
# Notas dos slides

APRESENTAÇÃO

O presente conjunto de slides pertence à coleção produzida para a disciplina *Introdução ao Processamento Paralelo e Distribuído* ofertada aos cursos de bacharelado em Ciência da Computação e em Engenharia da Computação pelo Centro de Desenvolvimento Tecnológico da Universidade Federal de Pelotas.

Os slides disponibilizados complementam as videoaulas produzidas e tratam de pontos específicos da disciplina. Embora tenham sido produzidos para ser assistidos de forma independente, a sequência informada reflete o encadeamento dos assuntos no desenvolvimento do conteúdo programático previsto para a disciplina.

computação
Universidade Federal de Pelotas

universidade federal de pelotas
CDTec
centro de desenvolvimento tecnológico



# Programação em Ambiente com Memória Distribuída

Dos modelos às estratégias de implementação

> I'll send an SOS to the world
> I'll send an SOS to the world
> I hope that someone gets my
> I hope that someone gets my
> I hope that someone gets my
> Message in a bottle
> Message in a bottle.

The Police

# Notas da videoaula

DESCRIÇÃO

Nesta videoaula são apresentados os principais conceitos associados ao modelo de programação multithread.

OBJETIVOS

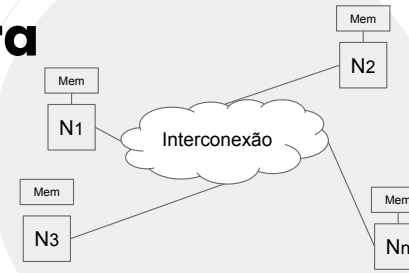
Nesta videoaula o aluno compreenderá a diferença entre as diferentes estratégias para implementação de ferramentas para programação em ambientes com memória distribuída bem como as questões a serem consideradas na implementação de programas em tais ambientes.



# Modelo de Arquitetura

A arquitetura básica é um **multicomputador**:

Um conjunto de nós interligados por uma rede de comunicação que oferece substrato para comunicação entre as partes do programa/aplicação em execução nos diferentes nó



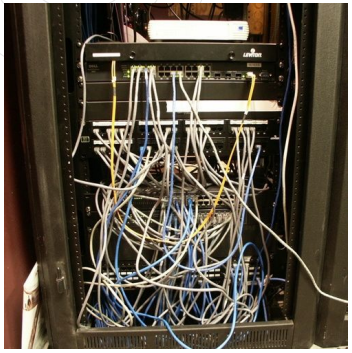



# Umas imagens: Cluster

A arquitetura mais popular para processamento distribuído é o cluster, ou aglomerado de computadores. Trata-se da união de diversos nós de processamento em uma estrutura explorada como um se fosse um computador único. Como se fosse, pois, na verdade, cada nó é um computador autônomo, rodando sua própria instância do sistema operacional. Uma camada de abstração, oferecida por uma biblioteca ou um framework, permite a visão de computador único.

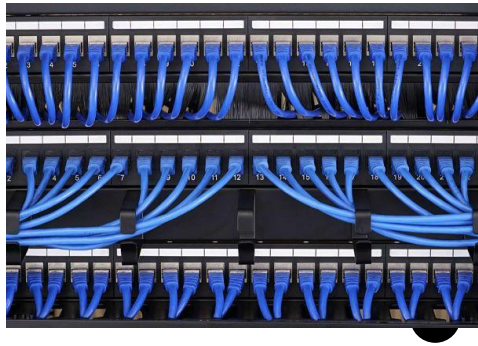


# Umas imagens: Patch Panel



Como inexiste memória compartilhada, toda interação entre as partes de um programa ou aplicação em execução deve se dar via uma estrutura de rede.

O investimento na tecnologia de rede retorna em ganhos de desempenho (e horas de sono, em alguns casos!).



# Modelo de Arquitetura

**Endereçamento**

A comunicação é um processo ativo que exige que as partes envolvidas conheçam-se mutuamente para endereçar mensagens com dados.

**Responsividade**

O nó receptor de uma comunicação deve disparar a execução de um serviço para completar a comunicação iniciada pelo nó que iniciou a comunicação.

**Sincronização**

A comunicação em um ambiente distribuído promove, em algum nível, a sincronização entre as partes envolvidas.




# Modelo de Arquitetura

**Gerência Distribuída**

Os nós envolvidos na computação tem certo nível de autonomia nas decisões locais. Pode haver coordenação global conforme a ferramenta utilizada.

**Latência**

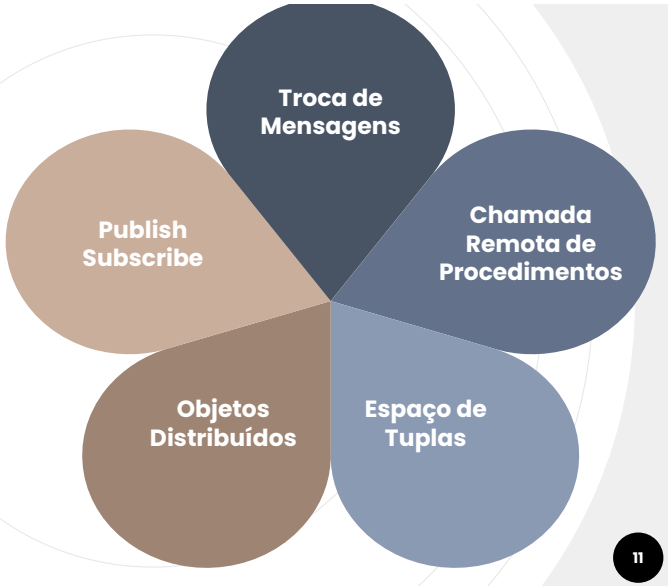
Promover uma comunicação implica em introduzir os tempos, normalmente altos, envolvidos sejam somados ao tempo total de execução.

**Granularidade**

É de se considerar que o volume de dados comunicados e a quantidade de cálculo entre duas comunicações sejam *grandes*.



# Modelos






# Troca de Mensagens

Um processo participa de uma comunicação no papel de origem de dados (*sender*) ou destino (*receiver*).

É necessário o estabelecimento de alguma conexão entre os partícipes para que eles se reconheçam.

Necessidade de endereçamento.

Pelo menos um *sender* e um *receiver* devem estar envolvidos em uma comunicação, mas nada impede comunicações coletivas.

Um protocolo deve ser estabelecido, na camada de aplicação, para dar semântica as mensagens.

# Troca de Mensagens

Tecnologia mais básica, em geral apoiada sobre mecanismos oferecidos pelas camadas de software mais básicas (como *sockets*).

Não oferece muitos recursos de programação, no entanto, oferece flexibilidade e (potencialmente) desempenho.

É conveniente observar que, em algum nível, este mecanismo de comunicação sempre ocorre, pois sobre ele estão assentados os modelos mais abstratos.

Outro aspecto relevante é que a troca de mensagens não é passiva no *receiver*, pois ele deve processar a mensagem recebida para promover a comunicação conforme o protocolo da aplicação.

Troca de Mensagens



# Troca de Mensagens

**Síncrono *vs.* Assíncrono**

Em um extremo, uma troca de mensagens **síncrona** requer que o envio, processado pelo *sender*, seja completado apenas quando o *receiver* completar a respectiva recepção.

Em outro extremo, uma troca de mensagem **assíncrona** implica em que a operação de envio seja considerada completa quando o mecanismo de transmissão assumiu o compromisso do envio propriamente dito. Mesmo que a mensagem não tenha saído do nó onde o *sender* esteja executando.

Troca de Mensagens



# Troca de Mensagens

**Síncrono *vs.* Assíncrono**

Entre estes extremos, é possível encontrar meio-termos, com diferentes níveis de confiabilidade e desempenho.

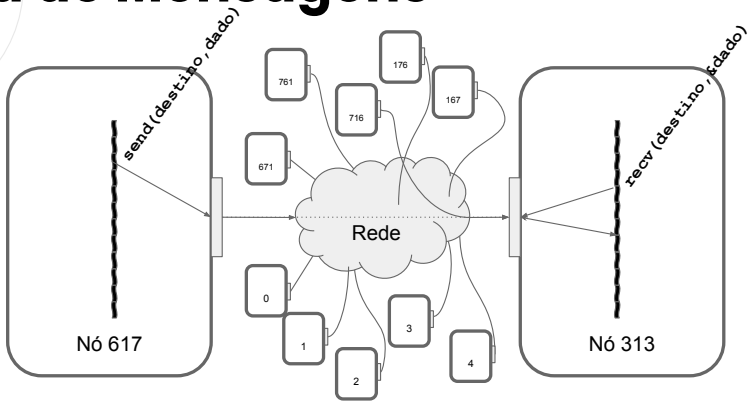
SÍNCRONO

ASSÍNCRONO

Troca de Mensagens



# Troca de Mensagens



# Troca de Mensagens

**Sockets**: API construída sobre TCP, comunicação sobre a Internet

**MPI**: Message Passing Interface, permite criar uma arquitetura virtual sobre uma rede local



Troca de Mensagens

# RPC

Primeira abstração de alto nível para desenvolvimento de programas em ambientes distribuídos: RPC.

Oferece uma interface de programação próxima àquela que os programadores já estavam acostumados: Chamada de Procedimentos, com passagem de parâmetros e retorno de resultados.

Voltado à implementação de aplicações no modelo cliente-servidor.

Um servidor RPC oferece um serviço associado a um endereço de rede.

Não é totalmente transparente.

O cliente precisa conhecer o servidor e a interface (endereço, nome, parâmetros, retorno) do serviço desejado.



Chamada Remota de Procedimentos

# RPC

Não é um padrão de fato, o que dificulta interoperabilidade.

Compatibilidade de dados: XDR

Parâmetros (e retorno) por valor.

RPC Síncrono vs. Assíncrono.

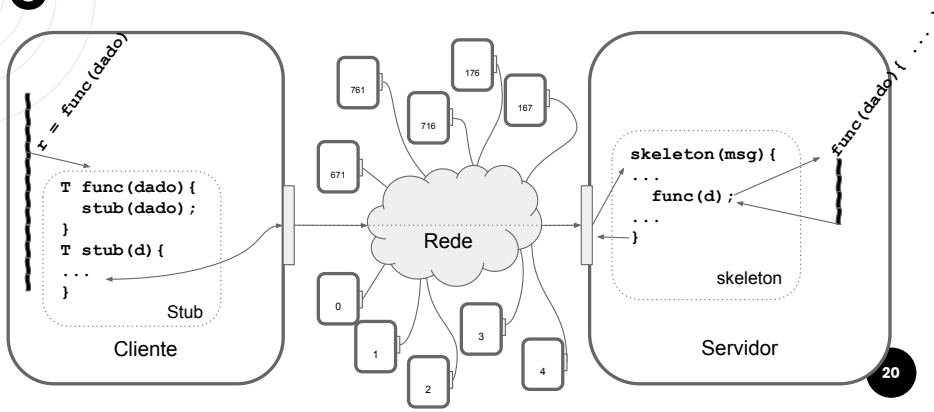
Abstração promovida pelo uso de *stub* e *skeleton*, responsáveis pelo encapsulamento da troca de mensagens do lado do cliente e do servidor, respectivamente.

RPCL - remote procedure call language - suportada pelo rpcgen provê uma interface de alto nível para geração de *stub* e *skeleton*.



Chamada Remota de Procedimentos

# RPC



# Objetos Distribuídos

A visão de rede é escondida do programador, que representa as comunicações por meio de chamadas de métodos de objetos remotos.

Variante de RPC.

Requer conhecimento, da parte do *sender*, do objeto destino.

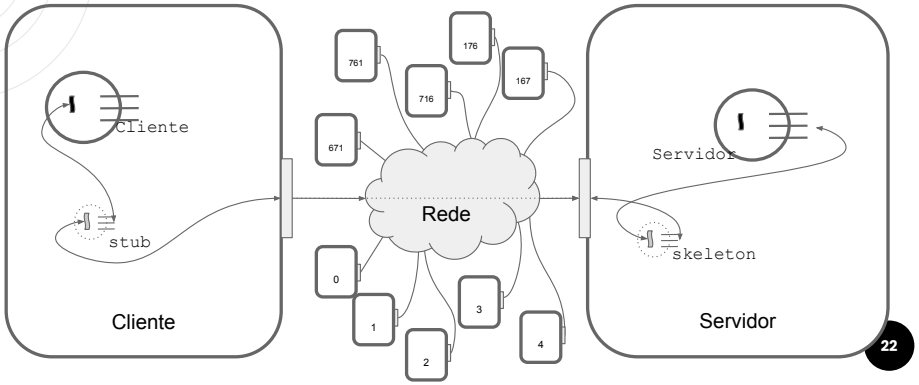
Podem existir um diretório, no qual são listados os endereços dos objetos provedores de serviços.

Grande vantagem: reflete integralmente o modelo Orientado a Objetos, promovendo o desenvolvimento de software em massa.

O desempenho pode não ser dos melhores.




# Objetos Distribuídos






# Objetos Distribuídos

**CORBA**: Common Object Request Broker Architecture

**RMI**: Java Remote Method Invocation




# Publish/Subscribe

Alternativa ao modelo cliente/servidor.

Publicador: fornece/produz a informação

Assinante: requisita/consome a informação.

Evento ou publicação: informação fornecida pelo publicador.

Assíncrono e unidirecional.

Escalável: o processamento do publisher é dedicado a produção de conteúdo e o número de assinantes pode ser adequado à aplicação.

# Publish/Subscribe

Modelo fracamente acoplado onde o publicador não conhece os assinantes, bem como um assinante não sabe quem fornece um conteúdo.

A inscrição de um assinante não é a um publicador, mas sim a uma categoria de conteúdo.

Serviço de Notificação (Canal Pub/Sub) intermedia inscrições e entrega de eventos.

Papel do Serviço de Notificação:

- Receber pedidos de assinaturas a conteúdos
- Manter as informações sobre assinaturas de conteúdo efetivadas
- Receber as notificações dos publicadores
- Encaminhar aos assinantes as informações aguardadas




# Publish/Subscribe

Tolerância a falhas e garantia de entrega devem ser suportadas pelo middleware.

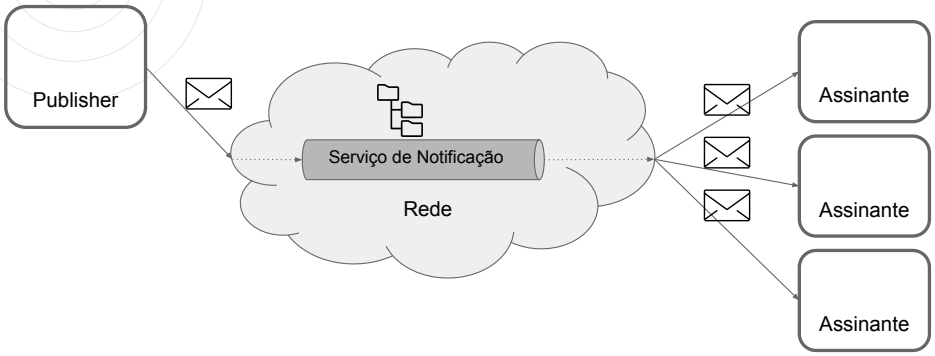
Necessidade de um mecanismo ativo, no assinante, para recebimento das postagens.

Implementação do middleware:

- Multicast: todos assinantes de um assunto compõem um grupo. Solução voltada a uma rede local.
- Broker: utiliza um protocolo de mais baixo nível (como TCP ou HTTP) para encaminhamento das mensagens.
- P2P: permite a inundação dos assinantes com a mensagem sobre o evento.




# Publish/Subscribe






# Publish/Subscribe

**WebSub**: Um protocolo sobre Internet

**RSS**: Recurso para implementação de web feed (agregador de notícias)

**JMS**: Java Message Service (Jakarta Message API_

# Espaço de Tuplas

Modelo de Memória Distribuída.

Tupla: `<chave,dado>`

`chave`: corresponde à identificação (endereço) da informação.

`dado`: é a informação armazenada.

A memória consiste em um grande quadro no qual tuplas são compartilhadas entre os processos da aplicação.

Comunicação essencialmente assíncrona, reflexo do fraco acoplamento entre as partes da aplicação.



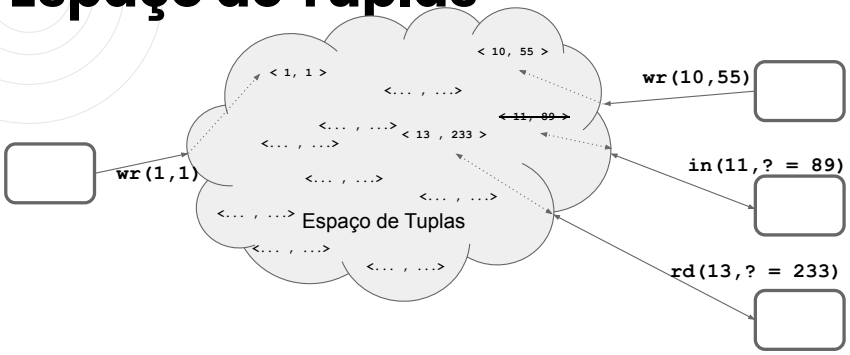

# Espaço de Tuplas

Operações:

- **`RD(chave,?)`**
  - Retorna o **`dado`** associado à **`chave`**
- **`IN(chave,?)`**
  - Retorna o **`dado`** associado à **`chave`**, retirando a tupla do espaço de tuplas
- **`WR(chave,dado)`**
  - Escreve uma nova tupla
- **`EVAL(chaveA,func(<chaveB,?>))`**
  - Escreve uma nova tupla no espaço de tuplas, sendo o **`dado`** o retorno da função **`func`** executada no espaço de tuplas.




# Espaço de Tuplas






# Espaço de Tuplas

**LINDA**: Linguagem de Coordenação que implementa o modelo

**JavaSpace**: API em Java que oferece recurso para operações atômicas sobre o espaço de tuplas

**Obrigado**!