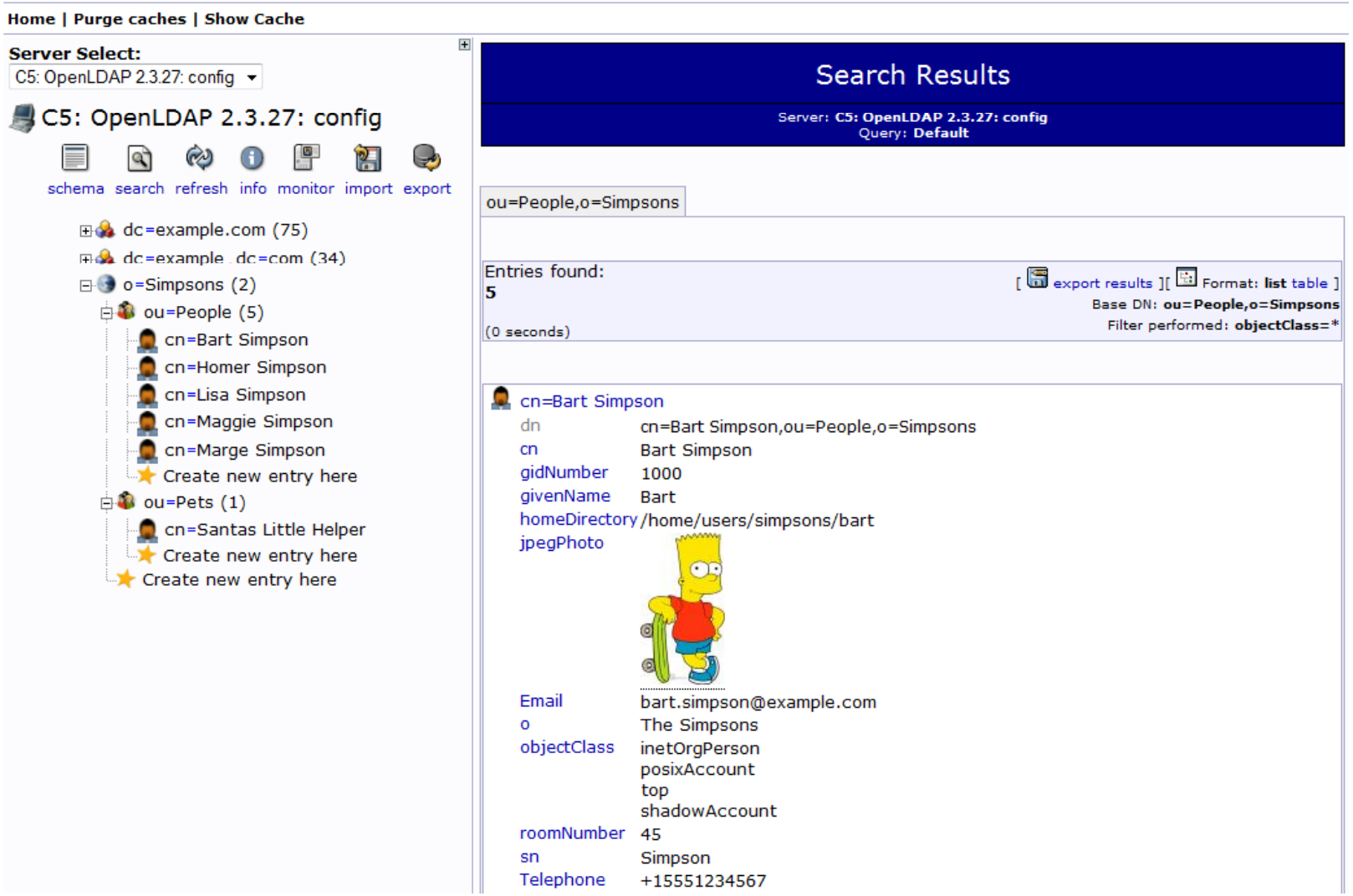
Actualização de Conteúdo LDAP

- Para criar um directório o modificar o seu conteúdo existe um formato específico que permite criar scripts de gestão
  - **LDAP Data Interchange Format (LDIF)**
- Permite a importação ou exportação de conteúdos inteiros de um directório
- Utiliza uma sintaxe simples baseada na utilização dos identificadores do DIB
  - **`dn: distinguished name`**
  - **`dc: domain component`**
  - **`ou: organizational unit`**
  - **`cn: common name`**
- As operações são efectuadas sobre os campos indicados
  - **`add, replace, delete`**

Exemplos de LDIF

```
dn: cn=John Smith,ou=Legal,dc=example,dc=com
changetype: modify
replace:employeeID
employeeID: 1234
-
replace: employeeNumber
employeeNumber: 98722
-
replace: extensionAttribute6
extensionAttribute6: JSmith98
-
dn: cn=Peter Michaels,ou=Artists,l=San Francisco,c=US
changetype: modify
add: telephonenumber
telephonenumber: +1 415 555 0002
-
```

Exemplo de Front End: phpLDAPadmin



http://wurley.demo.phpldapadmin.info/

Referências

- Referências X.500
  - http://sec.cs.kent.ac.uk/x500book
- Referências LDAP
  - http://docstore.mik.ua/orelly/perl/sysadmin/appb_01.htm
  - http://www.openldap.org
  - http://www.novell.com/products/edirectory
  - http://www.isode.com/products/m-vault.html
- Trabalho Complementar
  - Ler o tutorial sobre o JNDI (Java Naming and Directory Interface) particularmente a secção sobre *Naming and Directory Concepts*
  - http://java.sun.com/docs/books/tutorial/jndi/concepts/index.html

Fim do Capítulo IV

- Capítulo bastante extenso
  - Aprendizagem do *Core* da Computação Distribuída
- Resumo dos conhecimentos adquiridos
  - Modelo de execução de RPC
  - Definição de Interfaces
  - Registo e Descoberta
  - Plataforma de RPC
    - Exemplo Sun RPC
  - Objectos Distribuídos
    - Exemplo Java RMI
  - Nomeação
  - Serviços de Directório
- Preparação para abordar aplicações reais
  - NFS
  - WebServices
  - Grid Computing