Java Cá & Lá

Distributed Multitask Framework

Introdução

* Tradicionalmente: Software Sequenciais (Mercado e Acadêmia);
* Lei de Moore não é mais válida;
* Aumento do número de núcleos nas atuais CPUs;
* Solução? Programação Concorrente!

Programação Concorrente

* **Computação Paralela:** Único processador ou vários processadores em um único equipamento utilizando uma memória compartilhada.
* **Computação Distribuída:** Vários processadores distribuídos por uma rede utilizando memória distribuída (Clusters e Grids).
* **Modelos Híbrido:** Combinação dos modelos anteriores;

Liguagens Concorrentes

* Paralelismo Implícito (Paralelização por conta do compilador);
* Utilização de recursos das linguagens;
  * Threads;
  * Sockets;
* Utilização de Bibliotecas e Frameworks;
  * MPI;
  * Hadoop;
  * Corba;

Dificuldades da Computação Concorrente

* Sincronização e alocação de recursos;
* Overhead da rede;
* Configuração dos recursos;
* Atingir níveis de transparências;
* Portabilidade;
* Desenvolvimento de algoritmos paralelos;

O PROGRAMADOR TEM QUE SER NINJA JEDI!!!

MPI – Message Passing Interface

* O objetivo de MPI é prover um amplo padrão para escrever programas com passagem de mensagens de forma prática, portátil, eciente e flexível.
* No padrão MPI, uma aplicação é constituída por um ou mais processos que se comunicam, acionando-se funções para o envio e recebimento de mensagens entre os processos. Inicialmente, na maioria das implementações, um conjunto fixo de processos é criado.

MPI – Message Passing Interface

Example 4.12 The reverse of Example 4.5, page 158. The root process scatters sets of 100 ints to the other processes, but the sets of 100 are *stride* ints apart in the sending buffer, where *stride* $\geq$ 100. This requires use of MPI_SCATTERV. See Figure 4.8.

```
MPI_Comm comm;
int gsize,*sendbuf;
int root, rbuf[100], i, *displs, *scounts;

...

MPI_Comm_size( comm, &gsize);
sendbuf = (int *)malloc(gsize*stride*sizeof(int));
...
displs = (int *)malloc(gsize*sizeof(int));
scounts = (int *)malloc(gsize*sizeof(int));
for (i=0; i<gsize; ++i) {
    displs[i] = i*stride;
    scounts[i] = 100;
}
MPI_Scatterv( sendbuf, scounts, displs, MPI_INT, rbuf, 100, MPI_INT,
                                                        root, comm);
```

MPI – Message Passing Interface

* Código se torna confuso e complicado;
* Necessário o conhecimento de Sistemas Distribuídos como balanceamento de carga, particionamento de Dados e tolerância à falhas;
* Necessário o uso de ferramentas auxiliares, como o DEINO-MPI;
* Exige recompilação do código a cada modicação e distribuição do executável;
* Na prática signica aprender uma nova linguagem de programação. É um padrão da indústria com várias implementações individuais.

Objetivo

Desenvolver um Middleware para computação paralela (Shared e Distributed Memory) capaz de abstrair todo o conceito de tarefas concorrentes e recursos distribuídos do usuário.

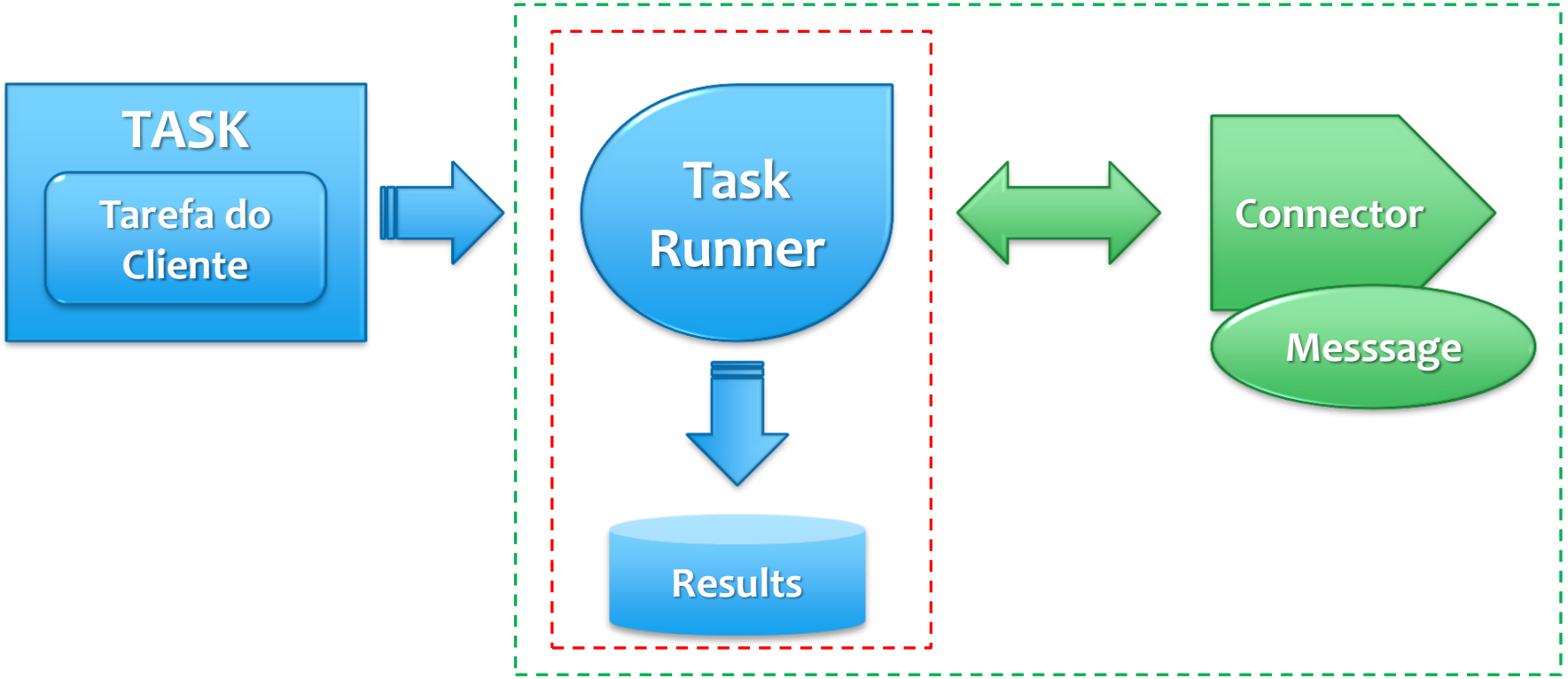
Propriedades

* Máquina Assíncrona Shared e Distributed;
* Máquina Síncrona Shared e Distributed;
* Conector;
* Auditoria;
* Manipulação de Classes;
* Classes todas com Interfaces definidas;
* Biblioteca;
* Servidor Genérico com uso de Workers;

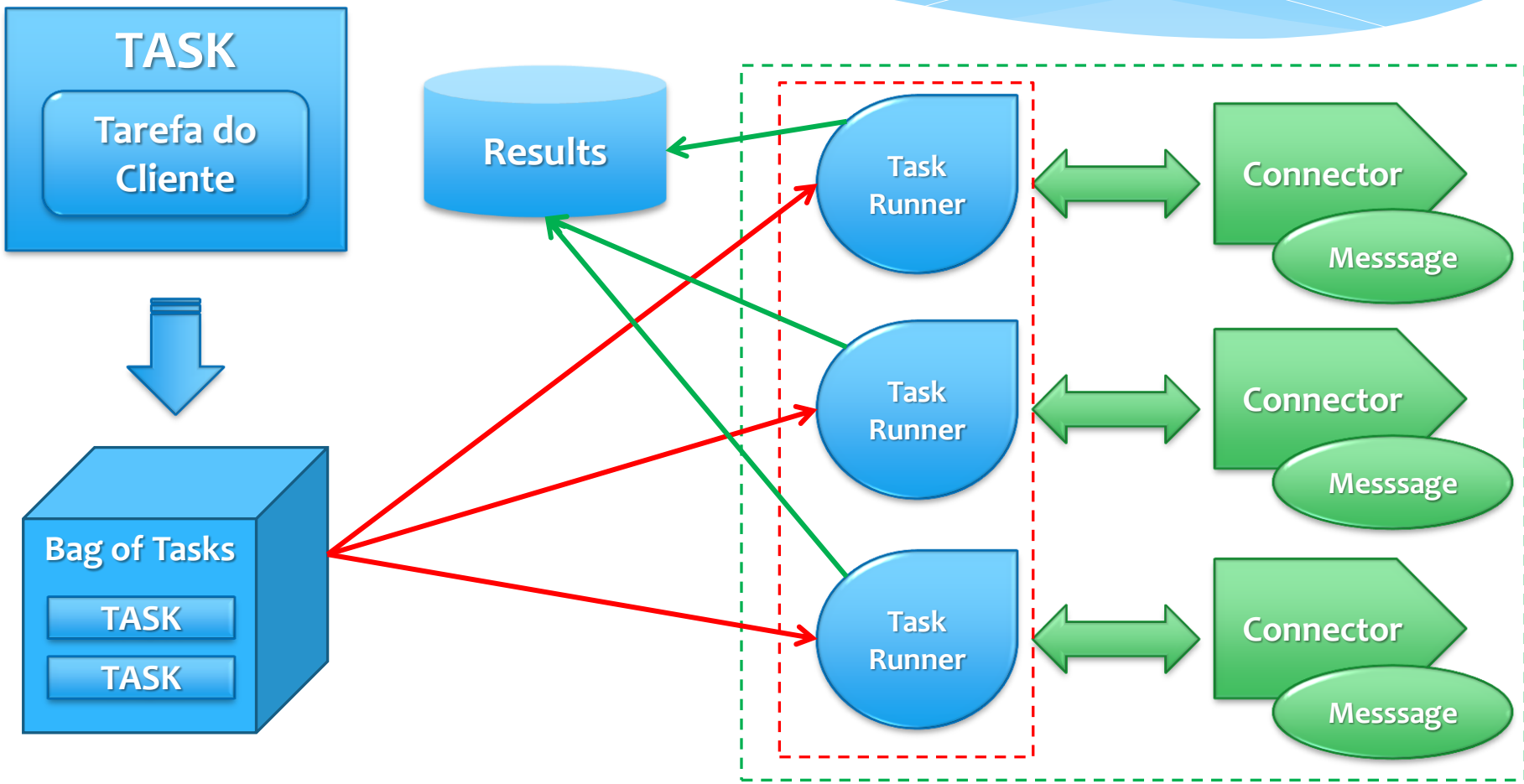
Níveis de Transparência Alcançados

* Transparência de Acesso;
* Transparência de Concorrência;
* Transparência de Localização;
* Transparência de Replicação;
* Transparência à Falha;
* Transparência de Migração;

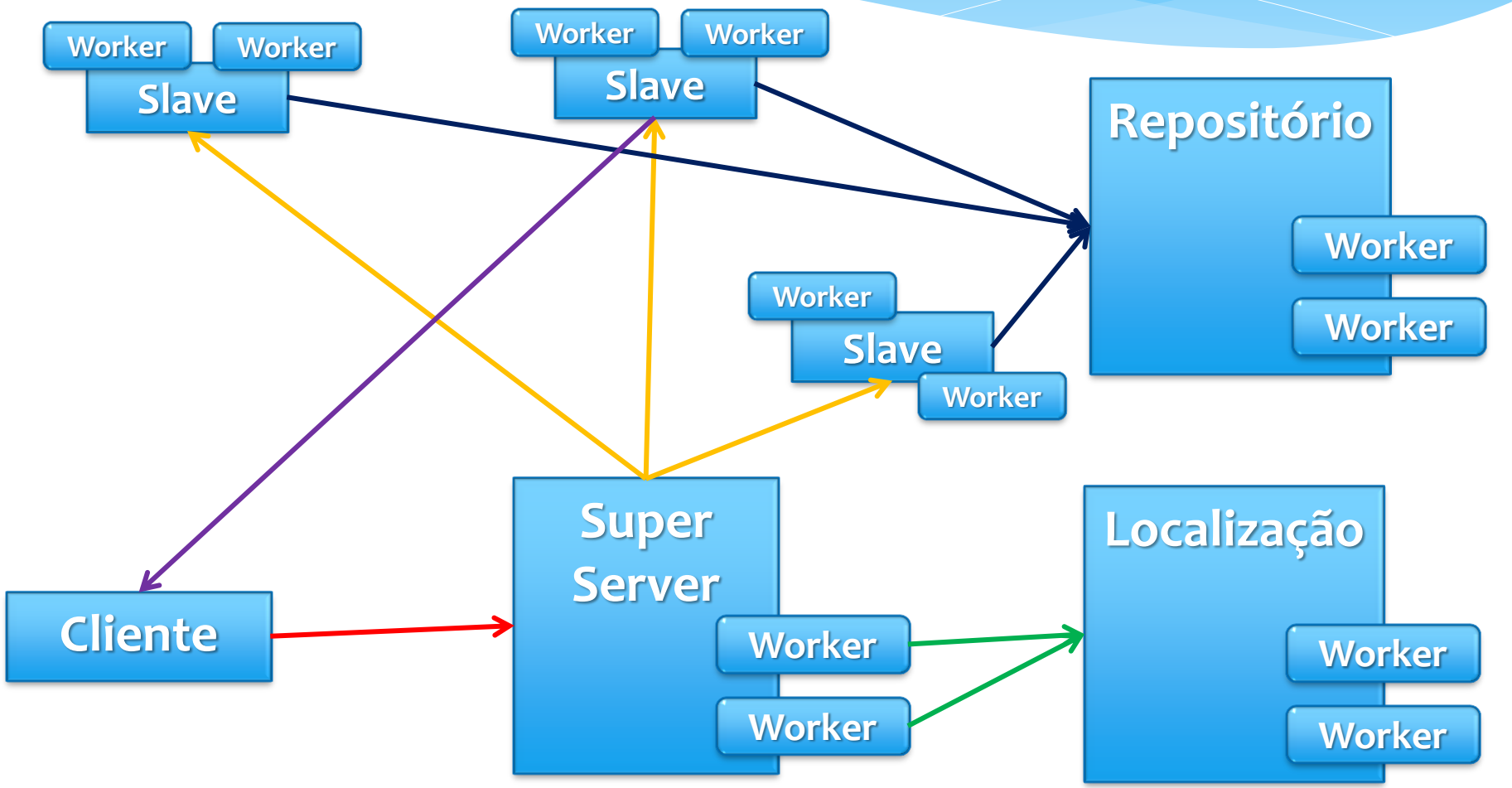
Arquitetura Cliente – Sync



Arquitetura Cliente - Assync



Arquitetura Servidor



Características da Solução

* Arquitetura de Componentes;
* Connectores;
* Migração do Código;
* Localização;
* Nomeação;
* Auditoria por Logging;

Dúvidas