Computação Distribuída – Cap. II

Licenciatura em Engenharia Informática

Universidade Lusófona

Prof. José Rogado
Prof. José Faísca

Arquitecturas e Modelos de Comunicação Distribuída

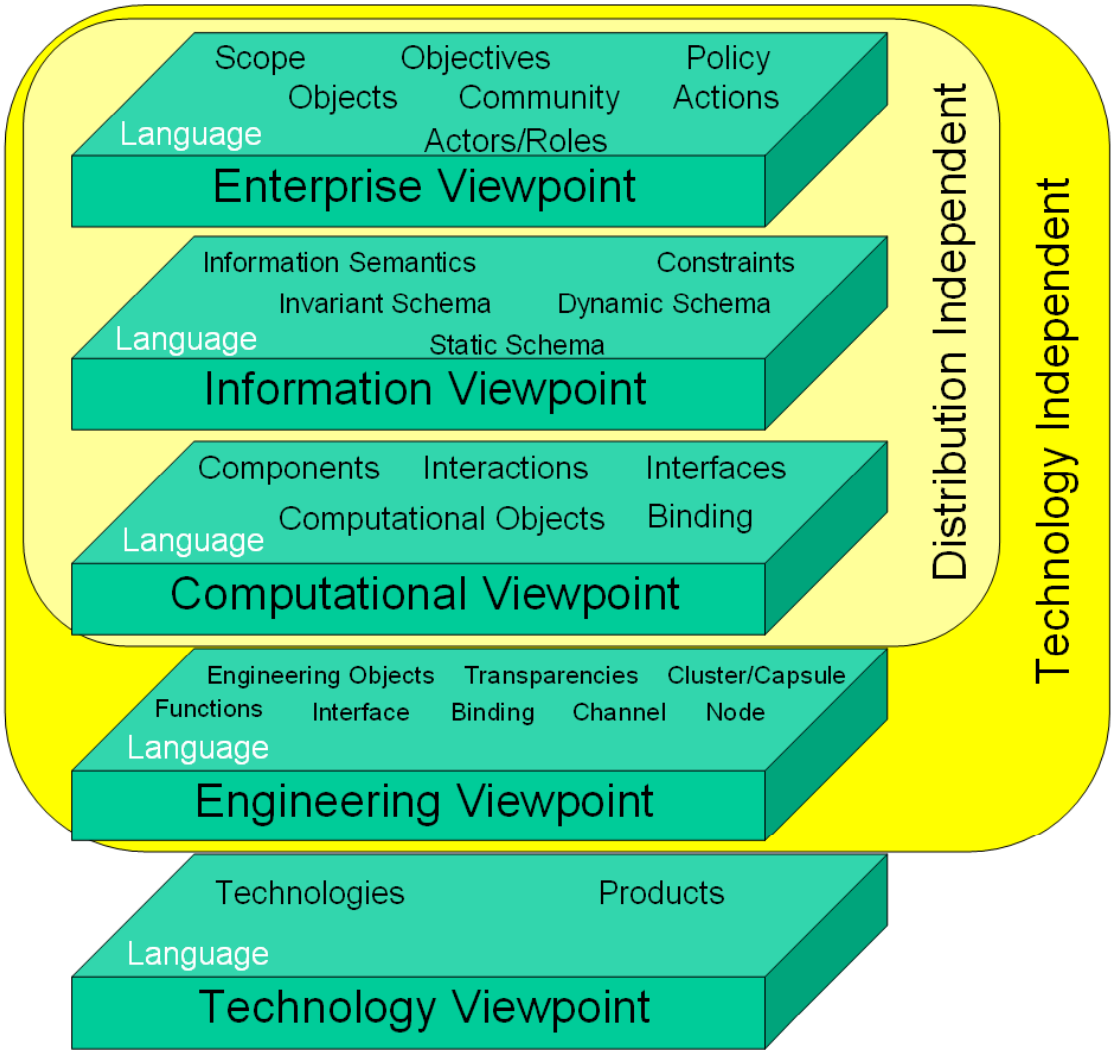
- Arquitecturas de sistemas
- Níveis de software
- Interfaces e objectos
- Modelos de Funcionamento
  - Interacção
  - Falhas
  - Segurança

Noção de Arquitectura

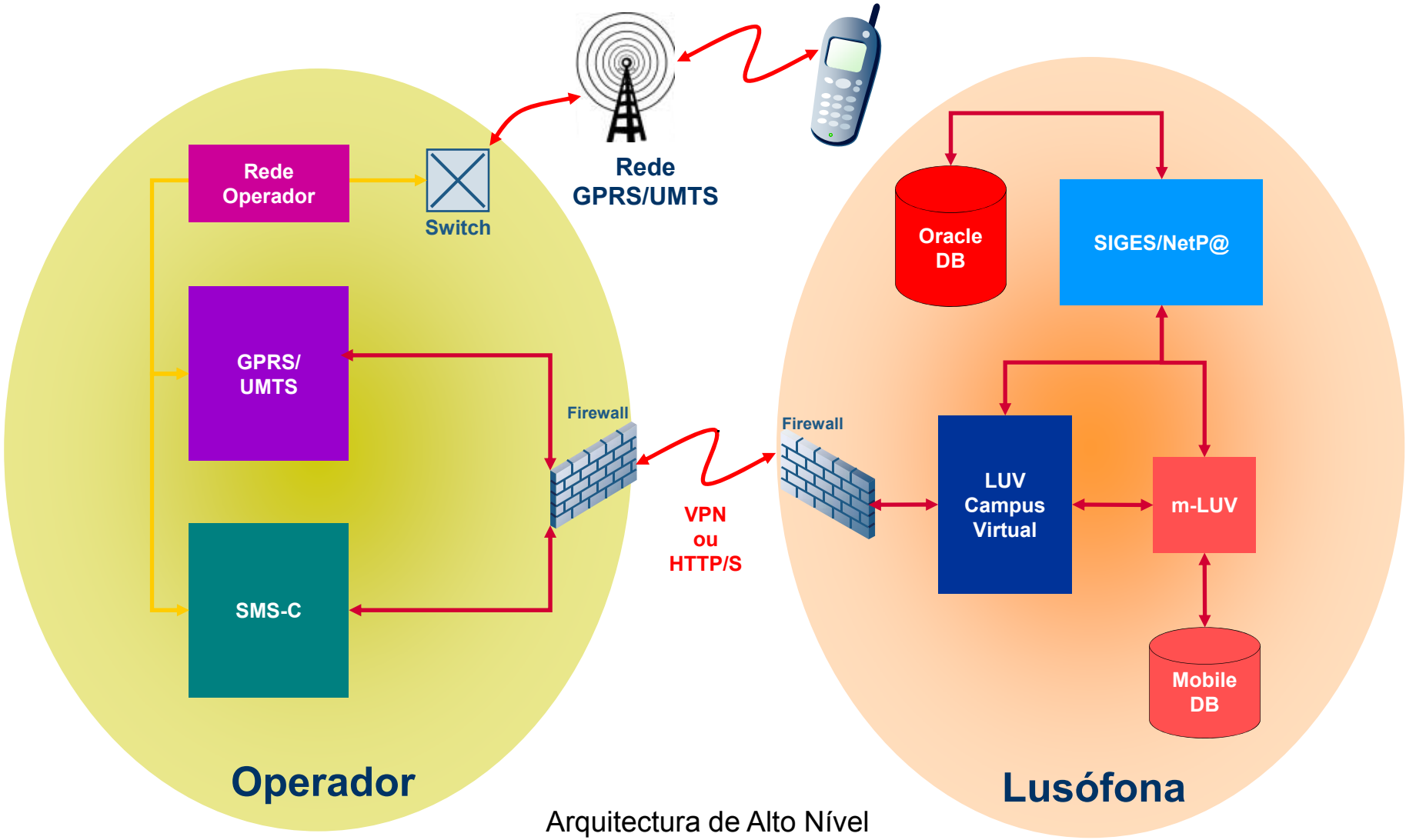
- A arquitectura define a estrutura de um sistema distribuído em termos de componentes especificáveis separadamente. Fornece um modelo para:
  - Especificação
  - Desenvolvimento
  - Funcionamento
  - Evolução
- O modelo de arquitectura
  - Define a funcionalidade e localização dos componentes do sistema
  - Identifica as relações e os padrões de comunicação existentes entre eles
- A correcta definição de uma arquitectura é importante para garantir:
  - Desempenho
  - Fiabilidade
  - Segurança
  - Modularidade
- Analogia: arquitectura de um edifício

Exemplo de Modelo de Definição de Arquitectura

- RM-ODP: Reference Model for Open Distributed Processing
- Norma conjunta do ITU e da OSI publicada em 1996
- Framework que inclui linguagens, conceitos e regras par definir correctamente uma arquitectura a vários níveis de representação
- Utilizada em projectos de larga escala em que componentes do sistema são desenvolvidos por entidades distintas

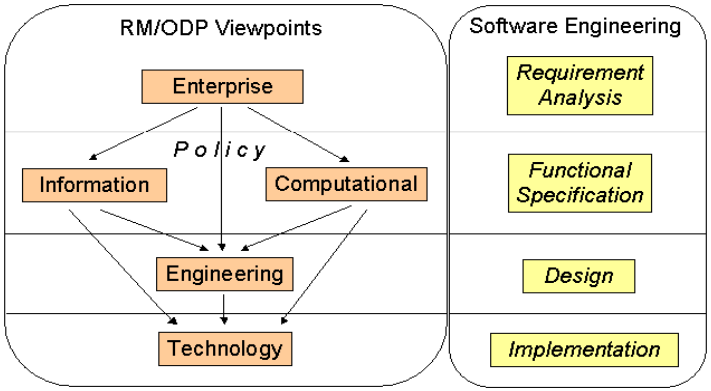


Exemplo de Arquitectura Simples: portal móvel mLUV

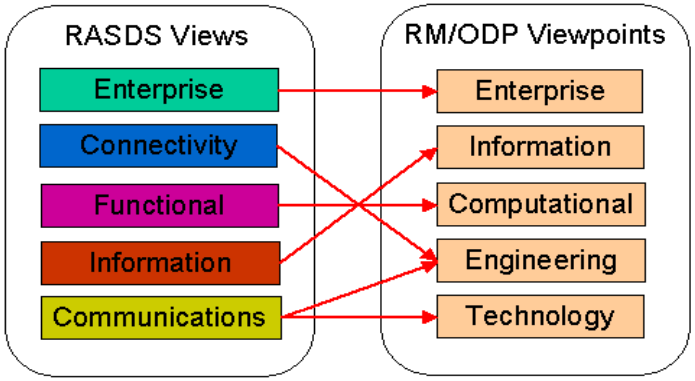


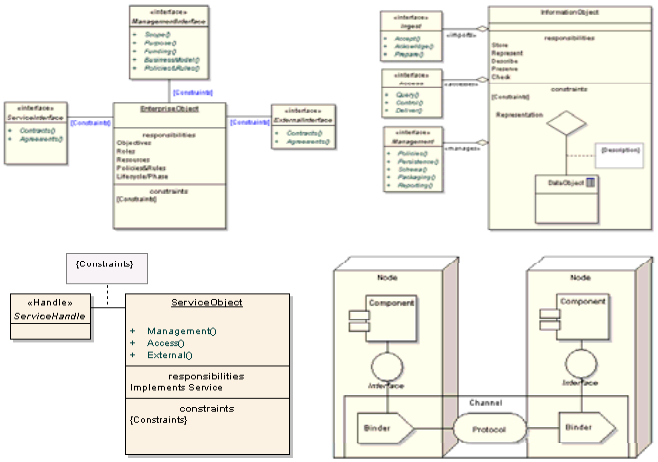
Arquitectura de Alto Nível

Exemplo Complexo: Ground Systems Harmonization

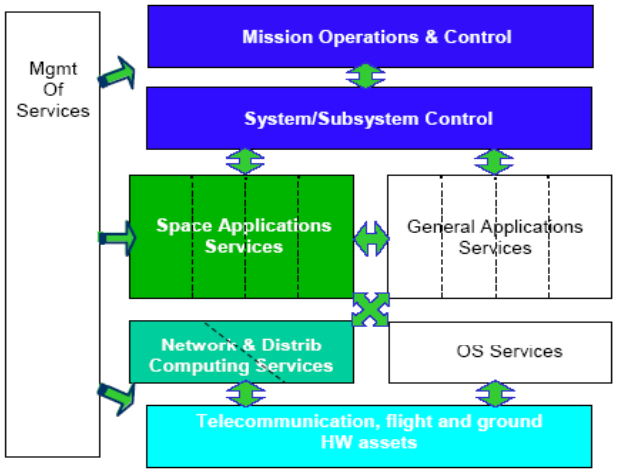


Ground Systems Harmonization Framework



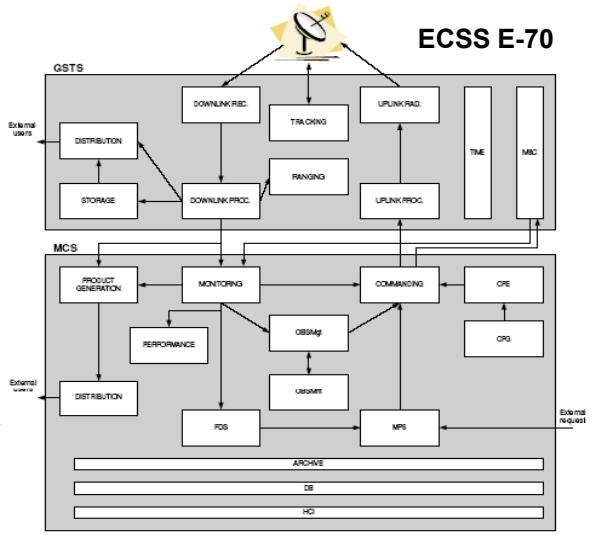


OMG Space Reference Architecture



Source: OMG Space DTF

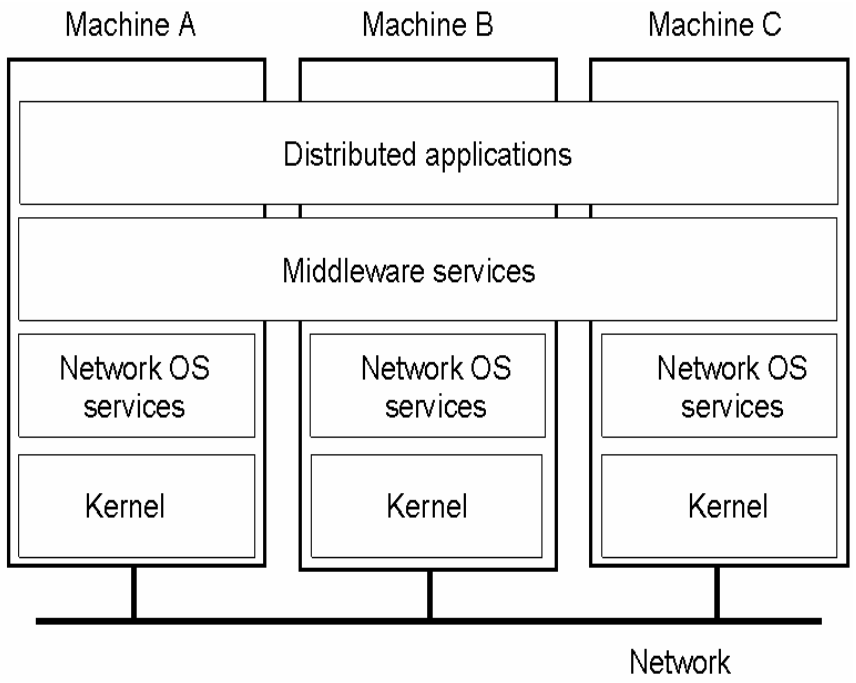
ECSS E-70

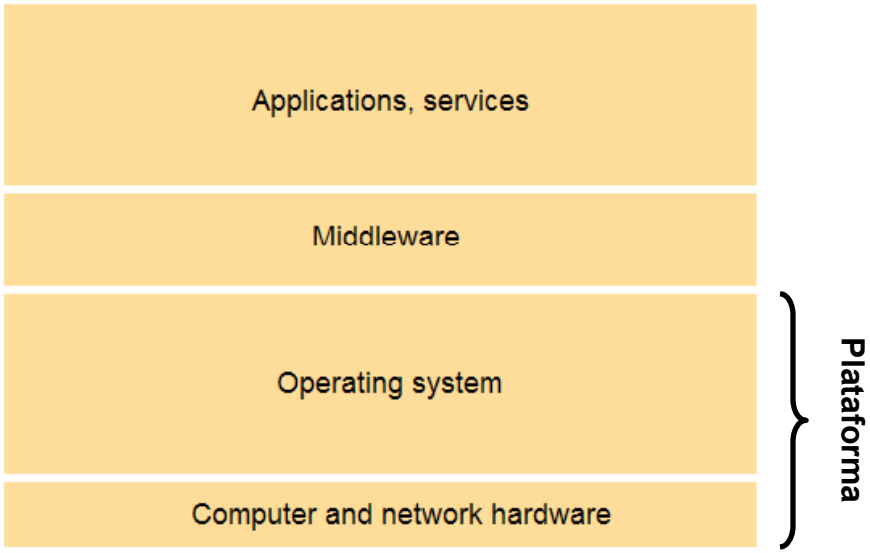


Componentes da Arquitectura

- Processo
  - Unidade de execução de programas num sistema
- Objecto
  - Unidade de encapsulamento de código e/ou dados
  - Os objectos são instanciados e executados em processos
- Serviço
  - Unidade funcional de distribuição, que gere recursos e fornece funcionalidades a aplicações e utilizadores
  - É implementado por objectos executados num ou vários processos
  - É definido por uma ***interface*** e um protocolo
  - Deve poder ser identificado e localizado
- Servidor
  - Componente que fornece um ou mais serviços, podendo abranger um ou mais componentes hardware
- Cliente
  - Entidade que invoca os serviços num servidor

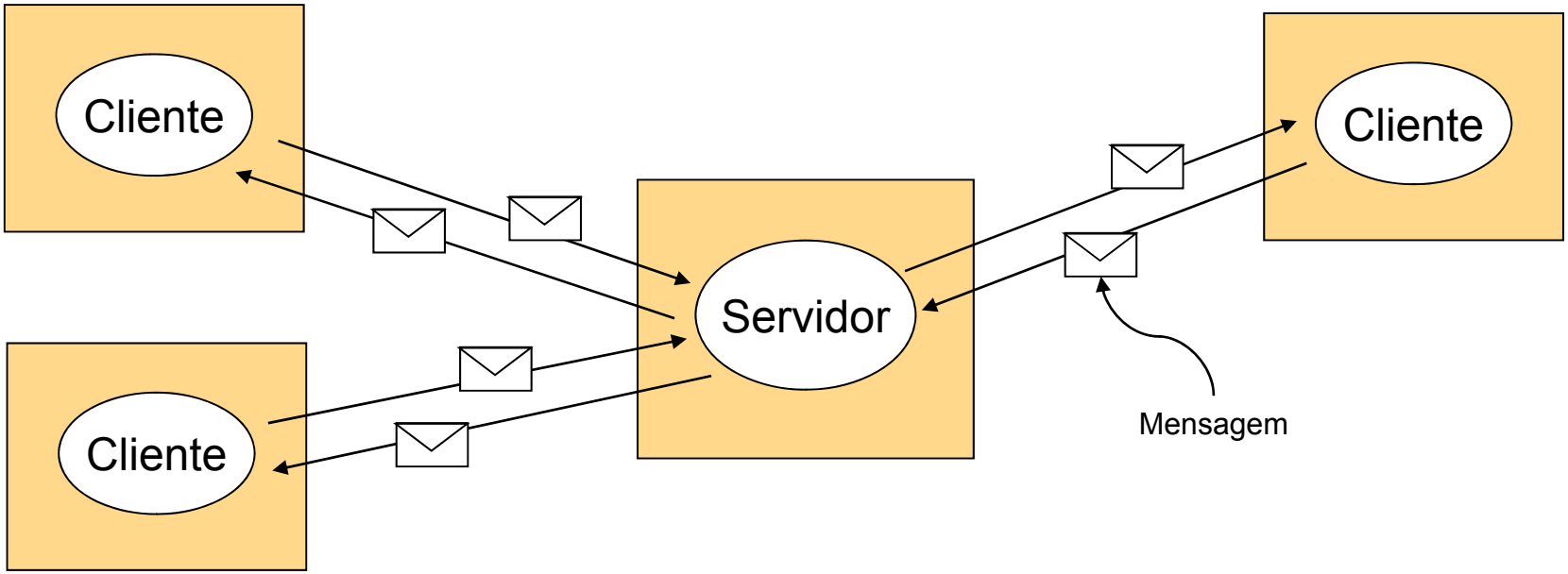
Níveis de Software e Serviços





- Níveis de software
  - Os serviços podem ser organizados em camadas lógicas com níveis de abstracção distintos
  - Um nível pode abranger um ou mais componentes do sistema
- Plataforma
  - Camadas mais baixas de software/hardware que fornecem os serviços básicos
  - Ex: Intel x86/Windows, Intel x86/Linux, PowerPC/MacOS

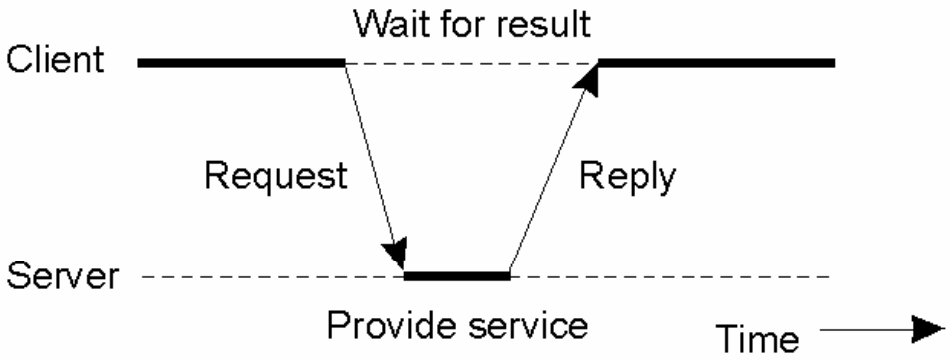
Revisão da Arquitectura Cliente/Servidor



- Arquitectura tradicional dos Sistemas Distribuídos
  - Caracteriza um sistema em que só há dois tipos de componentes
    - Arquitectura simples e fácil de implementar
  - Servidor: executa operações invocadas pelos clientes retornando resultados
  - Cliente: invoca operações em servidores
  - Mensagem: dados trocados entre cliente e servidor obedecendo a um formato especifico do serviço

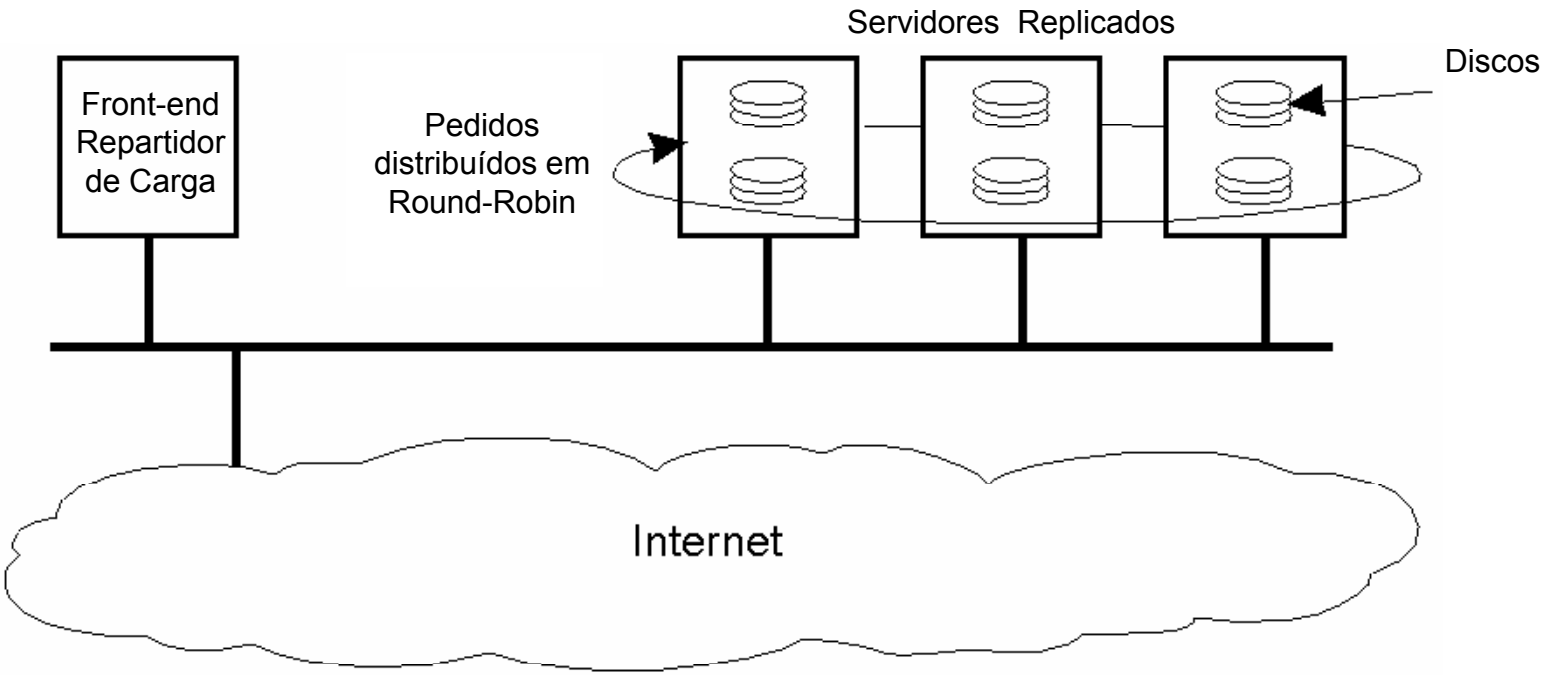
Características C/S

- Modelo de invocação de *request/reply* com envio e retorno de mensagens
- Modelo de interacção é geralmente síncrono: o cliente espera pelo retorno da invocação



- Relação N para 1 pode trazer problemas de desempenho e fiabilidade
  - Toda a carga é concentrada no servidor: *bottleneck*
  - A escalabilidade é problemática
  - Ponto de falha único: servidor
- Na realidade, a arquitecturas utilizadas são geralmente variantes do C/S
  - Cliente/Servidor-Múltiplo
- Exemplos
  - Browser/Servidor Web
  - Etc…

Variantes Cliente/Servidor (i)



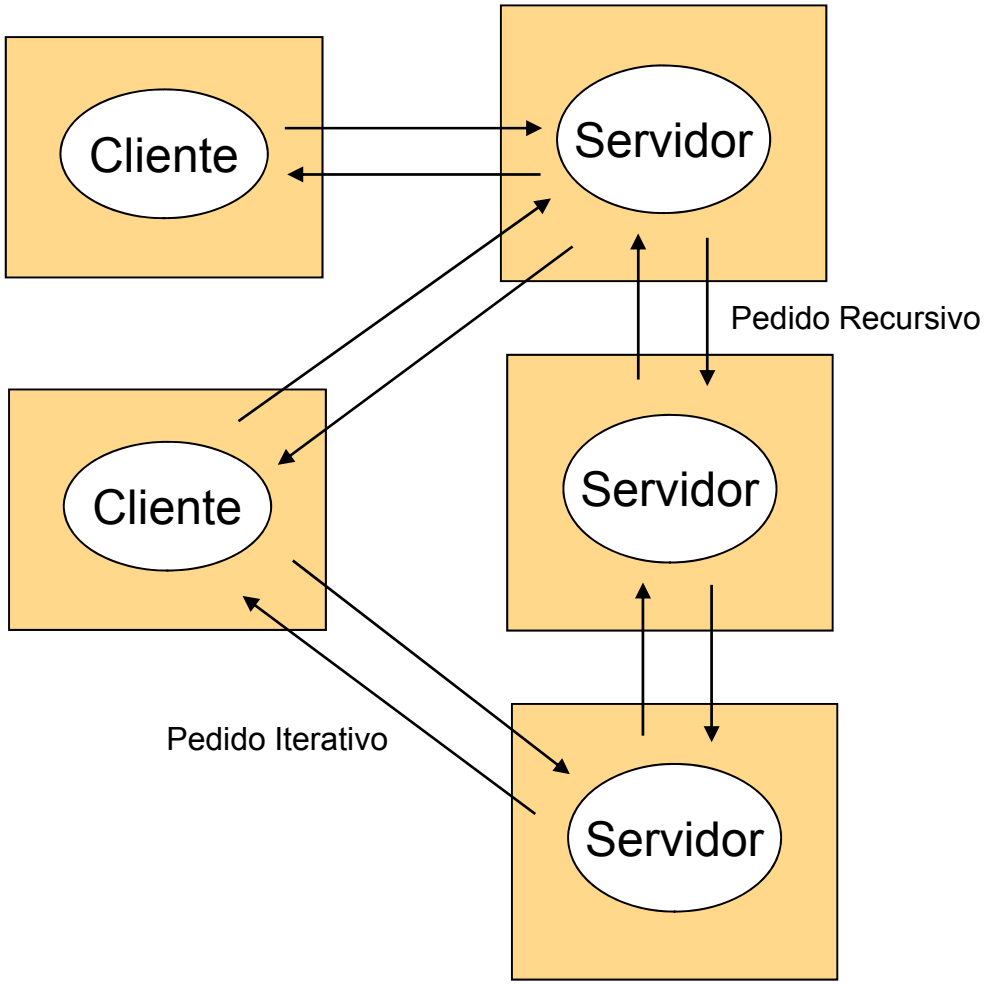
Servidores Redundantes

- O mesmo serviço é fornecido por vários servidores
  - Distribuição da carga, evitando ponto único de falha
  - Utilização de Repartidores de Carga (*Load-Balancers*)
  - Solução utilizada nos grandes portais
- Dificuldade em gerir a coerência de estado em cada réplica
  - Problema das Sessões

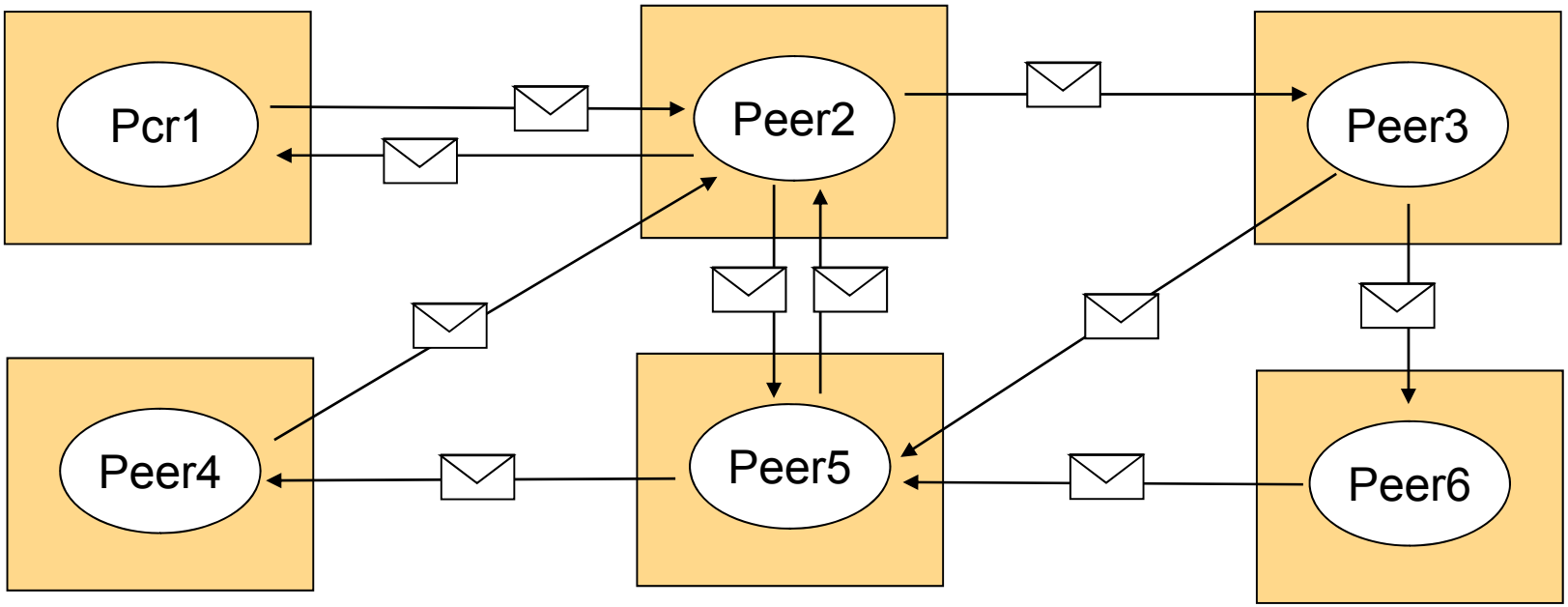
Variantes Cliente/Servidor (ii)

Servidores Repartidos

- O serviço é repartido por vários servidores
  - Cada servidor implementa uma parte do serviço
  - Distribuição da carga, evitando ponto único de falha
  - Solução utilizada no serviço de nomes (DNS)
- A invocação dos servidores pode ser feita
  - directamente pelos clientes por redirecção
    - Serviço iterativo
  - Pelo servidor inicial
    - Serviço recursivo



Arquitectura P2P (*Peer-to-Peer*)

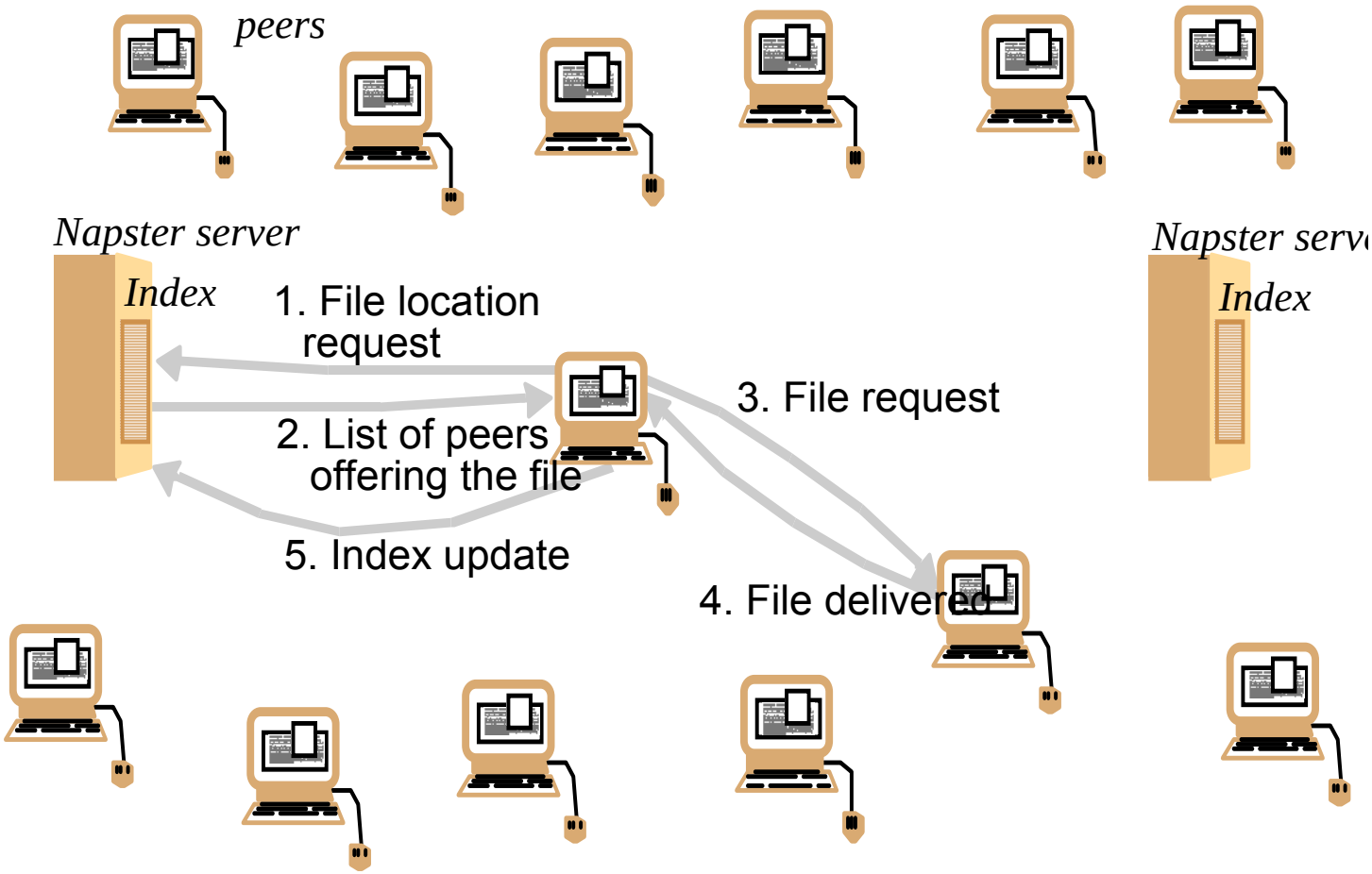


- Arquitectura baseada em componentes que desempenham papéis idênticos (*pares*)
  - Interagem cooperativamente assumindo o papel de cliente e/ou servidor simultaneamente
  - Permite explorar a capacidade de processamento e armazenamento de múltiplos computadores numa rede

Características P2P

- Modelo de interacção complexo
  - Dificuldade de implementação
  - Administração problemática
- Dispersão e número elevado de componentes
  - Problemas de Segurança
- Vantagens
  - Escalabilidade facilitada
  - Maior fiabilidade do serviço como um todo
    - Não existe ponto único de falha
- Os problemas de registo e pesquisa de serviço necessitam de um serviço centralizado, ou a utilização de algoritmos distribuídos complexos
  - Tema de investigação actual
  - Arquitecturas Híbridas com um ponto único de registo
  - *Overlay* de Roteamento Distribuído
- Muitas aplicações P2P actuais Web utilizam este último modelo
  - O BitTorrent é um dos exemplos mais conhecidos

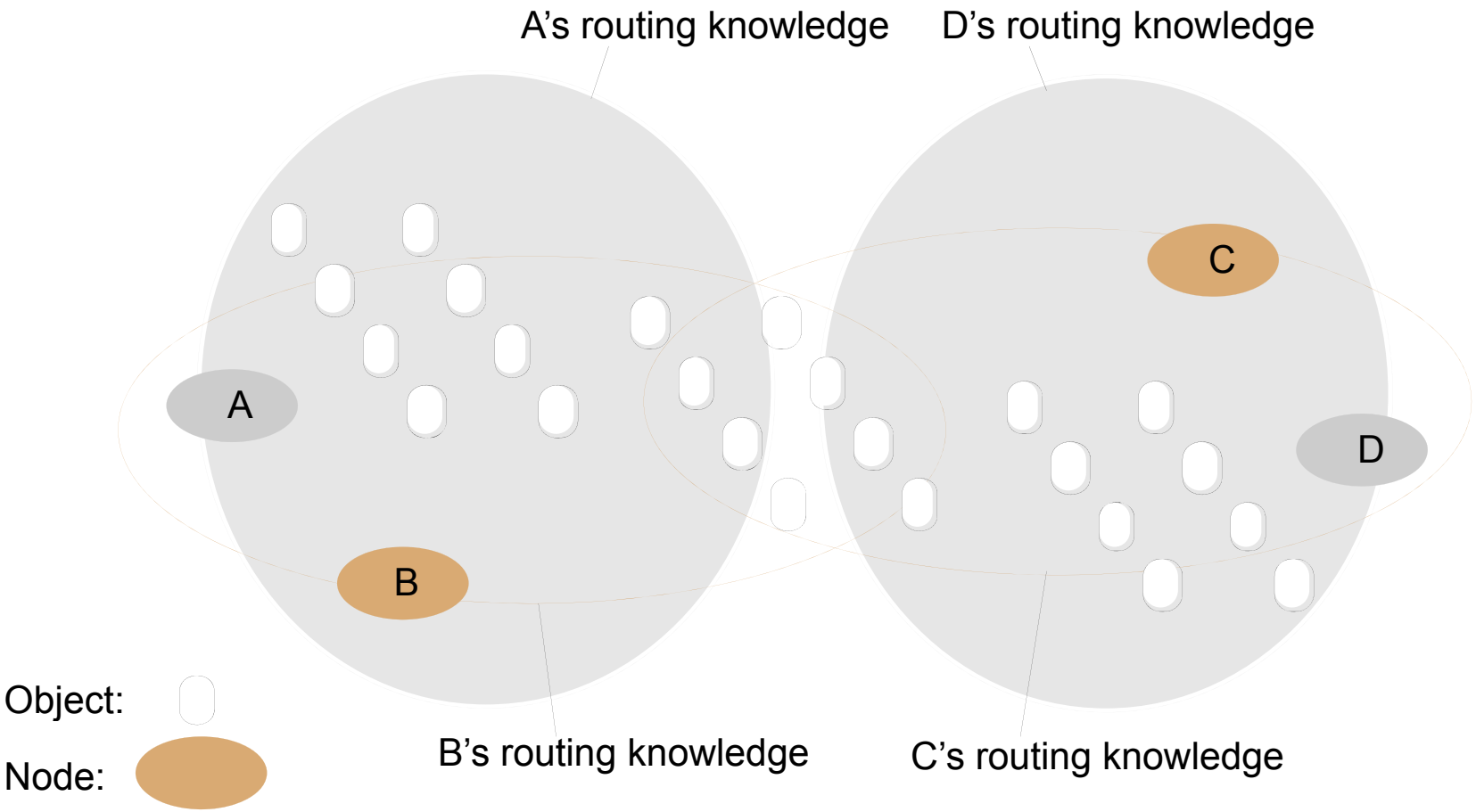
Napster: Arquitectura Híbrida



Middleware P2P: Routing Overlay

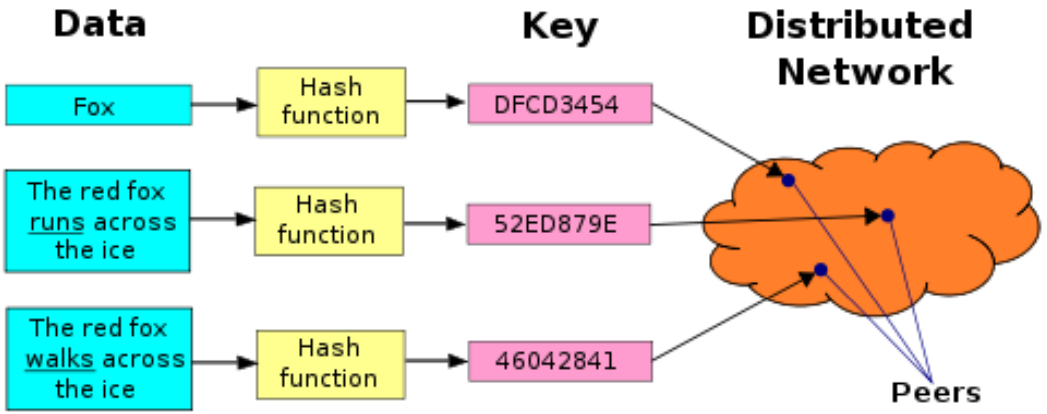
- Middleware P2P distribuído que permite a identificação, armazenamento e acesso à informação. É constituído por:
- Um **Global Unique Identifier** (GUID) que identifica univocamente a informação armazenada, obtido a partir do *digest* dessa a informação
  - Utilização de uma função de *Hash* (p.ex. SHA1 -> gera chaves de 160 bits)
  - O conjunto de todos GUIDs possíveis designa-se por **espaço de chaves** (*keyspace*)
- Um **particionamento** do *keyspace*, ficando cada nó (peer) do sistema responsável pela gestão de uma partição
  - A uma partição está associado um algoritmo que permite estabelecer a que partição pertence uma determinada chave
- Um mecanismo de **roteamento** no nível aplicacional, que permite a localização dos nós, o armazenamento e acesso à informação
  - O *overlay* responde a *queries* sobre identificadores de informação (GUIDs)
  - Cada nó contém a chave procurada ou um link para um nó que está mais perto do nó que a contém (*greedy algorithm*)

Distribuição da Informação de Roteamento



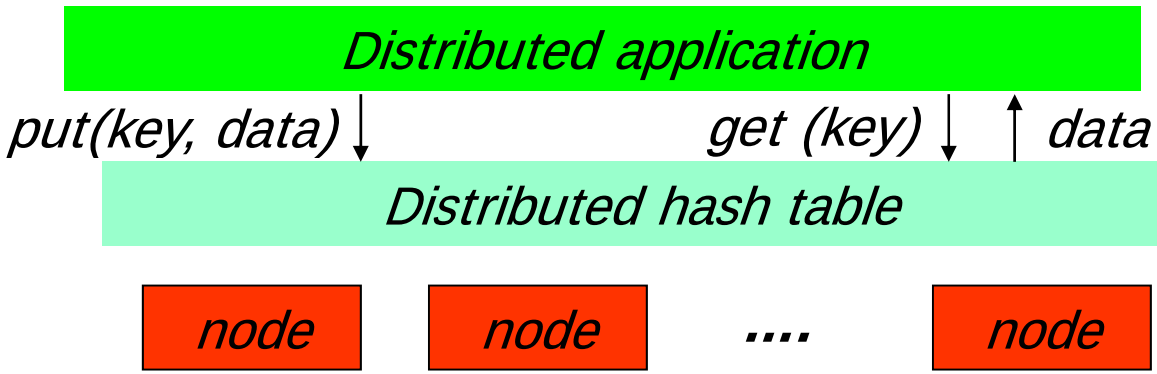
Ex. de Implementação: Distributed Hash Table

Keyspace



Source: Wikipedia

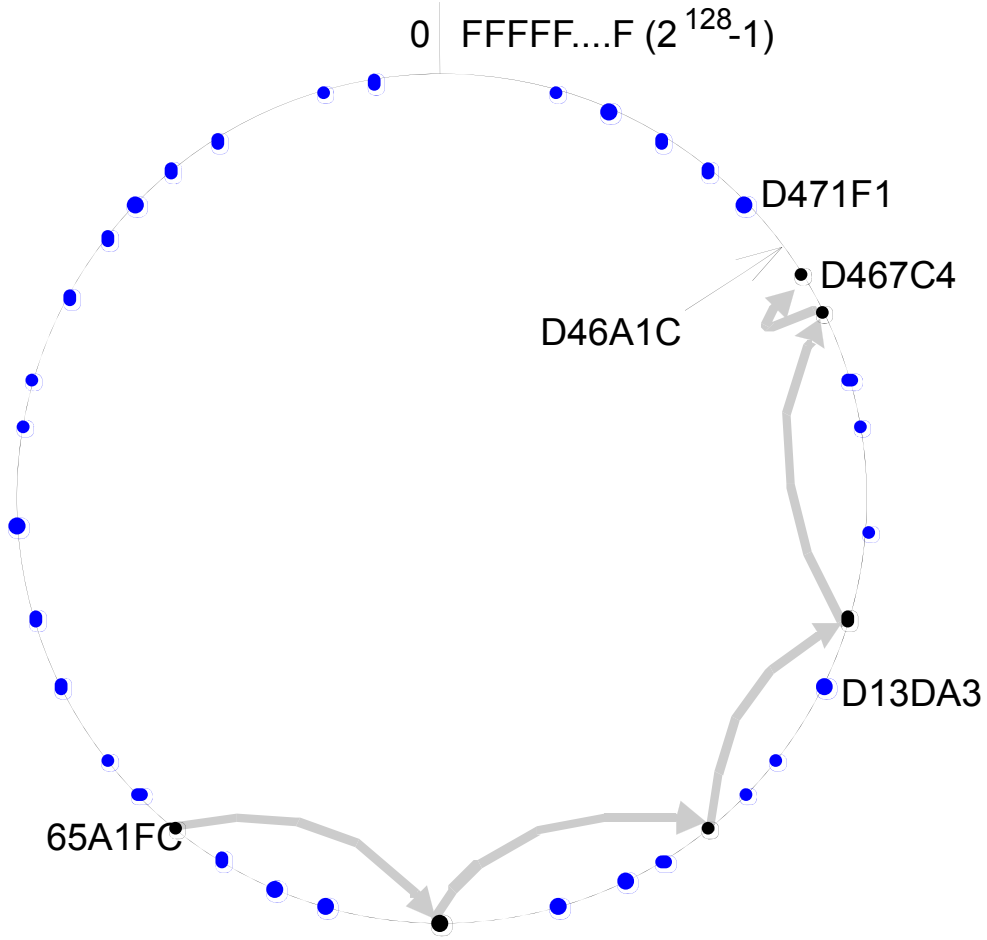
Primitivas



Source: Frans Kaashoek

Exemplos: Pastry (Squirrel), Tapestry, Chord, Kademlia (Torrent), ...

Exemplo de Encaminhamento Circular



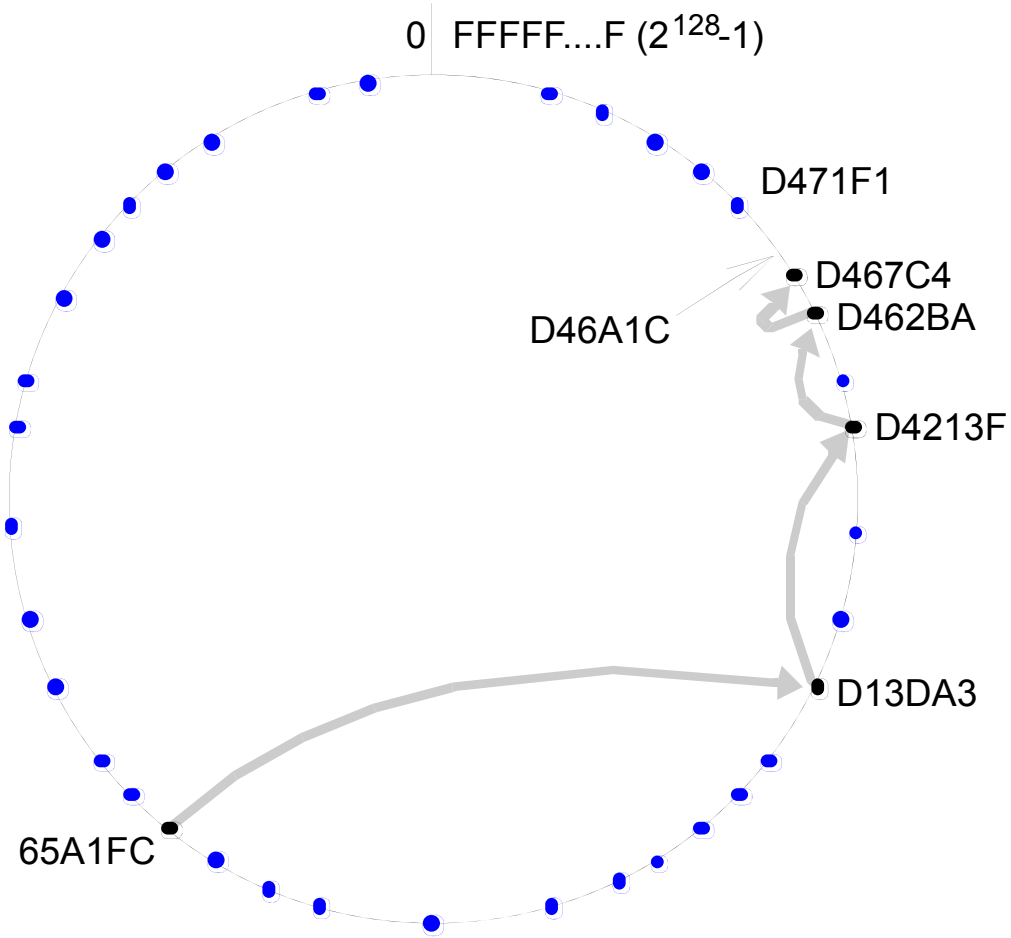
- Keyspace de 128 bits
- Os pontos representam nós activos.
- O Keyspace é circular: o nó 0 é adjacente ao nó $2^{128}-1$
- Cada nó contem a lista dos 2L nós vizinhos (L para cada lado)
- Uma mensagem é encaminhada para o nó cujo GUID é mais próximo do nó final
- Na figura, uma mensagem é encaminhada do nó 65A1FC para o nó D46A1C uma lista com L = 4
- Este algoritmo não é eficiente pois o número de saltos (*hops*) aumenta linearmente com a distância entre o nó de origem e de destino -> *O (N/2L)*
- Na realidade utilizam-se tabelas de encaminhamento mais complexas em cada nó

Exemplo de Tabela de Encaminhamento (Pastry)

p =	*GUID prefixes and corresponding nodehandles n*															
0	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*A*	*B*	*C*	*D*	*E*	*F*
	n	*n*	*n*	*n*	*n*	*n*		*n*	*n*	*n*	*n*	*n*	*n*	*n*	*n*	*n*
1	*60*	*61*	*62*	*63*	*64*	*65*	*66*	*67*	*68*	*69*	*6A*	*6B*	*6C*	*6F*	*6E*	*6F*
	n	*n*	*n*	*n*	*n*		*n*	*n*	*n*	*n*	*n*	*n*	*n*	*n*	*n*	*n*
2	*650*	*651*	*652*	*653*	*654*	*655*	*656*	*657*	*658*	*659*	*65A*	*65B*	*65C*	*65D*	*65E*	*65F*
	n	*n*	*n*	*n*	*n*	*n*	*n*	*n*	*n*	*n*		*n*	*n*	*n*	*n*	*n*
3	*65A0*	*65A1*	*65A2*	*65A3*	*65A4*	*65A5*	*65A6*	*65A7*	*65A8*	*65A9*	*65AA*	*65AB*	*65AC*	*65AD*	*65AE*	*65AF*
	n		*n*	*n*	*n*	*n*	*n*	*n*	*n*	*n*	*n*	*n*	*n*	*n*	*n*	*n*

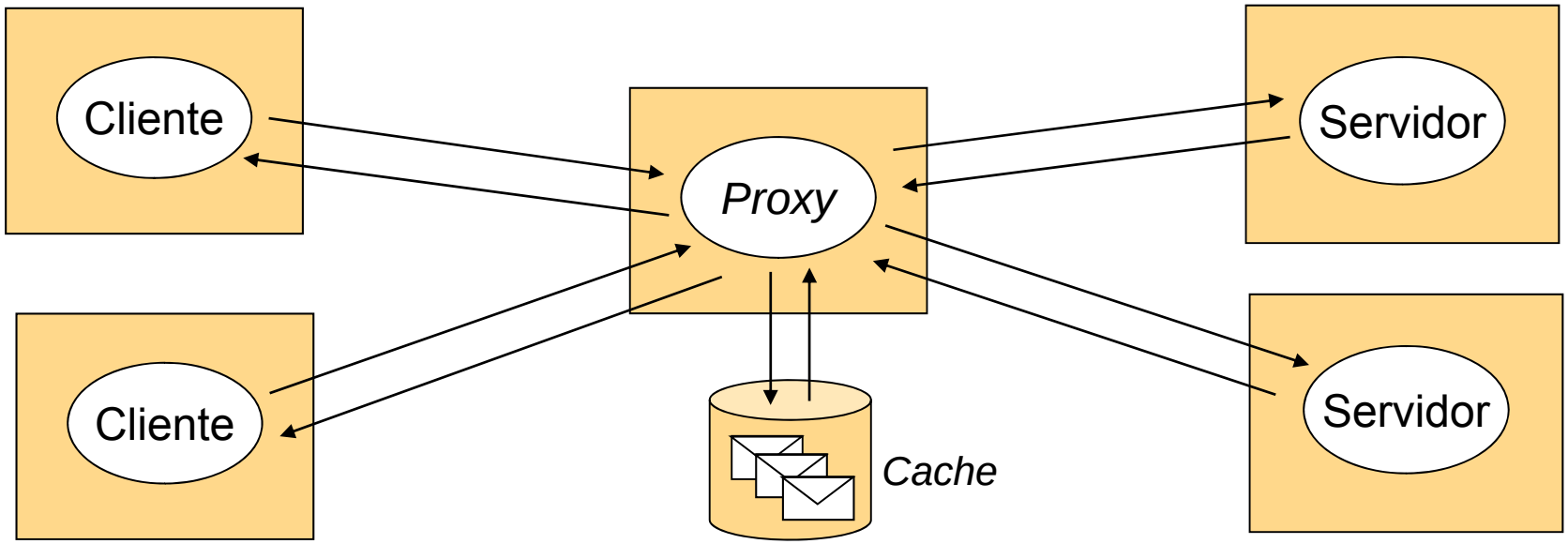
Tabela do nó cujo GUID começa por 65A1: *n* representa o par [GUID, IP address] indicando o próximo passo para as mensagens endereçadas a GUIDs com esse prefixo. Os valores em sombreado indicam correspondência de prefixo com o do presente GUID até ao número *p* de dígitos: enquanto o prefixo for comum, a tabela é percorrida por linha, caso contrário por coluna. Para um GUID de 128 bits haverá 32 linhas , uma por cada grupo algarismo hexadecimal (4 bits)

Encaminhamento com Tabela (Pastry)



- Encaminhamento do nó 65A1FC para o nó D46A1C utilizando tabelas de encaminhamento correctamente preenchidas
- O número de saltos (*hops*) para encaminhar uma mensagem é O ($\log_{16}(N)$)
- Baseado em Rowstron and Druschel [2001]

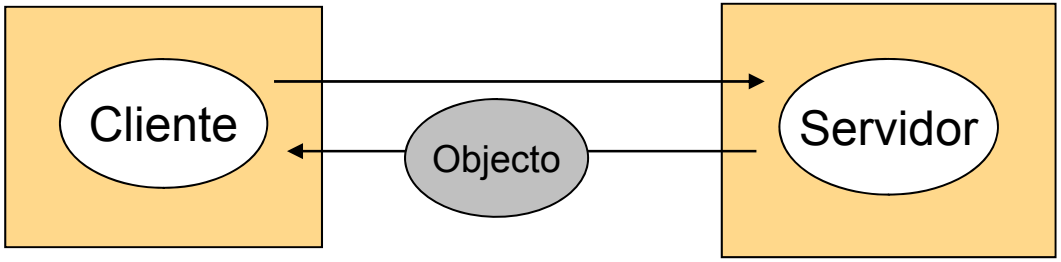
Utilização de *Proxy* e *Cache*



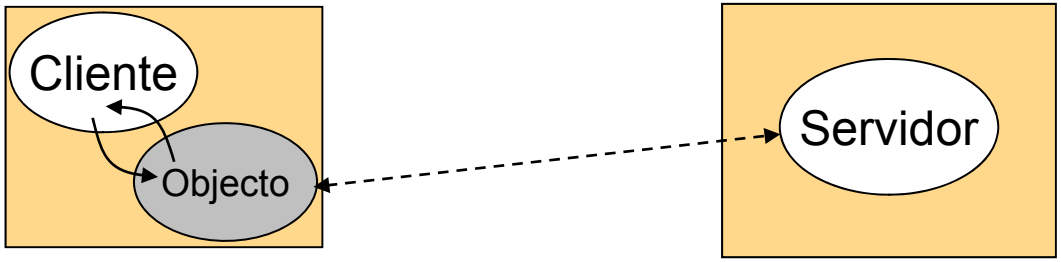
- O *proxy é um* representante de um serviço situado na proximidade imediata dos clientes
  - Simultaneamente cliente e servidor
- O *proxy* gere um *cache* que é um repositório de objectos mais frequentemente utilizados
  - Permite diminuir os tempos de acessos aos objectos e a carga dos servidores, introduzindo um certo grau de tolerância a falhas
  - A gestão do *cache* e a sua dimensão são essenciais para garantir a sua eficácia, devendo existir mecanismos para assegurar a sua coerência
- Ex: Proxy Web Server, *cache* NFS

Código Móvel

1. O pedido do cliente desencadeia o *download* de um objecto executável

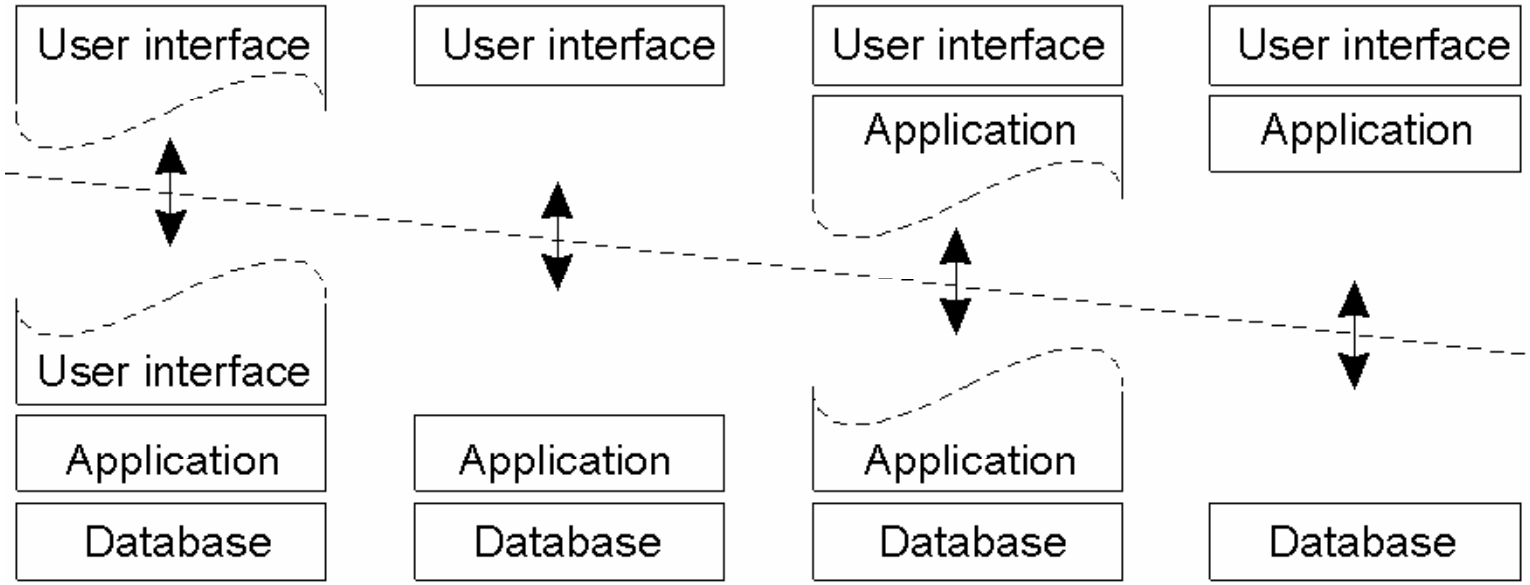


2. O cliente interage localmente com o objecto



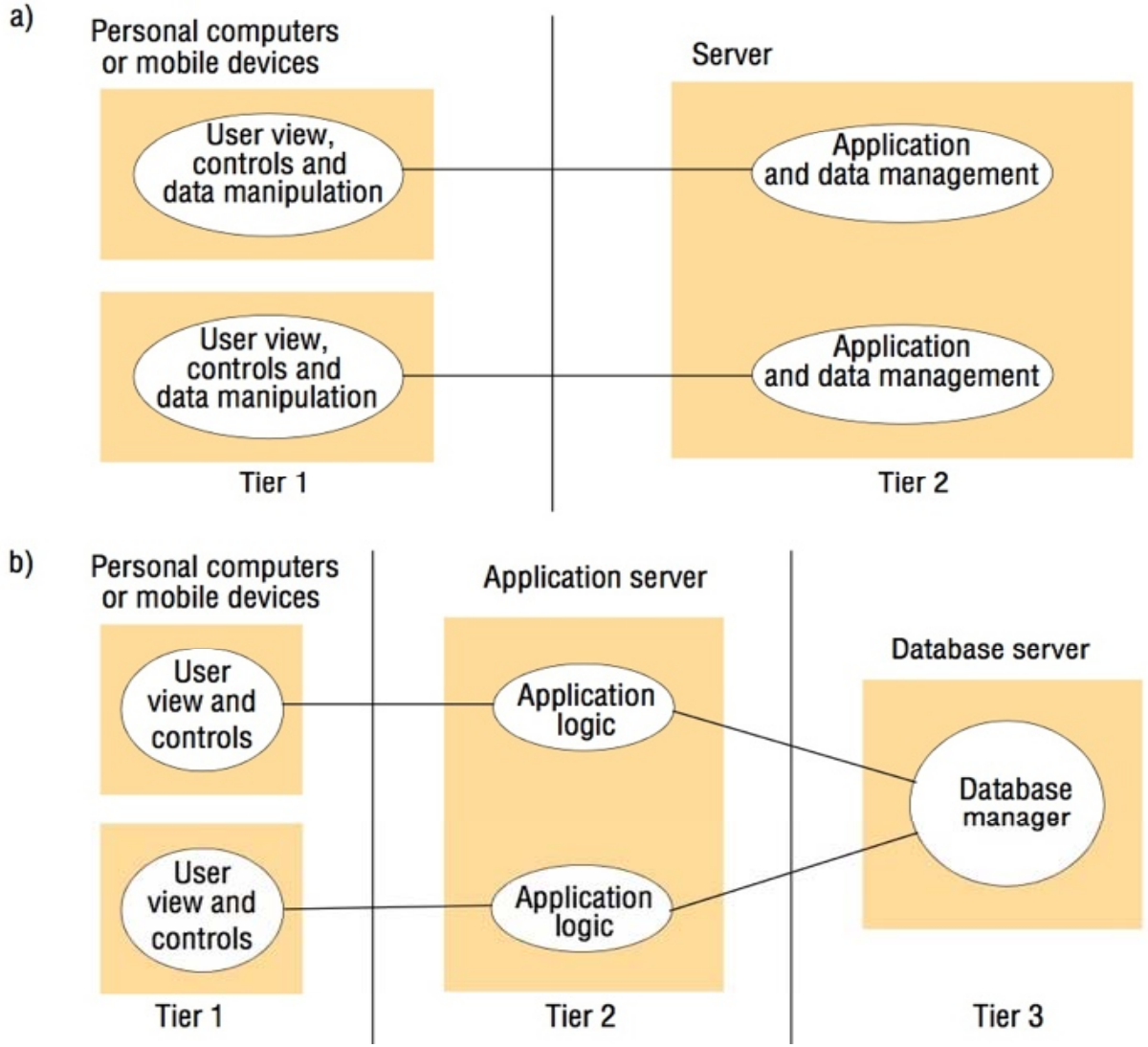
- O objecto é enviado para o cliente e executado localmente (Ex: Applet Java)
- A interacção com o cliente é feita localmente com o objecto
  - Melhora o desempenho e interactividade
  - Incrementa dinamicamente a funcionalidade do cliente
- O objecto pode interagir com o servidor para prestar serviços adicionais
- Segurança:
  - Necessidade de regras de carregamento e acesso do código móvel a recursos locais

Arquitecturas Multi-Níveis (*Multitier*)

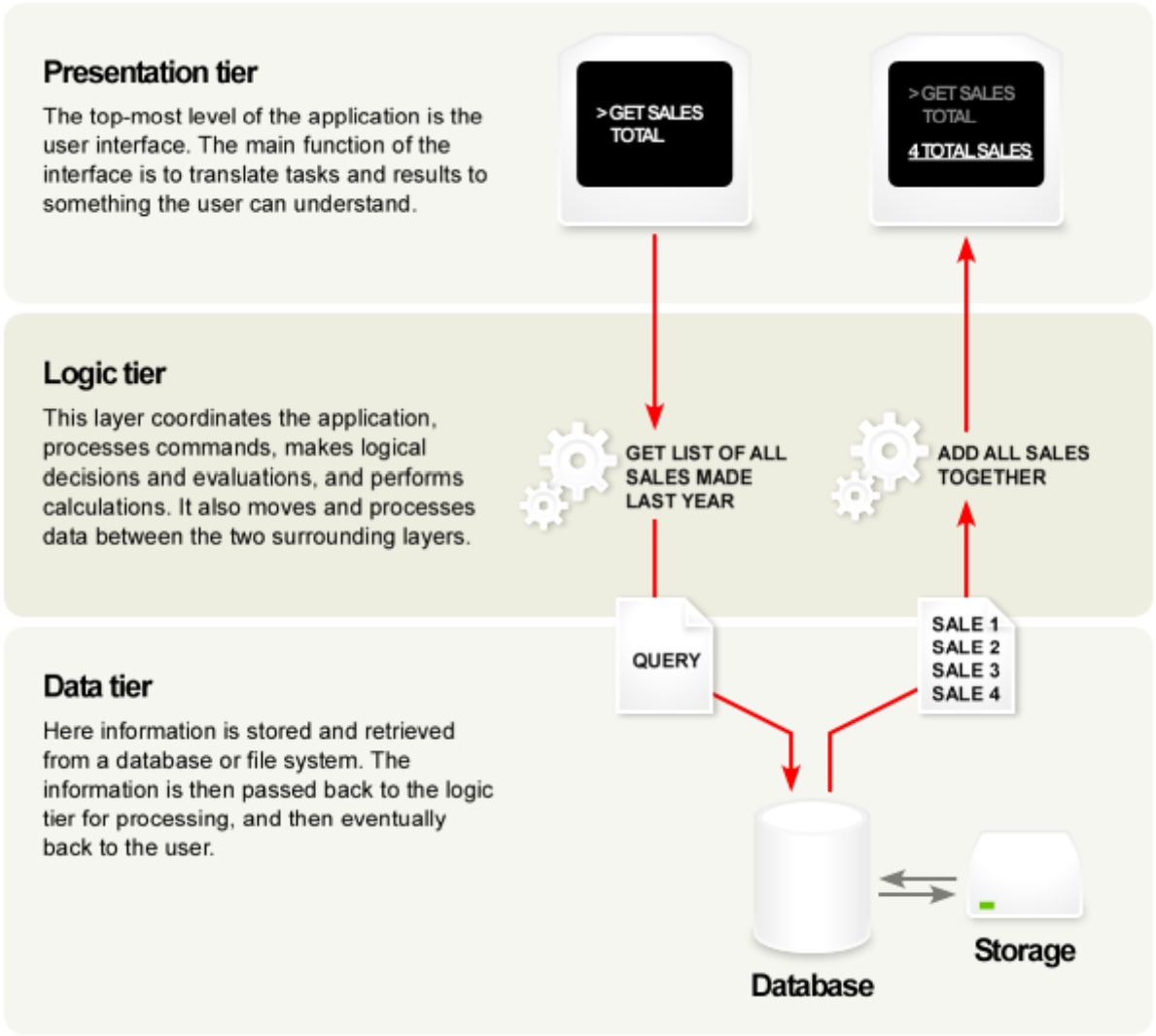


- A partição das funcionalidades entre cliente e servidor pode ser feita a vários níveis:
  - Terminais alfanuméricos clássicos
  - Terminais X – cliente corre OS e um servidor X
  - *Network Computers* – cliente corre OS mas não tem disco
  - Browser (*Thin Client)* – cliente só executa interface utilizador
  - Aplicações empresariais interactivas têm geralmente 3 camadas

Arquitecturas com 2 e 3 Níveis de Execução



Three-Tier Architecture

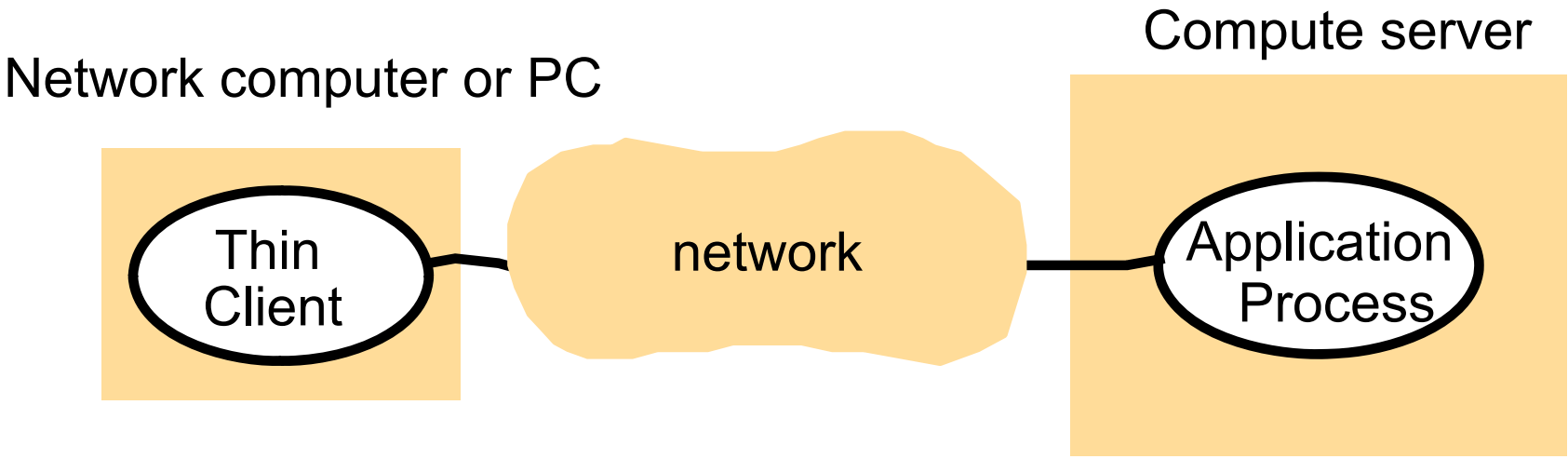


Apresentação

Lógica Aplicacional

Repositório de Dados

Cloud Computing



- Utilização das seguintes tecnologias
  - Computador standalone com sistema mínimo e browser
  - Virtualização de Sistemas (*hypervisor*)
  - Virtualização do Desktop
  - Gestão de Identidade Distribuída
- Modelos de utilização
  - IaaS (Infrastructure as a Service)
  - PaaS (Platform as a Service)
  - SaaS (Software as a Service)

Elementos da Arquitectura de Sistemas Distribuídos

- A definição da arquitectura de invocação de um Sistema Distribuído especifica um conjunto de elementos que caracterizam a forma como os seus serviços são invocados
  - Nível de Abstracção do Acesso
  - Interfaces dos Serviços
  - Objectos e Componentes
  - Modelo de Funcionamento

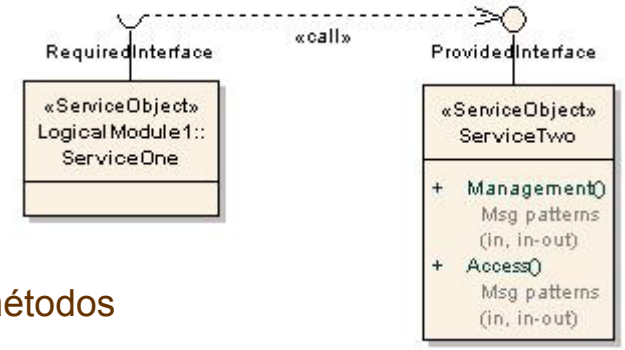
Nível de Abstracção

O acesso a serviços remotos pode ser feito:

- Por acesso directo ao protocolo de transporte
  - Ex: sockets com TCP e UDP
- Por invocação programática de funções remotas
  - Ex: SUN RPC, DCE/RPC, MS-RPC
- Por invocação de métodos de objectos remotos
  - Ex: Corba, Java RMI, DCOM, .NET
- Cada modo de acesso corresponde a um nível de abstracção diferente

Acesso a Serviços: Interfaces

- O conjunto de funcionalidades disponíveis num servidor é especificado através da definição das suas interfaces
  - Um serviço pode ter várias interfaces
  - As interfaces devem ser definidas formalmente



- Uma interface especifica
  - A semântica de invocação
    - Acesso directo às mensagens ou invocação remota de métodos
  - O tipo de ligação estabelecido
    - Persistente ou temporária
  - O formato das mensagens
    - Sequência (*stream*) de bytes ou dados estruturados pela aplicação
  - A semântica do envio
    - Síncrona ou assíncrona
- A interface tem um Identificador Único
  - Ex: socket (ip + porto)
- Uma interface pode permitir a invocação remota de métodos
  - Noção de RPC (*Remote Procedure Call*)
  - É necessário *middleware* no cliente e servidor - Ex: SUN RPC
  - Utilização de uma Linguagem de Definição de Interfaces - IDL

Características das Interfaces (i)

- Canal com ou sem ligação.
  - Com ligação - Estabelecido previamente entre processos interlocutores. Fiável, Bidireccional, Garante a sequencialidade das mensagens.
  - Sem ligação ou datagrama - Múltiplos interlocutores, mensagens curtas. (Cliente - Servidor)
- Portos de comunicação.
  - Objectos de interface entre os S.O. locais e os Protocolos de comunicação. Dois identificadores: Ref. local S.O; Identif. de rede associado ao protocolo de transporte.
- Semântica de envio.
  - Transf. Mensagem do espaço de endereçamento proc. emissor até ao porto receptor. Sincronização - Detecção de falhas de envio:
    - Assíncrona - Emite e desliga s/ garantia de chegada ao destino (Comunicação unidireccinal - FTP, Telnet).
    - Síncrona - Emissor bloqueado até receber confirmação do porto receptor.
    - Cliente/Servidor - Emissor bloqueado até receber mensagem de resposta.
- Estrutura da mensagem.
  - Estrutura individualizada ou Sequências arbitrárias de octetos de tamanho variável (byte stream) => Protocolo Superior Aplicação.

Características das Interfaces (ii)

- Semântica de recepção.
  - Retirar primeira mensagem no receptor ou bloquear porto na ausência de mensagens.
  - Existência de um ou vários canais (Modelo com ou sem ligação)
  - Exemplos (Select - sockets - Recepção múltipla - threads).
- Paralelismo.
  - Sincronização de primitivas de envio e recepção.
  - Concorrência de processos e tarefas.
  - Programação de processos que actuam como servidores.
- Faltas de comunicação.
  - Impossibilidade de entrega de mensagens
  - Quebra do canal de comunicação.
  - Normalmente sem mec. tolerância a falhas (dep. c/ ou s/ ligação).
  - Signals (Unix)
- Difusão de mensagens.
  - Envio de mensagem a múltiplos receptores (restritos a grupos de portos).
  - Com garantia de entrega fiável a todos os destinatários.
  - Envio melhor esforço, s/ garantia de entrega.

Acesso a Serviços: Objectos

- Os serviços distribuídos podem ser acedidos através da instanciação e invocação de objectos remotos
  - Transparência de acesso e localização
- O modelo de objectos distribuídos é uma extensão do modelo clássico de programação por objectos
  - A possibilidade de invocar objectos remotos é implementada por uma camada de *middleware*
  - Java RMI e Corba: ver Capítulo 4
- Para invocar um objecto remoto
  - O objecto remoto regista-se num servidor de objectos (*Broker*)
  - O objecto cliente procura o objecto no servidor, recebe um identificador (referência) e realiza a sua instanciação
  - Os métodos do objecto acessíveis remotamente são definidos pela *interface* (conceito OO) do objecto

Modelos de Funcionamento

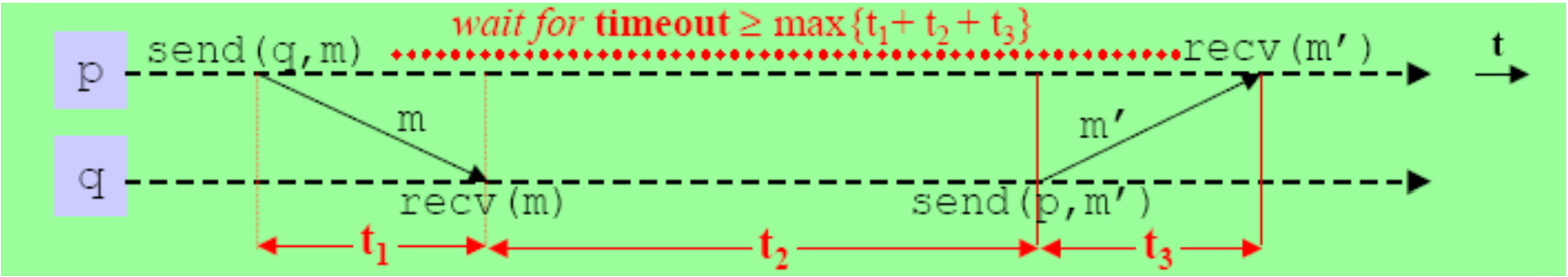
- As arquitecturas analisadas são constituídas por processos que comunicam através de mensagens
  - Definem apenas aspectos estáticos e topológicos
- Para caracterizar efectivamente um SD é necessário definir o seu comportamento dinâmico em cada um dos aspectos fundamentais:
  - Modelo de Interacção
  - Modelo de Falhas
  - Modelo de Segurança

Modelo de Interacção

- O funcionamento de um sistema corresponde à execução de um algoritmo distribuído
  - Inclui execução de código e envio e recepção de mensagens
  - Transferência de informação e de controlo (eventos)
- A interacção entre processos é condicionada por
  - Desempenho dos canais de comunicação
    - Latência
    - Largura de Banda
    - *Jitter* (variância da latência de um conjunto de mensagens)
  - Relógios e eventos temporais
    - É impossível coordenar os relógios de todos os computadores numa rede
    - Existem técnicas para definir uma temporalidade comum
- O modo como estes aspectos são geridos define o tipo de interacção
  - Síncrona / Assíncrona

Sistemas Distribuídos Síncronos

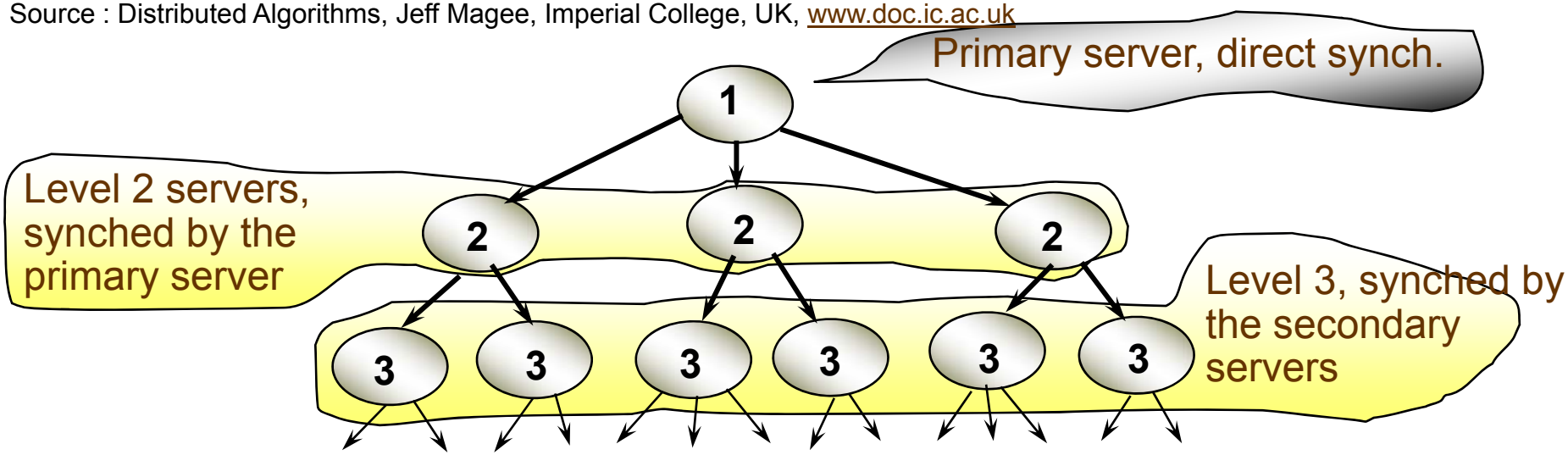
- Podem ser estabelecidos limites mínimos e máximos para:
  - O tempo de execução de cada passo de um processo
  - O tempo de transmissão das mensagens
  - A deriva dos relógios dos processadores
- A implementação de sistemas síncronos é complexa
  - Tema de investigação
  - Conceito utilizado para efeitos de modelização
- Exemplo: dois processo p e q trocam mensagens
  - Sendo conhecidos t_1 , t_2 e t_3 pode-se determinar o *timeout* para detectar falhas



Source: Distributed Algorithms, Jeff Magee, Imperial College, UK, www.doc.ic.ac.uk

O Network Time Protocol (NTP)

Source : Distributed Algorithms, Jeff Magee, Imperial College, UK, www.doc.ic.ac.uk

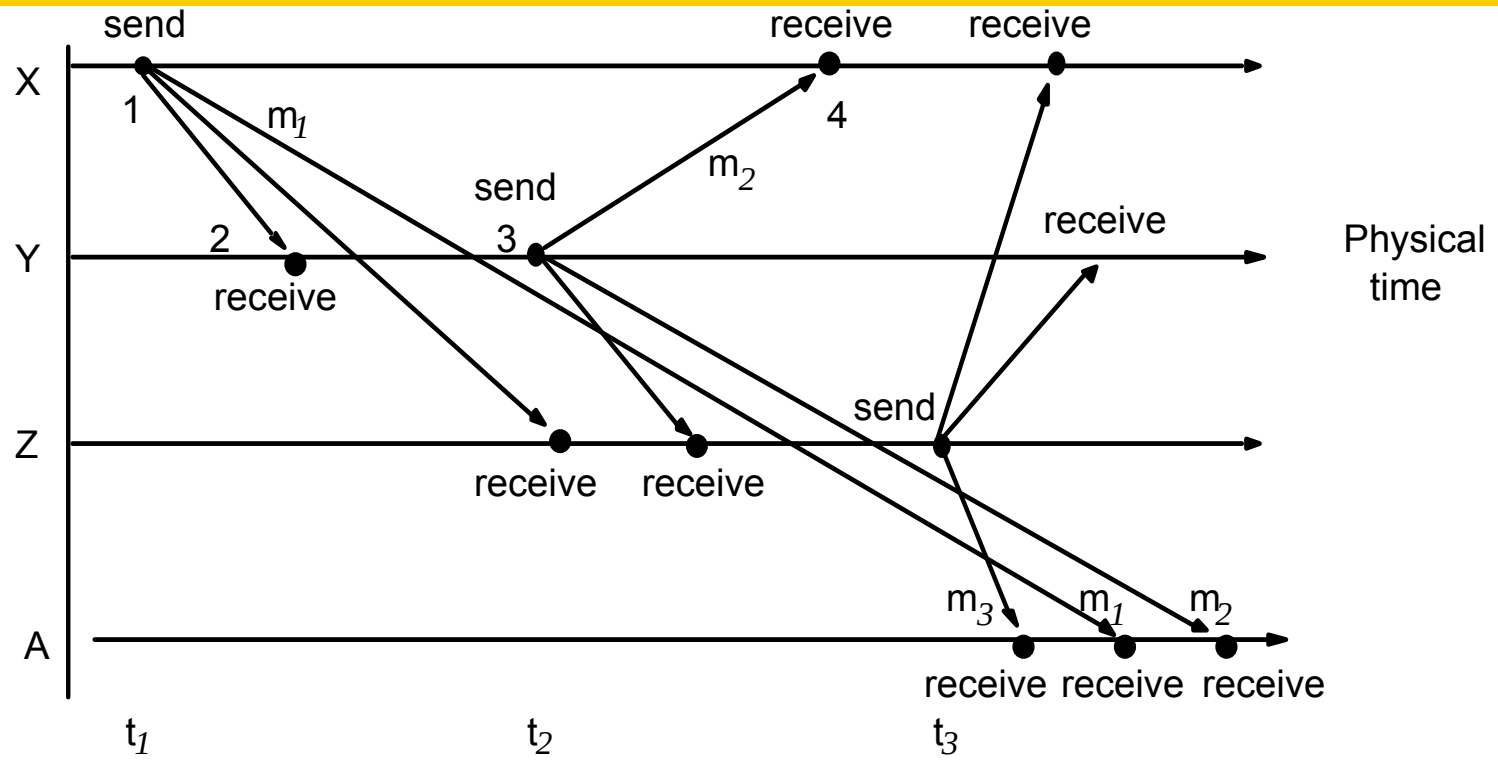


- Utiliza um conjunto de servidores para sincronizar os nós de uma rede
- Os servidores de tempo estão ligados por uma sub-rede de sincronização, e cada nó sincroniza os seus descendentes
- Devido à impossibilidade de prever o tempo que a propagação leva de uns nós para os outros, não é absolutamente preciso
- Não permite criar uma base temporal em relação à qual todos os eventos do sistema distribuído possam ser referenciados

Sistemas Distribuídos Assíncronos

- Não podem ser estabelecidos limites temporais
  - O tempo de execução de um processo é desconhecido
    - a única certeza é que tem tempo finito
  - Não existe limite temporal para o tempo de transmissão
  - Não existe sincronização de relógios
    - no entanto é possível estabelecer uma ordenação lógica de eventos
- É o caso da maioria dos sistemas existentes
  - Há problemas que não podem ser resolvidos por sistemas assíncronos
- É possível estabelecer uma noção lógica do tempo
  - Algoritmos de sincronização temporais (Lamport)
    - Introduzindo relações de ordem entre eventos (*aconteceu antes*)
      - Se E2 decorre de E1, então E1 aconteceu antes de E2
    - Utilizando relógios lógicos
      - Estampilhas temporais incrementadas a cada evento

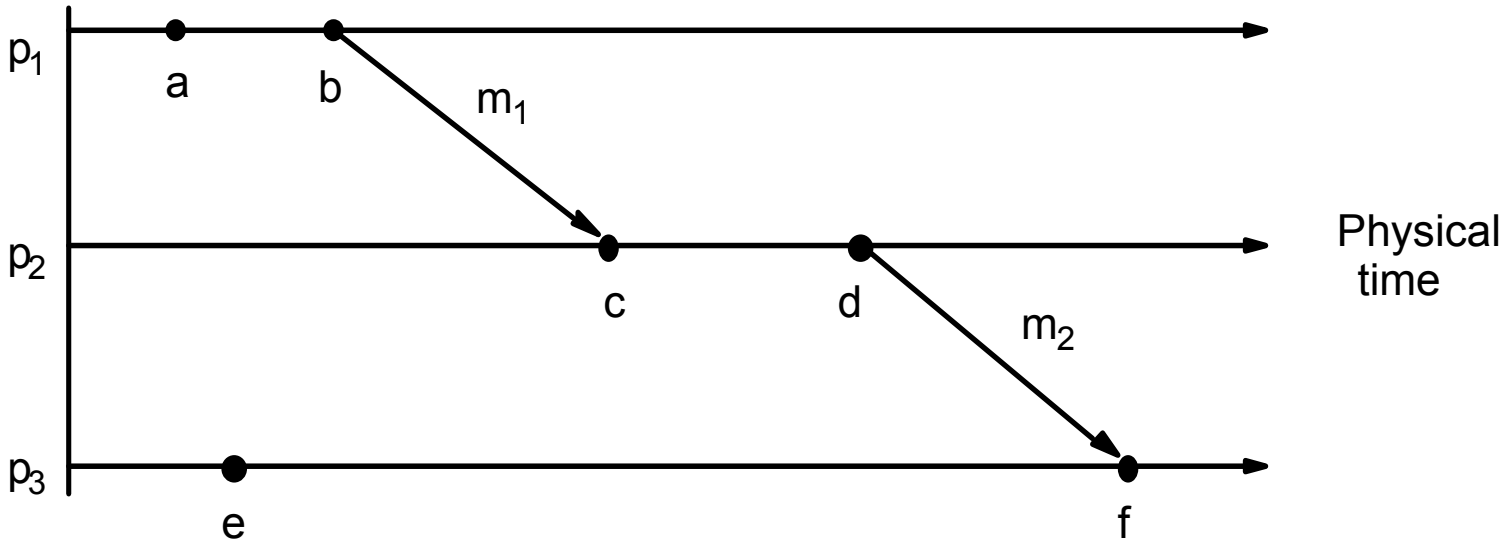
Ordenação de Eventos



- Supondo que X, Y, Z e A estão numa mesma lista de e-mail, se X enviar uma mensagem para a lista, esta é recebida por Y, Z, e A
- Se Y e Z responderem para a lista, A pode receber mensagens de resposta antes da mensagem original devido à diferença de latências
- Isto pode ser evitado se a cada mensagem estiver associada uma estampilha temporal estabelecida através dos algoritmos de Lamport

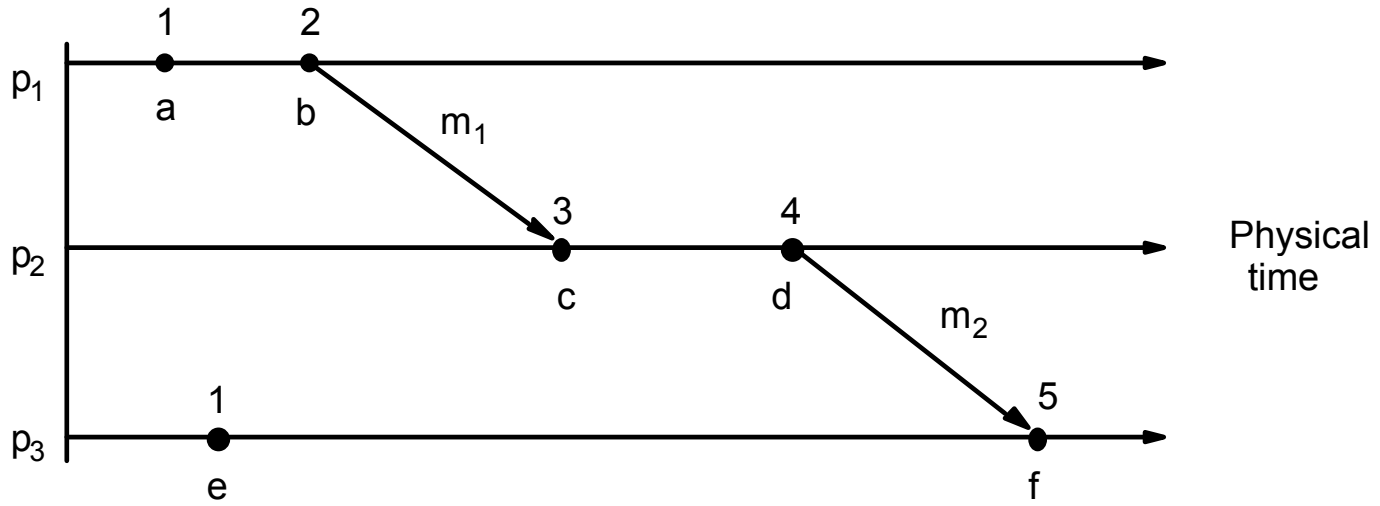
Relação *aconteceu antes*

- A relação aconteceu antes (→) é definida por
  - Se E1 e E2 ocorreram no mesmo processo e E1 anterior a E2, então E1 → E2
  - Se E1 e E2 correspondem ao envio e recepção de uma mensagem, então E1 → E2
  - Se E1 → E2 e E2 → E3, então E1 → E3
- Na figura temos:
  - a → b; b → c; c → d; d → f
  - Pode-se concluir: a → b → c → d → f



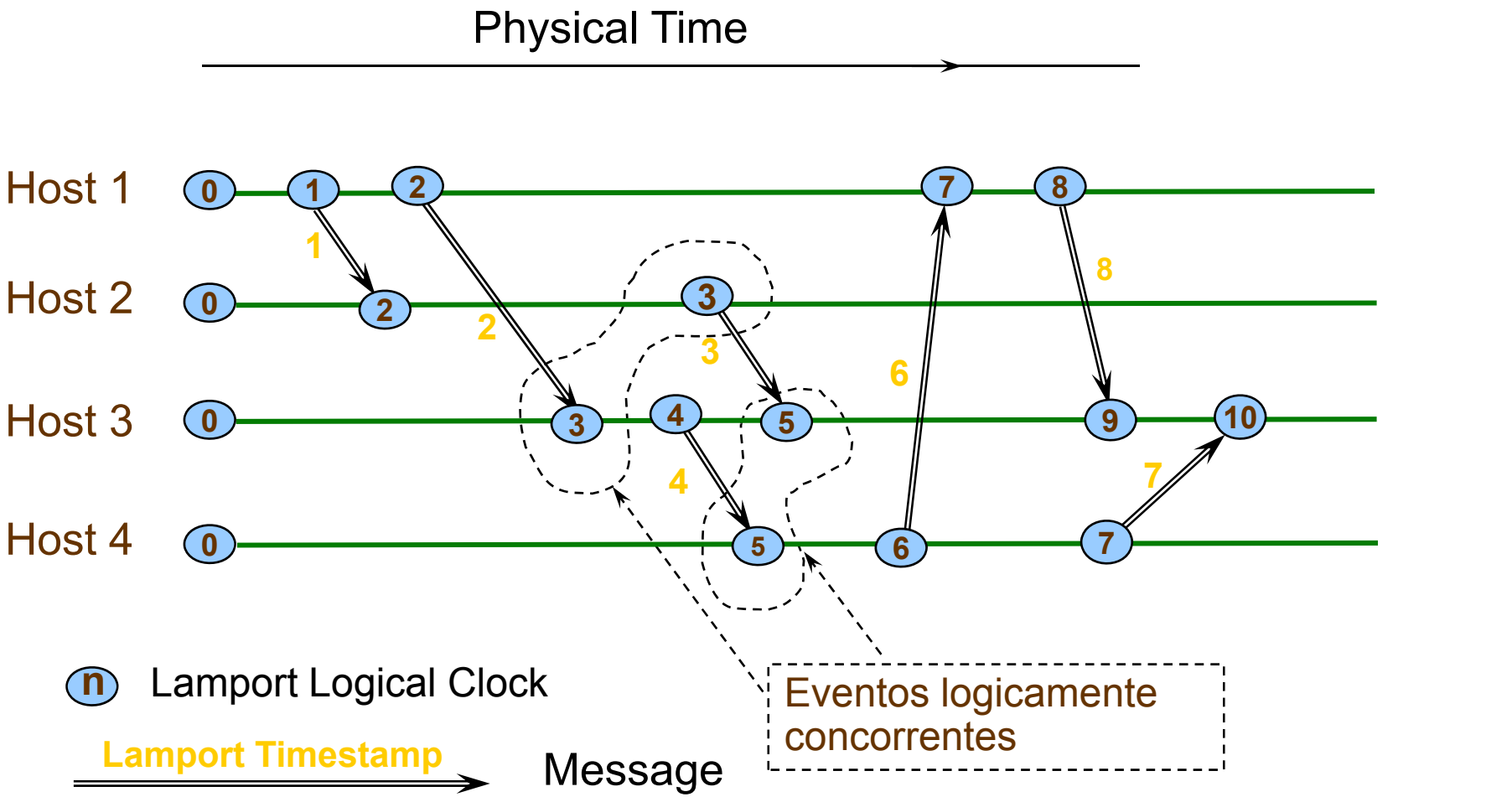
Relógio Lógico de Lamport

- Utiliza um contador crescente e monotónico para estender a relação *aconteceu antes* a outros processos:
  - O processo **i** incrementa o seu contador a cada evento: **Li = Li + 1**
  - O processo **i** envia o valor do seu contador em cada mensagem: **send(m, Li)**
  - O processo **j** que recebe calcula **Lj = max(Li, Lj)** e incrementa-o devido ao evento de recepção
- Na figura temos:
  - P1: 1 → 2
  - P2: 2 → 3 → 4
  - P3: 1 → 5; 4 → 5
  - Assim se Ei → Ej, então Li < Lj
  - Infelizmente se Li < Lj não é garantido que Ei → Ej
  - Para isso são necessários relógios vectoriais



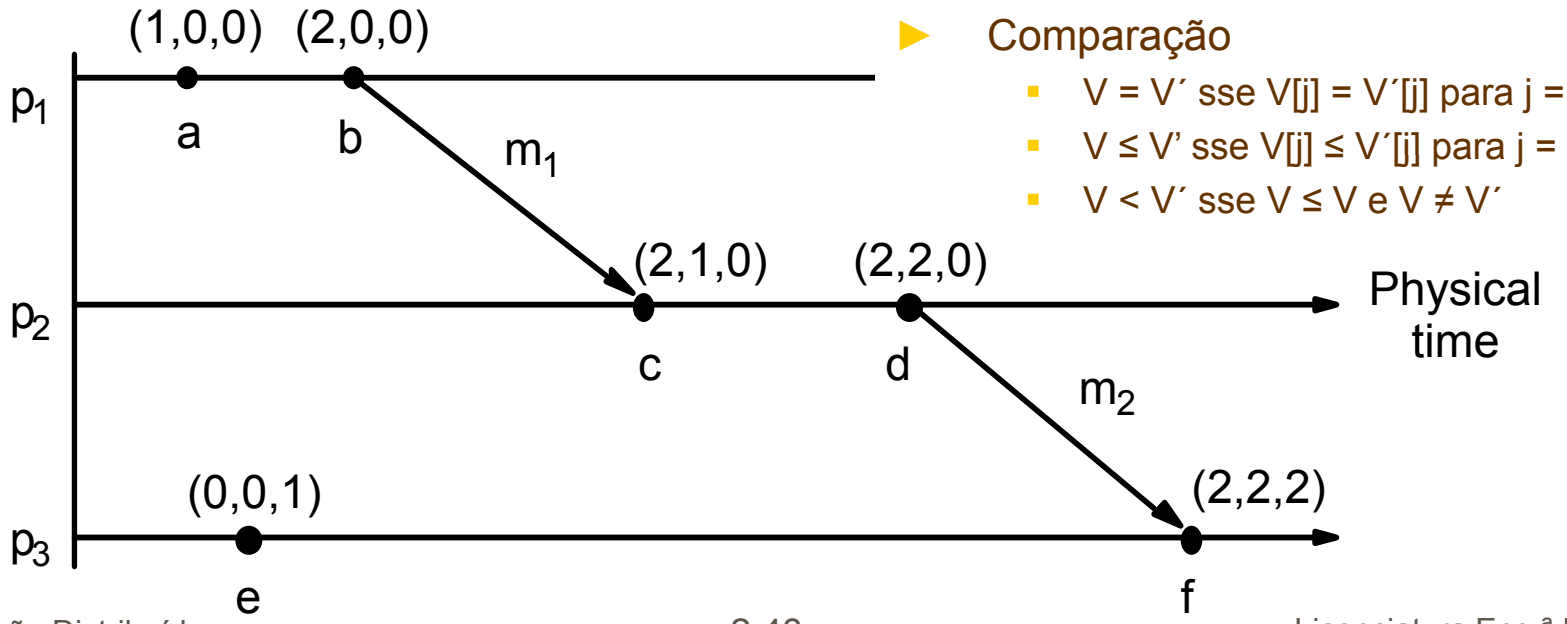
Exemplo de Relógio de Lamport

Source : Distributed Algorithms, Jeff Magee, Imperial College, UK, www.doc.ic.ac.uk



Relógios Vectoriais

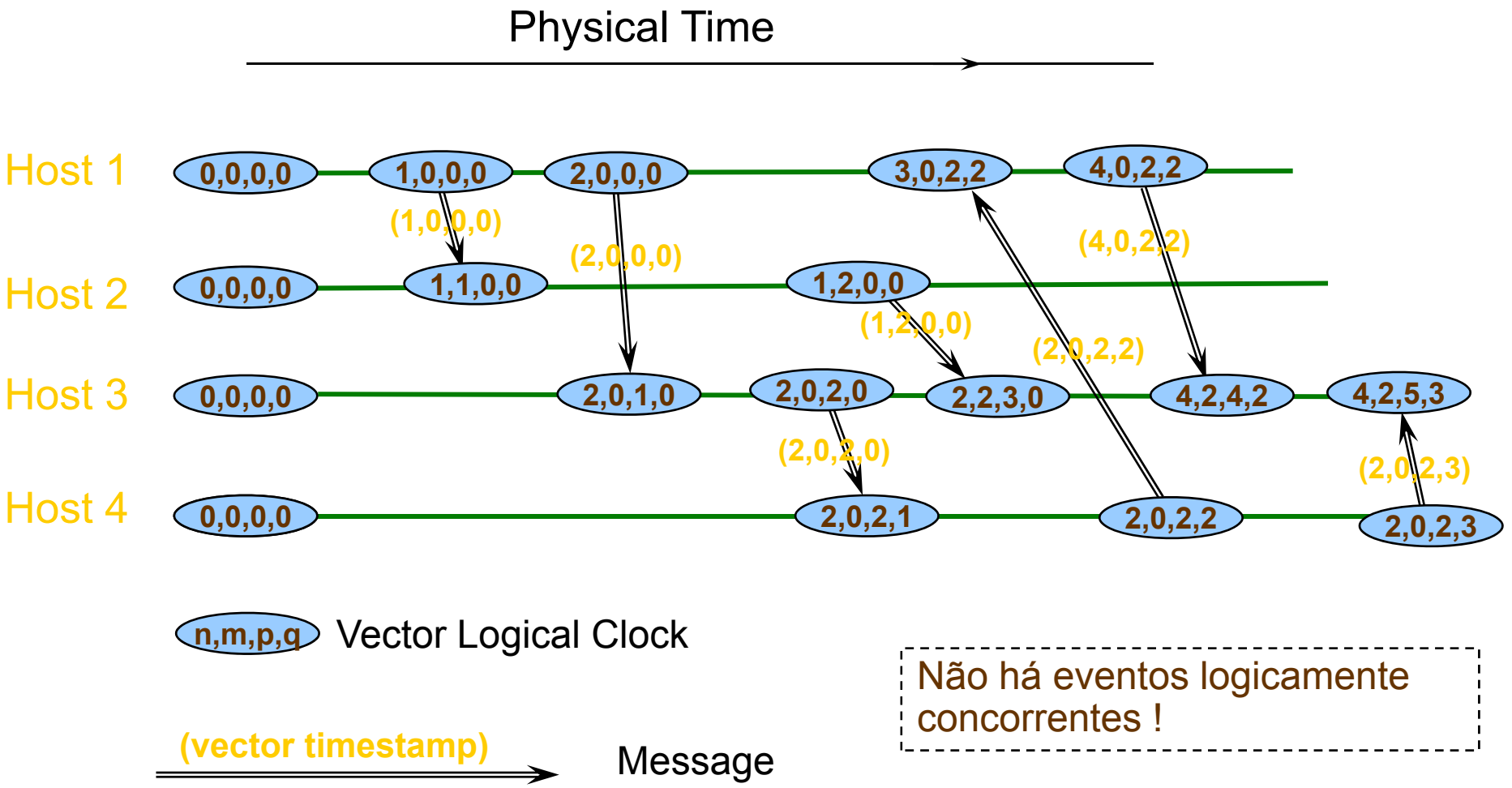
- Vector com dimensão igual ao número de processos N
  - $\mathbf{V^i}$ designa o valor do relógio no processo **i**
  - Inicialmente $\mathbf{V^i[j]} = 0$ para todos processos i = [1,…,N]
  - Em cada evento, um processo **i** incrementa o conteúdo do seu indíce: $\mathbf{V^i[i] = V^i[i] + 1}$
  - Cada processo inclui o valor de $\mathbf{V^i}$ nas mensagem que envia
  - Quando um processo **i** recebe uma mensagem de **j**, faz $\mathbf{V^i[j] = max\,(V^i[j], V^j[j])}$ e incrementa $\mathbf{V^i[i]}$
- Em cada processo
  - $\mathbf{V^i[i]}$ é o número de eventos ocorridos no processo **i**
  - $\mathbf{V^i[j]}$ com $\mathbf{i \neq j}$, é o número de eventos do processo **j** de que **i** teve conhecimento



- Comparação
  - V = V´ sse V[j] = V´[j] para j = {1, … , N}
  - V ≤ V' sse V[j] ≤ V´[j] para j = {1, … , N}
  - V < V´ sse V ≤ V e V ≠ V´

Exemplo de Relógio Vectorial

Source : Distributed Algorithms, Jeff Magee, Imperial College, UK, www.doc.ic.ac.uk



Modelo de Falhas

- Num SD tanto os processos como os canais de comunicação estão sujeitos a falhas
  - Podem ter comportamentos diferentes dos esperados
  - A probabilidade de ocorrência de falhas é sempre positiva, podendo só ser minimizada
- O modelo de falhas define
  - Os tipos de falhas que podem ocorrer num sistema
  - Que falhas podem ocorrer nos seus diferentes componentes
  - Definição do comportamento do sistema em presença das falhas previstas
- Tipos de Falhas
  - Por omissão
  - Temporais
  - Arbitrárias ou bizantinas

Falhas por Omissão

- Designam os casos em que um processo ou um canal falha a realização de uma acção de deveria executar
- Falha de um Processo
  - A falha por omissão típica de um processo é o *crash*
  - O estado de falha nítida de um processo designa-se por *fail-stop*
  - A detecção e obtenção de consenso sobre um *fail-stop* é difícil num sistema assíncrono
    - Pode ser um *crash* ou um canal saturado
- Falha de Mensagem
  - Mensagem não entregue
  - A falha pode ter lugar na emissão (falha de envio) ou na recepção (falha de recepção)
- As falhas por omissão são falhas benignas
  - No sentido em que provocam comportamentos previsíveis

Falhas Temporais e Arbitrárias

- Falhas temporais: designam a não ocorrência de eventos dentro do tempo que estava previsto para a sua ocorrência
  - Tem sobretudo significado num sistema síncrono
  - Num sistema assíncrono, as falhas temporais não têm significado e são assimiláveis a falhas por omissão
  - Os sistemas em tempo real são particularmente sensíveis a falhas temporais
- Falhas Arbitrárias ou Bizantinas: designam a ocorrência do pior cenário de falha
  - Um processo retorna valores errados na invocação de um serviço
    - Pode ser detectada muito depois da sua ocorrência
  - Um canal transmite uma mensagem com dados corrompidos
    - As falhas arbitrárias de transmissão são mais raras devido ao controle exercido pelos protocolos de comunicação

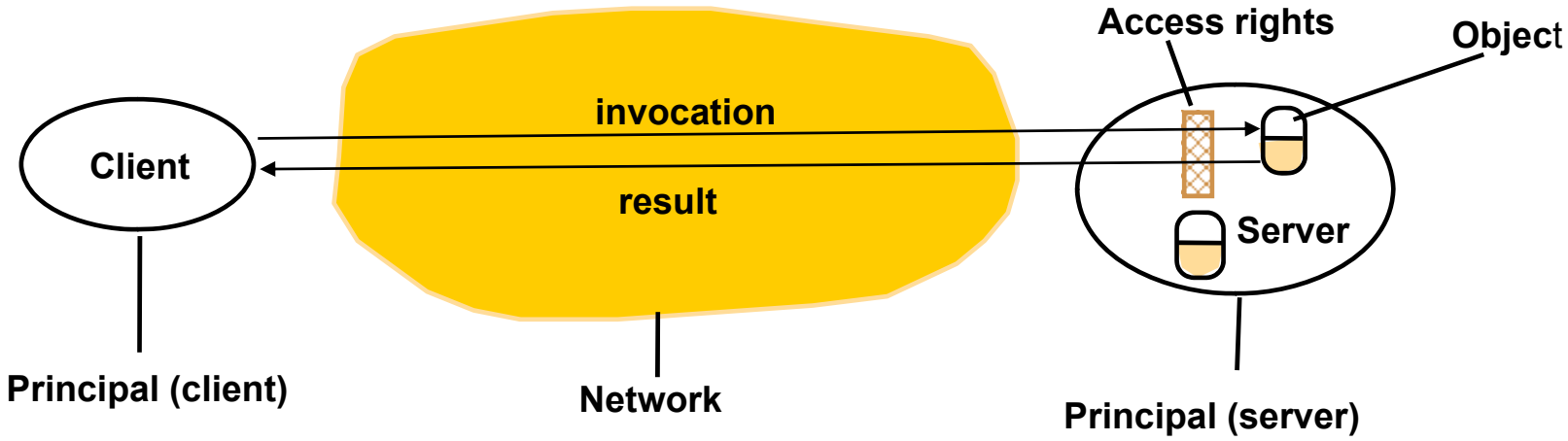
Máscara de Falhas

- Apesar das falhas serem inevitáveis, é possível construir sistemas fiáveis
- Certos tipos de falhas podem ser mascaradas
  - Reduzindo os seus efeitos sobre o funcionamento do sistema
- As falhas por omissão podem ser mascaradas por repetição ou replicação
  - Ex: repetir um mensagem não recebida, utilizar canal redundante
- Certas falhas arbitrárias podem ser transformadas em falhas por omissão
  - Um pacote com dados corrompidos é descartado através do CRC
  - Um processo que falhou pode ser reinicializado a partir de dados de sincronização (*checkpoint*)
- As falhas temporais são as mais difíceis de mascarar
  - Sistemas em Tempo Real

Modelo de Segurança

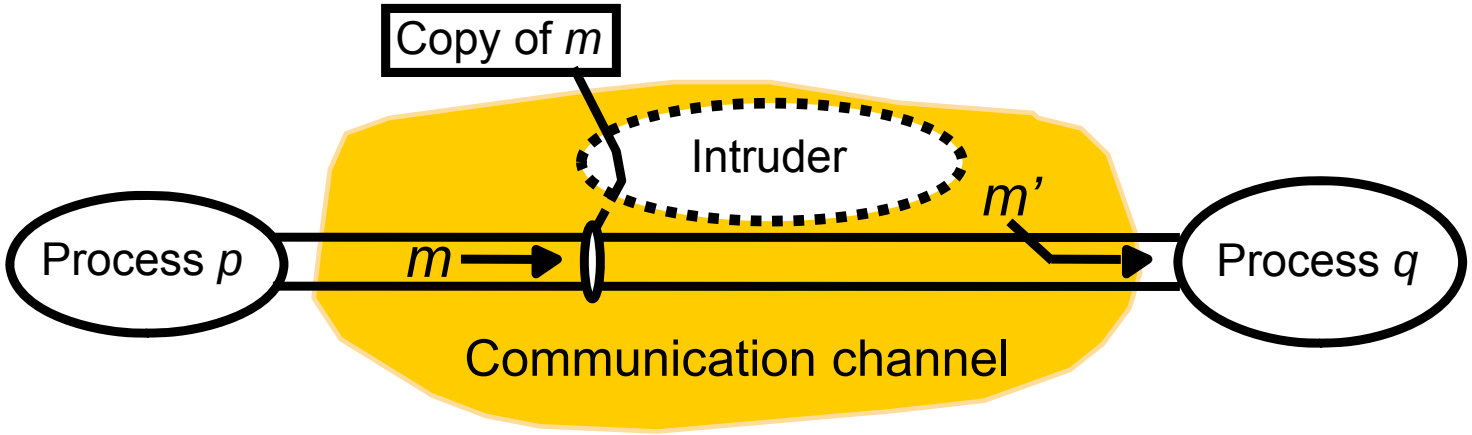
- Uma das razões para a utilização de Sistemas Distribuídos é a partilha de informação
  - Informação que é valiosa para os utilizadores
- O Modelo de Segurança permite ao sistema identificar as ameaças e os meios para as evitar
- A segurança num Sistema Distribuído pode ser obtida:
  - Protegendo os objectos (dados e código) encapsulados pelos processos contra acessos não autorizados
  - Protegendo os processos e os canais de comunicação utilizados para a sua interacção

Protecção dos Objectos



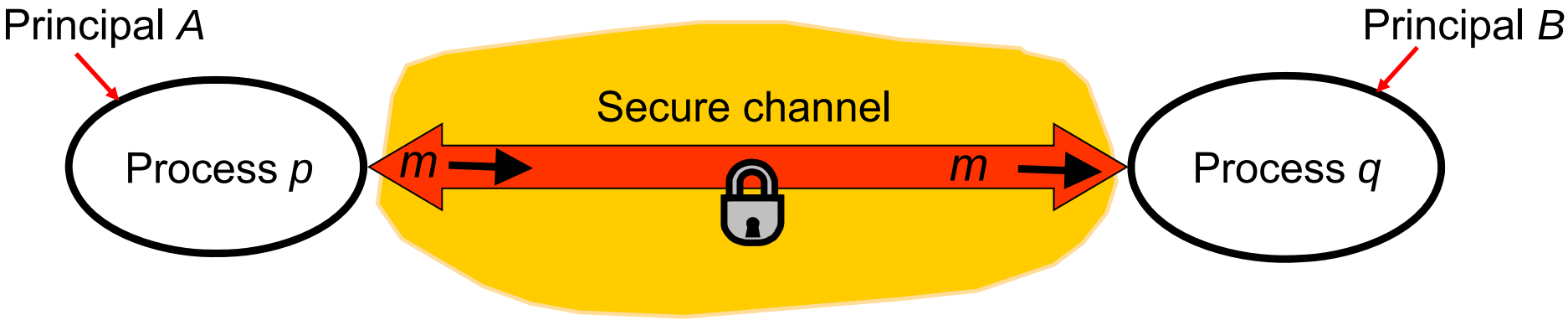
- Num Sistema Distribuído os objectos são encapsulados por processos e fornecem serviços a clientes
- Para poder estabelecer que clientes (utilizadores ou processos) têm o direito de invocar que objectos é preciso conhecer
  - A identidade dos intervenientes deve ser estabelecida =► noção de ***principal***
  - Os direitos de acesso devem ser respeitados =► controle de acessos

Ameaças aos Canais de Comunicação



- Para modelar as ameaças de segurança, introduz-se a noção de **Intruso malévolo** que personaliza qualquer entidade capaz de realizar acções não autorizadas no objectivo de comprometer os dados ou o funcionamento do sistema
- O Intruso pode:
  - Enviar mensagens a qualquer processo com interfaces públicas
    - Forjando endereços e identidade fazendo-se passar por quem não é
  - Aceder aos conteúdo do canal de comunicações
    - Copiar, modificar, remover e reenviar mensagens
  - Saturar o canal ou o processo enviando mensagens repetidamente

Canais de Comunicação Seguros



- A utilização de canais de comunicação seguros permite evitar algumas das ameaças conhecidas
- Para estabelecer canais de comunicação seguros são utilizadas técnicas de segurança
  - Criptografia de chaves secretas ou públicas
    - Encriptação, certificação e integridade de mensagens
    - Autenticação dos intervenientes (***principals***)
  - Protocolos de Segurança
    - *SSL - Secure Sockets Layer, TLS - Transport Level Security*
    - *VPNs - Virtual Private Networks*

Mas não resolvem:

- *Denial of Service Attacks*
- Certificação Código Móvel
- Imaginação dos *Hackers*…
- Ingenuidade utilizadores…

Fim do Capítulo 2

- Resumo dos conhecimentos adquiridos
  - Arquitecturas de Sistemas Distribuídos
    - Principais variantes e características
  - Métodos de Acesso a Serviços
    - Mensagens, Interfaces e Objectos
  - Modelos de Interacção
    - Sistemas Síncronos
    - Sistemas Assíncronos
    - Noções de Temporalidade
  - Modelo de Falhas
    - Tipos de Falhas
    - Máscara de Falhas
  - Modelo de Segurança
    - Ameaças
    - Protecções
- Trabalho Individual Complementar
  - Ver página de Leslie Lamport
    - http://research.microsoft.com/users/lamport/pubs/pubs.html
    - Tentar ler **"Time, Clocks and the Ordering of Events in a Distributed System"**