✓Modelos e Arquitecturas de Sistemas Distribuídos

Modelos de Sistemas Distribuídos

MODELOS FISICOS

- **Tipos de computadores , periféricos e conectividade entre eles;**

MODELO ARQUITECTURAL

- **Descreve o sistema em termos das tarefas e da comunicação entre elas.**
- **Quais as partes envolvidas e os protocolos de interacção entre elas:**
  - ***Client-Server*;**
  - ***Peer-to-Peer;***
  - **etc.**

MODELOS ABSTRACTOS

- **Perspectiva abstracta das soluções para os problemas e desafios existentes:**
  - **inexistência de um relógio global;**
  - **comunicação e interacção;**
  - **segurança;**
  - **tolerância a falhas;**
  - **etc.**

A grande questão

Onde colocar e como relacionar as componentes do Sistema Distribuído ?

- **Onde colocar as componentes do Sistema Distribuído na rede de computadores, tendo em conta os padrões de distribuição de dados e carga (*workload*) no acesso aos mesmos, garantindo que o sistema:**
  - **É fiável (*Reliable*) - probabilidade de não falhar**
  - **É fácil de gerir;**
  - **É adaptável (expansível)**
  - **É efectivo em custos**

Se considerarmos um sistema distribuído com 4 componentes com probabilidade de não falhar de 99% (muito bom), temos 0,99x0,99x0,99x0,99=0,96, isto é, o sistema distribuído tem só 96% de probabilidade de não falhar.

As partes constituintes dos Sistemas Distribuídos

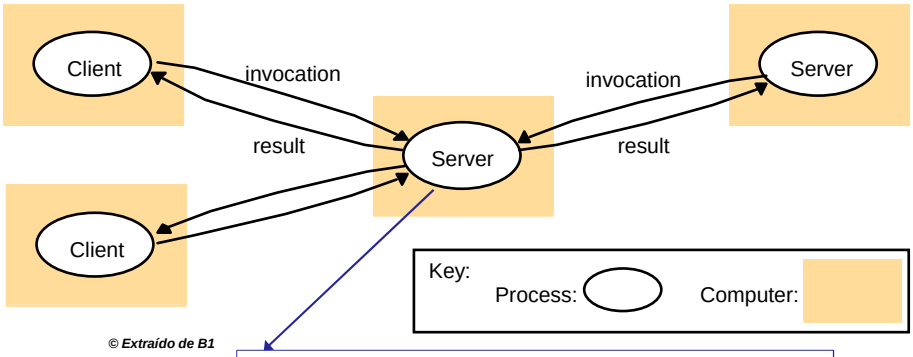
Quais os componentes constituintes que se relacionam, tendo em conta as funcionalidades e os padrões de comunicação entre eles ?

Processo, Objecto ou Serviço:

- **Servidor - Aceita pedidos, processa-os e devolve um resultado;**
- **Cliente – Realiza pedidos e obtém os resultados;**
- **Peer - Coopera com outros pares de forma simétrica por forma a executar uma tarefa;**

Modelo Cliente Servidor

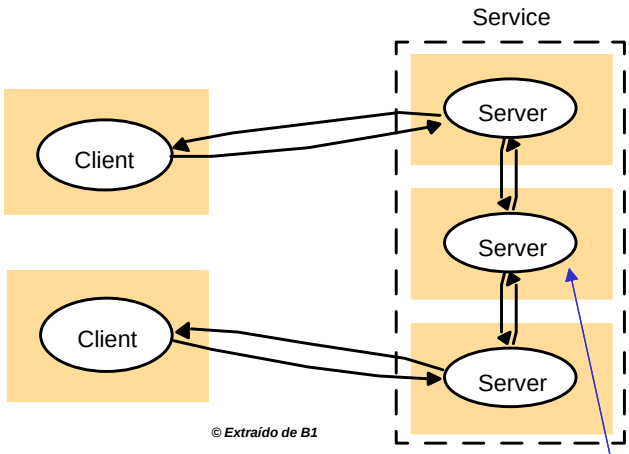
Os clientes invocam servidores individuais. Note-se que um servidor pode também ser cliente de outro servidor



© Extraído de B1

Função de intermediário: *Broker, Router, Load Balancer, Integrator, Transformer, Proxy, etc.*

Múltiplos servidores disponibilizam um Serviço

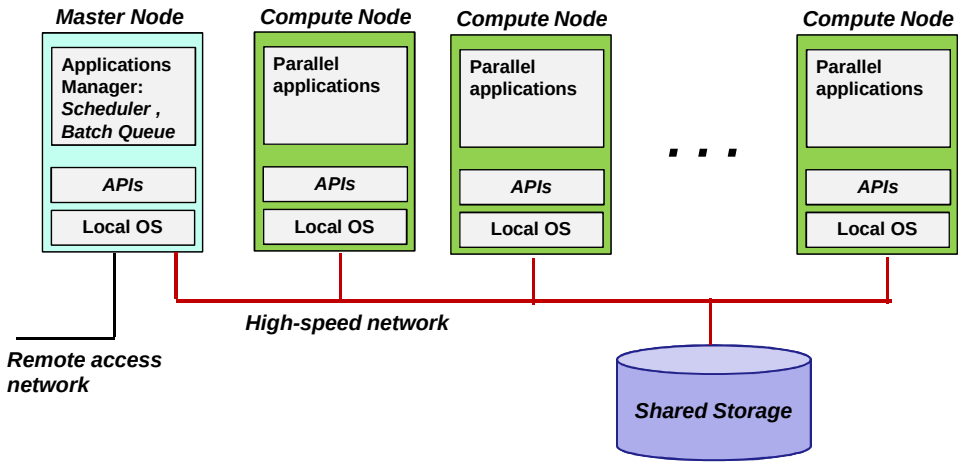


© Extraído de B1

Exemplos:

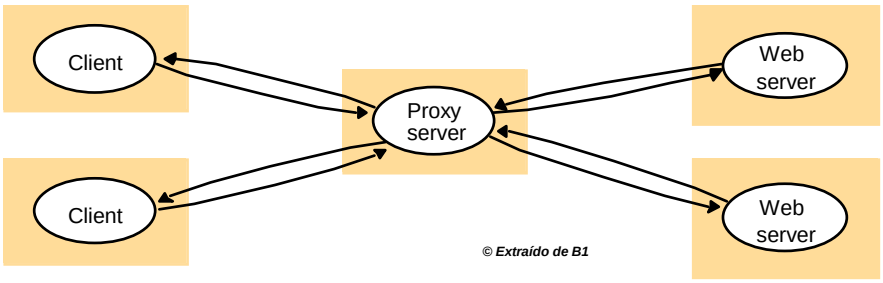
- **2 servidores Web usam um servidor de Base de Dados;**
- **Vários servidores acedem a um servidor de resolução de nomes.**

Cluster Computing



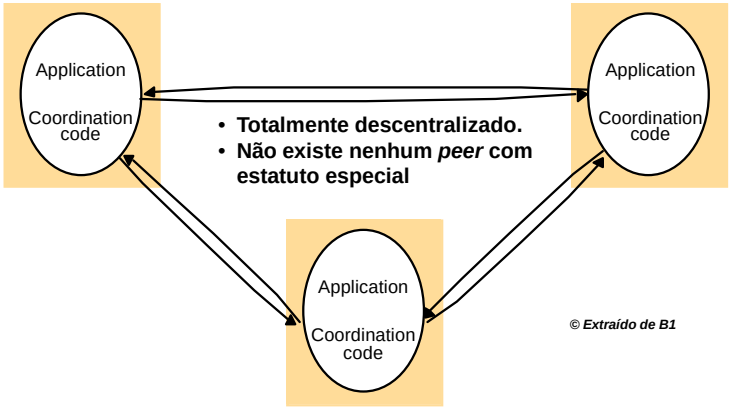
- Linux-based Beowulf clusters (http://www.beowulf.org/)
- MOSIX (http://www.mosix.org/)

Servidores *Proxy*



- **Um servidor *proxy* mantém um *cache* (réplica) de recursos existentes noutros servidores, por exemplo páginas Web, por forma a aumentar a disponibilidade e o desempenho.**
- **Permite gerir a restrição de acessos a determinados serviços/conteúdos**

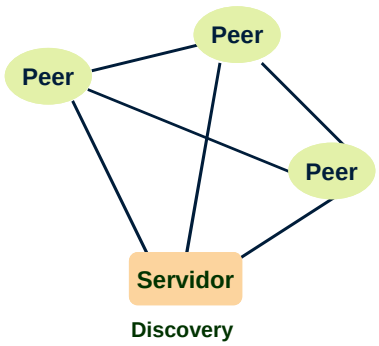
Modelo *Peer to Peer (P2P)*



- **No caso de N *Peers*, cada um pode interactuar com qualquer outro, sendo o padrão de comunicação dependente dos requisitos da aplicação;**
- **Este modelo totalmente descentralizado pode ser extremamente complexo, pelo que se recorre a soluções híbridas para solucionar alguns problemas**

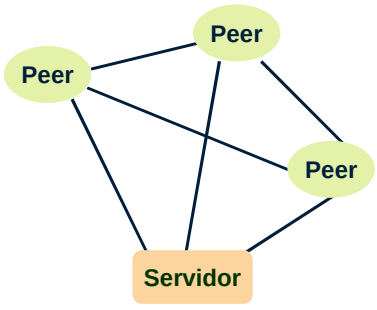
Modelos híbridos *Peer to Peer* (P2P)

P2P com um servidor de *Discovery* (Simple Discovery Server)



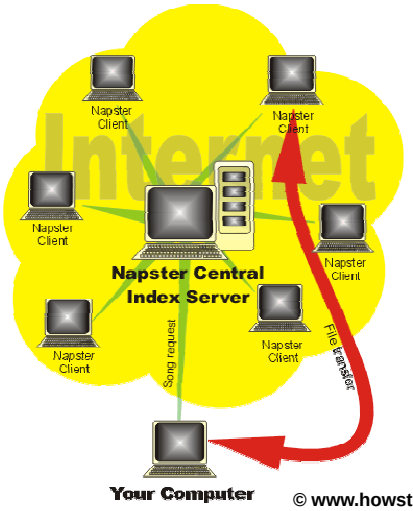
Discovery

P2P com um servidor de *Discovery*, *Lookup* e de conteúdos (persistência)



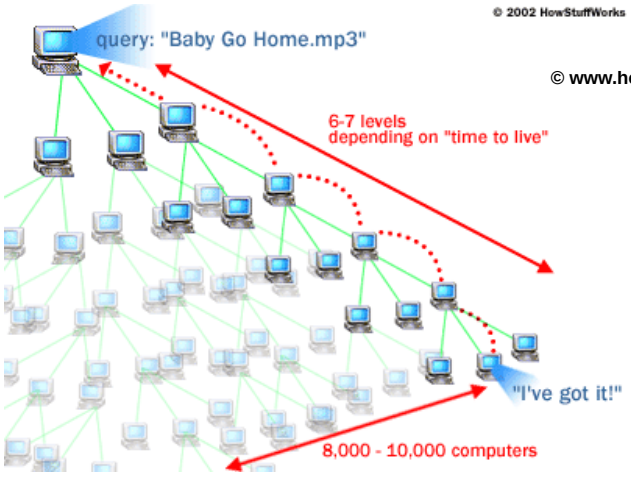
Discovery
Lookup
Content

Modelo híbrido P2P - *Napster*



© www.howstuffworks.com

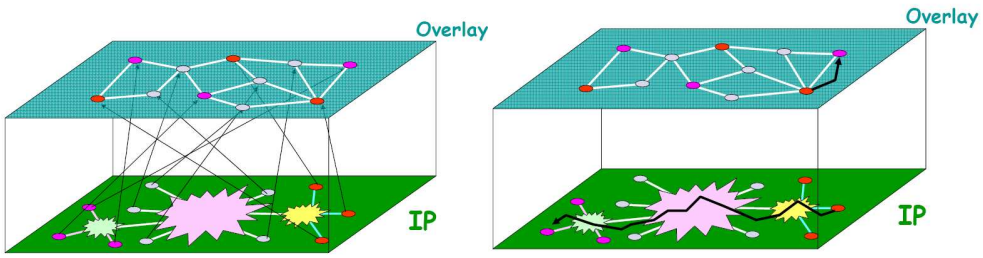
Modelo Puro P2P – *Gnutella; BitTorrent*



© www.howstuffworks.com

Routing Overlay

- **Algoritmo distribuído de localização de *Peers* e Recursos;**
- **Garante que qualquer nó pode aceder a qualquer recurso, fazendo o redireccionamento de cada pedido através de uma sequência de *Peers* vizinhos (mantêm garantia de conectividade frequente)**

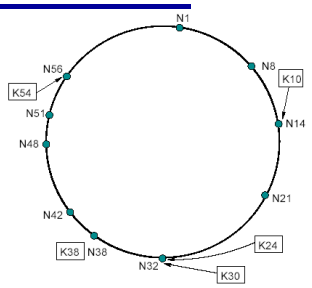


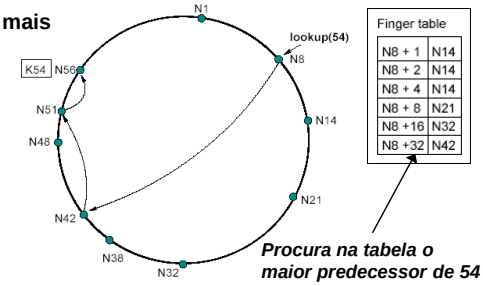
Routing é uma área de investigação ainda em aberto

Chord - Distributed lookup protocol

http://pdos.csail.mit.edu/chord/papers/paper-ton.pdf

- **Dada uma chave K determina qual o nó N que detém o conteúdo;**
- **Uma função de Hash *(SHA-1 (Secure Hash Algorithm)* atribui a cada nó N e a cada chave K um identificador de m bit (160 bit), suficientemente grande para evitar colisões;**
- **A chave k é atribuída ao nó cujo identificador é igual ou, no caso deste nó não existir, o primeiro sucessor mais próximo, no sentido dos ponteiros do relógio.**
- **Para acelerar o processo de *lookup* cada nó tem uma tabela de *routing* *(finger table*) com m (160) entradas;**
- **A entrada i do nó n contém o primeiro nó s, sucessor no anel: s = successor (n + 2^{i-1})**

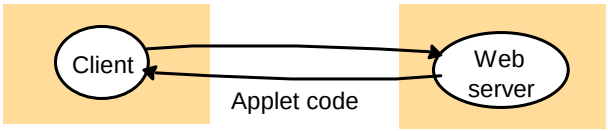




Procura na tabela o maior predecessor de 54

Migração de Código

a) O pedido do cliente resulta no carregamento (*download*) do código de um *applet*

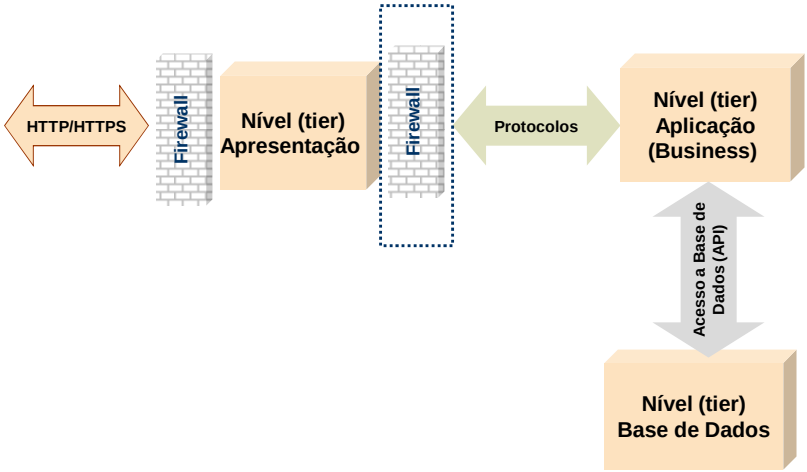


© Extraído de B1

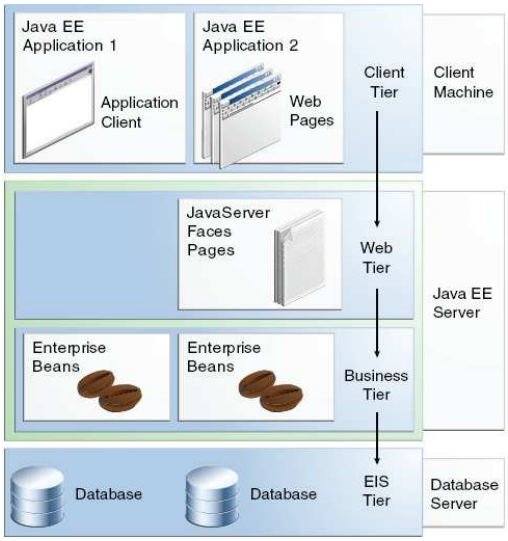
b) O cliente interage com o *applet*

c) Agentes móveis – programa (código e dados) que migra (viaja) entre computadores no cumprimento de uma tarefa. Levanta problemas complexos de segurança e suporte à heterogeneidade.

Arquitecturas de Múltiplos Níveis (n-Tier Computing)

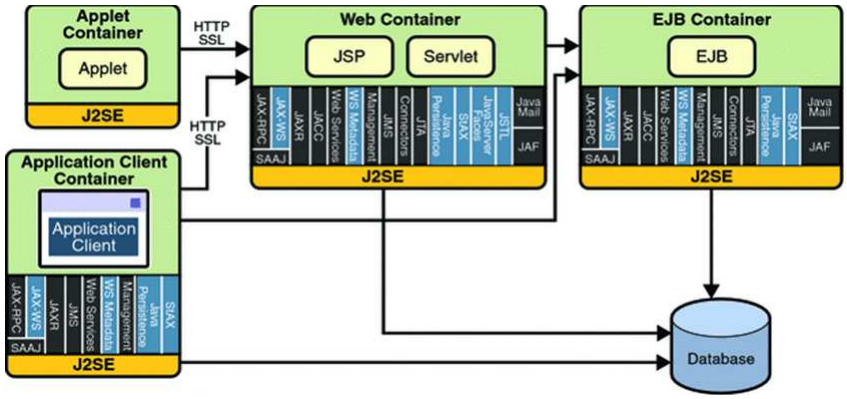


Arquitectura n-Tier – *Java Enterprise Edition (JEE)*



http://download.oracle.com/javaee/6/tutorial/doc/javaeetutorial6.pdf

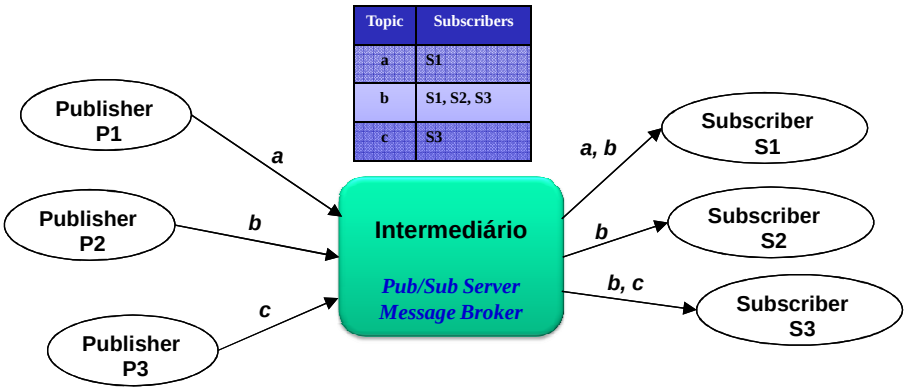
Java Enterprise Edition (JEE) Containers & APIs



Alguns serviços disponíveis num servidor JEE

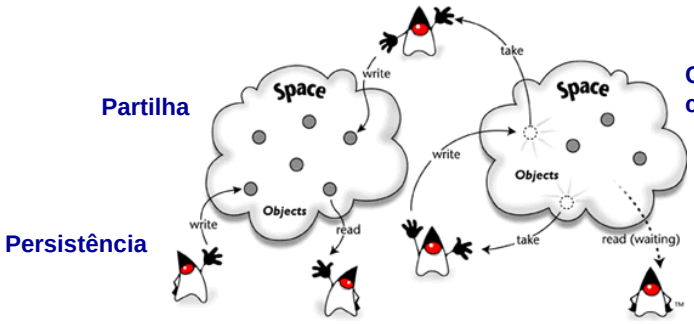
- ***Controlo de acessos (autorização) a componentes;***
- ***Transacções;***
- ***Serviço de nomes/directoria (Java Naming and Directory Interface – JNDI)***
- ***Conectividade remota***

Modelo Publishing/Subscribe



Arquitectura de espaços partilhados - *JavaSpaces*

- **Mecanismo uniforme e dinâmico de comunicação, coordenação e partilha de objectos;**
- **Numa aplicação distribuída um *Space* actua como um espaço virtual partilhado entre fornecedores e consumidores de recursos (objectos);**



Partilha

Concorrência e controlo transaccional

Operações atómicas

Persistência

Acesso associativo

Troca de objectos executáveis

http://java.sun.com/developer/technicalArticles/tools/JavaSpaces/

JavaSpaces - Operações

write: Coloca um objecto (*Entry*) no *Space*.

read: Recebe um *template* e tenta encontrar no *Space* um objecto com o padrão do *template* (acesso associativo). Caso exista um objecto é retornada uma cópia do objecto.

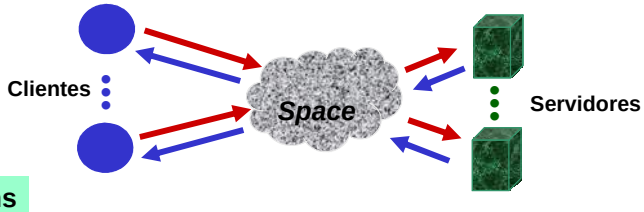
take: Idêntica à operação *read*, excepto que se existir um objecto no *Space* com o padrão do *template*, este é removido.

notify: Pretensão de ser notificado, através de eventos, quando for colocado no *Space* um objecto que respeite o padrão do *template* fornecido.

Outros casos

- IBM Tspaces – Servidor e APIs de acesso do conceito de Tuple Spaces.
  http://www.almaden.ibm.com/cs/TSpaces/
- Blitz JavaSpaces – Implementação open-source do JavaSpaces
  http://www.dancres.org/blitz/blitz_js.html
- SQLSpaces – Implementação do conceito de Tuple Spaces com interface para outras linguagens (C#, Ruby etc.)
  http://sqlspaces.collide.info/
- *MozartSpaces - Implementação em Java (open source) desenvolvido na Universidade de Tecnologia de Viena.*
  http://www.mozartspaces.org/

Utilização em arquitecturas distribuídas



Vantagens

- Não é necessário os clientes e servidores estarem activos em simultâneo;
- Balanceamento de carga e tolerância a falhas implícito. Possível adicionar servidores de forma transparente para os clientes;
- Desenho de aplicações mais simples com persistência, acesso concorrente e transaccional garantidos pelo *Space*;

Desvantagens

- *Overhead* devido à indirecção (acessos ao *Space*) da comunicação entre clientes e servidores;
- *Bottleneck* no acesso ao *Space* ;
- Dificuldade de manter consistência entre múltiplos *Spaces.*

Exemplo com Tspaces - Cliente

Lançar o servidor TSpaces: *\bin\tspaces tspaces.cfg*
TSpaces Server Status: *http://ServerName:8201/*

```
import  com.ibm.tspaces.*;

public class Main {

public static void main(String[] args) {
      String user="luis";
      if (args.length > 0) user=args[0];
      try {
            TupleSpace ts = new TupleSpace("MySpace“, “SpaceServerName”);

             for (int i=0; i < 100;i++)  {
                 Tuple req = new Tuple(user,"Req#"+i, new Integer(i), new Integer(1));
                  ts.write(req);
              }
             for (int i=0; i <100; i++)  {
                  Tuple template = new Tuple(user, "Req#"+i, new Field(Integer.class));
                   Tuple rpy=ts.waitToTake(template);
                   System.out.print(rpy.getField(1).getValue().toString()+":");
                   System.out.println(rpy.getField(2).getValue().toString()+"\n");
              }
      } catch ( Exception t ) {
             System.out.println("erro a abrir o space\n");

      }
   }
}
```

Exemplo com Tspaces - Server

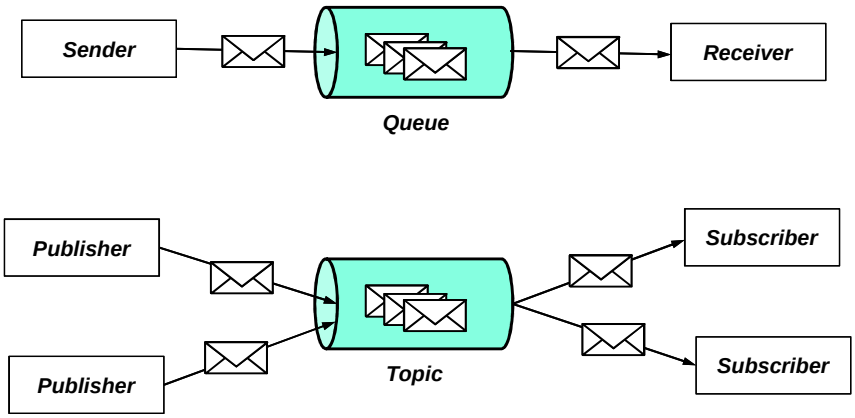
Retira pedidos do *Space*, processa-os e escreve as respostas no *Space*

```
import  com.ibm.tspaces.*;

public class Main {

public static void main(String[] args) {
    try {
        TupleSpace ts = new TupleSpace( "MySpace", "SpaceServerName");

    for (;;) {                          Pedido -> (UserNAme, NumRequest, Op1, Op2)
          Tuple template = new Tuple(new Field(String.class),new Field(String.class),
                                                  new Field(Integer.class),new Field(Integer.class));
          Tuple req=ts.waitToTake(template); //take(template);
          System.out.println("Mais um request from:"+req.getField(0).toString());
          System.out.println(req.getField(1).toString());
          Integer op1=(Integer)(req.getField(2).getValue());
          Integer op2=(Integer)(req.getField(3).getValue());
          Integer res=new Integer(op1.intValue()+op2.intValue());
          Thread.sleep(1000); // simula operação morosa
          Tuple rpy=new Tuple(req.getField(0), req.getField(1), res); // resposta
          ts.write(rpy);

        }
     } catch ( Exception t ) {  System.out.println("erro a abrir o space\n");    }
  }
```


Message Oriented Middleware (MOM)

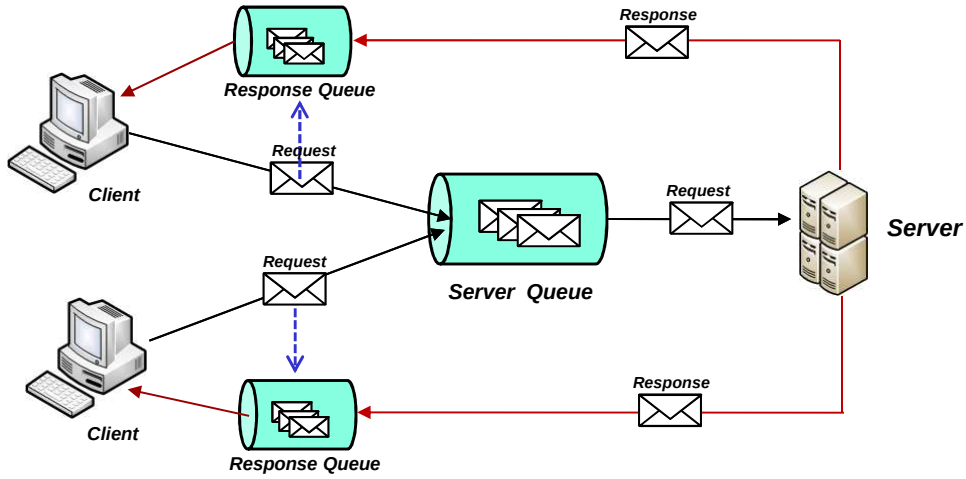
- Camada de software que suporta a troca de mensagens
  - Ponto a Ponto – Ambas as partes têm de estar em execução;
  - Filas de Mensagens – Suporta o desacoplamento entre Produtor e Consumidor das Mensagens:
    - Java Message System (JMS)
    - Microsoft Message Queue (MSMQ)
    - IBM WebSphere MQSeries
    - *Advanced Message Queuing Protocol (AMQP)* - Especificação aberta com várias implementações (Fedora *AMQP Infrastructure; Apache Qpid; RabbitMQ; etc.)*

Modelos baseados em Filas (Queues)



Modelos baseados em Filas (Queues)

- Desacoplamento entre clientes e servidores
- Transacções, tolerância a falhas e balanceamento de carga



Exemplo em MSMQ

```
<?xml version="1.0"?>
<Request   xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
           xmlns:xsd="http://www.w3.org/2001/XMLSchema">
      <reqID>3</reqID>
      <operacao>mult</operacao>
      <op1>10</op1>
      <op2>2</op2>
</Request>
```

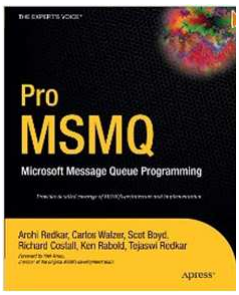
```
[Serializable]
public class Request
{
   public string reqID;
   public string operacao;
   public double  op1;
   public double op2;
}
```

```
<?xml version="1.0"?>
<Response  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
     <reqID>3</reqID>
     <res>20</res>
</Response>
```

```
[Serializable]
public class Response
{
   public string reqID;
   public double res;
}
```

Estudar exemplo: MSMQClientServer.zip

Para aprofundar o estudo de MSMQ



Pro MSMQ: Microsoft Message Queue Programming
Aretha Redkar , Ken Rabold , Richard Costall ,
Scot Boyd , Carlos Walzer

✓Caracterização da Programação Distribuída

Evolução da Programação Distribuída

- Centralizado;
  - Em Rede
    - Cliente Servidor
    - Descentralizado ou Distribuído

As palavras da moda (*Buzzwords*):

- Enterprise Programming;
- Distributed Programming;
- Web enabling
- n-tier Architectures;
- Scalability, fault-tolerance;
- Service Oriented Architectures (SOA)
- Peer-to-Peer (P2P)
  - Grid Computing
  - Cloud Computing

As perguntas Chave:

- Porquê aplicações distribuídas?
- O que é Programação Distribuída?
- Como desenhar aplicações distribuídas?

Caracterização da Programação Distribuída

Necessidade de executar tarefas em várias componentes fisicamente distintas, trabalhando juntas como um sistema único.

A Promessa

Se um computador pode completar uma tarefa em 5 segundos, então 5 computadores em paralelo podem completar essa tarefa num segundo. **?**

O Problema é "trabalhar juntos em paralelo" ?

1. **Decomposição do trabalho em partes que possam ser distribuídas;**
2. **Eficiência das comunicações entre as partes;**
3. **Coordenação do trabalho em cada parte e coordenação global**

1. Decomposição em partes que possam ser distribuídas

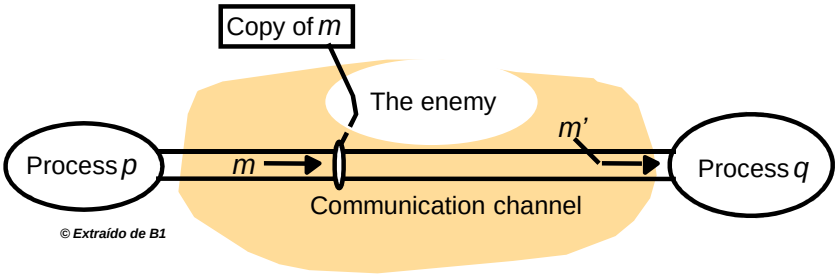
Ao longo dos anos ficou claro que a maioria das aplicações consistem em três partes principais:

a) **Apresentação – Visualização e entrada de dados. Para o utilizador este nível é a aplicação;**
b) **Regras de negócio (lógica) das aplicações – Onde os programadores gastam mais tempo e fazem maior esforço;**
c) **Dados (tipicamente em Base de Dados) – O programador tem de desenhar o modelo de dados, bem como as *queries* para acesso aos mesmos;**

2. Eficiência das comunicações entre as partes

Embora a comunicação entre as partes introduza peso na eficiência dos sistemas distribuídos, com a evolução dos últimos anos deixou de ser o grande desafio no desenvolvimento de sistemas distribuídos;

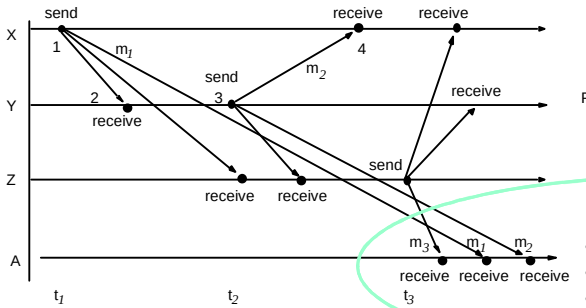
No entanto, a comunicação apresenta desafios de segurança.
Note-se que a existência de comunicações seguras, introduz *overhead*.



© Extraído de B1

3. Coordenação do trabalho em cada parte e coordenação global

A coordenação entre as partes, a existência de estados globais e a ordenação de eventos levantam grandes desafios e complexidade na programação de sistemas distribuídos;



- **m1- mensagem de convocação de uma reunião**
- **m2 e m3 - respostas.**

- **Resposta de Z para uma reunião ?**
- **Marcação de uma reunião por X**
- **Resposta de Y para a reunião**

© Extraído de B1

Necessidade de capturar dependências causais

Interfaces bem definidas

- **A programação distribuída caracteriza-se, assim, por dividir a aplicação em níveis, por exemplo, não misturar regras de negócio com código de apresentação.**
- **Isto não quer significar que cada nível deva executar-se em máquinas separadas ou em processos separados;**
- **O código de um nível só deve interactuar com outro nível através de uma interface bem definida**